Bidirektionale integrierte Entwicklungsumgebung

Bachelorarbeit

Jan-Philipp Schramm
Matrikelnummer: 4516495

1. Gutachter: Prof. Dr. Rainer Koschke
2. Gutachter: Dr. Olaf Bergmann

Fachbereich 3: Mathematik und Informatik

14. Februar 2022

Selbstständigkeitserklärung

Ich versichere, den Bachelor-Report oder den von mir zu verantwortenden Teil einer Gruppenarbeit[1] 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 14.02.2022	
Ort, Datum	Unterschrift

[1] Bei einer Gruppenarbeit muss die individuelle Leistung deutlich abgrenzbar und bewertbar sein und den Anforderungen entsprechen.

Danksagung

Als allererstes möchte ich Prof. Dr. Rainer Koschke meinen Dank für die Betreuung während meiner Bachelorarbeit aussprechen. Für die vielen hilfreichen Tipps und Anregungen, die ich während des Erstellens meiner Bachelorarbeit erhalten habe.
Auch meiner Familie möchte ich meinen besonderen Dank dafür aussprechen, dass sie mich während meines Informatikstudiums immer unterstützt hat.
Zum Schluss möchte ich mich bei allen bedanken, die an meiner Studie teilgenommen haben und somit mitgeholfen haben, meine Forschungsfrage zu beantworten. Auch bei allen anderen, die mir während der Erstellung meiner Bachelorarbeit geholfen haben, gilt dieser Dank.

Abstract

Softwarevisualisierung hilft bei der Beurteilung von Softwarequalität eines Projektes. So wird von der Arbeitsgruppe Softwaretechnik die Software SEE entwickelt, um Quellcode eines Softwareprojektes als eine „Stadt“ darzustellen. In dieser Bachelorarbeit wird die Anwendung SEE so erweitert, dass eine Interprozesskommunikation mit einer IDE ermöglicht werden kann. Für die Erweiterung und somit der Kommunikation mit einer IDE wird Visual Studio verwendet.
Das Interesse bei dieser Implementierung liegt darin herauszufinden, ob die Interprozesskommunikation für den Endbenutzer einen Unterschied macht. Genauer gesagt, ob sie einen Mehrwert für Softwareanalyse mit einer IDE bringt. Dafür wird eine empirische Benutzerstudie durchgeführt, in der die Bearbeitung von Aufgaben auf Bearbeitungszeit, Benutzerfehler, Gebrauchstauglichkeit und Zufriedenheit untersucht wird. Bei der Bearbeitung der Aufgaben gibt es zwei Durchläufe, einmal mit und einmal ohne die neue Erweiterung. Die Durchführung der Studie mit 10 Probanden ergab, dass die geschriebene Erweiterung gerade bei den subjektiven Faktoren (Zufriedenheit und Gebrauchstauglichkeit) einen deutlichen Mehrwert bietet.

Anmerkung

In dieser Abschlussarbeit wird aus Gründen der besseren Lesbarkeit bei personenbezogenen Hauptwörtern das generische Maskulinum verwendet. Alle anderen Geschlechter werden selbstverständlich durch diese Ausdrucksweise eingeschlossen.

INHALTSVERZEICHNIS

KAPITEL 1
Einleitung

Diese Abschlussarbeit widmet sich der Kommunikation zwischen einer Integrierte Entwicklungsumgebung (IDE) und einer Anwendung zur Softwarevisualisierung. Einleitend möchte ich dafür meine Motivation, die Zielsetzung zusammen mit der Forschungsfrage sowie den Aufbau dieser Arbeit genauer erläutern.

1.1 Motivation

Um die Softwarequalität zu wahren, gibt es verschiedene Herangehensweisen. Eine Möglichkeit stellt die Softwarevisualisierung dar, welche die grafische Veranschaulichung verschiedener Aspekten der Software ermöglicht. Darunter fallen die Visualisierung der statischen Struktur, Ausführung des Quellcodes sowie Evolution über die Zeit [vgl. Die03, S. 257]. Im Speziellen geht es in dieser Abschlussarbeit um die Visualisierung von Software in Form einer „Stadt“ als Metapher – eine sogenannte Code-City. Das Ziel dieser Darstellung ist dabei die Verständlichkeit eines Softwareprojektes zu erhöhen.
Eine solche Visualisierung bietet das Projekt Software-Engineering-Experience (SEE). Dieses wurde von der Arbeitsgruppe Softwaretechnik an der Universität Bremen entwickelt und besitzt neben der Visualisierung noch Möglichkeiten zur kollaborativen Betrachtung eines Softwareprojektes. Dabei stellt sich jedoch die Frage, wie sich die Softwarevisualisierung besser in die Softwareentwicklung integrieren lassen kann. Als Ansatz dafür ist die bidirektionale Integration der Softwarevisualisierungsanwendung in eine IDE möglich. Wenn ein Entwickler zum Beispiel eine Klasse, an der gerade gearbeitet wird, mit andern vergleichen möchte, kann eine bereitgestellte Option in der IDE ein Hervorheben in der Visualisierung ermöglichen. Umgekehrt sind mehrere Klassen auffällig dargestellt in der Visualisierung. Hier könnte durch die Auswahl der Klasse, diese in der IDE geöffnet werden.

1.2 Zielsetzung und Forschungsfrage

Ziel dieser Abschlussarbeit wird es konkret sein, eine nahtlose Möglichkeit der Interaktion zwischen einer Softwarevisualisierungsanwendung und einer Integrierte Entwicklungsumgebung zu entwickeln. Als Grundlage für diese prototypische Erweiterung wird die IDE Visual Studio dienen sowie die Anwendung SEE als Visualisierungsprogramm. Die Anforderungen für die Erweiterung sind wie folgt definiert:

1. Die Interprozesskommunikation zwischen SEE und Visual Studio implementieren.
2. Beide Anwendungen erweitern, damit folgendes ermöglicht wird:
    - Öffnung von Visual Studio mit geladenem Projektordner sowie gewünschten Quellcode
    - Das Springen zu einer gewünschten Zeile im Quellcode

- Das Hervorheben von Knoten in SEE mithilfe von Visual Studio
- Abhängigkeiten mithilfe von Visual Studio visualisieren

3. *Optional:* Das gemeinsame Betrachten eines Desktops soll über die Anwendung SEE möglich gemacht werden.

Nachdem die Gestaltungsziele umgesetzt worden sind, muss die entstandene Implementierung evaluiert werden. Die Evaluation bezieht sich dabei auf den erhaltenen Mehrwert einer solchen Erweiterung. Die Abschlussarbeit beruht somit auf der Forschungsfrage: *„Bietet die bidirektionale Integration von SEE in eine IDE einen Mehrwert für den Benutzer?“*. Dafür wurde eine Benutzerstudie erstellt, in der die objektiven und subjektiven Eigenschaften der Usability einer Anwendung getestet worden. Wichtig anzumerken ist, dass es sich hierbei um eine vergleichende Studie zwischen zwei verschiedene Interaktionsarten handelt, um die Softwarequalität eines Projektes zu beurteilen. Ob die bidirektionale Kommunikation einen Mehrwert besitzt, wird durch den Vergleich der genannten Faktoren bestimmt. Näheres findet sich in Kapitel 5.

1.3 Aufbau der Abschlussarbeit

Nachdem die Einleitung kurz die Beweggründe und Vorgehensweise für meine Bachelorarbeit dargelegt hat, möchte ich die Grundlagen für ein besseres Verständnis der späteren Implementierung genauer erläutern. Darin enthalten ist die Vorstellung der verwendeten Software, eine Begriffserklärung für den entfernten Zugriff auf einen Desktop sowie ähnliche Forschungsprojekte. Vor der Implementierung gibt es noch den Entwurf, in der meine Lösungsidee zum Problem geschildert wird. Im nächsten Kapitel folgt dann die finale Implementierung sowie die Probleme, die bei der Implementierung entstanden sind. Nachdem der Aufbau und die Funktionsweise der Erweiterung abgeschlossen ist, folgt der wichtigste Teil: Die Evaluation, um meine in Abschnitt 1.2 vorgestellte Forschungsfrage zu beantworten. Das beinhaltet den kompletten Aufbau sowie Durchführung der Evaluation, bevor zum Schluss die Ergebnisse aufgezeigt werden. Abschließend für diese Abschlussarbeit folgt dann das Fazit.

KAPITEL 2
Grundlagen

In diesem Kapitel möchte ich die Grundlagen für ein besseres Verständnis der Abschlussarbeit erklären. Das bedeutet, dass alle Technologien, Software und Bibliotheken, die für die Erstellung der Erweiterung benötigt wurden, dargelegt werden. Zu jeder Software, die nicht bereits im bestehenden Projekt SEE vorhanden war, gibt es noch eine kleine Erklärung für die Wahl dieser spezifischen Software. Zum Schluss folgt noch die Nennung zu Forschungsarbeiten, die einen ähnlichen Ansatz verfolgt haben. Eine Verlinkung zu den jeweiligen Internetseiten ist in den Fußnoten zu finden.

2.1 Unity

Seifert und Wislaug fassen die Anwendung Unity[1] gut zusammen. Bei Unity handelt es sich um eine Entwicklungsumgebung für die Erstellung von 2D- und 3D-Anwendungen. Dabei ist der Hauptfokus ein Werkzeug für die Spielentwicklung bereitzustellen. Alternativ zur Spielentwicklung findet Unity beispielsweise für die Architekturvisualisierung Verwendung. Durch Unity können Anwendungen für eine Vielzahl von Plattformen erstellt werden. Darunter fallen typische Desktop-Betriebssysteme wie Windows, Mac, Linux, aber auch IOS, Android und andere Plattformen unterstützt Unity [vgl. SW17, S. 1 ff.].
Dabei wird während der Entwicklung auf unterschiedliche Konzepte gesetzt. Unity Technologies zufolge ist das wichtigste Konzept bei der Entwicklung einer Anwendung mit Unity das *GameObject*. Jedes Objekt innerhalb einer Szene ist ein *GameObject*, wie zum Beispiel Bäume, Licht und vieles mehr. Allerdings spiegelt ein Objekt nur eine Hülle ohne Funktion wider. Um einem Objekt nun Funktionen zu verleihen, müssen Komponenten hinzugefügt werden, welche diese Funktionalitäten implementieren [vgl. Tec22b]. Diese Komponenten sind in C# geschriebene Skripts, welche von einer speziellen Klasse namens *MonoBehaviour* erben [vgl. Tec22a].

2.2 SEE

Das Programm Software-Engineering-Experience (SEE)[2] ist eine mit Unity entwickelte Anwendung zur Visualisierung von Software. Wie schon in Abschnitt 1.1 erklärt, setzt SEE auf die Visualisierung eines Softwareprojektes mithilfe einer Code-City. Diese stellt Klassen oder Methoden als Gebäude dar, welche über die Farbe, Größe und Breite verschiedene Metriken angeben können. So kann eine Klasse die besonders viele Code-Zeilen besitzt rot dargestellt werden, wo hingegen eine Klasse mit weniger Zeilen eher als weiß dargestellt wird. Es gibt auch die Möglichkeit der Darstellung von Softwareerosion. Dies alles ermöglicht dem Softwareentwickler eine gute Übersicht über den aktuellen Qualitätsstatus der eigenen Software zu gewinnen.

[1] https://unity.com/ (besucht am 18.01.2022)
[2] https://see.uni-bremen.de/ (besucht am 18.01.2022)

SEE ist für Windows, Linux, Mac OS verfügbar und kann mithilfe verschiedener Interaktionsmöglichkeiten bedient werden. Darunter fallen klassisch die Bedienung mit Maus und Tastatur, aber auch die Verwendung von VR ist möglich. Außerdem können, falls gewünscht, mehrere Teilnehmer gleichzeitig dieselbe Code-City betrachten. In den nachfolgenden Kapiteln befinden sich einige Bilder, die die Anwendung SEE zeigen.

2.3 Visual Studio und Erweiterung

Wie bereits im Abschnitt 1.2 erwähnt, wird eine prototypische Erweiterung für eine IDE erstellt. Dabei fiel die Wahl auf die IDE Visual Studio[3], da diese auch während der Entwicklung des Softwareprojekts SEE eine wichtige Rolle spielt. Bei Visual Studio handelt es sich um eine von Microsoft entwickelte IDE. Diese stellt viele Funktionalitäten, wie Debuggen, kompilieren und bearbeiten von Software bereit [vgl. Mic21e].
Eine Erweiterung von Visual Studio fügt neue Funktionen hinzu. In der Dokumentation sind diese unter den Namen Befehle (Commands) bekannt. Mit Befehlen können anschließend Menüs oder Toolbars erweitern werden. Um einen Befehl zu erzeugen, wird ein *MenuCommand* oder *OleMenuCommand* mit einem eigenem Event-Handler erstellt und registriert [vgl. Mic21a]. Um den Befehl jetzt für den Benutzer zugänglich zu machen, muss dieser an eine sinnvolle Stelle platziert werden. Dafür ist ein sogenanntes Visual-Studio-Command-Table zuständig. Dies ist eine XML-Datei, welche die Befehle, die eine Erweiterung für Visual Studio erstellt, beinhaltet [vgl. Mic21d]. In Abbildung 2.1 ist Visual Studio 2019 mit einem Beispielprojekt zu sehen.

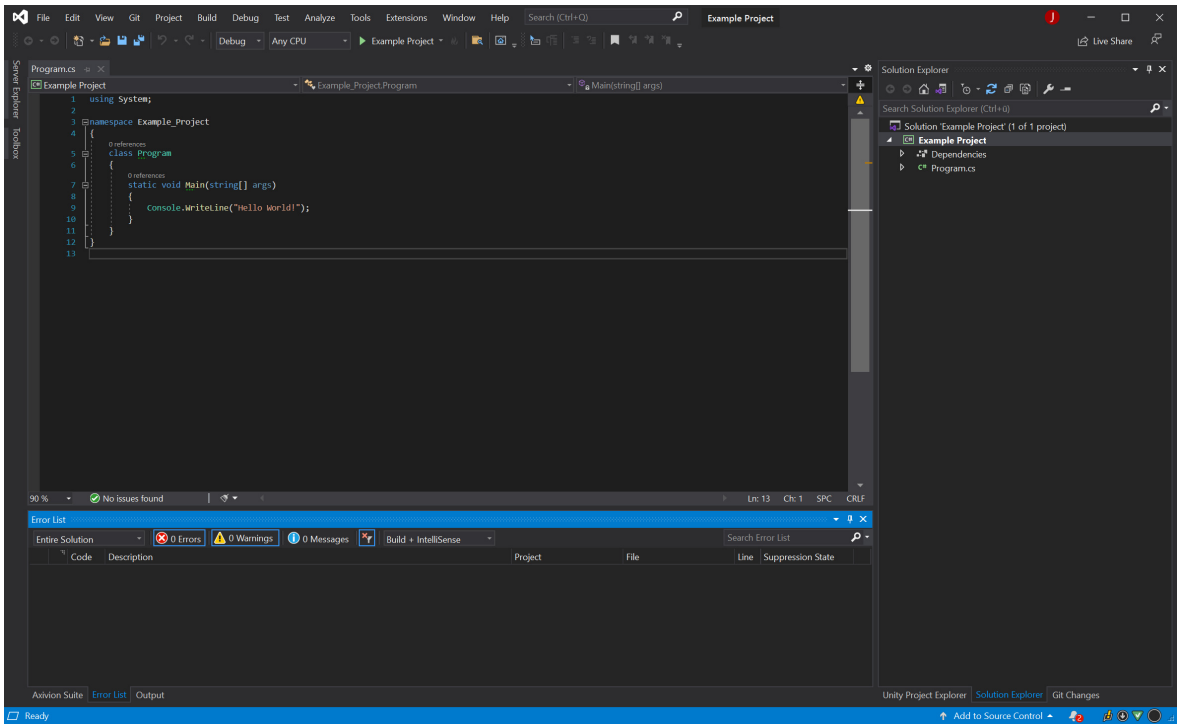

Abbildung 2.1: Visual Studio 2019

[3]https://visualstudio.microsoft.com/ (besucht am 20.01.2022)

2.4 Visual Studio Locator

Ein Problem, was nach Microsoft aufgrund von Änderungen an den letzten Versionen von Visual Studio entstanden ist, ist das Finden der Programmpfade. Um diesem Problem aus dem Weg zu gehen, hat Microsoft das Projekt vswhere entwickelt. Dieses stellt das Programm *„vswhere.exe“* zur Verfügung, welches durch Programmvariablen Anweisungen zur gesuchten Version sowie den genau benötigten Informationen gegeben werden kann [vgl. Micb]. Ein Beispiel für einen Befehl wäre:

```
vswhere.exe -version [16.0,17.0} -format value -property productPath
```

Dadurch erhält der Benutzer den Pfad zur *„devenv.exe“* von Visual Studio 2019, welche das Starten einer neuen Instanz von Visual Studio ermöglicht.

2.5 Interprozesskommunikation

Die Interprozesskommunikation beschreibt den Austausch von Daten zwischen verschiedenen Prozessen. Dabei werden die einzelnen Prozesse normalerweise als Client oder Server kategorisiert, wobei viele Anwendungen auch als beides agieren [vgl. Mic21b]. Mandl nennt dafür drei verschiedene Schnittstellen [vgl. Man14, S. 199 f.]:

- **Pipes:** Sind Kommunikationskanäle, die eine Datenstrom nur in eine Richtung transportieren können. Eine vom Prozess erstellte Pipe wird nach Terminierung beider Gesprächspartner wieder geschlossen. Eine besondere Form stellen die Named Pipes dar. Diese erhalten, wie der Name schon sagt, einen Namen und können über die Lebenszeit eines Prozesses hinaus weiterexistieren.
- **Message Queues:** Grenzt die Daten in Nachrichtenstücke ein und stellt diese in einer Nachrichtenwarteschlange den Prozessen zur Verfügung.
- **Sockets:** Eine Möglichkeit der Interprozesskommunikation, in der auch Prozesse über ein Netzwerk miteinander kommunizieren können. Grundlage dafür sind die Netzwerkprotokolle UDP oder TCP. Prozesse verwenden zur Identifizierung des Sockets IP-Adressen sowie Portnummern. Aufbauend auf Sockets kann Remote Procedure Call (RPC), ein Mechanismus zum Aufrufen entfernter Prozeduren, verwendet werden.

In den Unterabschnitten sind die verwendeten Technologien für die Implementierung der Kommunikation zwischen SEE und Visual Studio aufgelistet.

2.5.1 TCP-Socket

Für den Austausch von Informationen habe ich mich für den TCP-Socket entschieden. Der Hauptgrund, weswegen ich mich gegen eine Benutzung von Pipes entschieden habe, ist deren Inkompatibilität mit Anwendung, die mithilfe von Unity entwickelt wurden. So haben Beispielsweise Named Pipes im Mono Framework, welches Unity für C# verwendet, keine Unterstützung in der 64-Bit-Version einer Anwendung [vgl. Pro]. Für die Kommunikation wird ein Server erstellt, der auf *localhost* laufen wird. Die Informationen sind in Form eines Datenstroms zu versenden, dabei wird dieser Datenstrom in C# über einen *NetworkStream* geregelt [vgl. Mica].

2.5.2 StreamRpc

Der in Unterabschnitt 2.5.1 vorgestellte Kommunikationskanal zwischen zwei Anwendungen benötigt für eine einfache Interpretation von Informationen noch eine weitere Technologie.

Diese nennt sich StreamRpc[4] und ermöglicht das entfernte Aufrufen (RPC) von Methoden in anderen Programmen. Somit ist eine einfachere Interprozesskommunikation möglich, die keiner aufwendigen Interpretierung von übergebenen Daten bedarf.
StreamRpc ist ein Fork von dem Projekt StreamJsonRpc[5], welches vom JSON-RPC Wire Protocol Gebrauch macht und als Ziel hat, die Software-Abhängigkeiten auf ein Minimum zu reduzieren [vgl. Sha]. Bei der Kommunikation wird das Datenaustauschformat JSON verwendet, um Funktionen in der gewünschten Anwendung ausführen zu können. JSON selbst ist eine Syntax, die die Grundlagen für einen einfachen programmiersprachenunabhängigen Datenaustausch erbringt [vgl. Int17, S. 1].

