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

Entwicklung eines IDE-Plugins zur Integrierung von Erklärungsbedarf in Codeabschnitte

Development of an IDE plugin that integrates the need for explanation into code sections

Bachelorarbeit

im Studiengang Informatik

von

Lucas Burgath

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Dr. Jil Klünder
Betreuer: Hannah Deters

Hannover, 11.12.2024

Erklärung der Selbstständigkeit

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

Hannover, den 11.12.2024

Lucas Burgath

Zusammenfassung

Entwicklung eines IDE-Plugins zur Integrierung von Erklärungsbedarf in Codeabschnitte

Endnutzer können häufig nicht alle Aspekte und Funktionen einer Software verstehen, insbesondere die Gründe für bestimmte Resultate sind dabei manchmal unklar. Daher sollten Entwickler in ihren Anwendungen Erklärungen bereitstellen. Im Rahmen dieser Arbeit wurde ein Konzept entwickelt und als Proof-of-Concept implementiert, das den Prozess der Erklärungsintegrierung unterstützt. Dabei wird die Zuordnung von Use Cases und Erklärungen aus erweiterten Use Case-Tabellen verwendet, um Codeabschnitten entsprechende Use Cases und somit Erklärungen zuzuordnen. Die Gesamtfunktionalität wurde in zwei Bereiche gegliedert: Die Geschäftslogik, welche in einem separaten Package umgesetzt wurde, und die Benutzerschnittstelle, welche als Plugin für eine integrierte Entwicklungsumgebung bereitgestellt wird. Mithilfe aspektorientierter Programmierung lassen sich gezielt einzelne Funktionen mit Use Cases verknüpfen, wodurch passende Erklärungen für diese Codeabschnitte gespeichert werden können. Die Verwaltung erfolgt durch das Singleton-Entwurfsmuster, das die Speicherung und das Abrufen von Erklärung zentral steuert. Diese beiden Prinzipien bilden die Grundlage des Packages und wurden hier in C# implementiert. Um die Anwendung für Entwickler zu erleichtern, wurde ein Plugin für den VS Code Editor entwickelt. Es ermöglicht die Zuordnung von Erklärungen zu Codeabschnitten durch visuelle Markierungen im Editor. In einer kleinen Nutzerstudie wurde gezeigt, dass die entwickelte thread-safe Lösung nutzbar ist. In dieser Arbeit konnte erfolgreich demonstriert werden, dass ein effektives und thread-safes Plugin entwickelt werden kann, um Erklärungsbedarf direkt in Codeabschnitte zu integrieren. Dies verbessert nicht nur die Wartbarkeit und Konsistenz des Codes, sondern erleichtert auch den Entwicklungsprozess.

Abstract

Development of an IDE plugin that integrates the need for explanation into code sections

End users often find it difficult to understand all aspects and functions of a software, especially the reasons for certain results remain unclear in some cases. So developers should provide explanations within their applications. In the context of this work, a concept was developed and implemented as a proof-of-concept to support the process of integrating explanations. The approach involves using mappings of use cases and explanations from extended use case tables to associate sections of code with the corresponding use cases and therefore explanations. The functionality was divided into two sections: The business logic, which was implemented in a separate package, and the user interface, which is provided as a plugin for an integrated development environment. Using aspect-oriented programming, specific functions can be specifically linked with use cases, allowing suitable explanations for these code sections to be stored. Management is handled through the singleton design pattern, which primarily controls the storage and retrieval of explanations. These two principles form the foundation of the package and were implemented in C#. To simplify usage for developers, a plugin for the VS Code editor was developed. It enables the assignment of explanations to code sections using visual highlighting in the VS Code editor. A brief user study proofed that the developed thread-safe proof-of-concept solution is usable. This work successfully demonstrated that an effective and thread-safe plugin can be developed to integrate explanations directly into code sections. This not only improves the maintainability and consistency of the code but also facilitates the development process.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Softwaresysteme integrieren sich immer mehr in den Alltag der Gesellschaft. Die Herausforderungen und Probleme, die hierbei gelöst werden müssen, führen aufgrund ihrer Schwierigkeit auch zu immer komplexerer Software. Die Komplexität ist dabei nicht ausschließlich auf den Einsatz von künstlicher Intelligenz (KI) zurückzuführen. [1][2] Auch ohne den Einsatz von KI kann es selbst für Experten herausfordernd sein, die Funktionsweise eines vorliegenden Softwaresystems (vollständig) zu erfassen. [3]