2.6 VNC

Um die Funktion für den Zugriff auf einen entfernten Computer hinzuzufügen, wird das sogenannten Virtual-Network-Computing (VNC) verwendet. VNC ist ein Plattformunabhängiges Protokoll, welches den Zugriff auf den kompletten Desktop eines Computers über das Netzwerk ermöglicht. Dabei können auch mehrere Systeme gleichzeitig auf denselben Desktop zugreifen [vgl. Ric+98, S. 33 f.]. Anwendungen, die dieses Protokoll einbinden, sind zahlreich vorhanden und können von Endanwender frei gewählt werden.

2.7 Ähnliche Forschungsprojekte

Abschließend zu der Vorstellung der verwendeten Software innerhalb dieser Arbeit möchte ich noch zwei weitere Projekte vorstellen. In diesen wurde ein ähnlicher Ansatz wie in dieser Abschlussarbeit verfolgt.

CodeMetropolis: Eclipse over the City of Source Code In dieser wissenschaftlichen Ausarbeitung präsentieren Balogh, Szabolics und Beszédes eine Erweiterung zur Integrierung des Projektes CodeMetropolis[6], eine Software zur Generierung einer Code-City eines Softwareprojektes im Videospiel Minecraft, in der IDE Eclipse. Eclipse erhält dabei Funktionen, um die Erstellung einer Code-City zu veranlassen und auch zum Navigieren in der generierten Code-City. Als Hauptverwendungszweck für die Integration wurde die Verwendung zu Lehrzwecken und zum Einarbeiten in ein neues Projekt angeführt [vgl. BSB15].

Embedding Spatial Software Visualizationin the IDE: an Exploratory Study Alternativ zu der Verwendung von einer Code-City zur Visualisierung bietet sich das Projekt von Kuhn, Erni und Nierstrasz an. Diese haben eine Erweiterung für Eclipse entwickelt, die eine räumliche Visualisierung von Software direkt in der IDE ermöglicht. Dabei wird die Software Codemap integriert. Anhand dieser Integrierung wurde anschließend eine Benutzerstudie durchgeführt, um den Nutzen festzustellen. So wurde durch die Erweiterung ein guter Eindruck über Menge und Verbreitung erzielt, jedoch wurde die gewählte Art der Visualisierung oft als verwirrend empfunden [vgl. KEN10].

[4] https://github.com/shana/StreamRpc (besucht am 27.12.2021)
[5] https://github.com/microsoft/vs-streamjsonrpc (besucht am 27.12.2021)
[6] http://codemetropolis.github.io/CodeMetropolis/about/ (besucht am 10.02.2022)

KAPITEL 3
Entwurf

In diesem Kapitel geht es um die Vorstellung meiner Lösungsidee zu den in Abschnitt 1.2 definierten Zielsetzungen dieser Arbeit. Dafür wird zunächst eine Unterteilung in SEE und Visual Studio unternommen, damit dort eine genauere Erläuterung der jeweiligen Erweiterung geschehen kann. Dabei wird die Erweiterung für Visual Studio hier und im Folgenden VsSeeExtension genannt. Der Entwurf zielt, aufgrund der Verwendung von Visual Studio, nur auf die Implementierung für das Betriebssystem Windows ab. Die Implementierung soll dabei aber leicht für die Unterstützung von anderen Betriebssystem erweiterbar sein.

3.1 SEE

Aus den in diesem Unterabschnitt erklärten Ideen zur Umsetzung soll anschließend eine Komponente entstehen. Für die Aktivierung soll ein Entwickler diese Komponente an ein beliebiges *GameObject* hängen.

3.1.1 Anpassung der Oberfläche

Für eine Erfüllung der in Abschnitt 1.2 gegebenen Anforderungen ist die Anpassung der Oberfläche geplant. Genauer gesagt die Oberfläche des Code-Windows, die in Abbildung 3.1 zu sehen ist. Dieses kann der Benutzer durch das anklicken eines Knotens in der Code-City öffnen, wenn in den Modus „Show Code“ gewechselt wird. Das geöffnete Fenster zeigt den Quellcode an sowie Name der Klasse/Methode des ausgewählten Knotens.

SetupVR (PlayerSettings.cs)

```
215     {
216         if (Ground == null)
217         {
218             throw new InvalidOperationException("A Ground must be assigned in the PlayerSettings. Use the Inspector.");
219         }
220         else
221         {
222             // Create Teleporting game object
223             PrefabInstantiator.InstantiatePrefab("Prefabs/Players/Teleporting").name = "Teleporting";
224             {
225                 // Attach TeleportArea to floor
226                 // The TeleportArea replaces the material of the game object it is attached to
227                 // into a transparent material. This way the game object becomes invisible.
228                 // For this reason, we will clone the floor and move the cloned floor slightly above
229                 // its origin and then attach the TeleportArea to the cloned floor.
230                 Vector3 position = Ground.transform.position;
231                 position.y += 0.01f;
232                 GameObject clonedFloor = Instantiate(Ground, position, Ground.transform.rotation);
233                 clonedFloor.AddComponent<TeleportArea>();
234             }
235             {
236                 // Assign the VR camera to the chart manager so that charts can move along with the camera.
237                 GameObject chartManager = GameObject.Find(ChartManagerName);
238                 if (chartManager)
239                 {
240                     ChartPositionVr chartPosition = chartManager.GetComponentInChildren<ChartPositionVr>();
241                     if (chartPosition)
242                     {
243                         chartPosition.enabled = true;
244                         chartPosition.CameraTransform = player.GetComponentInChildren<Camera>().transform;
```

Abbildung 3.1: Das vorhandene Code-Window in SEE

Wie in Abbildung 3.1 zu sehen ist, befindet sich auf der linken Seite der Leiste ein Icon mit dem Namen des gewählten Objekts sowie wie deren dazugehörige Datei. Auf der rechten Seite ist noch viel Platz, weshalb dort ein neuer Button zum Öffnen einer Datei oder starten von Visual Studio platziert werden soll. Da das Code-Window als Prefab (Vorlage) gespeichert ist, sollte eine Erweiterung ebenfalls problemlos funktionieren.

3.1.2 Hervorheben der Knoten und Kanten

Innerhalb einer Code-City stellen die Knoten unter anderem Methoden, Klassen und Namespaces dar. Die Knoten können ihre Zugehörigkeit zueinander durch die Darstellung von Kanten zeigen. Ein kleines Beispiel ist in Abbildung 3.2 zu sehen. Durch die Erweiterung soll Visual Studio diese Elemente einer Code-City hervorheben können. Aktuell können diese Objekte sowohl durch den Benutzer, als auch durch weitere Benutzer, wenn SEE über ein Netzwerk erreichbar ist, hervorgehoben werden. Diese bereits bestehende Möglichkeit der Selektion sollte nun so erweitert werden, dass auch Visual Studio diese Funktion nutzen kann. So wird dem Benutzer das gewünschte Element, welches in der IDE ausgewählt wurde, in SEE hervorgehoben.

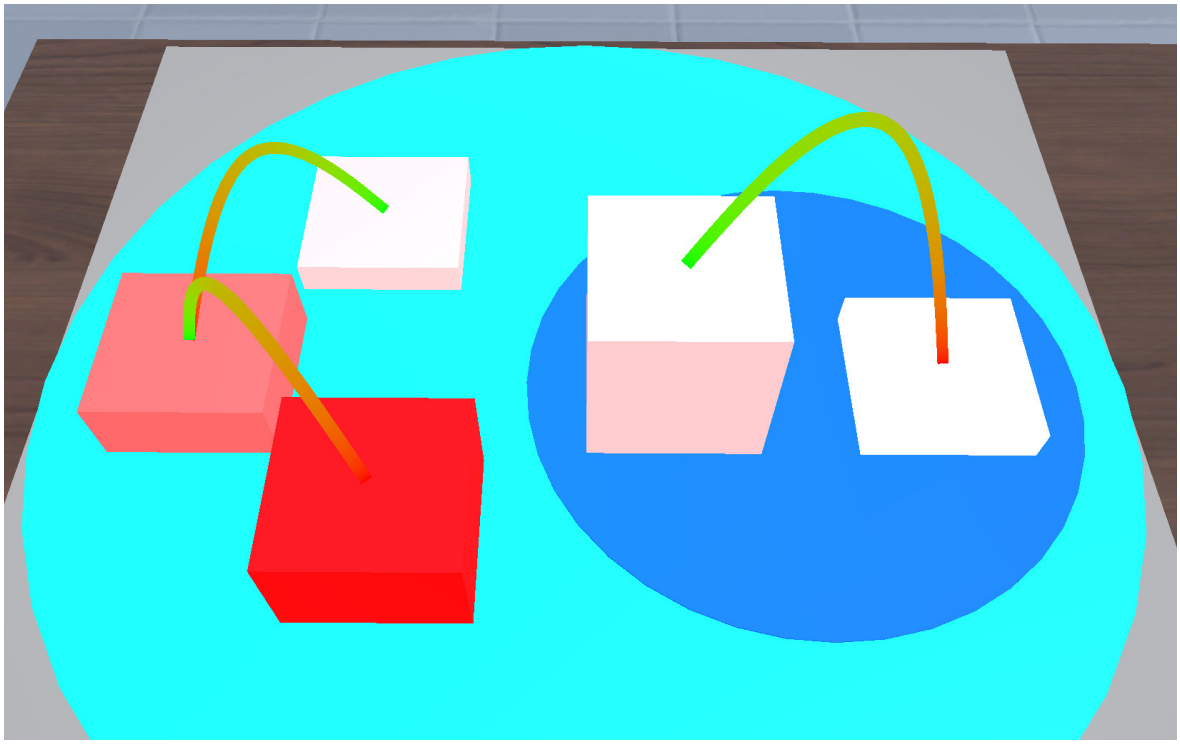

Abbildung 3.2: Beispiel für eine Code-City

3.1.3 Kommunikation mit Visual Studio

Dieser Abschnitt stellt den Hauptteil meiner Implementierung dar. Dabei handelt es sich hierbei um die gesamte Kommunikation sowie Verwaltung der Instanzen von Visual Studio. Die Kommunikation sollen bereits vorhandener Technologien der Interprozesskommunikation ermöglichen. Das heißt, dass auf Sockets zum Austausch von Daten sowie StramRpc zum Interpretieren dieser gesetzt wird. StramRpcs Aufgabe ist das Methodenaufrufen im jeweils anderen Prozess zu ermöglichen. Dafür benötigt StreamRpc ein Objekt, in dem es die Methoden aufrufen kann [vgl. Mic20]. Da die Kommunikation nach dem Client-Server-Modell

stattfinden wird, wäre es am sinnvollsten SEE als Server für die gesamte Kommunikation zu wählen. Vor allem, weil mehrere Instanzen von Visual Studio gleichzeitig laufen können, wäre die Verwendung einer IDE als Server ungeeignet. Das liegt daran, dass ein Server eindeutig identifizierbar sein muss, da es ansonsten zu Problemen mit der Erstellung führen kann. Eine Mehrfachausführung von SEE ist zwar ebenfalls möglich, aber da die Anwendung mit einer festgelegten Code-City und Einstellungen kompiliert wird, wäre eine Mehrfachausführung nicht sinnvoll. Zudem soll, wenn gewünscht, einer weiteren Instanz von SEE vor der Erstellung ein alternativer Port zugewiesen werden können. Wenn noch keine Instanz von Visual Studio auffindbar ist, da noch keine initiiert wurde, sollte das automatische öffnen beim Klicken des in Unterabschnitt 3.1.1 hinzugefügten Knopfs erfolgen.
Aufgrund der Wahl von SEE als Server, muss dieser auch alle Verwaltungsaufgaben für die Interprozesskommunikation übernehmen. Das bedeutet, wenn eine neue Instanz von Visual Studio sich Verbinden möchte, muss SEE das geladene Projekt überprüfen. Dieses sollte das gleiche wie bei der geladenen Code-City sein. Dementsprechend sollte SEE auch beim Wechseln des Projektes in Visual Studio die Verbindung trennen.

3.1.4 Entfernter Zugriff

Der entfernte Zugriff soll durch VNC ermöglicht werden. Konkret ist vorgesehen einen neuen Modus zu implementieren, der das Herstellen einer Verbindung zu einem Host ermöglicht. Mit Host ist der Computer gemeint, der eine von SEE externe Software installiert hat, mit der das Bereitstellen und Interagieren mit dem Desktop durch einen anderen Rechner ermöglicht wird. Der neu hinzugefügte Modus in SEE ist also nur zum Herstellen einer Verbindung zu einem bestehenden Host angedacht. SEE selbst wird keine Funktionen bereitstellen, um einen Bildschirm direkt zu teilen. Wenn nun der Benutzer eine Verbindung starten will, wird ein Webbrowser innerhalb von SEE geöffnet (im neu hinzugefügten Modus), in dem ein clientloses Plugin geladen wird. Bei diesem Plugin handelt es sich um eine Software, die das Interagieren mit dem Host-Desktop über eine Webbrowser ermöglicht.

3.2 Visual Studio

Das resultierende Ergebnis aus diesem Entwurf soll die neue Erweiterung VsSeeExtension sein. Der Endanwender erhält dabei eine ausführbare Datei, die die neuen Funktionen zu Visual Studio hinzufügt.

3.2.1 Anpassung der Oberfläche

Für eine gute Benutzererfahrung müssen die neuen Befehle an passenden Stellen platzierte werden. Das bedeutet, die VsSeeExtension wird in den diversen Menüs von Visual Studio neue Einträge erstellen. So sollte generell unter den Erweiterungen das Verändern des Konnektivitätsstatus möglich sein. Wenn der Benutzer ein Element innerhalb des Editors hat, welches hervorgehobene werden soll, soll ein rechter Mausklick das Kontextmenü hervorbringen, in dem sich ein neuer Eintrag mit einer entsprechenden Aktion befindet. Dabei sollte Visual Studio nur die Option bereitstelle, wen das Kontextmenü im Editor innerhalb des Elements geöffnet wurde. Mit andern Worten, wenn das Kontextmenü innerhalb einer Methode geöffnet wurde, so kann dort der Eintrag zum Hervorheben dieser angezeigt werden. Neben dieser Möglichkeit eine Aktion in SEE auszuführen, gibt es ebenfalls noch eine Erweiterung des Kontextmenü der Tab-Ansicht sowie der Projekt-Ansicht. Dort sollen Aktionen ausgeführt werden, die die gesamte Klasse betreffen.

3.2.2 Kommunikation mit SEE

Wie bereits in Unterabschnitt 3.1.3 genannt, handelt es sich bei SEE um die Implementierung des Servers der bidirektionalen Kommunikation. Jede geöffnete Instanz von Visual Studio wird dementsprechend als Client mit ähnlicher Technologie operieren. Dieser Client soll die Optionen bekommen sich entweder automatisch mit dem Server zu verbinden oder dies dem Benutzer als manuelle Option zu überlasen. Die automatische Verbindung soll den Umgang mit der Erweiterung vereinfachen. Falls es jedoch gewollt ist, keine Verbindung einzugehen, soll das automatische Verbinden deaktiviert werden können. Für den Fall, dass SEE selbst eine Visual Studio-Instanz starten möchte, sollte SEE ein Flag übergeben. Damit soll SEE ein automatisches Verbinden erzwingen, auch wenn die Einstellungen von Visual Studio etwas anderes besagen. Als Client muss die IDE keine Verwaltung von Verbindungen übernehmen, sollte jedoch Informationen an SEE zur Veränderung des Projektstatus weiterleiten. Innerhalb dieser Erweiterung muss ebenfalls StreamRpc ein Objekt erhalten, in denen alle für SEE aufrufbaren Methoden enthalten sind.

3.2.3 Datei öffnen

Wenn SEE ein Signal gesendet hat, also eine entsprechende Methode in Visual Studio aufgerufen hat, so soll diese zum Öffnen der angeforderten Datei führen. Das bedeutet Visual Studio wird einen neuen Tab für die entsprechende Datei öffnen. Damit diese geöffnet werden kann, muss auf die Services von Visual Studio zurückgegriffen werden. Da SEE generell die Möglichkeit gibt unterschiedliche Arten von Knoten innerhalb einer Code-City darzustellen, reicht ein einfaches Öffnen der Datei nicht aus. Dem Benutzer soll die Suche nach dem gewünschten Element innerhalb der Datei nicht selbst überlassen sein. Dementsprechend soll Visual Studio zur Zeile, die SEE bei seinem Aufruf mitgeliefert hat, springen und diese zur besseren Kenntlichmachung auch markieren. Somit kann von SEE aus mit Angabe der Datei und der Zeile eines Elements, dasselbige in Visual Studio hervorgehoben werden.

KAPITEL 4
Implementierung

In diesem Kapitel gehe ich auf die genaue Implementierung, die nach den Lösungsideen in Kapitel 3 entwickelt wurden, ein. Dabei wird auf die in Kapitel 2 vorgestellten Technologien zurückgegriffen. Auch findet eine Aufteilung der Abschnitte in SEE und Visual Studio statt. Die Unterabschnitte unterscheiden sich aber teilweise von denen im Entwurf, da eine Unterteilung in weitere Unterabschnitte den Sachverhalt klarer darstellt. Aufgrund der Überschätzung der benötigten Zeit fällt die in Abschnitt 1.2 vorgestellte optionale Anforderung weg.

4.1 SEE

Da es sich bei dem Projekt SEE um eine Anwendung handelt, die mit Unity entwickelt wurde, wird für die Erweiterung C# verwendet. Während der Entwicklung habe ich die Spielentwicklungsumgebung Unity und die IDE Visual Studio verwendet.

4.1.1 Server und StreamRpc

Wie in Unterabschnitt 2.5.1 erklärt, soll die Implementierung des Servers mit einem TCP-Socket erfolgen. Eine Verwendung von Named Pipes ist dabei ausgeschlossen. Dennoch wurde im ursprünglichen Konzept die Verwendung verschiedener Kommunikationskanäle einbezogen. Deshalb habe ich eine abstrakte Klasse *JsonRpcServer* geschrieben, die alle benötigten Funktionen bereitstellt. Die Implementierung der konkreten Kommunikationstechnologie, welche den Datenstrom beinhaltet, findet in der erbenden Klasse statt. Der *JsonRpcSocketServer* ist die konkrete Server-Implementierung, welche Verwendung vom TCP-Socket macht. Auch wenn es potenziell die Möglichkeit gibt andere Technologien für die Interprozesskommunikation zu verwenden, so sollte der TCP-Socket für die Kommunikation zwischen SEE und Visual Studio vollkommen ausreichen.
Für eine bessere Verwaltung aller Informationen zu einer spezifischen Verbindung lagere ich diese in die Klasse *JsonRpcConnection* aus. Auch hier handelt es sich um eine abstrakte Klasse, damit eine spezifische Server-Implementierung ihr Kommunikationsobjekte direkt dieser Verbindung zuordnen kann. So speichert der TCP-Socket-Server die einzelnen Client-Verbindungen in seiner konkreten Implementierung der *JsonRpcConnection* ab. Insgesamt wird dadurch eine unkomplizierte Verwaltung aller verbunden Clients erreicht. *JsonRpcConnection* stellt Events bereit, bei denen sich ein Server registrieren kann. Diese Events verwendet die Klasse, um auf die Veränderung des Verbindungsstatus hinzuweisen. Wenn eine Verbindung erfolgreich war, so wird sie der internen Menge aller Verbindungen hinzugefügt, andernfalls entfernt. Neben den bereits genannten Informationen speichert die Verbindung ebenfalls die StreamRpc-Instanz. Innerhalb der *JsonRpcConnection* kann dieser StreamRpc-Instanz ein Objekt, dass von einem Client aufgerufen wird, hinzugefügt werden. Das hinzugefügte Objekt wird so für den entfernten Prozeduraufruf verwendet.
Der Server kann so erstellt werden, dass sich nur eine bestimmte Anzahl an Clients mit diesem Verbinden kann. Dafür sendet der Server zu Beginn eines Verbindungsversuches dem Client

ein Steuersignal. Mit diesen Signalen gibt der Server an, ob schon die maximale Anzahl an Clients mit dem Server verbunden sind, oder die Verbindungsanfrage akzeptiert wird. Bei erfolgreichen Verbinden löst die Server-Implementierung ein Event aus und überreicht allen Abonnenten des Events die dazugehörige *JsonRpcConnection*. Gleiches gilt ebenfalls für die Trennung der Verbindung. Wenn SEE eine Prozedur in einem verbundenen Client ausführen möchte, so muss eine entsprechende Methode in *JsonRpcServer* gewählt werden. Dabei gibt es die Auswahl in allen verbundenen Clients eine spezifische Methode aufzurufen oder diese in nur einem Client aufzurufen. Für das Aufrufen muss allerdings der Name für die aufzurufende Methode sowie die nötigen Parameter übergeben werden.