Speziell den Endnutzern ist es dabei häufig nicht möglich zu verstehen, warum ein Programm ein bestimmtes Resultat erzeugt bzw. wie einzelne Komponenten der Software grundsätzlich funktionieren. Sie müssen also zum Teil kritische Entscheidungen treffen, obwohl die Software für sie an sich eine ‚Blackbox' ist. [4] Damit die Funktionsweise eines Programms nicht erst durch zeitintensives Training erlernt oder Nachfragen bei Kollegen erfahren werden muss, sollte die Software sich selbst erklären können. Dabei muss für die nicht-funktionale Anforderung Erklärbarkeit zunächst abhängig von Kontext und Nutzer definiert werden, was erklärt werden soll. Diese Erklärungen müssen dann noch abrufbar in die Software integriert werden. [2]

1.2 Vorarbeit

Um eine klare Planung in der Softwareentwicklung zu ermöglichen, wird im Vorfeld erarbeitet, was der Nutzer später mit der Software erreichen kann. Dabei wird auch festgelegt, wie die Software sich in bestimmten Nutzungssituationen verhält. Diese Nutzungs- bzw. Interaktionsszenarien zwischen Akteur und System, werden auch Use Cases genannt. Diese werden in Tabellenform festgehalten. [5][6][7]

Erklärbarkeit ist vor allem nutzer- und kontextabhängig (dies wird in Kapitel 2 noch genauer beleuchtet). [2] Daher kann bereits in der

Planungsphase mithilfe der Use Cases Erklärungsbedarf für die einzelnen Schritte angegeben werden. [7] In einer vorangegangenen Bachelorarbeit von Denise Behrens [7] wurde bereits erarbeitet, wie die Use Case-Tabellen um Erklärungsbedarf erweitert werden können. Für den nachfolgenden Proof-of-Concept wird angenommen, dass die Tabellen als eine auf das Wesentliche reduzierte CSV-Datei vorliegen (Schrittid und Erklärung). Eine detailreiche Beschreibung der Inhalte der CSV-Datei erfolgt in Abschnitt 4.4.

1.3 Problemstellung

Innerhalb der Planungsphase werden die Use Cases entworfen und die Daten entsprechend in die jeweiligen Use Case–Tabellen aufgenommen. Für die Schritte in den Tabellen wird Erklärungsbedarf erhoben (vgl. Abbildung 1.1 Schritt 1). Ein Explainability Engineer verfasst den Bedarfen entsprechende Erklärungen (vgl. Abbildung 1.1 Schritt 2). Danach müssen die Erklärungen der Use Case Schritte in die entworfene Applikation integriert werden (vgl. Abbildung 1.1 Schritt 3). Dies kann manuell eine herausfordernde, fehleranfällige und zeitintensive Aufgabe sein, welche die Entwickler von anderen Implementierungsaufgaben abhält.

Ziel dieser Arbeit ist es, einen Lösungsansatz für folgende Aufgabe herauszuarbeiten: Entwickler sollen auf Basis der erstellten erweiterten Use Case–Tabelle einer bestimmten Codestelle einen bestimmten Use Case Schritt zuweisen. Es sollte dabei gewährleistet sein, dass die Erklärung aus dieser Tabelle abgerufen und angezeigt werden kann, sobald sich die Software an dieser entsprechenden Codestelle befindet und der Nutzer eine Erklärung anfordert.