4.1.2 Verwaltung der IDE

Im Namespace *Controls* habe ich aufgrund der benötigten Verwaltung von der bidirektionalen Integration die Komponente *IDEIntegration* hinzugefügt. Diese erbt von *MonoBehaviour*, da diese Klasse als Komponente an ein beliebiges *GameObject* in der Szene angefügt werden soll. Somit ist die Komponente auch im Unity-Editor verwendbar und konfigurierbar.
In der *IDEIntegration* befindet sich eine Server-Implementierung der abstrakten Klasse *JsonRpcServer*. Im Falle der Kompatibilität mit Visual Studio wird der *JsonRpcSocketServer* verwendet. Die Erstellung dieser Komponente sollte aber nur einmal in einer Szene geschehen, da Mehrfacherstellung des Servers zu Problemen führen kann. *IDEIntegration* ist bei sämtlichen Events, die den Server betreffen, registriert und handelt beim Aufkommen dieser entsprechend.
Wann immer eine IDE versucht sich mit SEE zu verbinden, folgt zunächst eine Abfrage der Version der IDE sowie das aktuell geöffnete Projekt. Wenn diese mit der von SEE aktuell gewünschten IDE übereinstimmen, folgt die Etablierung der Verbindung. *IDEIntegration* speichert die Verbindung dann in einem *Dictionary* mit dem Pfad des Projekts. Der Benutzer erhält anschließend eine Benachrichtigung, dass die IDE sich erfolgreich verbunden hat. Wenn es zum Abbruch der Verbindung kommt, entfernt *IDEIntegration* diese, nachdem der Server dies bekannt gegeben hat, aus dem internen Speicher und teilt dem Benutzer mithilfe einer Benachrichtigung mit, dass SEE sich von der IDE getrennt hat.
Zum Vergleichen des Projekts mit dem geöffneten in der IDE muss SEE den Pfad zum gesuchte Projekt erhalten. Dies geschieht über eine neue Variable, die in *AbstractSEECity* hinzugefügt wurde. Die *AbstractSEECity* beinhaltet Informationen über die Code-City. Außerdem habe ich diese Variable unter dem Namen *Solution file* im Insepctor von Unity editierbar gemacht, wie in Abbildung 4.1 zu sehen ist.

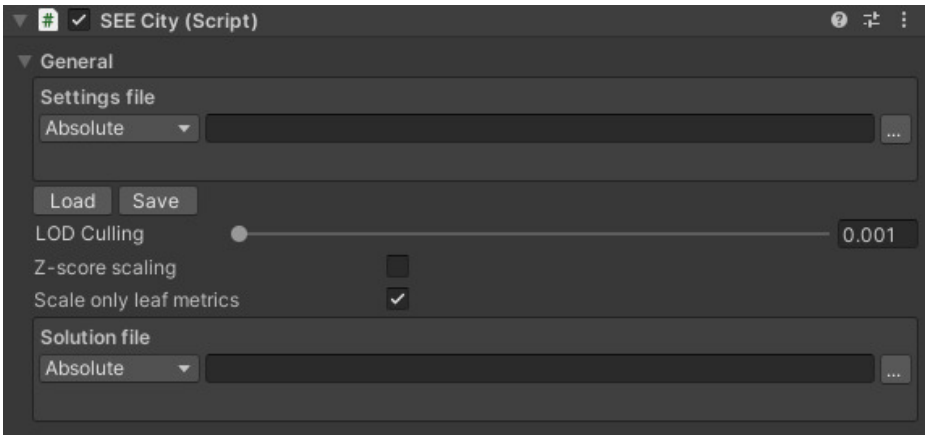

Abbildung 4.1: Einstellung für den Solution-Pfad

4.1.3 Anpassung der Benutzeroberfläche

Wie in Unterabschnitt 3.1.1 beschrieben, habe ich das bereits in SEE vorhandene Code-Window um ein neues Steuerelement erweitert. Die Erweiterung erfolgte mithilfe des Unity-Editors, in dem ich das Prefab des Code-Windows erfolgreich um eine Standardvorlage eines Knopfes erweitert habe. Das neu hinzugefügte Element ist jetzt allerdings noch ein Objekt ohne Funktionen. Das bedeutet, dass in der Klasse *CodeWindow*, welches den Inhalt und Funktionen eines Code-Window repräsentiert, die neue Funktion hinzugefügt werden muss. Dort wird dem Event, welches durch ein Betätigen des neuen Knopfs ausgelöst wird, eine anonyme Methode übergeben. Diese anonyme Methode ruft eine Methode in der Klasse *IDEIntegration* auf, was letztendlich zum gewünschten Aufruf der Datei in Visual Studio führen wird. In Abbildung 4.2 ist die Veränderung im Code-Window zu sehen.

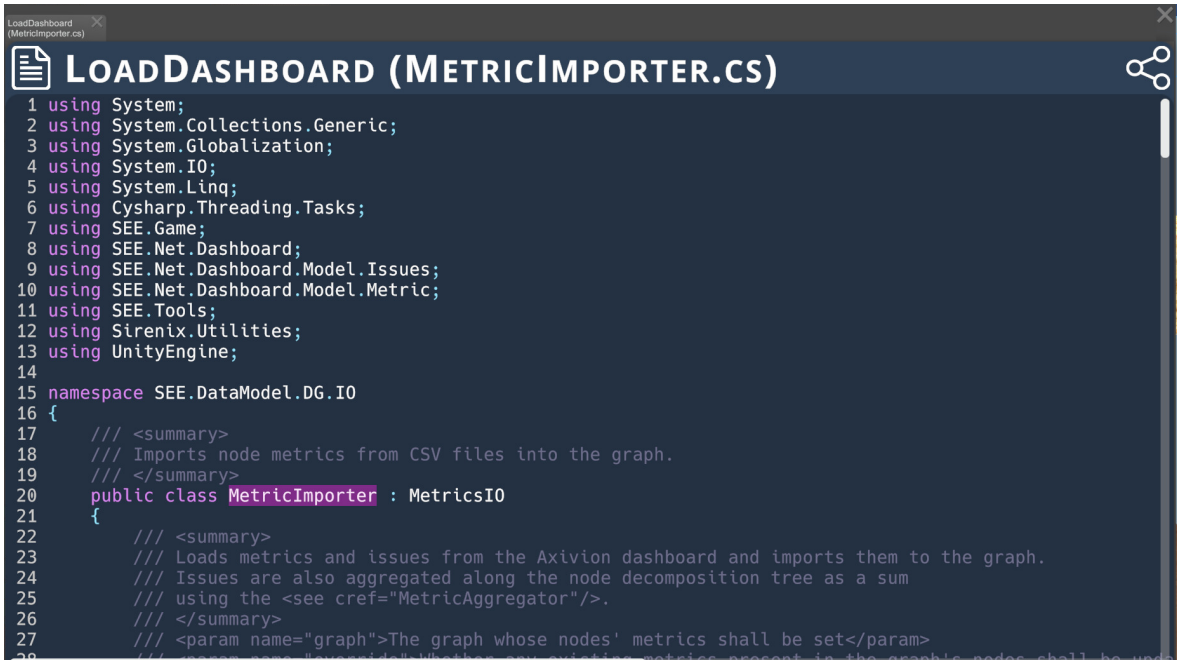

Abbildung 4.2: Das neue Code-Window in SEE

4.1.4 Visual Studio öffnen

Für die Suche nach den bereits vorhandenen Installationen von Visual Studio habe ich eine neue statische Klasse namens *VSPathFinder* erstellt. In dieser Klasse wird der Zugriff auf das in Abschnitt 2.4 beschriebene externen Programm vswhere definiert. Diese Klasse stellt das Enum *Version* bereit, in der alle aktuell unterstützten Versionen von Visual Studio angegeben sind. Um den absoluten Pfad einer gewünschten Instanz von Visual Studio zu erhalten, überreicht das Programm die gewünschte *Version* als Parameter an die Methode *GetVisualStudioExecutableAsync(...)*. Intern wird asynchron ein neuer Prozess mit den nötigen Argumenten erstellt und gewartet, bis dieser eine Rückgabe liefert. Die Ausgabe des Programms ist damit auch gleichzeitig die Rückgabe an die aufrufende Methode. Zu erwähnen ist, dass diese Implementierung aufgrund der *vswhere.exe* nur in Windows funktionieren wird. Für eine Verwendung unter Mac OS fehlt noch die Implementierung.
Der so erhaltene Pfad wird nun verwendet, um eine neue Instanz von Visual Studio zu starten. Visual Studio erhält dabei zwei Argumente. Einmal den Pfad zur C#-Solution, der wie in Unterabschnitt 4.1.2 beschrieben, vom Benutzer angegeben wird. Des Weiteren wird

/VsSeeExtension als Flag übergeben. Mit diesem Flag teilt SEE der neu erstellten Instanz von Visual Studio mit, einen direkten Verbindungsversuch zu starten. Dies geschieht ungeachtet, ob ein automatisches Verbinden aktiviert ist oder nicht.

4.1.5 Hervorheben von Konten und Kanten

Bevor Visual Studio ein Element in der Code-City hervorheben kann, muss diese möglichst effizient auffindbar sein. Eine Code-City generiert sich dabei aus einer vordefinierten Datei, was bedeutet, dass alle Informationen zu einem Softwareprojekt bereits vor dem Erstellen der Code-City bekannt sind. Also wird die Erweiterung alle Elemente des Typs Kante und Knoten heraussuchen und diese in einem *Dictionary* in der *IDEIntegration* abspeichern. Der Schlüssel, der dabei generiert wird, setzt sich aus den Filteroptionen des Benutzers zusammen. Eine nähere Beschreibung zu den Optionen ist in Unterabschnitt 4.1.6 genannt.
Wie auch in Unterabschnitt 3.1.2 erklärt, verwende ich das bereits bestehende System zum Hervorheben von Elementen. Dafür wurde die Klasse *SelectAction* verwendet. Diese stellt die Funktionalität zum Selektieren eines Knotens oder einer Kante bereit. In der Klasse selbst befindet sich eine Update-Methode, die ich um die Abfrage der *IDEIntegration* erweitert habe. So wird, wenn nicht bereits eine andere Aktion getätigt wurde, eine Liste mit allen zu selektierenden Objekten abgefragt und hervorgehoben. Bei bereits selektierten Objekten hebt diese Aktion das Hervorheben dieser auf. Wenn keine Objekte in der Liste vorhanden sind, passiert nichts.

4.1.6 Konfiguration

Bevor ein Entwickler die *IDEIntegration* im Unity-Inspector konfigurieren kann, muss das hinzufügen zu einem *GameObject* stattfinden. Unter *Type* ist die IDE ausgewählt, mit der sich standardmäßig verbunden werden soll. Des Weiteren ist der Port sowie die Anzahl an maximalen Clients einstellbar. Für den Fall, dass es egal ist, mit welchem Typen sich SEE verbindet, kann sich durch die Auswahl von *Connect To Any* jede IDE mit der VsSeeExtension verbinden. Die letzten beiden Abschnitte geben die Filteroptionen an, nach denen die *IDEIntegration* die von der IDE übermittelten Werte interpretiert. So sorgt *Use Element Position* dafür, dass die exakte Position eines Elements zur Identifizierung verwendet wird. *Use Element Range* hingegen verwendet die Startzeile eines Elementes mit dem Versatz zur Endzeile. Die Einstellungsmöglichkeiten sind in Abbildung 4.3 zu sehen.

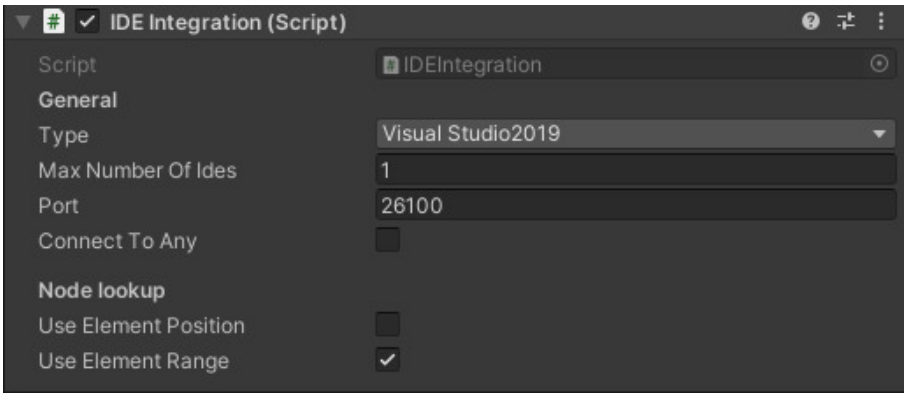

Abbildung 4.3: Konfiguration im Unity-Inspector

4.1.7 Funktionen für Visual Studio

In Unterabschnitt 4.1.1 habe ich schon das Hinzufügen eines Objektes für StreamRpc genannt. In diesem befinden sich zur Wahrung der Übersichtlichkeit alle Methoden, die ein Client aufrufen kann. Dieses Objekt wird jeder Verbindung eines Clients über die *IDEIntegration* hinzugefügt. Wenn gewünscht, können spezielle Verbindungen zusätzliche Objekte erhalten. Damit können zum Beispiel einige IDEs zusätzliche Funktionen erhalten. In der jetzigen Implementierung stellt SEE Funktionen bereit, um den Zustand von *IDEIntegration* zu verändern. Eine IDE kann folgende Methoden in SEE aufrufen:

- **HighlightNode(...)**
  Sorgt für das Hervorheben eines Knotens in einer Code-City.
- **HighlightNodeReferences(...)**
  Sorgt dafür, dass alle Kanten, die die Referenzen eines Objektes angeben, hervorgehoben werden.
- **HighlightNodes(...)**
  Hebt eine Liste von angegebenen Knoten hervor.
- **SolutionChanged(...)**
  Für den Fall, dass sich während einer aktiv bestehenden Verbindung zwischen SEE und Visual Studio die geöffnete Solution verändert, kann dies SEE über das Aufrufen dieser Methode mitgeteilt werden. SEE wird dementsprechend eine Prüfung durchführen und, falls nötig, eine Trennung von der IDE vornehmen.

4.1.8 Ungelöste Probleme

Aus Benutzerperspektive ist das in den Fokus setzen der aktuellen SEE-Instanz sinnvoll. Das bedeutet, wenn in Visual Studio der Befehl zum Hervorheben von Knoten oder Kanten an SEE übermittelt wurde, dieser auch gleichzeitig SEE in den Vordergrund setzt. Dadurch wird dem Benutzer direkt die Auswirkung, der gerade getätigten Aktionen, ersichtlich. Unity selbst scheint keine Funktion bereitzustellen, um das Anwendungsfenster in den Vordergrund zu bringen. Alternativ dazu muss auf die Verwendung von plattformspezifischen Bibliotheken zurückgegriffen werden. So bietet die *User32.dll* für Windows eine Funktion namens *SetForegroundWindow(...)*, mit der das Fokussieren einer Anwendung möglich sein sollte [vgl. Mic21c]. Allerdings lieferte eine erste Verwendung nicht das gewünschte Ergebnis und wurde aufgrund der restlichen Implementierung nicht weiterverfolgt.

4.2 Visual Studio

Genauso wie auch die Erweiterung für SEE, wird auch die Implementierung der Erweiterung von Visual Studio in C# geschrieben.

4.2.1 Client und StreamRpc

Wie bereits in Unterabschnitt 4.1.1 beschrieben, findet die Kommunikation über einen TCP-Socket statt. Die Realisierung des Clients ist ähnlich zur Server-Implementierung in SEE. Es gibt eine abstrakte Klasse, die alle Grundfunktionen bereitstellt. Die verwendete Kommunikationstechnologie wird wieder in einer erbebenden Klasse implementiert. Die resultierende Klasse heißt *JsonRpcSocketClient*. Insgesamt bietet die Client-Implementierung Methoden zum Starten eines Verbindungsversuches und auch zum Beenden einer aktuellen Verbindung.

Genauso wie beim Server kann ebenfalls eine Prozedur in SEE durch die Übergabe des dazugehörigen Methodennamens und deren Parameter in der Client-Implementierung ausgeführt werden. Jedoch befindet sich die nötige StreamRpc-Instanz dafür nicht ausgelagert in einer extra Klasse, sondern direkt im Client. Der Grund dafür ist, dass sich Visual Studio nur mit einem Server verbinden kann und deshalb auch keine Verwaltung verschiedener Verbindung benötigt wird. Aus den gleichen Gründen geschieht die Registrierung des aufrufbaren Objektes für die StreamRpc-Instanz auch direkt bei der Erstellung des Clients.
Die Implementierung des TCP-Socket-Clients startet einen Verbindungsversuch direkt beim Starten von Visual Studio, wenn das automatische Verbinden aktiviert ist. Wenn SEE nicht aktiv ist oder schon die maximale Anzahl an Clients erreicht wurde, wird der Verbindungsversuch als fehlgeschlagen angesehen und der Client versucht nach drei Sekunden eine neue Verbindung zu starten. Dies geschieht so lange, bis eine Verbindung etabliert wurde. Anders sieht es aus, wenn automatisches Verbinden deaktiviert ist. Dort wird nach einem fehlgeschlagenen Verbindungsversuch direkt aufgehört. Dem Benutzer ist es dann überlassen einen neuen Versuch zu starten.

4.2.2 Verwaltung von SEE

Neben der eigentlichen Implementierung des Clients müssen noch ein paar spezifische Funktionen für die Verwendung von SEE hinzugefügt werden. Dafür ist die Klasse *SeeIntegration* zuständig. Sie registriert, wenn sich SEE verbindet oder trennt und aktualisiert dementsprechend die Benutzeroberfläche von Visual Studio. Somit aktiviert oder deaktiviert die VsSeeExtension alle Befehle, die Aktionen in SEE ausführen, damit diese nur während einer aktiven Verbindung vom Endnutzer verwendbar sind. Zudem kümmert sich diese Klasse um das Starten des Servers, wenn die Instanz von Visual Studio von SEE gestartet wurde und das automatische Verbinden deaktiviert ist.

4.2.3 Neue Befehle

Abschnitt 2.3 beschreibt kurz die Klassen, die für die Erzeugung eines neuen Befehls in Visual Studio zu verwenden sind. Da die letztendlich dargestellten Menüeinträge dynamisch veränderbar sein sollen, wird die Klasse *OleMenuCommand* verwendet. Der Klasse müssen *EventHandler* übergeben werden, die sich um das Ausführen einer Aktion sowie das Abfragen des Status von einem Befehl kümmern. Diese *EventHandler* werden zusammen mit einer Methode zum Initialisieren vom *OleMenuCommand* in die abstrakte Klasse *Command* ausgelagert. Jeder neue Befehl, der für die VsSeeExtension geschrieben wurde, erbt von *Command* und wird als Singleton implementiert. Die Implementierung als Singleton soll sicherstellen, dass wirklich nur ein Befehl des gleichen Typs existiert. Des Weiteren gibt es Abstrahierungen, um Funktionen bei ähnlichen Befehlen auszulagern. Fast gleich implementiert sind zum Beispiel die Befehle zum Hervorheben eines Elements in SEE. Dort wird lediglich unterschieden, wann der Befehl in der IDE angezeigt werden soll. Die hinzugefügten neuen Befehle sind:

- **AboutCommand**
  Beim Verwenden dieses Befehls, wird ein Fenster aufgerufen, in der eine kurze Beschreibung zur Erweiterung geschrieben ist.

- **ConnectToSeeCommand**
  Ein Befehl, um einen Verbindungsversuch mit SEE zu starten. Wird nur angezeigt, wenn noch keine Verbindung mit SEE besteht und deaktiviert dargestellt, wenn automatisches Verbinden aktiviert ist, aber noch keine Verbindung zu SEE besteht.