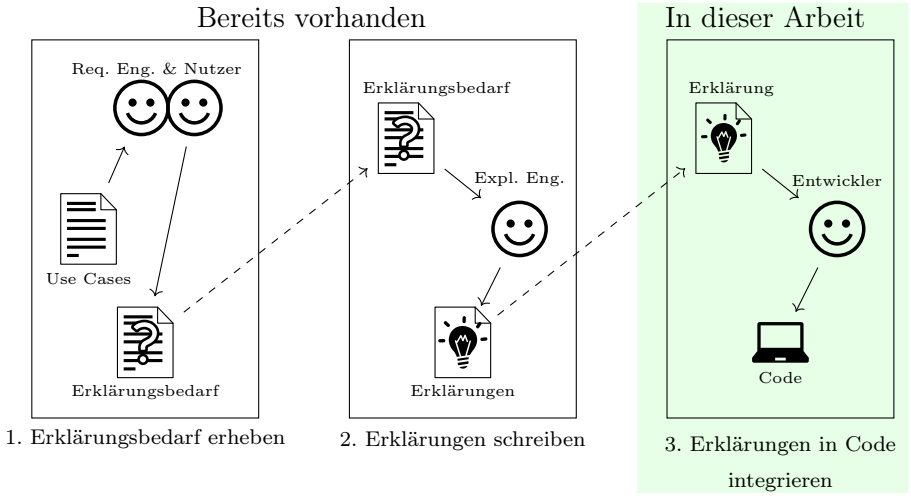


Abbildung 1.1: Ein Requirements Engineer und ein Nutzer erarbeiten zusammen Erklärungsbedarf in den Use Cases. Für diesen schreibt der Explainability Engineer dann die passenden Erklärungen. Diese werden dann vom Entwickler in den Code integriert werden.

1.4 Lösungsansatz

Vor dem genannten Hintergrund müssen zwei Probleme gelöst werden. Einerseits die Zuordnung von Codestellen zu Use Case Schritten. In diesem Fall soll die Zuordnung von Schritt zu Codestelle durch textuelles Markieren der Stelle umgesetzt werden. Dafür werden C#-Attribute genutzt, welche die Id des jeweiligen Use Case Schrittes enthalten (z. B. 3.4 → Im Use Case 3 Schritt 4). Dies wird kein eingebautes Basis-Attribut sein, denn diese bieten nur eine Möglichkeit Code deklarativ mit Informationen zu verknüpfen und können keine Funktionen gezielt auslösen. [8] [9] Diese Eigenschaft kann aber durch den Einsatz von bestimmten Packages erhalten werden. Das Attribut wird die Fähigkeit besitzen, ein anderes Objekt zu benachrichtigen (mehr dazu in den Abschnitten 2.2, 2.3, 3.2 und 4.1).

Andererseits braucht es ein Objekt, welches speichern oder in irgendeiner Weise nachverfolgen kann, ob sich die Ausführung in einem aktuellen Use Case Schritt mit Hilfestellung befindet und welche Erklärung damit ausgelöst wird. Dieses Objekt muss von jeder Stelle im Code aus ansprechbar sein, also für die Benachrichtigungen oder für das Abfragen der Hilfestellung zur Verfügung stehen. Zusätzlich muss dieses Objekt jederzeit den aktuellen Schritt kennen und in jedem Thread des Programms inhaltliche Konsistenz sicherstellen.

1.5 Struktur der Arbeit

In dieser Arbeit werden zunächst in Kapitel 2 die Grundlagen erläutert. Dazu erfolgt ein kurzer Einblick in die Themen Erklärbarkeit, Singleton, Aspektorientierung und IDE-Plugins. Anschließend wird das Konzept der Implementierung in Kapitel 3 detailliert vorgestellt. In diesem Kapitel wird zunächst eine allgemeine Übersicht über die konzipierte Architektur präsentiert, gefolgt von einer detaillierten Beschreibung der einzelnen Komponenten. Nachdem die Konzepte dargelegt wurden, werden in Kapitel 4 wesentliche Details der Implementierung präsentiert und dazu bestimmte Implementierungsentscheidungen begründet. Zuletzt wird mithilfe einer kurzen Nutzerstudie aufgezeigt, dass das Entwickelte benutzbar ist. Dafür wird die Nutzerstudie in Kapitel 5 genauer beschrieben und in Kapitel 6 die erhaltenen Ergebnisse vorgestellt und ausgewertet. Die Arbeit schließt mit einem Fazit (Kapitel 7), welches die wichtigsten Erkenntnisse zusammenfasst, und einen Ausblick auf zukünftige Forschungsarbeiten in diesem Themenbereich präsentiert.

Kapitel 2

Grundlagen

Um die späteren Teile der Konzeptionierung und Implementierung nachvollziehen und einordnen zu können, wird zunächst ein kurzer Einblick in relevante Konzepte gegeben. Dazu gehören Erklärbarkeit, Singleton, aspektorientiertes Programmieren und IDE-Plugins.