- **DisconnectFromSeeCommand**
  Ähnlich zu *ConnectToSeeCommand*, nur kann der Benutzer sich hiermit von SEE trennen.
- **HighlightClassCommand**
  Wenn innerhalb einer Klasse das Kontextmenü geöffnet wird, so wird dieser Befehl angezeigt. Beim Ausführen wird dafür gesorgt, dass das entsprechende Element in SEE hervorgehoben wird.
- **HighlightMethodCommand**
  Funktioniert genauso wie *HighlightClassCommand*, lediglich wird dieser Befehl nur bei Methoden angezeigt.
- **HighlightClassesCommand**
  Hebt alle Klassen innerhalb einer Quelldatei in SEE hervor.
- **HighlightMethodReferencesCommand**
  Ähnliche Funktion wie die bereits vorgestellten Befehle, nur das hier die Referenzen einer Methode (also die Kanten) in SEE hervorgehoben werden.

4.2.4 Anpassung der Benutzeroberfläche

So habe ich sowohl für die Quellcode-Ansicht, die Tab-Ansicht und die Projekt-Ansicht jeweils Menüeinträge hinzugefügt, um in SEE das gewünschte Element hervorzuheben. Des Weiteren finden sich unter *Extensions* → *SEE Integration* zwei zusätzliche Einträge. Diese führen zu weiteren Informationen über die Erweiterung, oder ermöglichen es sich mit SEE zu verbinden beziehungsweise die Verbindung zu trennen. Menüeinträge werden nur dann angezeigt, wenn auch die richtigen Elemente innerhalb des Quellcodes ausgewählt wurden, oder die Funktion generell aktiv ist. Der Befehl zum Verbinden/Trennen wird zum Beispiel ausgegraut, wenn *Auto Connect* aktiviert ist. Ein Beispiel der neu hinzugefügten Einträge ist in Abbildung 4.4 zu sehen.

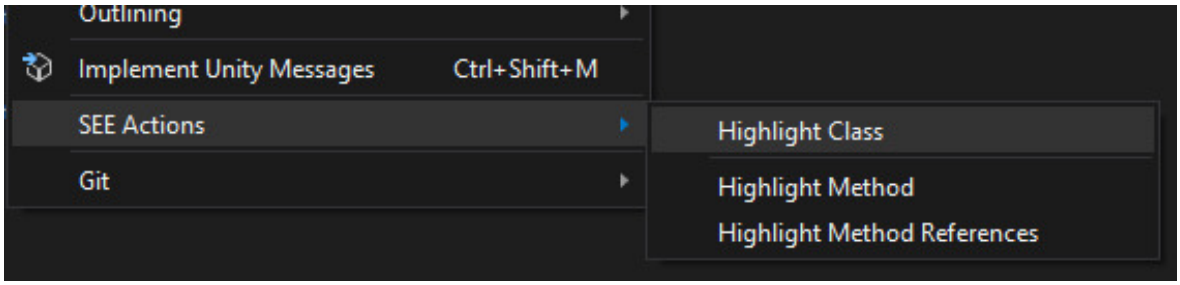

Abbildung 4.4: Neuer Menüeintrag in Visual Studio

4.2.5 Konfiguration

Die Optionsseite zum Konfigurieren muss der Erweiterung als Attribut bekannt gegeben werden. Dabei muss eine Klasse von *DialogPage* erben. Neue Optionen sind dann durch das Hinzufügen neuer Properties zu der Klasse erstellbar. Die Konfiguration kann anschließend vom Benutzer der VsSeeExtension unter *Tools* → *Options* → *SEE Integration* vorgenommen werden. Die Einstellungsmöglichkeiten sind hier begrenzt, da die Verarbeitung der relevanten Informationen in SEE stattfindet. Deshalb wird hier nur die Möglichkeit zwischen *Auto Connect* zu wechseln oder die Portnummer zu ändern angeboten. Die Optionen sind in Abbildung 4.5 zu sehen.

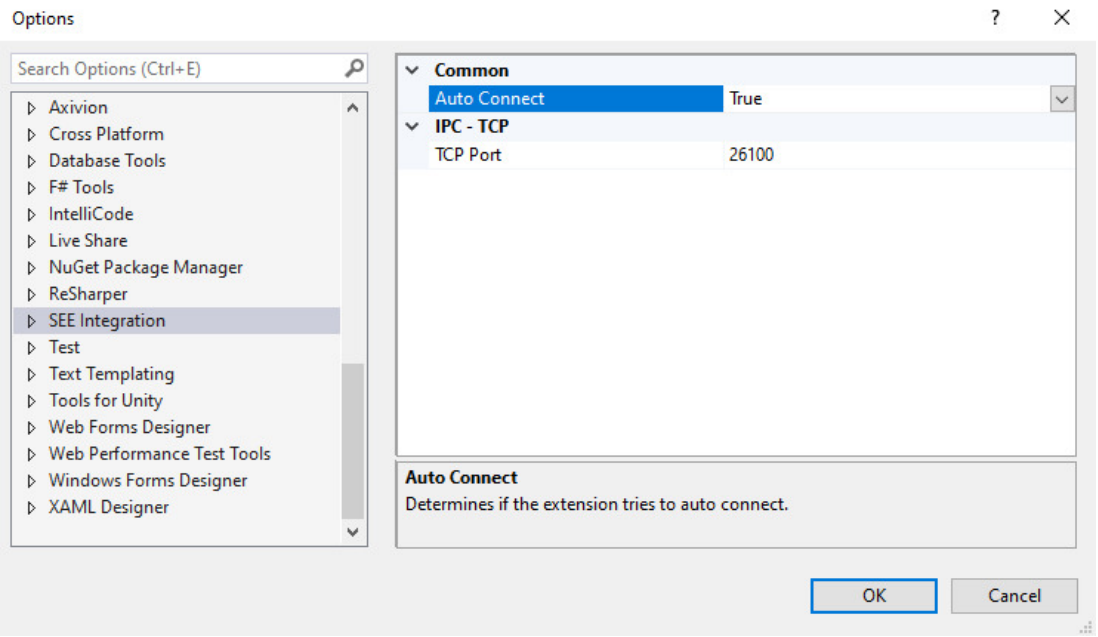

Abbildung 4.5: Optionen der VsSeeExtension

4.2.6 Funktionen für SEE

Da es sich um eine bidirektionale Kommunikation handelt, stellt auch Visual Studio Funktionen bereit, die von SEE aufgerufen werden können. Diese verwenden neu erstellte Hilfsklassen, die auf von Visual Studio bereitgestellten Services zugreifen, um die nötigen Funktionen zum Verwalten von Dokumenten, der Projektmappe und dem Fenster bereitzustellen. Die von SEE aufrufbaren Methoden sind folgende:

- **OpenFile(...)**
  Mit dieser Methode kann eine neue Datei in der IDE geöffnet werden. Zusätzlich gibt es die Möglichkeit noch die Zeilennummer anzugeben, um diese dem Benutzer hervorzuheben.
- **GetProject()**
  Gibt den gerade geöffneten Solution-Pfad zurück.
- **GetIdeVersion()**
  Gibt die aktuell verwendete Version der gerade verwendeten IDE an.
- **WasStartedBySee()**
  Gibt an, ob diese Instanz direkt durch SEE erstellt wurde oder nicht.
- **SetFocus()**
  Sorgt dafür, dass das Fenster von Visual Studio hervorgehoben wird, damit der Benutzer die Veränderung in der Datei direkt sieht. Somit wird eventuelle Verwirrung durch fehlende Ereignisse verhindert.
- **ChangeSolution(...)**
  Ändert die angezeigte C#-Solution in Visual Studio.
- **Decline()**
  Sorgt dafür, dass die verbundene Instanz von Visual Studio die Verbindung unterbricht und weitere Verbindungsversuche einstellt.

4.2.7 Sonstiges

Auch wenn es im ursprünglichen Entwurf nicht gefordert ist, habe ich zusätzlich noch ein About-Fenster, für weiter Informationen über die Erweiterung, hinzugefügt. Diese kann der Benutzer über *Tools* → *Options* → *SEE Integration* abrufen. Während der Entwicklung an der VsSeeExtension hat Microsoft eine neue Version von Visual Studio veröffentlicht. Die neue Version Visual Studio 2022 hat im Zuge dessen auch einen Anleitung herausgebracht. Diese erklärt die Überführung des alten Projektes in ein neues, welches mit der aktuellen Version von Visual Studio kompatibel ist. Aufgrund dessen hat sich die anfängliche Projektstruktur verändert. So existiert nun ein Hauptprojekt, welches alle Klassen die von beiden IDEs verwendet werden können beinhaltet sowie zwei weiter Projekte. In diesen befindet sich die jeweilige Einstellung der Versionen.

4.2.8 Ungelöste Probleme

Die in Unterabschnitt 4.2.7 angesprochene Veränderung des Projektes birgt allerdings noch ein Problem. Das Erstellen der Erweiterung ist zwar möglich, aber die Ausführung der VsSeeExtension führt noch zu Problemen in Visual Studio 2022. Das Hauptproblem ist das Erstellen einer Verbindung zum Server. Neben diesem spezifischen Problem für die neue Version der IDE gibt es auch noch ein Problem mit dem Fokussieren der Anwendung. Visual Studio schafft zwar sich in den Vordergrund zu bringen, dies aber nicht immer. Es kommt vor, dass nur das Symbol in der Taskleiste aufleuchtet, ohne die Anwendung zu fokussieren.

KAPITEL 5
Evaluation

Nachdem die Implementierung in sowohl SEE als auch Visual Studio abgeschlossen wurde, muss diese nun evaluiert werden. Dabei habe ich mich bei dem Studiendesign für eine Benutzerstudie, in der eine festgelegte Personengruppe bestimmte Aufgaben erledigen muss, entschieden. Eine erste Idee zum Aufbau einer Benutzerstudie habe ich durch die Erhebung von Galperin erhalten [Gal21, S. 31-39]. Diese habe ich durch meine weitergehende Recherche ausgebaut. Im folgenden Abschnitt ist diese genauer ausgeführt.

5.1 Hypothesen

Bei der in Abschnitt 1.2 eingeführten Forschungsfrage steht der Mehrwert (auch zusätzlicher Nutzen), den ein Benutzer beim Arbeiten mit der Software erhält, im Fokus. Zum Evaluieren des Mehrwertes, wird wie in Abschnitt 5.2 beschrieben, eine vergleichende Evaluation durchgeführt. Diese baut auf sowohl objektiven als auch subjektiven Daten auf. Daher findet die folgende Bildung meiner Hypothesen zum zusätzlichen Nutzen auf diesen Aspekten statt:

Bearbeitungszeit

$H1_0$	*Nullhypothese:* Die Bearbeitungszeit der Aufgaben mit der Erweiterung steigt oder bleibt gleich der Bearbeitungszeit von Visual Studio ohne Erweiterung.
$H1_1$	*Gegenhypothese:* Die Bearbeitungszeit der Aufgaben mit der Erweiterung sinkt im Vergleich zu der Bearbeitungszeit von Visual Studio ohne Erweiterung.

Benutzerfehler

$H2_0$	*Nullhypothese:* Die Fehlerquote bei der Bearbeitung der Aufgaben mit Erweiterung steigt oder bleibt gleich der Fehlerquote von Visual Studio ohne Erweiterung.
$H2_1$	*Gegenhypothese:* Die Fehlerquote bei der Bearbeitung der Aufgaben mit der Erweiterung sinkt im Vergleich zu der Fehlerquote von Visual Studio ohne Erweiterung.

Zufriedenheit

$H3_0$	*Nullhypothese:* Die Zufriedenheit bei der Bearbeitung der Aufgaben mit Erweiterung sinkt oder bleibt gleich der Zufriedenheit von Visual Studio ohne Erweiterung.
$H3_1$	*Gegenhypothese:* Die Zufriedenheit bei der Bearbeitung der Aufgaben mit Erweiterung steigt im Vergleich zu der Zufriedenheit von Visual Studio ohne Erweiterung.

Gebrauchstauglichkeit

$H4_0$	*Nullhypothese:* Die Gebrauchstauglichkeit von Visual Studio mit Erweiterung sinkt oder bleibt gleich der Gebrauchstauglichkeit von Visual Studio ohne Erweiterung.
$H4_1$	*Gegenhypothese:* Die Gebrauchstauglichkeit von Visual Studio mit Erweiterung steigt im Vergleich zu der Gebrauchstauglichkeit von Visual Studio ohne Erweiterung.

5.2 Methodik

Ziel ist es, eine vergleichende Evaluation zu gestalten. Bei einer vergleichenden Evaluierung zwischen verschiedenen Anwendungen werden diese nach den Aspekten der Usability verglichen [vgl. Nie93, S. 79 f.]. Diese umfassen sowohl objektiv als auch subjektiv messbare Werte. So sind Erlernbarkeit, Effizienz, Einprägsamkeit, Fehlerquote sowie Zufriedenheit klassische Attribute, die in der Usability enthalten sind [vgl. Nie93, S. 26].
Bei der Durchführung der Benutzerstudie fokussiere ich mich auf die Messung der Bearbeitungszeit und Fehlerquote als objektive Werte sowie Zufriedenheit und Gebrauchstauglichkeit als subjektive Werte. Die Gebrauchstauglichkeit beschreibt dabei die Benutzbarkeit des Gesamtsystems, wo sich hingegen die Zufriedenheit auf eine gerade erledigte Aufgabe mit dem System bezieht. Diese Unterteilung baut auf den erhaltenen Erkenntnissen in Unterabschnitt 5.4.1 auf.
Letztendlich lässt dies auf eine aufgaben-basierte Beobachtung für das Messen der benötigten Zeit und einer Umfrage für die Erfassung der Gebrauchstauglichkeit sowie Zufriedenheit als zu verwendende Methoden schließen. Eine Alternative zur Beobachtung wäre die Think-Aloud-Methode. Allerdings würde das Beschreiben der aktuellen Gedanken die Bearbeitungszeit beeinträchtigen [vgl. Rii18, S. 265].

5.3 Aufgabenstellung

In diesem Abschnitt möchte ich die Wahl meiner Aufgaben darlegen. Insbesondere die Vorgehensweise bei der Auswahl der Klassen und den dazugehörigen Dateien. Zudem wird zusätzliche Software genannte, die zu einer besseren Vergleichbarkeit führt. Anschließend eine kleine Vorstellung der letzendlich entstanden Code-City.

5.3.1 Zusätzliche Software

Bevor ich mit der genauen Beschreibung der Aufgaben beginne, möchte ich anmerken, dass für einen besseren Vergleich zwischen Nutzung von Visual Studio und Visual Studio mit der VsSeeExtension eine zusätzliche Erweiterung für die IDE installiert wurde. Dabei handelt es sich um das Axivion Visual Studio Plugin, welches Visual Studio die Möglichkeit gibt, Softwareerosion darstellen zu können. Die unterschiedlichen Typen der Softwareerosion sind in Visual Studio mit verschiedenen Symbolen gekennzeichnet. So ermöglicht die Erweiterung das Hervorheben mithilfe von Symbolen am linken Rand des Programmierfensters. Ein Beispiel der Darstellung ist in Abbildung 5.1 zu sehen.

```
134     /// <summary>
135     /// getter for the fields of a ParsedJLG
136     /// </summary>
        1 reference
137     public IList<string> FilesOfProject { get => filesOfProject; }
        0 references
138     public IList<string> LocationLookupTable { get => locationLookupTable; }
        0 references
139     public IList<string> FieldLookupTable { get => fieldLookupTable; }
        21 references
140     public List<JavaStatement> AllStatements { get => allStatements; }
        1 reference
141     public Stack<string> ReturnValues { get => returnValues; set => returnValues = value; }
142
143     /// <summary>
144     /// This Methode creates the string, that visualizes the runtime data in the small text window.
145     /// </summary>
146     /// <param name="statementCounter"></param>
147     /// <param name="AddReturnValueToStack">This should be true when the Visualization is running f
148     /// <returns></returns>
        2 references
149     internal string CreateStatementInfoString(int statementCounter, Boolean AddReturnValueToStack)
150     {
151         JavaStatement js = allStatements[statementCounter];
152         string info = "Line " + js.Line + Environment.NewLine;
```

Abbildung 5.1: Darstellung der Softwareerosion in Visual Studio

Im Gegensatz dazu stellt SEE die Softwareerosion direkt über den einzelnen Elementen dar (siehe Abbildung 5.2). Auch hier wird auf die Darstellung mithilfe von Axivion gesetzt. Somit ist eine Vergleichbarkeit durch diese Erweiterung gegeben.

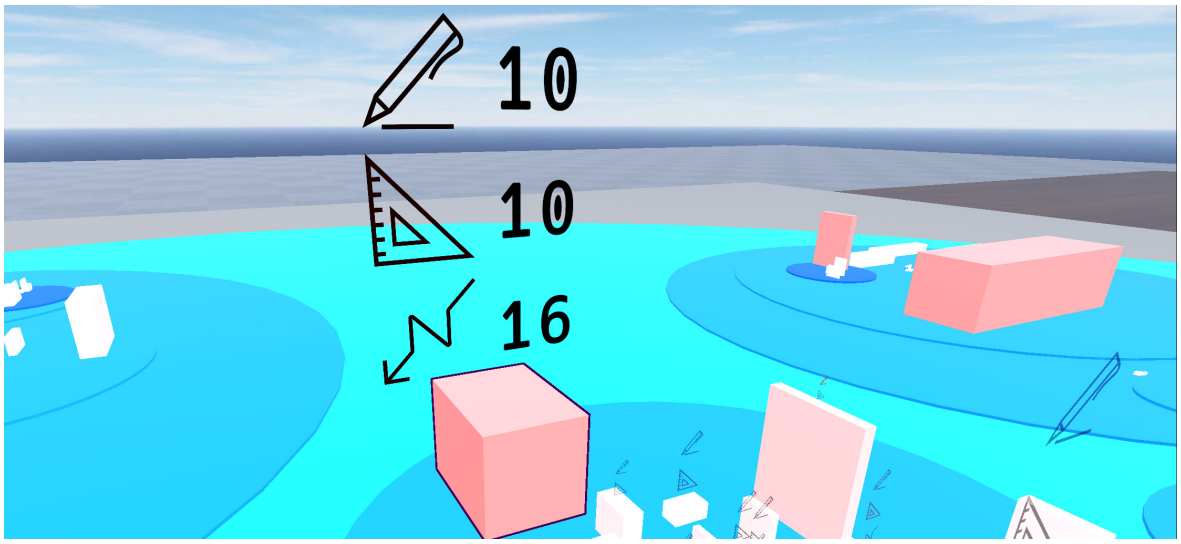

Abbildung 5.2: Darstellung der Softwareerosion in SEE

Beginnend von oben bedeuten die Symbole Stil-Verletzung, Metrik-Verletzung und Architektur-Verletzung. Es gibt zwar noch mehr Arten, diese sind aber im Rahmen der, in Unterabschnitt 5.3.2, beschriebenen Aufgaben nicht relevant.

5.3.2 Aufgabe

Bei den Aufgabenstellungen handelt es sich um Aufgaben, die während der Softwarequalitätssicherung auftauchen könnten. Um die Teilnehmer dieser Studie motiviert zu behalten, soll diese nicht länger als ungefähr 40–60 Minuten dauern. Das schließt die Erklärung der

Studie, Bearbeitung der Aufgaben und anschließende Umfrage mit ein. Greifeneder stellt dabei die Anforderungen, dass die Aufgabenstellungen eindeutig formuliert sind, sodass diese von Probanden ohne Hilfe gelöst werden können. Die Ergebnisse der Studie sollen auch über einen längeren Testzeitraum immer gleich bleiben [vgl. Gre13, S. 270]. Somit habe ich mich für folgende Aufgaben entschieden:

Aufgabe 1 Sie arbeiten gerade an einer Klasse und würden gerne wissen, ob es sich bei dieser um die größte Klasse im Ordner, in dem sie sich befinden, handelt. Die Vorgehensweisen sind:

- **Visual Studio:** Jede Klasse in dem aktuellen Ordner wird mit der Ausgangsklasse verglichen.
- **VsSeeExtension:** Die Ausgangsklasse wird mithilfe der Erweiterung in SEE hervorgehoben. Dort kann die Klasse aufgrund der dargestellten Größe mit den anderen Dateien verglichen werden. Wenn dabei eine andere Klasse größer ist, müssen die Probanden diese mithilfe von SEE in Visual Studio öffnen.