2.1 Erklärbarkeit

Die nicht-funktionale Anforderung (NFR) Erklärbarkeit gewinnt vor allem in Zeiten des vermehrten Einsatzes von KI an Bedeutung. Aber auch unabhängig vom Einsatz der KI wird Erklärbarkeit immer relevanter. Um die Substanz der Erklärbarkeit besser erfassen zu können, wird zunächst eine allgemeine Definition betrachtet: Ein System ist genau dann in einem Aspekt erklärbar, wenn ein Corpus an Informationen an den Adressaten gegeben wird, welcher es dem Adressaten erlaubt, den Aspekt zu verstehen. [4]

Explizit auf Software übertragen, bedeutet das: Erklärbarkeit ist die Fähigkeit einer Software, Erklärungen bereitzustellen, welche Information über beliebige Systemaspekte verständlicher für die Stakeholder machen. Erklärbarkeit hat dabei einen großen Einfluss auf andere Qualitätsaspekte von Systemen bzw. Software. So ist sie z. B. ein Mittel, um das Vertrauen in das System zu stärken, könnte aber in anderen Fällen das Vertrauen schwächen. [4] [10]

Der Erklärungsbedarf kann nach Unterbusch et al. [11] in unterschiedliche Teilbereiche aufgeteilt werden:

- **Training** beinhaltet Erklärungsbedarf, der auftritt, wenn Nutzer mit dem System bzw. Feature nicht vertraut sind, weil dieses neu oder geändert worden ist.

- **Interaction**, also Erklärungsbedarf, der sich bei normaler Nutzung eines mit dem System vertrauten Nutzers ergibt.

- **Business** ist jener Erklärungsbedarf, der nicht zwingend durch Systeminteraktion, sondern durch Businessziele, Projekte oder Prozessanforderungen auftritt. Zum Beispiel, warum die Software für eine Aktion benötigt wird, welche zuvor auf eine andere Art und Weise durchgeführt wurde.

- **Dissatisfaction** umfasst die Forderung nach Erklärung aufgrund eines subjektiven Mangels an der Software aus Sicht des Nutzers. Zum Beispiel den Grund einer Änderung am Workflow bzw. der UI.

- **Errata** schließt die Forderung nach Erklärung ein, welche aufgrund eines objektiven Mangels am System auftritt. Zum Beispiel eine den Nutzer fehlleitende Fehlermeldung, wie „Du bist nicht mit dem Internet verbunden“. Dabei hört dieser gleichzeitig Musik über einen Streamingdienst und ist somit für sich selbst erkennbar mit dem Internet verbunden.

In den ersten drei Kategorien (Training, Interaction, Business) ist der Erklärungsbedarf das Hauptanliegen. Bei den beiden letzten Kategorien (Dissatisfaction, Errata) ist dieser zweitrangig, da es außer dem fehlendem Wissen andere Probleme mit dem System gibt.

Auch Droste et al. [10] haben ähnliche Kategorien für Erklärungsbedarf identifiziert: Systemverhalten, Interaktion, Domänenwissen, Privacy & Security und User Interface. Einzig die Businessziele von Unterbusch et al. [11] sowie die Kategorie Privacy & Security von Droste et al. [10] stellen jeweils eine Kategorie dar, die in der anderen Kategorisierung nicht enthalten ist.

2.2 Singleton

Die Werkzeuge der Programmierung gehen noch weit über spezifische Sprachen, ihre Anweisungen und Schlüsselwörter hinaus. Dazu gehören auch eher theoretische Konzepte, wie die sog. Entwurfsmuster (Design Patterns). Dies sind Pläne bzw. Konzepte, wie bestimmte Problemstellungen auf eine objektorientierte Weise angegangen werden sollten. Dabei ist es wichtig zu wissen, dass diese Designs eher auf langjähriger praktischer Erfahrung und nicht auf einem tatsächlichen Beweis beruhen. [12]

Ein Singleton ist ein solches Design Pattern, welches zu den Erzeugungsmustern gehört. Diese befassen sich mit der Abstraktion des Kreierens bzw. der Instanziierung von Objekten. [12][13] Kurz gesagt, wie und wann Objekte erstellt werden. [13]

Mithilfe des Singleton-Musters wird dafür gesorgt, dass nur eine einzige Instanz einer Klasse kreiert werden kann. Im gesamten Programm existiert damit nur eine einzige Instanz der Klasse, welche über einen globalen Zugriff erreicht werden kann. Dabei hat diese Singleton-Klasse die Aufgabe eine

Instanz von sich selbst zu erstellen, zu speichern und den Zugriff auf die Instanz bzw. Daten der Instanz zu regulieren. Gleichzeitig muss sie aber auch die Instanziierung durch andere Programmteile verbieten. [12][13][14]