Anschließend soll der Klassenname und die dazugehörige Größe der größten Klasse niedergeschrieben werden.

Aufgabe 2 Sie arbeiten gerade mit drei Klassen, von denen Sie jeweils gerne die Methode mit den meisten Stil-Verletzungen herausfinden möchten. Die Vorgehensweisen sind:

- **Visual Studio:** Mithilfe des Axivion Visual Studio Plugins werden die Verletzungen angegeben. Die Stil-Verletzungen müssen nun in jeder Methode gezählt und eingeordnet werden.
- **VsSeeExtension:** Die Ausgangsklassen müssen in SEE hervorgehoben werden. Dort lässt sich die Anzahl der Stil-Verletzungen ablesen.

Anschließend werden die Methoden sowie die Anzahl der Stil-Verletzungen in absteigender Reihenfolge niedergeschrieben.

Aufgabe 3 Sie haben die Aufgabe in drei Klassen nach fehlender Dokumentation an den Methoden zu suchen. Die Vorgehensweisen sind:

- **Visual Studio:** Jede gegebene Klasse muss auf fehlende Dokumentation durchsucht werden.
- **VsSeeExtension:** Die Ausgangsklassen müssen in SEE hervorgehoben werden. Dort können die Probanden anhand der Metriken die Klassen ohne Dokumentation ablesen.

Anschließend wird der Klassenname und die Anzahl der Methoden ohne Dokumentation in absteigender Reihenfolge niedergeschrieben.

5.3.3 Wahl der Dateien

Eine vergleichende Studie setzt die Verwendung der gleichen Aufgaben für das jeweilige System voraus. Das bedeutet, dass die Dateien oder Ordner für die Aufgaben unterschiedlich aber vergleichbar sein müssen. Deshalb wird als Grundlage dieser Evaluation das Projekt SEE verwendet. Dieses bietet ausreichend Ordner und Quelldateien, sodass die Bearbeitung der Aufgaben in einem angemessenen Rahmen stattfinden kann.

In der ersten Aufgabe spielt die Anzahl der Klassen eine wichtige Rolle, weshalb hier zunächst nur die Anzahl der Dateien in einem Ordner betrachtet wird. Dabei sollen die Probanden nicht zu viele Dateien untersuchen müssen, da nur eine begrenzte Zeit zur Verfügung steht. Außerdem sollten die Probanden keine zu komplexe Struktur betrachten, da diese die Übersichtlichkeit verringert. Aufgrund dieser Kriterien wurden nur die Ordner der ersten Ebene in Betracht gezogen. In Tabelle 5.1 ist die Anzahl der Dateien innerhalb eines Ordners ersichtlich. Aus dieser Menge wurden die beiden Verzeichnisse *Layout* und *Net* gewählt, da diese den gesuchten Kriterien am ehesten entsprechen. Die Auswahl der Ausgangsklasse erfolgte per Zufall und ist in Tabelle 5.2 einzusehen.

Ordner	**Dateianzahl**
DataModel	4
Tools	4
CameraPaths	5
Layout	8
Net	9
Controls	13
GameObjects	18
Game	32
Utils	38

Tabelle 5.1: Anzahl der Elemente in den Hauptordnern

Für Aufgabe zwei und drei spielen jeweils die Anzahl der Zeilen einer Klasse sowie die Methoden-Anzahl eine wichtige Rolle. Das Bearbeiten der Klassen sollte von den Probanden nicht zu schnell gehen, daher wäre eine Klasse mit zum Beispiel 50 Zeilen eher ungeeignet. Beim Zählen der Stil-Verletzungen sollten nicht zu viele Methoden in der Klasse vorkommen, da hier das Zählen innerhalb einer Methode wichtig ist. Für die Auswahl der Klassen mit fehlender Dokumentation ist hingegen eine höhere Anzahl von Methoden wünschenswert, da die Aufgabe ansonsten zu schnell lösbar wäre. Eine Auswahl an Dateien, die diese Ansprüche erfüllen, ist in Tabelle 5.3 zu sehen. Die Zuordnung der Dateipaare kann in Tabelle 5.2 betrachtet werden.

Auf.	**Visual Studio**	**SEE + Visual Studio**
1	LayoutNodes.cs	Player.cs
2	ParsedJLG.cs	TextFactory.cs
2	PainReceiver.cs	FilePicker.cs
2	PathReplay.cs	HidePropertyDialog.cs
3	Protal.cs	PersonalAssistantBrain.cs
3	MappingAction.cs	SEECity.cs
3	GraphReader.cs	ConfigMenu.cs

Tabelle 5.2: Zuordnung der Paare zu den Aufgaben

Datei	Zeilenanzahl	Methodenanzahl
ParsedJLG.cs	192	5
PaintReceiver.cs	198	5
PathReplay.cs	255	6
Portal.cs	260	10
MappingAction.cs	429	21
GraphReader.cs	526	16
TextFactory.cs	163	6
FilePicker.cs	208	5
HidePropertyDialog.cs	315	6
PersonalAssistantBrain.cs	259	11
SEECity.cs	454	16
ConfigMenu.cs	668	20

Tabelle 5.3: Ausgewählte Dateien für die Evaluation

5.3.4 Code-City

Für die Benutzerstudie werden zwei verschiedene SEE-Instanzen gebraucht. Die erste Instanz wird für die Darstellung der Klassen benötigt (siehe Abbildung 5.3). Dort stellt eine Code-City die Blattknoten (Blöcke) als Klasse eines Softwareprojektes dar. Die zweite Instanz ist für die Darstellung der Methoden wichtig. Denn für die Aufgaben zwei und drei müssen nähere Informationen zu einer Methode ersichtlich sein. Der Grund für die Unterscheidung in zwei Anwendungen ist die Darstellung von inneren Knoten in SEE. Ein innerer Knoten kann eine Metrik nur durch seine Farbe darstellen, dieser sollte aber auf den verschiedenen Ebenen immer unterschiedliche Farben haben. In der ersten Aufgabe sollen die Probanden die Klasse mit der meisten Zeilenanzahl herausfinden. Dies würde jedoch für die inneren Knoten bedeuten, dass höchstwahrscheinlich verschiedene Farben in einer Ebene verwendet werden, wenn die Farbe die Zeilenanzahl widerspiegelt.
Wie schon in Unterabschnitt 5.3.3 beschrieben, wurden die SEE-Instanzen auf dem Quellcode des Projektes SEE erstellt. In der Code-City der ersten Aufgabe, in der die Blattknoten für die Darstellung der Klasse zuständig sind, spiegelt die Höhe der Knoten auch die Anzahl der Codezeilen einer Klasse wider. Die als Kreise dargestellten innern Knoten sind dementsprechend für die Darstellung der Ordnerstruktur dar. Für die zweite und dritte Aufgabe wurde eine Code-City generiert, in denen Methoden einer Klasse möglichst gut sichtbar sind. In diesem Fall spielt die Höhe als Metrik überhaupt keine Rolle, viel wichtiger ist die Farbkodierung. Diese gibt an, wie viele Kommentare eine Methode besitzt. Eine dunkelrote Färbung weist auf keinerlei Kommentare hin, während eine weiße Färbung auf mehrere Kommentare hindeutet. In Abbildung 5.4 und Abbildung 5.5 kann die Code-City für Aufgabe zwei und drei mit Darstellung der Softwareerosion betrachtet werden.
Die Code-City für Aufgabe zwei besitzt 603 Knoten, von denen 96 innere Knoten sind und 507 Blattknoten. Im Gegensatz dazu sind in Aufgabe zwei und drei 3536 Knoten generiert worden. Davon sind 554 innere Knoten und 2982 Blattknoten.

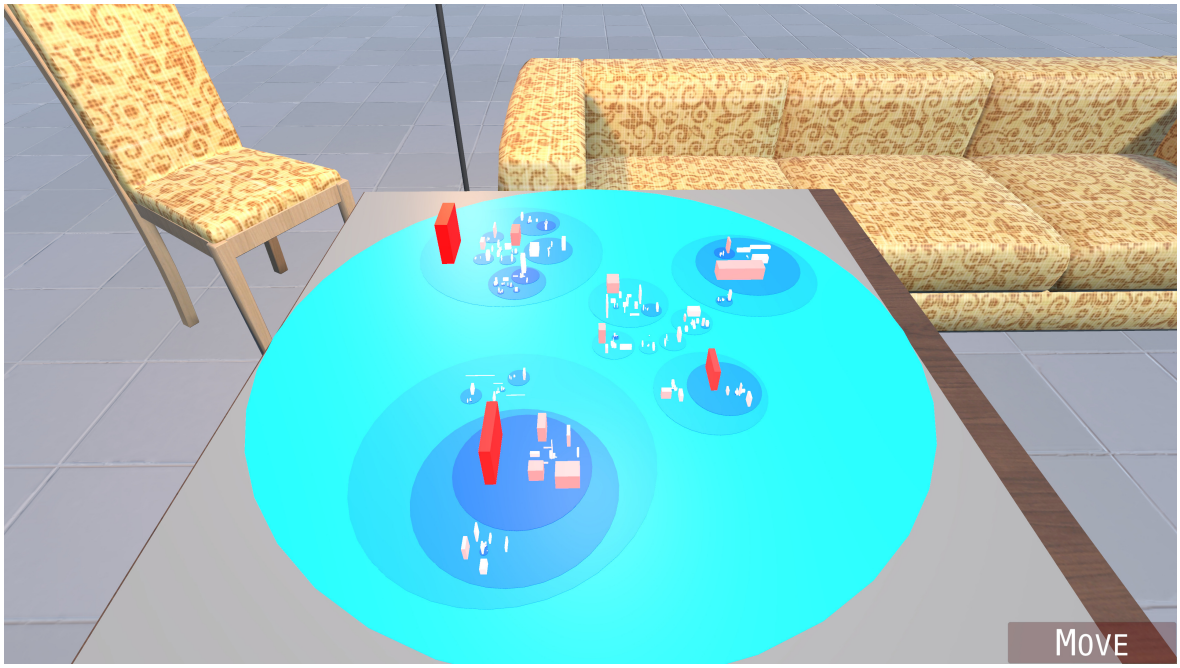

Abbildung 5.3: Code-City für Aufgabe 1

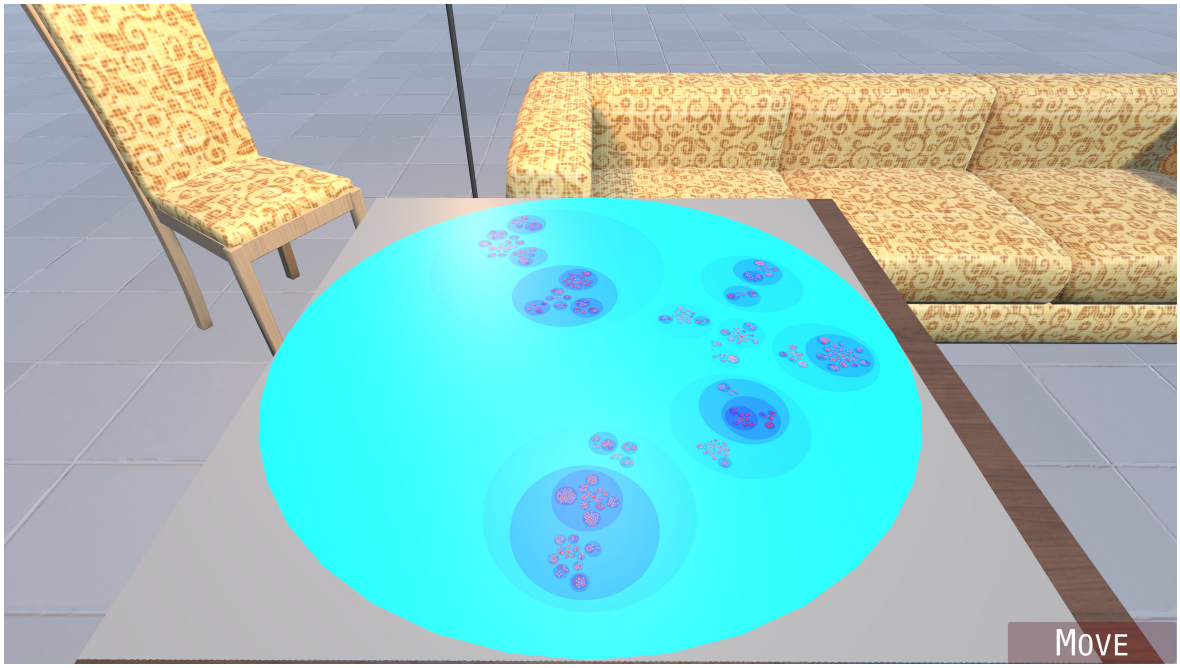

Abbildung 5.4: Code-City für Aufgabe 2 und 3

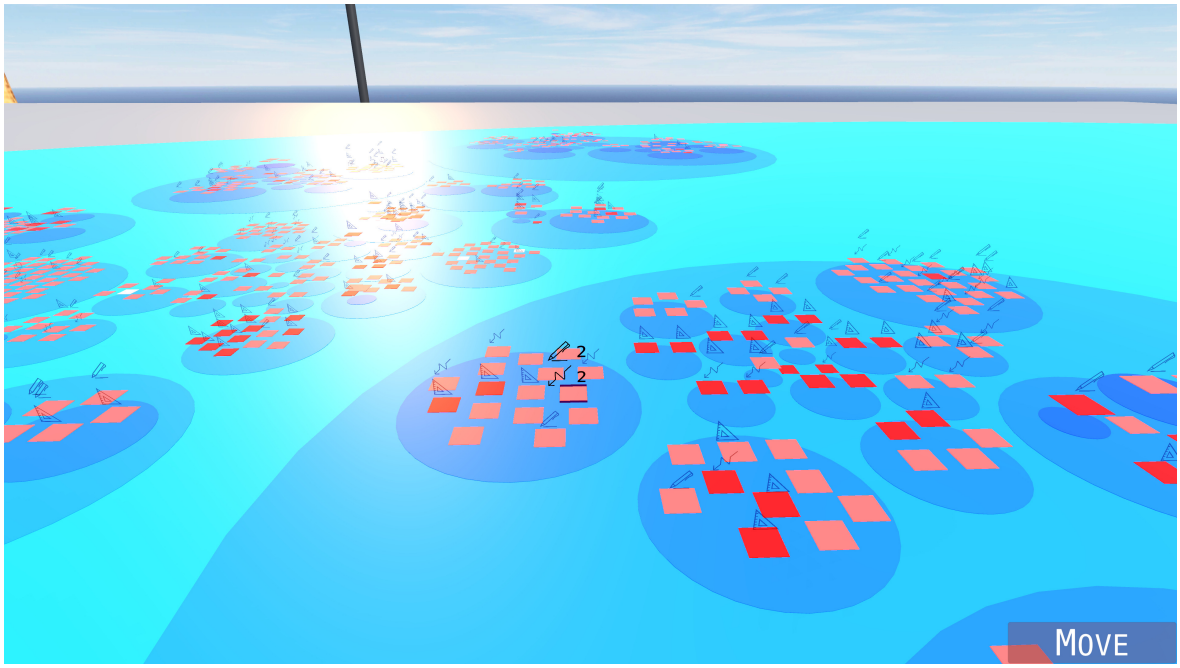

Abbildung 5.5: Darstellung der Softwareerosion in Aufgabe 2 und 3

5.4 Erhebung

In diesem Abschnitt möchte ich den für die Studie vorgesehenen geplanten Ablauf erklären. Dafür wird die Auswahl der einzelnen Software sowie generelle Werkzeuge zur Erhebung der Daten genauer erläutert. Aufgrund der anhaltenden Corona-Pandemie ist eine Durchführung Online mittels geeigneter Software beabsichtigt. Das bedeutet, eine Umgebung, in der bereits alles vorinstalliert ist, soll für die Evaluierung bereitgestellt werden. Das ist vor allem deshalb gewünscht, da die benötigte Software (SEE, Visual Studio, VsSeeExtension und Axivion Visual Studio Plugin) eine potenziell lange Installationsdauer innehaben. Dies könnte besonders Benutzer mit einer langsameren Internetleitung abschrecken.

5.4.1 Ablauf der Umfrage

Laut Riihiaho beginnt eine Umfrage normalerweise mit dem Einholen der Hintergrundinformationen einer Person. Dazu können auch Erwartungen zum System abgefragt werden. Nachdem die Aufgaben vom Benutzer erledigt wurden, folgt die Evaluierung der Benutzererfahrung durch einen Fragebogen [vgl. Rii18, S. 262]. Nach Greifeneder soll eine Umfrage mit einem Begrüßungstext Informationen zur Studie bereitstellen. Darunter fallen Informationen wofür die Studie ist, was herausgefunden werden soll, wie lang die Umfrage dauert und eine Aufklärung über den Datenschutz. Zudem sollen auch weitere Hinweistexte verwendet werden, um über den weiteren Verlauf der Studie in den Aufgaben zu informieren [vgl. Gre13, S. 262 f.]. Außerdem soll laut Sauro und Lewis auch direkt nach einer Aufgabe ein Fragebogen bereitgestellt werden, um die Zufriedenheit direkt nach der Bearbeitung einer Aufgabe zu testen [vgl. SL09, S. 1617].
Aus diesen Erkenntnissen ist meine Umfrage wie folgt aufgebaut. Als Erstes wird ein Hinweistext mit allen nötigen Informationen die Umfrage einleiten. Als Nächstes sollen die Probanden eine Verbindung zur Testumgebung herstellen. Wenn dies erledigt ist, beginnen die Probanden mit dem Ausfüllen der demografischen Daten. Bevor mit der Bearbeitung der Aufgaben begonnen wird, müssen die Probanden den Umgang mit der Software erlernen.

Dafür wird eine Anleitung zur Bedienung in der Umfrage beschrieben. Damit einhergehend auch Begriffserklärungen, wie Softwareerosion. Wenn es keine Fragen zur Verwendung der Software gibt, wird eine kurze Verständnisaufgabe in SEE und Visual Studio durchgeführt. Hiermit wird getestet, ob die Umgangsweise mit der Software auch verstanden wurde, damit ein späteres Unverständnis beim Lösen der Aufgaben nicht auftaucht. Wenn dies geschehen ist, werden die Aufgaben mit Hinweistexten gestellt, um die Software abschließend mit einem geeigneten Fragebogen zu evaluieren. Damit die Probanden eventuelle Anmerkungen oder Verbesserungsvorschläge zu der Software nennen können, wird dem Benutzer ebenfalls ein Eingabefeld bereitgestellt. Für den Vergleich folgt ein weiteres Mal die Aufgabenstellung, doch dieses Mal für die zweite Software. Wenn auch dort die Evaluierung abgeschlossen ist, schließt die Umfrage mit einem Text, der sich für die Teilnahme bedankt, ab.
Um einen Bias aufgrund der Reihenfolge auszuschließen, teile ich die Probanden in zwei Gruppen auf. Das verhindert besonders das Aufkommen des Aktualitätseffekts, wobei Probanden während der Benutzung einer Software immer sicherer und besser werden, was zum Beispiel die Bearbeitungszeit beeinträchtigen kann [vgl. NWC16, S. 87]. Somit beginnt die erste Gruppe zuerst mit der Bearbeitung der Aufgaben der IDE ohne Erweiterung und erst anschließend in einem zweiten Durchlauf mit der VsSeeExtension. Die zweite Gruppe dementsprechend umgekehrt.