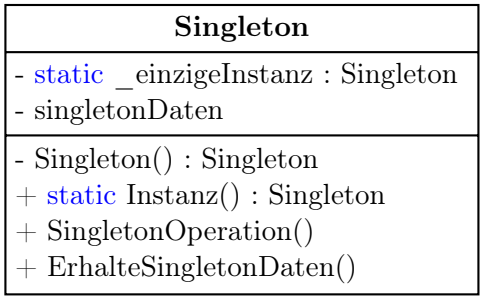


Abbildung 2.1: Struktur der Singleton-Klasse nach UML 2.4.1, angelehnt an [12, S. 173]

Durch eine genaue Betrachtung der Abbildung 2.1 lassen sich die genannten Eigenschaften in den Funktionen identifizieren. Die Klasse speichert eine private, statische Instanz von sich selbst (`_einzigeInstanz`). Um mit einer Instanz dieser Klasse arbeiten zu können, muss jedes andere Objekt dafür die statische `Instanz()`-Methode dieser Klasse aufrufen. Diese Methode gibt nur die gespeicherte statische Instanz der Klasse selbst (`_einzigeInstanz`) zurück. Dadurch, dass der normale Konstruktor der Klasse privat ist, kann kein anderes Objekt von außerhalb eine neue Instanz der Klasse erzeugen. Somit kann in dem gesamten Programm nur die statisch gespeicherte Instanz aus der Klasse genutzt werden. [12] Das `static` Schlüsselwort sorgt dafür, dass die Variable an die Klasse (also den Typ) gebunden wird und eben nicht an eine Instanz bzw. ein Objekt der Klasse. [15] Eine genauere Erläuterung der Implementierung in C# folgt in Abschnitt 4.1.

2.2.1 Threadsicherheit beim Singleton

Bei Betrachtung der Abbildung 2.2 lässt sich leicht feststellen, warum threadsafety eine große Rolle bei Singletons einnimmt. Wird nicht darauf geachtet, könnte es dazu kommen, dass ein anderer Thread die erste Instanziierung stört (dieser läuft nicht synchronisiert ab). Im schlimmsten Fall führt das Fehlen von Synchronizität zu der Situation auf der linken Seite der Abbildung 2.2. Jeder Thread hat festgestellt, dass es noch keine Instanz gibt, eine Instanz kreiert und arbeitet jetzt auf dieser. Damit existiert mehr als nur eine Instanz der Singleton-Klasse. Wird beim Erstellen des Singletons genau auf Threadsicherheit geachtet, ergibt sich die im rechten Teil der Abbildung dargestellte Situation: Alle einzelnen Threads „teilen“ sich eine Instanz der Singleton-Klasse. [13][14][16]

Wie dies im Detail umgesetzt wird und welche Probleme jeweils beim Multithreading auftreten können, ist abhängig von der Programmiersprache.

Zu C# mehr in Kapitel 4 (zu Java in [13] von Dooley oder in [16] von Musch).

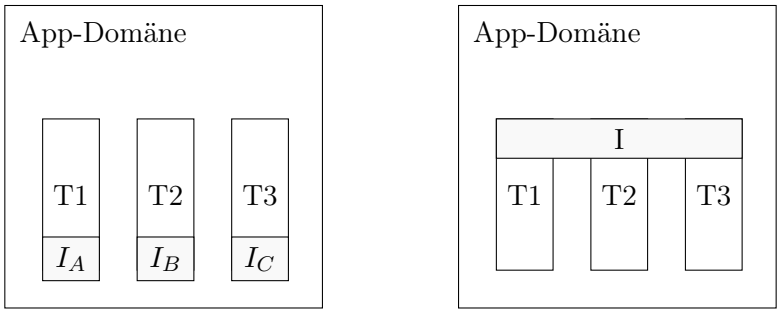


Abbildung 2.2: Stark vereinfachte schematische Abbildung der Nutzung von Instanzen in unterschiedlichen Threads einer App-Domäne. Links eine Singleton Implementierung ohne Beachtung von thread-safety, rechts mit.