5.4.2 Fragebogenwahl

Wie schon einleitend in Unterabschnitt 5.4.1 erklärt, sollen zunächst die Hintergrundinformationen und Erfahrung zum System gesammelt werden. Das bedeutet, es wird das Alter, Geschlecht, allgemeine Programmierkenntnisse, Erfahrung mit C#, Visual Studio- und SEE-Kenntnisse für die Evaluation gesammelt. Das Sammeln dieser Information dient letztendlich für die Überprüfung der Repräsentativität der Grundgesamtheit [vgl. Sea13, S. 49].
Bei der Wahl des Fragebogens für die Gebrauchstauglichkeit habe ich mich für die System Usability Scale (SUS) entschieden. Nach Brooke verwendet die SUS eine Likert-Skala, in der mit fünf Auswahlmöglichkeiten die Zustimmung oder Ablehnung zu einer Aussage angegeben wird. In der SUS sind zehn Aussagen, die die Teilnehmer bewerten. Die Aussagen sind prinzipiell immer gegensätzlich gestellt, um eine Verzerrung bei den Antworten zu verhindern [vgl. Bro96, S. 3 f.]. Brooke schrieb später in seinem Fazit zur SUS, dass sich zuverlässige Ergebnisse selbst bei einer kleinen Zahl von Teilnehmer erreichen lassen und der Fragebogen besonders gut für den Vergleich zwischen verschiedenen Versionen von einer Anwendung, die mit unterschiedlicher Technologie funktionieren, eignet [vgl. Bro13, S. 38]. Aufgrund dieser Merkmale eignet sich die SUS ideal für die Bewertung der Gebrauchstauglichkeit in meiner vergleichende Evaluierung der VsSeeExtension.
Da es sich bei SUS um einen Fragebogen handelt, der für die Evaluierung der Gebrauchstauglichkeit der gesamten Software da ist, muss ich noch einen weiteren standardisierten Fragebogen für die Evaluierung der Zufriedenheit nach einer Aufgabe wählen. Dieser soll schnell und einfach für die Teilnehmer zu bearbeiten sein. Deshalb habe ich mich für den After-Scenario-Questionnaire (ASQ) entschieden. Lewis definiert den ASQ, wie bei der SUS, als einen Fragebogen, der die Likert-Skala verwendet. Allerdings findet die Punktevergabe mit sieben anstelle von fünf Punkten statt. Außerdem bedeutet, im Gegensatz zu SUS, der erste Eintrag eine starke Zustimmung und der letzte eine starke Ablehnung. Grundlage dieses Fragebogens sind die drei Aspekte: leichte Lösbarkeit der Aufgaben, Bearbeitungszeit und Angemessenheit der unterstützenden Informationen. Die daraus vorkommenden drei Aussagen können dadurch schnell bearbeitet werden [vgl. Lew95, S. 62].
Beide Fragebögen sind ursprünglich nur in Englisch verfügbar, weshalb ich für diese nach Übersetzungen, die nicht den Sinn verändern, gesucht habe. Für die SUS habe ich mich nach

Recherche für die Version von Bernard Rummel entschieden. Diese baut auf dem Projekt von Wolfgang Reinhardt auf [vgl. Rum16]. Für das ASQ wurde keine standardisierte Übersetzung gefunden, weshalb die drei englischen Aussagen als Grundlage für eine Übersetzung ins Deutsche verwendet wurden. Diese wurden zusammen mit Prof. Dr. Rainer Koschke in möglichst leicht verständliche, aber dennoch sinnerhaltende Aussagen nach Lewis übersetzt [vgl. Lew91, S. 79]:

1. Die Aufgabe war mit Visual Studio (und SEE) leicht zu lösen.
2. Die Aufgabe war mit Visual Studio (und SEE) in vertretbarer Zeit zu lösen.
3. Ich bin mit den Hinweisen und Hilfen von Visual Studio (und SEE) zufrieden.

Die Aussagen beziehen sich anders als im Original direkt auf die benutzte Software, was weitere Unklarheiten beseitigt. Eine Übersetzung wurde vor allem deshalb erstellt, da der Fragebogen auf Deutsch gestaltet werden soll und eine Einführung von englischen Sätzen einen Bruch darstellen würde. Außerdem kann ich bei der Studie nicht das gleiche Verständnis für die Aussagen voraussetzen. Somit ist eine Übersetzung und damit auch Verständnis für alle Probanden gegeben. Zum Vergleich hier nochmal die Aussagen von Lewis aus dem Original [Lew91, S. 79]:

1. „Overall, I am satisfied with the ease of completing the tasks in this scenario.“
2. „Overall, I am satisfied with the amount of time it took to complete the tasks in this scenario.“
3. „Overall, I am satisfied with the support information (on-line help, messages, documentation) when completing the tasks?[sic!]“

5.4.3 Verwendete Software und Hardware

Um die Evaluation durchführen zu können muss weitere Software benutzt werden. So zum Beispiel eine Webseite, auf der die Probanden die Umfrage ausfüllen können und auch Anwendung um die Testumgebung zu erreichen. Dafür habe ich folgende Software verwendet:

Entferntes Zugreifen Für den entfernten Zugriff wurde die Software TeamViewer[1] gewählt. Diese wurde schon in vorigen Projekten verwendet und kann den Zugriff auf den eigenen Rechner für die Studie einschränken. Außerdem müssen die Probanden diese Software nicht installieren, lediglich das Herunterladen der Software wird benötigt.

Umfrage Bein Testen nach einer geeigneten Website für die Umfrage habe ich mich mit Google Formulare[2] und SoSciSurvey[3] beschäftigt. Dabei fiel die Auswahl auf SoSciSurvey, weil es kostenlos von der Universität Bremen bereitgestellt wird.

Kommunikation Die Kommunikation während der Durchführung der Umfrage sollte für die Probanden möglichst einfach sein. Des Weiteren sollte die Umfrage auch den Datenschutz beachten. Dafür wurden BigBlueButton[4] und Zoom[5] in Betracht gezogen. Beide Dienste werden von der Universität bereitgestellt. Letztendlich habe ich mich für die Verwendung von BigBlueButton entschieden, aufgrund der größeren Erfahrung mit dieser Software.

[1] https://www.teamviewer.com/de (besucht am 25.12.2021)
[2] https://www.google.com/intl/de/forms/about (besucht am 26.12.2021)
[3] https://survey.uni-bremen.de/sosci/admin (besucht am 26.12.2021)
[4] https://bbb.zfn.uni-bremen.de (besucht am 29.12.2021)
[5] https://uni-bremen.zoom.us/ (besucht am 29.12.2021)

Desktopumgebung Für die von den Probanden verwendete Desktopumgebung wurde ein neuer Benutzer angelegt. So kann gewährleistet werden, dass während der Durchführung keine unerwünschte Software für Ablenkung sorgt. Auf dem Desktop befinden sich lediglich Ordner, in denen alle für die Studie benötigten Daten enthalten sind. Für die Interprozesskommunikation verwenden die Probanden Visual Studio 2019. Diese ist bereits, wie die restlichen benötigten Anwendungen und Erweiterungen, vorinstalliert.
Der Computer, auf den zugegriffen wird, hat Windows 10 vorinstalliert. Als Prozessor ist ein AMD Ryzen 7 3700x installiert sowie als GPU eine AMD Radeon RX 5700 XT. Das System weist insgesamt 16 GB RAM vor.

5.4.4 Stichprobe

Da es im Rahmen dieser Studie nicht möglich ist, eine randomisierte Stichprobe zu erheben, wird auf die Stichprobentechnik Convenience Sample zurückgegriffen. Das Convenience Sampling umfasst das Einbeziehen von Probanden, die einfach zu Erreichen sind und sich auch bereit erklären an einer Studie teilzunehmen [vgl. TY07, S. 78]. Für die Suche nach Probanden bedeutet das in diesem Fall, dass hauptsächlich Studenten als Probanden infrage kommen. Dabei werden zum Teil die potenziellen Teilnehmer direkt angeschrieben sowie durch Bekanntgabe der Studie nach freiwilligen gesucht, die Interesse an einer Teilnahme äußern. Die zu testende Software richtet sich hauptsächlich an Softwareentwickler, weshalb alle Probanden mit der Softwareentwicklung vertraut sein müssen.

5.5 Pilotstudie

Die Pilotstudie wurde mit einer Person durchgeführt und hat einige Fehler aufgedeckt. Dabei stellte sich die Steuerung von SEE über TeamViewer als zu langsam und verzögert heraus. Das hatte ebenfalls zur Folge, dass an Objekte teilweise zu nah gezoomt wurde, was die Steuerung unpräziser machte. Deshalb habe ich mir eine alternative Steuerung überlegt. Die Probanden müssen hierbei auf die Stelle zeigen, auf die gezoomt werden soll. Außerdem wurde der Aufbau der Umfrage leicht angepasst. Nach der Willkommensseite folgt nun das Einrichten von TeamViewer, die demografischen Fragen und anschließend die Einführung in die Software. Zum Schluss wurden die Labels sowie unnötige Objekte in SEE entfernt, da sich diese als störend herausgestellt haben.

5.6 Auswertung

In diesem Abschnitt folgt die Aufzählung aller erhaltener Messwerte sowie die Auswertung dieser. Insgesamt haben an der Benutzerstudie zehn Probanden teilgenommen. Die befragten Personen waren hauptsächlich Studenten. Weitere Informationen zu den demografischen Daten sind in Unterabschnitt 5.6.1.

5.6.1 Demografische Daten

In Abbildung 5.6 befinden sich alle Diagramme, die die demografischen Daten darstellen. Ein zusätzliches Diagramm für die Geschlechterverteilung wurde nicht erstellt, da alle zehn befragten Probanden männlich waren. Fünf der Teilnehmer waren zwischen 20 und 24 Jahre alt. Insgesamt liegt die Alterspanne bei 20 bis 34 Jahren. Jeweils vier gaben an, keine oder weniger als ein Jahr Erfahrung mit C# gesammelt zu haben. Allerdings besitzen sieben, der befragten, mehr als drei Jahre generelle Programmiererfahrung. Keiner gab an überhaupt keine

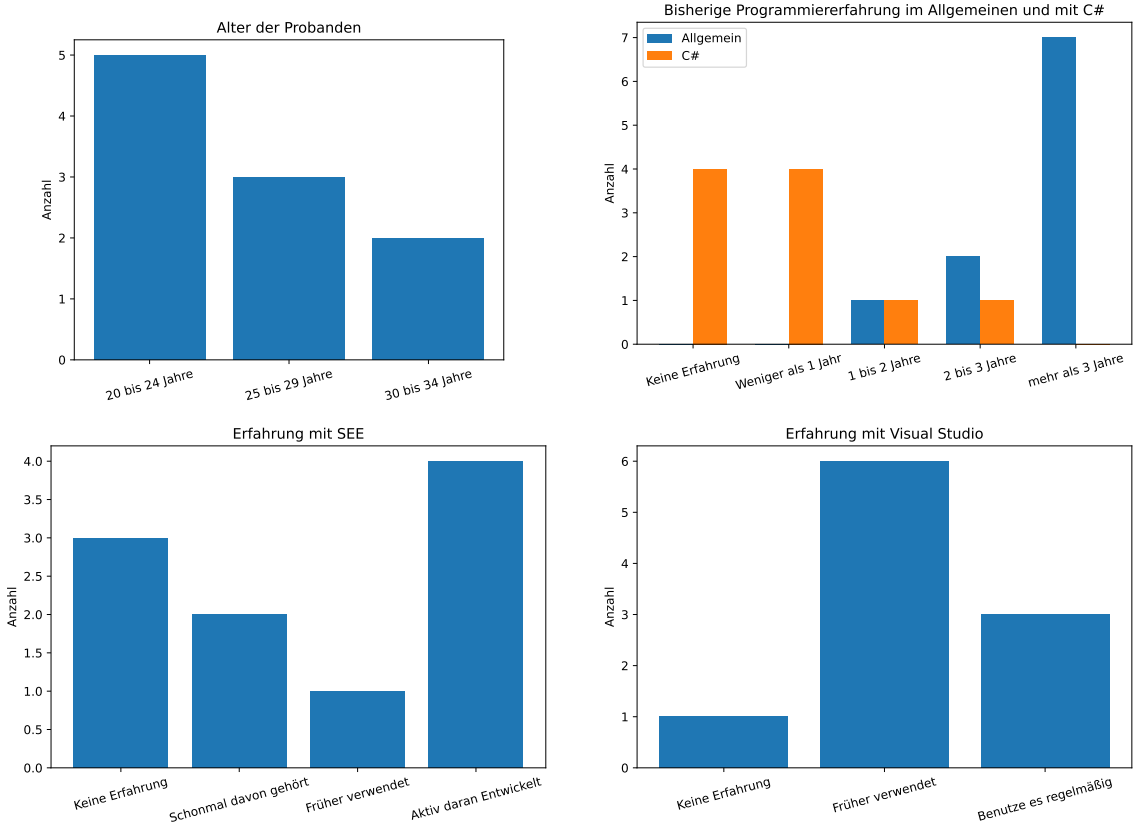

Abbildung 5.6: Darstellung aller erhobenen demografischen Daten

Erfahrung mit dem Programmieren gemacht zu haben. Somit ist bei allen die Stichproben-Voraussetzung erfüllt, dass die Probanden mit der Softwareentwicklung vertraut sein müssen, um die Konzepte von Klasse und Methode nachvollziehen zu können. Die Anzahl der Teilnehmer, die bereits mit SEE gearbeitet und daran entwickelt haben, ist gleich der Anzahl die keine Erfahrung gesammelt haben. In diesem Fall fasse ich keine Erfahrung und schon mal davon gehört zusammen, da beide Gruppen die genau Interaktionsweise nicht kennen. Sechs Probanden geben an, dass sie Visual Studio bereits in der Vergangenheit benutzt haben und nur eine Person hatte vor der Studie noch keine Möglichkeit Visual Studio zu verwenden.
Da bei der Studie hauptsächlich Studenten teilgenommen haben und dementsprechend eher eine junge Gruppe an Menschen befragt wurde, ist die Stichprobe nicht vollständig für die Grundgesamtheit der Softwareentwickler generalisierbar. Jedoch bietet diese Studie einen ersten Eindruck für den Unterschied zwischen beiden Verwendungsarten.

5.6.2 Quantitative Daten

In diesem Abschnitt präsentiere ich zunächst die erhaltenen Ergebnisse aus der Benutzerstudie, bevor diese in Unterabschnitt 5.6.4 auf ihre Signifikanz analysiert werden. Die Präsentation erfolgt mithilfe von Tabellen, in denen abschließend jeweils der Mittelwert sowie Median angegeben ist. Ebenfalls findet eine Visualisierung durch Boxplots statt, um die Verteilung für ein besseres Verständnis zu visualisieren. Alle Ergebnisse sind bis auf die zweite Nachkommastelle aufgerundet angegeben.

Ergebnisse der Aufgaben In Aufgabe eins und zwei ist bereits vor dem Signifikanztest absehbar, dass es zu keiner statistischen Signifikanz kommen kann. Beide Durchläufe der

ersten Aufgabe weisen denselben Mittelwert (0,10) sowie Median (0,00) auf. In der dritten Aufgabe gibt es im Mittel zwar eine leichte Verbesserung von 0,03, jedoch besitzen beide immer noch den gleichen Median von 0,00. Deshalb habe ich mich entschlossen, anstatt die Aufgaben einzeln zu betrachten, den gesamten Durchlauf für die Fehlerquote zu untersuchen. In Tabelle 5.4 sind alle Werte für die Bearbeitung der Aufgaben von den Probanden aufgelistet. Die Tabelle stellt jeweils die Werte für den gesamten Durchlauf dar.

Probanden	**Aufgaben**			
	ohne Erweiterung		**mit Erweiterung**	
	Zeit (Minuten)	**Fehler**	**Zeit (Minuten)**	**Fehler**
P01	12,47	0,33	11,53	0,00
P02	5,25	0,22	6,92	0,00
P03	7,23	0,11	5,43	0,22
P04	5,83	0,00	8,97	0,67
P05	10,58	0,11	7,47	0,00
P06	8,65	0,11	8,38	0,00
P07	9,97	0,00	6,90	0,11
P08	10,32	0,00	11,35	0,11
P09	8,03	0,11	6,40	0,00
P10	7,35	0,22	9,21	0,00
Mittelwert	8,57	0,12	8,26	0,11
Median	8,34	0,11	7,93	0,00

Tabelle 5.4: Ergebnisse der benötigten Zeit und Fehlerquote

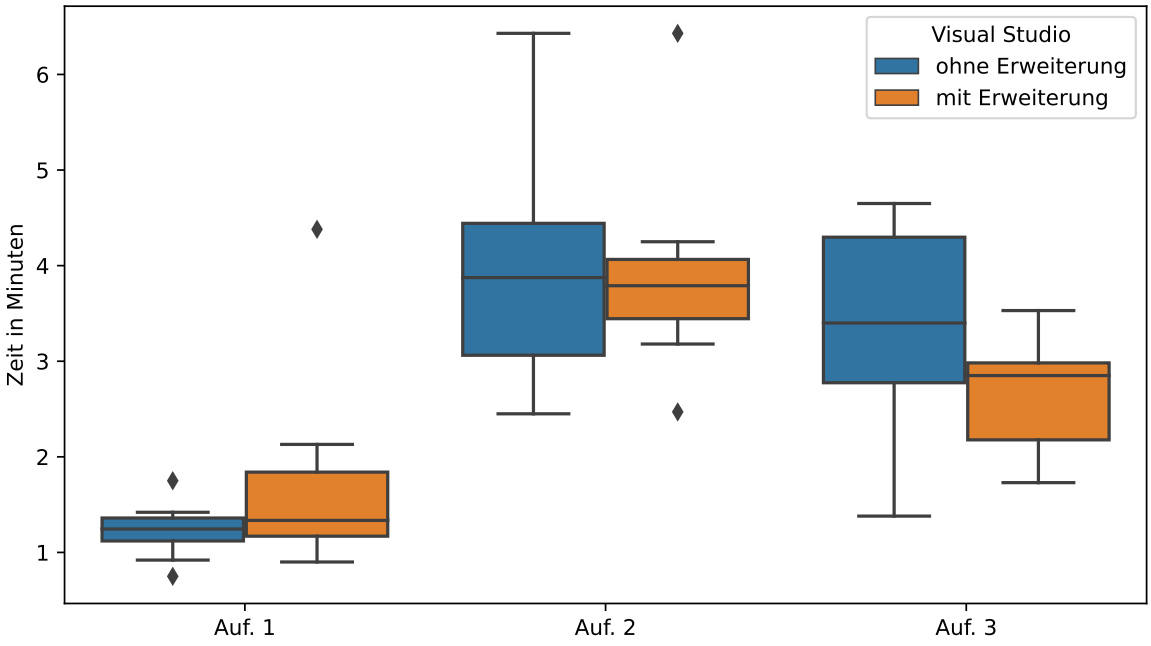

Abbildung 5.7: Vergleich zwischen Bearbeitungszeiten pro Aufgabe

Da auch die unterschiedlichen Bearbeitungszeiten innerhalb einer Aufgabe je Durchlauf interessant sind, sind diese Werte in Abbildung 5.7 visualisiert. Im Durchschnitt ist die Bearbeitung der ersten Aufgabe mit der Erweiterung 0,50 Minuten langsamer. Im Gegensatz dazu liegt der Unterschied zwischen beiden Median nur bei 0,16 Minuten, da die Aufgabe ohne Erweiterung im Median 1,24 Minuten gebraucht hat, während mit der VsSeeExtension 1,40 Minuten benötigt wurden. Die große Diskrepanz zwischen diesen Werten liegt an dem Ausreißer von 4,38 Minuten. Im Gegensatz zur ersten Aufgabe zeigt der Vergleich bei der zweiten eine leichte Verbesserung. So kann sich die Bearbeitungszeit durch die Erweiterung im Durchschnitt um 0,01 Minuten auf 3,90 Minuten verbessern. Im Median beträgt die Verbesserung 0,08 Minuten. Die Veränderung ist nicht besonders groß und wird aufgrund der Stichprobengröße auch nicht signifikant sein, aber der Signifikanztest folgt erst in den nächsten Abschnitten. Zum Schluss noch Aufgabe drei, in der nach fehlender Dokumentation gesucht wurde. Hier ist eine Verringerung der benötigten Zeit direkt ersichtlich. So hat sich die Zeit im Schnitt um 0,79 Minuten verbessert und im Median um 0,62 Minuten.
Die Fehlerquote spiegelt den durchschnittlichen Benutzerfehler pro Aufgabe wider. Um die Fehlerquote für eine Aufgabe zu berechnen, wird jede Angabe, die der Proband zu einer Aufgabe gemacht hat, mit der Lösung verglichen. Korrekt ist eine Angabe, wenn diese an der richtigen Position angegeben wurde und deren Werte stimmen. Rechtschreibfehler wurden dabei ignoriert. Die Gleichung für die Fehlerquote eine Aufgabe sieht wie folgt aus:

$$\text{Fehlerquote} = 1 - \frac{\text{Richtige Antworten}}{\text{Anzahl der Antworten}} \tag{5.1}$$

In Aufgabe drei kam es vor, dass beim Ermitteln der fehlenden Kommentare fälschlicherweise zu viele in Visual Studio gezählt wurde. Da mit SEE jedoch der gleiche Fehler passiert wäre, wird diese Antwort ebenfalls akzeptiert. Für eine Klasse in Aufgabe zwei gab es den Fall, dass es mehrere Methoden mit der gleichen maximalen Anzahl an Stil-Verletzungen gab. So das die Angabe eines Methodennamens von diesen als richtig Antwort gewertet.