2.3 Aspektorientiertes Programmieren

Aspektorientierung (AOP) wurde 1997 als ein weiteres Programmierparadigma neben Objektorientierung (OOP) und prozeduraler Programmierung bei Xerox PARC von Kiczales et al. [17] entwickelt. Eines der Hauptanliegen war der Umgang mit den sog. `cross-cutting concerns` in OOP. Also Funktionalitäten, die gleichzeitig in sämtlichen separaten Ebenen einer Anwendung eingesetzt werden, wie bspw. in der Logik oder der Benutzeroberfläche. Sie schaffen damit Abhängigkeiten zwischen eigentlich separierten Klassen und Methoden (starke Kopplung). Ein Beispiel dafür ist das Logging für Methoden oder UI-Events. [17][18][19]

AOP dient dazu, solche Funktionalitäten (Aspekte) klar und modular an einer Stelle zu definieren, um sie so unabhängig und zentral von der Geschäftslogik entfernt verwalten und gezielt einsetzen zu können. [17]

Um die Grundlagen von AOP zu verstehen, ist es wesentlich, die Begriffe Joinpoints, Advices und Pointcuts genauer zu betrachten:

- **Joinpoints** sind die Punkte, an denen bei Ausführung des Systems Code injiziert werden soll. Zu solchen Injektionspunkten gehören zum Beispiel Methodenein- und ausgang, Variablenzuweisung, Klasseninitialisierung. [17][19][20]

- **Advices** sind dabei die Codestücke, die in einem Joinpoint eingewebt werden sollen. Manchmal können diese sowohl vor als auch nach einem Joinpoint injiziert werden. [19][20]

- **Pointcuts** sind Sammlungen an Jointpoints für einen Advice und bilden damit die tatsächlichen Einstiegspunkte. Diese können häufig entweder direkt als Menge angegeben oder als Muster beschrieben werden. Solche Muster können von einfachem „nach jeder Methode“ hin zu „nach jeder Methode in Klasse XY, bis auf jene Methoden, die mit Y starten“ reichen. [18][19][20]

In vielen modernen objektorientierten Sprachen gibt es (spezielle) Frameworks, Packages oder Bibliotheken, die AOP in die jeweilige Programmiersprache integrieren. Dazu gehören unter anderem AspectJ oder Spring AOP für Spring in Java, AspectC++ in C++, Aspectlib in Python und Postsharp bzw. Metalama in C#. [18][19][21][22][23]

2.3.1 AOP in Anwendung

Nachfolgendes Beispiel inklusive der Abbildung 2.3 und 2.4 veranschaulicht die Anwendungsmöglichkeiten von AOP. Angenommen, die Aufgabe besteht darin, eine Steuerungssoftware für eine elektrische Schiebetür in Java zu entwerfen.

Damit genau Protokoll darüber geführt werden kann, wann die Tür geöffnet und geschlossen wurde, muss logging betrieben werden. Sobald sich die Tür öffnet oder schließt, wird dieser Vorgang vermerkt. Die Tür sollte sich nur dann öffnen, wenn mindestens fünf Personen vor ihr stehen. Und sie sollte so lange geöffnet bleiben, bis alle Personen die Tür passiert haben. Die Funktionalität für das Öffnen, Schließen und Zählen von (passierten) Personen ist bereits vollständig in der `DoorController` Klasse implementiert. Ein Beispiel für die Nutzung der Steuerungssoftware ist in Abbildung 2.3 zu sehen.

```
1     var controller = new DoorController();
2
3     int peopleCount = controller.countPeople();
4
5     if(peopleCount >= 5){
6         controller.activateDoor();
7
8         while(!controller.allPeoplePassed()) {};
9
10        controller.deactivateDoor();
11    }
```

Abbildung 2.3: Exemplarischer Codeausschnitt zur Steuerung einer Tür in Java.

Um im vorangegangenen Code das Logging zu implementieren, wird sich dafür entschieden, AOP zu verwenden. In einem hypothetischen Java AOP Framework wurden die Joinpoints auf den Eintritt der Funktionen gesetzt. Pointcuts werden hier unter anderem über Java Annotationen festgesetzt.

Unter der Annahme, dass bereits ein Logging-Aspekt `Logg` erstellt worden ist, könnte das Markieren folgendermaßen ablaufen:

```
@Logg
public void activateDoor() {...}

@Logg
public void deactivateDoor() {...}
```

Abbildung 2.4: Ausschnitt aus der DoorController Klasse.

Um einen besseren Überblick über die AOP-Struktur des Codes 2.3 zu erhalten, kann die Kette an Funktionsaufrufen in Blöcken darstellt werden. In diesem Beispiel haben alle Personen nach drei `allPeoplePassed()` Aufrufen die Tür passiert:

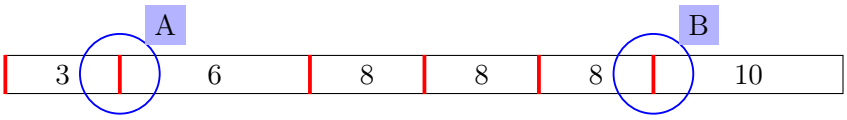


Abbildung 2.5: Ablaufkette an Funktionsaufrufen in Blöcken. Die Nummern in den Blöcken sind die Zeilennummern des Funktionsaufrufs, Joinpoints sind rot markiert und die Logg-Pointcuts blau umkreist.

Im Endeffekt werden AOP Advices in den tatsächlichen Code über die Pointcuts eingewebt. Der Advice, der sich hinter der Annotation `@Logg` verbirgt, wird hier vor activateDoor (Kreis bei `A`) und nochmal vor deactivateDoor (Kreis bei `B`) injiziert. Die Advices sind damit an vielen Stellen einsetzbar und können gekapselt vom Rest bearbeitet werden. Je nach Art des Frameworks geschieht das Einweben zur Kompilierung (statisch) oder zur Ausführung (dynamisch). [19]

2.4 IDE-Plugins

Um den Nutzen von IDE-Plugins grundsätzlich zu verstehen, ist es notwendig sich die beiden Bestandteile, also IDE und Plugin genauer anschauen.

Die **Integrated Development Environment** (IDE, zu dt. integrierte Entwicklungsumgebung) ist eine Softwareanwendung, welche Entwickler dabei unterstützt Anwendungen bzw. Applikationen zu entwickeln. IDEs stellen meist eine Vielzahl an Funktionen für den Entwicklungsprozess bereit. Diese reichen von schreiben bzw. bearbeiten des Codes bis hin zum Bauen, Analysieren und Testen der jeweiligen Anwendung. [24] Über dieses Maß an Funktionen hinaus lassen sich manche auch um benutzerdefinierte Funktionen ergänzen, wie z. B. mit Plugins.

An sich ist ein **Plugin** ein „kleines, optionales Programmteil, das die

einbettende Applikation um oft sehr spezialisierte [...] Funktionen erweitert“ (nach Fischer, [25]). Mit anderen Worten, Plugins sind Programmstücke, mit denen neue benutzerdefinierte Funktionen zu einer bestehenden Applikation hinzufügt werden können.

Durch die Zusammenführung beider Aspekte ergibt sich: Mithilfe von Plugins können die bestehenden Funktionen einer IDE um benutzerdefinierte Funktionen erweitert werden. Die hinzugefügte Funktionalität kann dabei so ausgestaltet werden, dass sie den persönlichen individuellen Anforderungen entspricht. Oder zumindest so weit umgesetzt werden, wie es die Anwendung zulässt. Solche Plugins können danach privat, über das Internet oder einen IDE internen Marktplatz verbreitet bzw. mit Lizenz vertrieben werden.

2.4.1 Die IDE dieser Arbeit: Visual Studio Code

Das Plugin dieser Arbeit wird für Visual Studio Code (VS Code) entwickelt. VS Code ist keine klassische IDE. Sie ist nicht speziell für eine Sprache konzipiert, sondern unterstützt von Haus aus die meisten in der Entwicklung gängigen Programmiersprachen. Dadurch wird jedoch keine umfassende Unterstützung bei der Entwicklung in den meisten Programmiersprachen bereitgestellt. Erst durch den Einsatz von Extensions (zu dt. Erweiterungen) kann tiefgreifende Unterstützung für die meisten Sprachen erhalten werden, wie zum Beispiel das Vorschlagen von sprachenspezifischen Keywords. Diese Extensions bilden den Grundstein für die Arbeit mit VS Code und sind somit eine zentrale Komponente dieser IDE. [26]

Im Grunde genommen ist Extension hier aber nur ein anderes Wort für Plugin. Die Extensions sind also die Plugins von VS Code. Diese können auch viel weniger fundamentale Funktionen als die genannte Sprachunterstützung implementieren. Durch sie können u. a. auch neue Kommandos oder Kontextmenüeinträge hinzugefügt werden. Darüber hinaus ermöglichen Extensions das Auslesen, Bearbeiten und Hinzufügen von Dateien aus dem aktuellen Arbeitsbereich sowie die Verarbeitung von Nutzereingaben über eine Eingabemaske. Sie werden grundsätzlich in TypeScript geschrieben. [27][28]

Damit eine Extension tatsächlich mit Elementen und Funktionen von VS Code interagieren kann, braucht es eine geeignete Schnittstelle. Diese Programmierschnittstelle ist hier die VS Code API.