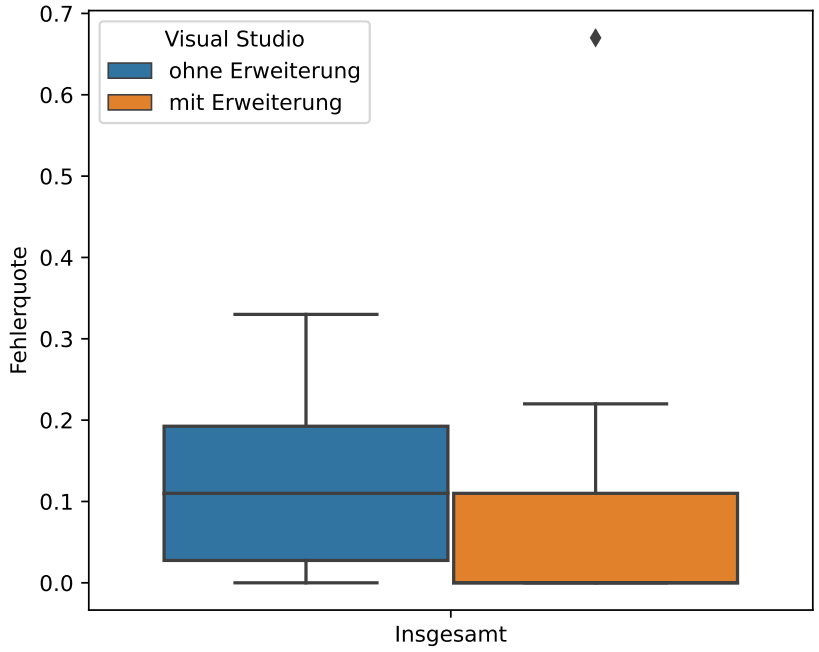

Abbildung 5.8: Vergleich der Fehlerquote in einem Durchlauf

Der Ausreißer mit einer Fehlerquote von 0,67 bei Visual Studio mit Erweiterung ist aufgrund

von falscher Zeilenangaben und Methodennamen entstanden. So kam es vor, dass die Animation zum Anzeigen von Stil-Verletzungen in SEE nicht schnell genug war. Dadurch ist der Eindruck entstanden, es gäbe nur eine Stil-Verletzung anstatt von 13. Die Veränderung der Fehlerquote kann in Abbildung 5.8 betrachtet werden. Insgesamt ist im Durchschnitt eine Verbesserung von 0,01 gegenüber Visual Studio ohne Erweiterung und im Median von 0,11 festzustellen.

Ergebnisse der ASQ Da mit dem After-Scenario-Questionnaire die Zufriedenheit direkt nach einer Aufgabe gemessen wird, kann so eine Visualisierung über die Aufgaben stattfinden. Nochmal zur Erinnerung, eine höhere Zufriedenheit bedeutet einen geringeren Score. Der ASQ-Score ergibt sich aus dem Mittelwert der Angaben zu den jeweiligen Aussagen [vgl. Lew95, S. 75]. Tabelle 5.5 gibt eine Übersicht über alle ASQ-Scores.

Probanden	**ASQ-Score**					
	ohne Erweiterung			**mit Erweiterung**		
	Auf. 1	**Auf. 2**	**Auf. 3**	**Auf. 1**	**Auf. 2**	**Auf. 3**
P01	1,67	4,00	4,00	4,67	2,00	2,33
P02	2,00	4,00	1,67	1,67	2,00	2,00
P03	2,00	4,00	3,33	1,00	1,33	1,00
P04	3,00	3,00	5,00	1,33	1,00	1,00
P05	1,67	2,67	2,00	1,00	1,00	1,00
P06	1,00	3,00	3,67	1,00	1,33	1,33
P07	4,00	6,33	6,67	3,33	3,33	2,33
P08	4,00	6,33	5,67	1,67	3,33	1,67
P09	3,00	6,33	6,67	1,00	1,00	1,00
P10	2,00	2,33	3,67	1,33	1,33	1,33
Mittelwert	2,43	4,20	4,23	1,80	1,77	1,50
Median	2,00	4,00	3,83	1,33	1,33	1,33

Tabelle 5.5: After-Scenario-Questionnaire-Score aller Probanden

So sind fast alle Probanden bei der Bearbeitung der unterschiedlichen Aufgaben mit Erweiterung deutlich zufriedener als ohne. Beim Vergleich des Durchschnitts der ersten Aufgabe kann so eine Differenz von 0,63 sowie im Median 0,67 Punkten festgestellt werden. Der Ausreißer, mit dem Wert 4,67 in der ersten Aufgabe mit Erweiterung könnte eventuell durch Verwirrung mit dem vorher ausgefüllten SUS-Fragebogen entstanden sein. Dies ist allerdings eine rein spekulative Annahme und hat keine Auswirkung auf die Evaluation. Aufgabe zwei und drei zeigen noch deutlichere Unterschiede der Zufriedenheit. So konnte sich Aufgabe zwei im Schnitt um 2,43 Score-Punkte verbessern und im Median um 2,67. Auch die letzte Aufgabe weist eine durchschnittliche Verbesserung der Zufriedenheit von 2,73 Punkten auf. Im Median hat sich die Zufriedenheit um 2,5 Punkte verbessert. In der späteren Auswertung könnten diese Werte durch die große Verbesserung signifikant sein. In der Visualisierung in Abbildung 5.9 werden die unterschiedlichen Zufriedenheiten nochmals verdeutlicht.

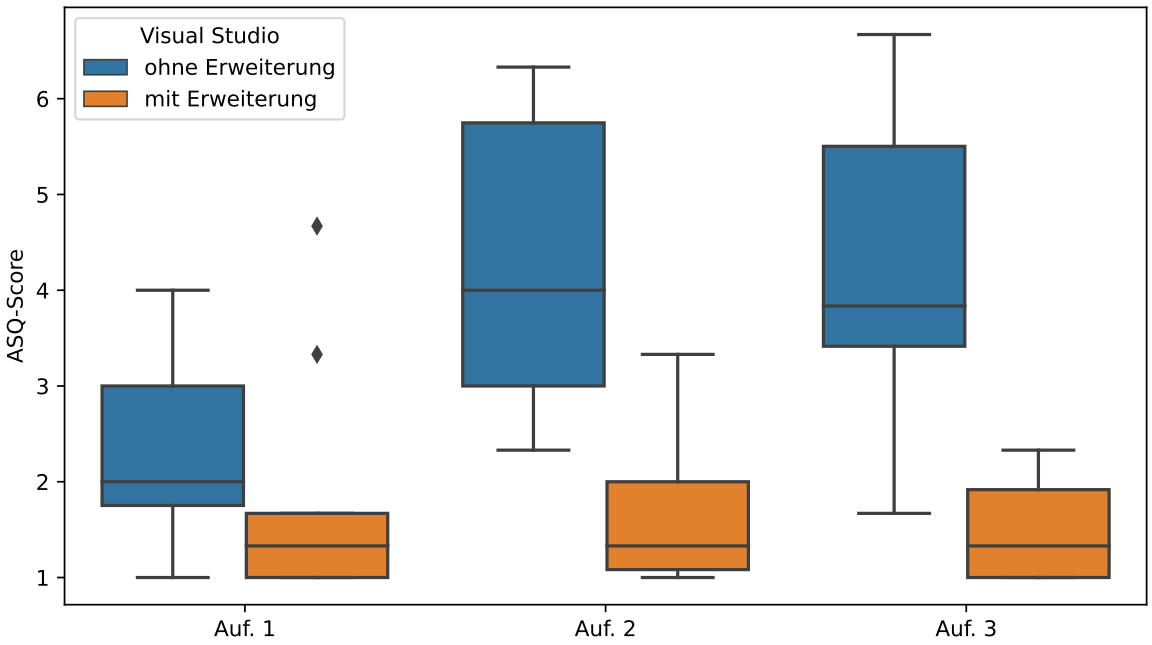

Abbildung 5.9: Vergleich der ASQ-Scores nach jeder Aufgabe

Ergebnisse der SUS Abschließend möchte ich die Ergebnisse für die System Usability Scale in Tabelle 5.6 angeben. Nach Brooke werden zunächst die Werte zu allen positiven Aussagen um eins verringert. Jede zweite Aussage wird umgewandelt, indem fünf minus dem Aussagenwert gerechnet wird. Aus den neu erhaltenen Scores wird die Summe gebildet, damit diese anschließend mit 2,5 multipliziert wird. So findet die Bildung des SUS-Scores in einem Bereich von 0 bis 100 statt [vgl. Bro96, S. 5]. Auch wenn es den Anschein haben mag, soll der Score keine Prozentzahl darstellen [vgl. Bro13, S. 35].

Probanden	**SUS-Score**	
	ohne Erweiterung	**mit Erweiterung**
P01	67,50	65,00
P02	77,50	90,00
P03	82,50	95,00
P04	77,50	92,50
P05	72,50	77,50
P06	80,00	82,50
P07	62,50	82,50
P08	57,50	87,50
P09	42,50	100,00
P10	45,00	97,50
Mittelwert	66,50	87,00
Median	70,00	88,75

Tabelle 5.6: System Usability Scale-Score aller Probanden

In Abbildung 5.10 findet sich die Visualisierung wieder. Dabei schneidet, mit einem durch-

schnittlichen SUS-Score von 87, Visual Studio mit der Erweiterung zur Interprozesskommunikation besser ab. Das entspricht einen Abstand von 20,5 Punkten zur Interaktion ohne Erweiterung. Dieser Unterschied ist zwar im Median nicht ganz so groß, aber mit 18,75 dennoch eine deutliche Verbesserung gegenüber der alten Interaktionsmöglichkeit.

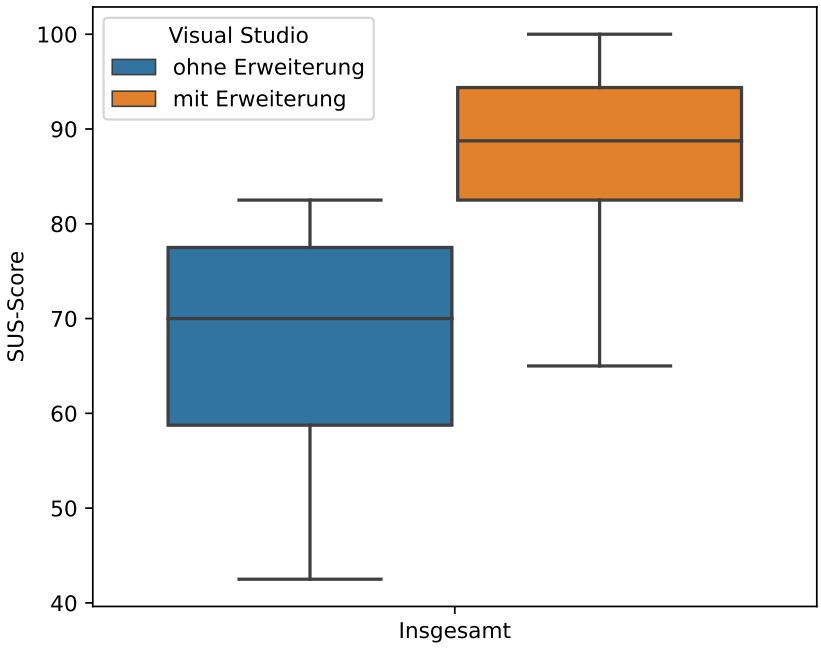

Abbildung 5.10: Vergleich zwischen den SUS-Score

Abschließend zu den präsentierten Ergebnissen kann gesagt werden, dass in fast allen Bereichen Verbesserungen vorgekommen sind, mit Ausnahme der Bearbeitungszeit von Aufgabe eins. Ob diese Daten nun eine statistische Signifikanz aufweisen wird in den nächsten Abschnitten geklärt.

5.6.3 Anmerkungen der Probanden

Durchlauf ohne Erweiterung Von den Probanden gab es zu der Bearbeitung der Aufgaben einige Anmerkungen. So waren die Symbole zur Darstellung von Softwareerosion teilweise nicht unterscheidbar, was möglicherweise der Verwendung von TeamViewer geschuldet war. Zudem wurde von einer Person die Einrückung einer Klasse als verwirrend bewertet. Als Verbesserungsvorschlag für den Umgang wurde eine Visualisierung aller Informationen als zusätzliche Option mit Filtermöglichkeit genannt.

Durchlauf mit Erweiterung So wurde bemängelt, dass die Steuerung von SEE zu komplex sei und die Metriken an einigen Stellen zu unübersichtlich sind, da die Blöcke teilweise überlappen. Viele der befragten Probanden wünschen sich eine automatische Zoomfunktion beziehungsweise eine bessere Hervorhebung des selektierten Elementes in SEE. Ein weiterer Kritikpunkt ist das wechseln zwischen den beiden Anwendungen. Dies sollte automatisch passieren und nicht von den Benutzern manuell. Trotz dieser Kritikpunkte wurde die VsSee-Extension von vielen befragten als sinnvolle Erweiterung befunden.

5.6.4 Signifikanztest

Um die in Abschnitt 5.1 Nullhypothesen zu testen, führe ich einen Signifikanztest auf den dazugehörigen Messwerten durch. Dabei werden die in Unterabschnitt 5.6.2 genannten Ergebnisse mit einem Signifikanzniveau von 5% untersucht. Der zugrundeliegende Datensatz besteht aus gepaarten Stichproben. Das bedeutet, dass beide Stichproben voneinander abhängig sind, da beide sich gegenseitig durch den Versuchsaufbau beeinflussen. Somit muss als Signifikanztest der Wilcoxon-Vorzeichen-Rang-Test verwendet werden.
Nach Cleff wird dieser Test auf gepaarten Stichproben mit mindestens ordinaler Skalierung getestet und stellt keine Anforderungen zur Verteilung auf. Aus den gepaarten Stichproben werden die Differenzen berechnet und anschließend mit Rängen versehen, wobei Nulldifferenzen ignoriert werden. Diese sind ausgehend vom Vorzeichen der Differenz positiv oder negativ. Aus den so gebildeten Rängen findet eine Bildung von positiven und negativen Rangsummen statt. Das Minimum beider Rangsummen ergibt einen empirischen Testwert, der anschließend mit einem kritischen Wert verglichen wird [vgl. Cle19, S. 164-168].
Im Folgenden benutze ich Python für die Berechnung der empirischen Signifikanz. Dafür wurde die Bibliothek *scipy.stats* mit der Funktion *wilcoxon(...)* verwendet. Aufgrund der gewählten einseitigen Hypothesentests gibt die Funktion *wilcoxon(...)* als Testwert die Summe aller positiven Ränge sowie die empirische Signifikanz zurück [vgl. com]. Bei der Untersuchung der Ergebnisse wird nur die empirische Signifikanz betrachtet. Für eine vollständige Angabe wird jedoch die Test-Statistik mitgenannt. Die Ergebnisse sind, falls nötig, bis auf die dritte Nachkommastelle gerundet angegeben.

Bearbeitungszeit Um zu überprüfen, ob die Bearbeitung signifikant gesunken ist, muss auf $H1_0 : B_O \leq B_M$ gegen $H1_1 : B_O > B_M$ getestet werden. Die Bearbeitungszeit ohne Erweiterung ist als B_O angegeben und mit Erweiterung als B_M.

- **Aufgabe 1:** Im Gegensatz zu den anderen Werten ist hier keine Verbesserung zu erkennen (siehe Abbildung 5.7). Somit ist ein Test auf eine signifikante Verringerung der Bearbeitungszeit nicht sinnvoll. Stattdessen wird auf eine signifikante Verschlechterung der benötigten Zeit getestet. Dafür muss ich zwei neue Hypothesen aufstellen und $H5_0 : B_O \geq B_M$ gegen $H5_1 : B_O < B_M$ testen. Die Berechnung ergab eine Test-Statistik von 12,0 und eine empirische Signifikanz von 0,065. Die neu erstellte Nullhypothese $H5_0$ wird daher weiter angenommen. Somit ist die Bearbeitung von Aufgabe 1 mit der neuen Erweiterung auch nicht signifikant langsamer.

- **Aufgabe 2:** Die Berechnungen ergaben, dass die Test-Statistik 28,0 beträgt sowie die empirische Signifikanz 0,5. Die Nullhypothese $H1_0$ wird daher für die zweite Aufgabe weiter angenommen und die Gegenhypothese $H1_1$ verworfen. Die Verringerung der Bearbeitungszeit hat somit keine statistische Signifikanz.

- **Aufgabe 3:** Die Anwendung der Berechnung ergab eine Test-Statistik von 46,0 und einer empirischen Signifikanz von 0,032. Aufgrund des festgelegten Signifikanzniveaus kann ich $H1_0$ verwerfen. Das bedeutet, dass die Bearbeitungszeit in Aufgabe drei statistisch signifikant geringer ist.

Benutzerfehler Als letzter objektiver Wert wird die Fehlerquote der Probanden betrachtet. Da, wie bereits geschrieben, die Fehlerquote im Einzelnen sich nicht groß unterscheidet, findet die Betrachtung auf den gesamten Durchlauf statt. Dabei wird die Fehlerquote auf $H2_0 : F_O \leq F_M$ gegen $H2_1 : F_O > F_M$ getestet. F_O stellt die Fehlerquote ohne Erweiterung

dar, während F_M die Fehlerquote mit VsSeeExtension darstellt. Der Wilcoxon-Vorzeichen-Rang-Test ergab, dass die Test-Statistik 34,5 beträgt sowie die empirische Signifikanz 0,278. Damit konnte keine statistische Signifikanz zur Verbesserung der Fehlerquote gezeigt werden.

Zufriedenheit Da die Steigerung der Zufriedenheit nach einer Aufgabe gemessen wird, findet der Signifikanztest nach jeder Aufgabe statt. Je höher die Zufriedenheit, desto geringer der ASQ-Score. Deshalb muss ich jede Aufgabe mit $H3_0 : Z_O \leq Z_M$ gegen $H3_1 : Z_O > Z_M$ auf ihre Signifikanz testen, wobei die Zufriedenheit ohne Erweiterung durch Z_O und mit Erweiterung durch Z_M bezeichnet wird. Die Ergebnisse, die mit dem Wilcoxon-Vorzeichen-Rang-Test berechnet wurden, sind wie folgt:

- **Aufgabe 1:** Die Berechnung der Differenzen enthielt eine Nulldifferenz, diese wurde in der weiteren Berechnung nicht mehr betrachtet. Das Ausführen der Funktion ergab eine Test-Statistik von 36,0 mit einem empirischen Signifikanzniveau von 0,064. Das empirische Signifikanzniveau überschreitet somit das festgelegte. Daher wird die Gegenhypothese $H3_1$ für die erste Aufgabe verworfen und weiterhin $H3_0$ angenommen.
- **Aufgabe 2:** Die Berechnung ergab eine Test-Statistik von 55,0 mit einem empirischen Signifikanzniveau von 0,001. Das empirische Signifikanzniveau unterschreitet somit das festgelegte Signifikanzniveau. Daher kann die Nullhypothese $H3_0$ für die zweite Aufgabe verworfen werden. Die Zufriedenheit bei der Bearbeitung von Aufgabe zwei mit Erweiterung ist signifikant höher.
- **Aufgabe 3:** Als Ereignisse wurde eine Test-Statistik von 54,0 mit einem empirischen Signifikanzniveau von 0,002 errechnet. Auch hier wird das festgelegte Signifikanzniveau unterschritten, was ebenfalls zur Verwerfung der Nullhypothese $H3_0$ führt. Damit hat sich auch für Aufgabe drei die Zufriedenheit signifikant verbessert.

Gebrauchstauglichkeit Für die Bewertung der statistischen Signifikanz der Gebrauchstauglichkeit, muss ich $H4_0 : G_O \geq G_M$ gegen $H4_1 : G_O < G_M$ testen, wobei die Gebrauchstauglichkeit ohne Erweiterung durch G_O und mit durch G_M bezeichnet wird. Die Berechnung des Wilcoxon-Vorzeichen-Rang-Test ergab eine Test-Statistik von 1,5 mit einem empirischen Signifikanzniveau von 0,002 und ist somit kleiner als das geforderte Signifikanzniveau. Somit kann die Nullhypothese $H4_0$ verworfen werden. Die Gebrauchstauglichkeit von Visual Studio mit Erweiterung ist signifikant höher.

5.7 Diskussion

Nachdem ich den Studienaufbau sowie deren Durchführung und Auswertung durch die Hypothesentests dargelegt habe, bleibt nur noch die Forschungsfrage dieser Abschlussarbeit zu beantworten: *„Bietet die bidirektionale Integration von SEE in eine IDE einen Mehrwert für den Benutzer?“*. Im Folgenden bezieht sich die Angabe von den Ergebnissen immer auf den Median. Bei Betrachtung der objektiven Werte konnte nur die Suche nach den fehlenden Kommentaren eine signifikante Verbesserung von 0,62 Minuten der Bearbeitungszeit vorweisen. Alle anderen objektiven Werte haben zwar kleine Verbesserungen oder Verschlechterungen, diese sind aber in keinem signifikanten Ausmaß. Den Ergebnissen der objektiven Werte sollte allerdings hinzugefügt werden, dass die Probanden durch die Verwendung einer Remote-Sitzung potenziell negativ beeinflusst wurden. So gab es eine Verzögerung, die die Bedienung und das Wechseln zwischen den Anwendungen erschwert hat.
Anders als die objektiven Faktoren sind die subjektiven Aspekte weniger durch die Verwendung der Remote-Sitzung beeinträchtigt. Beim Berechnen des Hypothesentests wurde

nur bei Aufgaben zwei und drei eine signifikante Verbesserung der Zufriedenheit festgestellt, auch wenn es insgesamt in allen Aufgaben zu einem Zuwachs an Zufriedenheit kam. Aufgabe zwei konnte sich dabei um 2,67 Punkte verbessern und Aufgabe drei um 2,5 Punkte. Die Gebrauchstauglichkeit hat sich sogar um 18,75 Punkten verbessert. So wurde die Verwendung von Visual Studio ohne die VsSeeExtension mit einem SUS-Score von 70 Punkten bewertet, hingegen hat die Benutzung der VsSeeExtension den Wert auf 88,75 angehoben.
Insgesamt konnte in dieser Benutzerstudie ein deutlicher Mehrwert in den subjektiven Aspekten festgestellt werden. Jedoch wären, aufgrund der genannten Probleme durch die Remote-Sitzung sowie die demografischen Daten der Stichprobe (siehe Unterabschnitt 5.6.1), ein Weiterführen der Studie mit mehr Probanden und direkten Zugriff auf den zu benutzenden Computer sinnvoll.

KAPITEL 6
Fazit

Abschließen möchte ich diese Arbeit mit einer kurzen Zusammenfassung zu den Ergebnissen sowie einen Ausblick auf mögliche Verbesserdungsideen für die Zukunft der VsSeeExtension.

6.1 Zusammenfassung

Im Rahmen dieser Abschlussarbeit war es mir möglich eine Erweiterung für SEE sowie Visual Studio zu entwickeln. Beide sorgen für einen bidirektionalen Austausch von Informationen über einen TCP-Socket. Zusätzlich zu diesem wird von RPC Gebrauch gemacht. Damit ist es möglich im jeweils andern Prozess vorher spezifizierte Methoden aufzurufen. Beide Erweiterungen sorgen für neue Möglichkeiten der Softwarequalitätsanalyse eines Projektes mithilfe der Kombination aus SEE und einer IDE. So lässt sich nun in der Code-City von SEE ein Element durch das Visual Studio hervorheben. Umgekehrt kann SEE Dateien in Visual Studio öffnen und dort Elemente selektieren.
Mithilfe dieser entwickelten Erweiterung konnte anschließend in einer Benutzerstudie mit 10 Personen geklärt werden, ob die Interprozesskommunikation zwischen beiden Anwendungen einen Mehrwert bietet. Die Probanden waren bei Bearbeitung der Aufgaben zur Softwarequalitätsanalyse deutlich Zufriedener mit der VsSeeExtension als ohne. Dies spiegelt sich auch in dem höheren SUS-Score wider. Auch wenn bei den meisten subjektiven Faktoren eine signifikante Verbesserung gefunden wurde, so hat sich die Bearbeitung bei den meisten objektiv messbaren Werten nicht signifikant verbessert oder verschlechtert. Nur die benötigte Bearbeitungszeit bei der Suche nach fehlenden Kommentaren hat sich so erkennbar verbessert. Zusammenfassend wird ein Mehrwert der Erweiterung in SEE sowie VsSeeExtension durch die subjektiven Faktoren erreicht.

6.2 Ausblick

Zukünftig könnte eine Verbesserung der entwickelten Erweiterung mit den Vorschlägen aus Unterabschnitt 5.6.3 erreicht werden. So zum Beispiel das Implementieren der häufig angemerkten Möglichkeit zum übersichtlicheren Darstellen des selektierten Elementes in SEE. Dabei ist das Heranzoomen an ein Objekt der Code-City die beste Option, um den Benutzern direkt die Auswahl zu präsentieren. Dafür muss die Ausgangsposition der virtuellen Kamera in einem flüssigen Übergang zur neuen Position des ausgewählten Elements wechseln. Damit die Position für den Benutzer nachvollziehbar bleibt, sollte die Kamera von der Ausgangsposition zu einer Übersichtsposition der Code-City wechseln. Erst anschließend sollte das Heranzoomen an das ausgewählte Element erfolgen. Die Berechnung des Kamerapfades kann durch die Bibliothek *TinySpline*[1] geschehen. Diese ist bereits im Projekt SEE vorhanden und wurde zum Beispiel in der Klasse *CameraPath* verwendet, um ein Präsentationsvideo zu erstellen. Die Zoomfunktion sollte aber nicht immer automatisch von der Anwendung ver-

[1] https://github.com/msteinbeck/tinyspline (besucht am 06.02.22)

wendet werden, da dies wieder zu einer subjektiven Verschlechterung führen kann. Dies ist der Fall, wenn ein Benutzer Klassen oder Methoden in der Gesamtübersicht der Code-City vergleichen möchte. Da außerdem die Möglichkeit besteht mehrere Elemente in der Code-City gleichzeitig durch die IDE hervorzuheben, sollte solch eine Zoomfunktion nur dann zugelassen sein, wenn diese sinnvoll ist. So zum Beispiel, wenn es vom Benutzer gewollt ist und nur ein Element selektiert ist.
Damit den Benutzern auch direkt eine Veränderung durch das Betätigen der neuen Schaltflächenelemente auffällt, sollte diese in Zukunft die verbundene Anwendung fokussieren. Zwar besitzt die VsSeeExtension solch eine Implementierung und somit auch die Funktion von SEE fokussiert zu werden, aber diese funktioniert nicht zuverlässig. Das Beheben dieser Problematik sowie das Fokussieren von SEE sollte in späteren Verbesserungen hinzugefügt werden.
Neben diesen zusätzlichen Funktionen, die Hauptsächlich die Bedienung für den Benutzer vereinfachen, gibt es aber auch andere Ideen, um den Funktionsumfang zu erweitern. So kann Visual Studio aktuell nur Methoden und Klassen in C# hervorheben. Hier könnte eine Ausweitung auf Interfaces, Konstanten und vieles mehr geschehen. Diese sollten auch einfach zu implementieren sein.

GLOSSAR

Bezeichnung	**Beschreibung**
After-Scenario-Questionnaire	Ein aus drei Aussagen bestehender Fragebogen, um die Zufriedenheit nach einer Aufgabe zu messen. 29, 35, 45, 49
Axivion Visual Studio Plugin	Erweiterung für Visual Studio zur Darstellung von Softwareerosion. 22, 24, 28
Code-City	Eine virtuelle Stadt, die ein gegebenes Softwareprojekt widerspiegelt. 1, 3, 4, 6–10, 12, 14, 15, 22, 26, 27, 41, 42, 47
Code-Window	Ein Fenster, in dem Quellcode innerhalb von SEE dargestellt werden kann. 7, 8, 13, 47
GameObject	Ein Objekt innerhalb einer Unity-Szene. Es erhält erst durch seine Komponenten Funktionen. 3, 7, 12, 14, 43
Interprozesskommunikation	Austausch von Daten zwischen zwei oder mehreren Anwendungen. iv, 1, 5, 6, 8, 9, 11, 31, 37, 41
Kontextmenü	Das Menü, welches durch den Rechtsklick (wie zum Beispiel im Editor) geöffnet wird. 9, 43
Likert-Skala	Eine für Fragebögen verwendete Skala, in der von „sehr schlecht“ bis „sehr gut“ bewertet werden kann. Dafür steht eine begrenzte Anzahl an Auswahlmöglichkeiten bereit. 29
MenuCommand	Ein Objekt, dass ein Befehl in Visual Studio repräsentiert. Kann zum Beispiel einem Kontextmenü hinzugefügt werden. 4, 43
Mono Framework	Open-Source-Implementierung des .NET Frameworks. 5
MonoBehaviour	Ein Skript, welches einem *GameObject* in Unity angefügt werden soll, erbt von dieser Klasse. 3, 12
OleMenuCommand	Ähnlich zu *MenuCommand*, jedoch können die Befehle automatisch geladen werden. 4, 16
Prefab	Vordefiniertes *GameObject* als Vorlage. 8, 13
Remote Procedure Call	Ermöglicht das entfernte Aufrufen von Methoden. 5, 45
Remote-Sitzung	Fernzugriff auf einen Computer durch einen andern. 39, 40

Bezeichnung	Beschreibung
Softwareerosion	Beschreibt die Verschlechterung der Software über die Zeit. Dies wird häufig durch die Modifikation dieser Software ausgelöst. Bei so einer Software wird häufig ein Re-Engineering-Prozess angewandt [vgl. dB12, S. 132]. 3, 22, 23, 26, 28, 29, 37, 43, 47
StreamRpc	Projekt, dass das entfernte Aufrufen von Methoden in anderen Prozessen ermöglicht. 6, 8, 10, 11, 15, 16
System Usability Scale	Systematischer Fragebogen zur Messung der Gebrauchstauglichkeit. 29, 36, 45, 49
Unity	Spiele-Engine speziell für die Entwicklung von Spielen. 3, 5, 11–15, 43, 47
Usability	Ein aus Erlernbarkeit, Effizienz, Einprägsamkeit, Fehlerquote sowie Zufriedenheit bestehende Eigenschaft eines Systems [vgl. Nie93, S. 26]. 2, 22
Virtual-Network-Computing	Protokoll, um auf den Desktop eines anderen Rechners zuzugreifen. 6, 45
Visual Studio	IDE entwickelt von Microsoft für diverse Programmiersprachen. iv, 1, 2, 4, 5, 7–19, 21–25, 28–32, 34, 35, 37, 39–44, 47
Visual-Studio-Command-Table	Beschreibt die Befehle, die in einer Erweiterung beinhaltet sind. Ist eine XML-Datei, die später kompeliert wird [vgl. Mic21d]. 4
VsSeeExtension	Eine im Laufe dieser Abschlussarbeit entwickelte prototypische Implementation für die Lösung der Fragestellung. 7, 9, 14, 16–19, 22, 24, 28, 29, 34, 37, 39–42, 47
vswhere	Programm zum Finden der Programmpfade von den installierten Visual Studio Instanzen. 5, 13
Wilcoxon-Vorzeichen-Rang-Test	Signifikanztest für die Untersuchung von zwei gepaarten Stichproben, über die keine Verteilungsannahme getroffen werden kann und mindestens ordinalskaliert sind [vgl. Cle19, S. 164]. 38, 39

ABKÜRZUNGSVERZEICHNIS

Bezeichnung	**Beschreibung**
ASQ	After-Scenario-Questionnaire 29, 30, 35, 36, 39, 47
IDE	Integrierte Entwicklungsumgebung iv, 1, 2, 4, 6, 8–12, 14–16, 18, 19, 22, 29, 39, 41, 42, 44
RPC	Remote Procedure Call 5, 6, 41
SEE	Software-Engineering-Experience iv, 1–5, 7–18, 21, 23–26, 28–32, 34, 35, 37, 39, 41–43, 47
SUS	System Usability Scale 29, 35–37, 40, 41, 47
VNC	Virtual-Network-Computing 6, 9

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

Literaturverzeichnis

[Lew91] James R. Lewis. "Psychometric Evaluation of an After-Scenario Questionnaire for Computer Usability Studies: The ASQ". In: *SIGCHI Bull.* 23.1 (Jan. 1991), S. 78–81. DOI: 10.1145/122672.122692.

[Nie93] Jakob Nielsen. *Usability Engineering*. 1. Aufl. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[Lew95] James R. Lewis. "IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use". In: *International Journal of Human–Computer Interaction* 7.1 (1995), S. 57–78. DOI: 10.1080/10447319509526110.

[Bro96] John Brooke. *SUS: A quick and dirty usability scale*. 1996.

[Ric+98] Tristan Richardson u. a. "Virtual network computing". In: *IEEE Internet Computing* 2.1 (1998), S. 33–38. DOI: 10.1109/4236.656066.

[Die03] Stephan Diehl. "Softwarevisualisierung". In: *Informatik-Spektrum* 26.4 (2003), S. 257–260. DOI: 10.1007/s00287-003-0314-4.

[TY07] Charles Teddlie und Fen Yu. "Mixed Methods Sampling: A Typology With Examples". In: *Journal of Mixed Methods Research* 1.1 (2007), S. 77–100. DOI: 10.1177/1558689806292430.

[SL09] Jeff Sauro und James R. Lewis. "Correlations among Prototypical Usability Metrics: Evidence for the Construct of Usability". In: CHI '09. Boston, MA, USA: Association for Computing Machinery, 2009, S. 1609–1618. DOI: 10.1145/1518701.1518947.

[KEN10] Adrian Kuhn, David Erni und Oscar Nierstrasz. "Embedding Spatial Software Visualization in the IDE: An Exploratory Study". In: *Proceedings of the 5th International Symposium on Software Visualization*. SOFTVIS '10. Salt Lake City, Utah, USA: Association for Computing Machinery, 2010, S. 113–122. DOI: 10.1145/1879211.1879229.

[dB12] Lakshitha de Silva und Dharini Balasubramaniam. "Controlling software architecture erosion: A survey". In: *Journal of Systems and Software* 85.1 (2012). Dynamic Analysis and Testing of Embedded Software, S. 132–151. DOI: 10.1016/j.jss.2011.07.036.

[Bro13] John Brooke. "SUS: A Retrospective". In: *Journal of usability studies* 8.2 (Feb. 2013), S. 29–40.

[Gre13] Elke Greifeneder. "Benutzerforschung". In: *Handbuch Methoden der Bibliotheks- und Informationswissenschaft: Bibliotheks-, Benutzerforschung, Informationsanalyse*. Hrsg. von Konrad Umlauf, Simone Fühles-Ubach und Michael Seadle. De Gruyter Saur, 2013, S. 257–283. DOI: 10.1515/9783110255546.257.

[Sea13] Michael Seadle. "Entwicklung von Forschungsdesigns". In: *Handbuch Methoden der Bibliotheks- und Informationswissenschaft Bibliotheks-, Benutzerforschung, Informationsanalyse*. Hrsg. von Konrad Umlauf, Simone Fühles-Ubach und Michael Seadle. De Gruyter Saur, 2013, S. 41–63. DOI: 10.1515/9783110255546.41.

[Man14] Peter Mandl. *Grundkurs Betriebssysteme*. 4. Aufl. Springer Vieweg, Wiesbaden, 2014. DOI: 10.1007/978-3-658-06218-7.

[BSB15] Gergő Balogh, Attila Szabolics und Arpád Beszédes. "CodeMetropolis: Eclipse over the city of source code". In: *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE. 2015, S. 271–276.

[NWC16] Divya Natesan, Morgan Walker und Shannon Clark. "Cognitive Bias in Usability Testing". In: *Proceedings of the International Symposium on Human Factors and Ergonomics in Health Care* 5.1 (2016), S. 86–88. DOI: 10.1177/2327857916051015.

[Rum16] Bernard Rummel. *System Usability Scale – jetzt auch auf Deutsch*. 1. Feb. 2016. URL: https://blogs.sap.com/2016/02/01/system-usability-scale-jetzt-auch-auf-deutsch/ (besucht am 05. 01. 2022).

[Int17] ECMA International. *Standard ECMA-404 - The JSON Data Interchange Syntax*. 2. Aufl. Dezember 2017. URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/.

[SW17] Carsten Seifert und Jan Wislaug. *Spiele entwickeln mit Unity 5*. 3. Aufl. Carl Hanser Verlag München, 2017.

[Rii18] Sirpa Riihiaho. "Usability Testing". In: *The Wiley Handbook of Human Computer Interaction*. Hrsg. von Kent L. Norman und Jurek Kirakowski. 1. Aufl. Bd. 1. John Wiley & Sons Ltd., 2018, S. 255–275. DOI: 10.1002/9781118976005.ch14.

[Cle19] Thomas Cleff. *Angewandte Induktive Statistik und Statistische Testverfahren. Eine computergestützte Einführung mit Excel, SPSS und Stata*. 1. Aufl. Springer Gabler, Wiesbaden, 2019. DOI: 10.1007/978-3-8349-6973-6.

[Mic20] Microsoft. *Establishing a JSON-RPC connection*. 6. Okt. 2020. URL: https://github.com/microsoft/vs-streamjsonrpc/blob/main/doc/connecting.md (besucht am 11. 02. 2022).

[Gal21] Falko Galperin. "Visualisierung von Code-Smells in Code-Cities". Unveröffentlichte Bachelorarbeit. Universität Bremen, 2021.

[Mic21a] Microsoft. *Commands, menus, and toolbars*. 6. Aug. 2021. URL: https://docs.microsoft.com/en-us/visualstudio/extensibility/internals/commands-menus-and-toolbars (besucht am 17. 01. 2022).

[Mic21b] Microsoft. *Interprocess Communications*. 7. Jan. 2021. URL: https://docs.microsoft.com/en-us/windows/win32/ipc/interprocess-communications (besucht am 04. 02. 2022).

[Mic21c] Microsoft. *SetForegroundWindow function (winuser.h)*. 13. Okt. 2021. URL: https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setforegroundwindow (besucht am 10. 02. 2022).

[Mic21d] Microsoft. *Visual Studio Command Table (.Vsct) Files*. 6. Aug. 2021. URL: https://docs.microsoft.com/en-us/visualstudio/extensibility/internals/visual-studio-command-table-dot-vsct-files (besucht am 15. 01. 2022).

[Mic21e] Microsoft. *Welcome to the Visual Studio IDE*. 14. Sep. 2021. URL: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide (besucht am 27. 12. 2021).

[Tec22a] Unity Technologies. *Creating and Using Scripts*. 6. Feb. 2022. URL: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html (besucht am 29. 01. 2022).

[Tec22b] Unity Technologies. *GameObjects*. 15. Jan. 2022. URL: https://docs.unity3d.com/Manual/GameObjects.html (besucht am 18. 01. 2022).

[com] The SciPy community. *scipy.stats.wilcoxon*. URL: https://docs.scipy.org/doc/scipy-1.8.0/html-scipyorg/reference/generated/scipy.stats.wilcoxon.html (besucht am 08. 02. 2022).

[Mica] Microsoft. *TcpClient Class*. URL: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.tcpclient (besucht am 18. 01. 2022).

[Micb] Microsoft. *Visual Studio Locator*. URL: https://github.com/Microsoft/vswhere (besucht am 26. 12. 2021).

[Pro] Mono Project. *System.IO.Pipes.NamedPipeServerStream Class*. URL: http://docs.go-mono.com/?link=T%3aSystem.IO.Pipes.NamedPipeServerStream (besucht am 20. 01. 2022).

[Sha] Shana. *StreamRpc*. URL: https://github.com/shana/StreamRpc (besucht am 27. 12. 2021).