2.4.2 Arbeiten mit der VS Code API

Die VS Code API besteht aus unterschiedlichen Bereichen, den sog. Namespaces. In diesen sind unterschiedliche Befehle thematisch gebündelt. Wenn beispielsweise innerhalb einer Erweiterung eine Error-Nachricht angezeigt werden soll, dann wird die entsprechende Funktion aus dem window-Namespace aufgerufen. Diese Funktion löst dann die eigentliche Funktionalität in der Applikation aus, wie in Bild 2.6 dargestellt. [29]
Innerhalb der Extension „BeispielErweiterung“ soll eine Fehlermeldung an

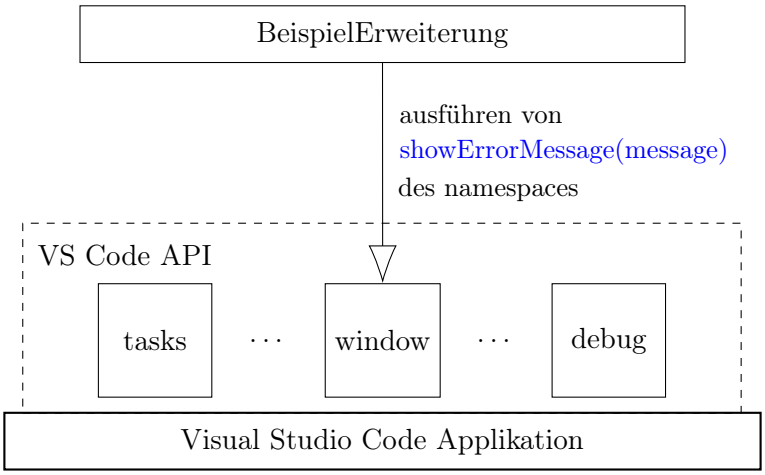


Abbildung 2.6: Vereinfachte strukturelle Darstellung der Nutzung der VS Code API basierend auf der VS Code API Referenz. Dabei sind „tasks“, „window“ und „debug“ drei ausgewählte Namespaces der API. [29]

den Nutzer verschickt werden. Dazu wird die `showErrorMessage` Funktion aus dem „window“ Namespace der VS Code API genutzt. Diese zeigt dann in VS Code ein separates Nachrichtenfenster mit der `message` Fehlermeldung an.

2.4.3 Bereitstellung der Erweiterung

Damit die innerhalb dieser Bachelorarbeit entwickelte Erweiterung von anderen genutzt werden kann, sollte sie installierbar sein. Dazu wird das von Microsoft bereitgestellte Tool „Visual Studio Code Extension Manager“ verwendet. [30] Dafür muss dieses Tool installiert sein und mit dem Terminal in das Projektverzeichnis navigiert werden. Durch Ausführen des Kommandos `vsce package` wird dann eine in VS Code installierbare .vsix Datei erstellt. [31]

Wichtig, am Ende wird mit der elektronischen Abgabe dieser Arbeit (dem Gitlab-Repository) auch die gepackte Extension (.vsix) ausgeliefert. Sie wird also nicht auf dem Visual Studio Code Marktplatz veröffentlicht.

2.5 NASA TLX-Scores

Mithilfe des NASA Task Load Index (TLX) wird die mentale Belastung der Nutzer geschätzt. [32] Eine genauere Beschreibung der Anwendung des TLX erfolgt in dem Kapitel der Nutzerstudie (Kapitel 5). Der NASA TLX wird hier eingesetzt, um die Verwendbarkeit der Erweiterung aufzuzeigen.

Dabei wird die Arbeitslast wie folgt kategorisiert: Ein Score von 0 bis 20 bedeutet sehr niedrige mentale Belastung, von 21 bis 40 niedrige, von 41 bis 60 mittlere, von 61 bis 80 hohe und von 81 bis 100 sehr hohe Arbeitslast.

Diese gleichmäßige Aufteilung erleichtert es, im Rahmen der Studie einen Proof-of-Concept zu liefern, indem sie einen klaren und einfachen Überblick über die mentale Belastung der Nutzer bietet. Gleichzeitig stellt sie ein vorläufiges und flexibles Framework dar, das bei Bedarf mit weiteren Daten spezialisiert werden kann.

Kapitel 3

Konzept

In diesem Kapitel werden die in diesem Projekt eingesetzten Prinzipien und Techniken beschrieben. Die Funktionalitäten werden in zwei Bereiche aufgeteilt: Geschäftslogik und UI. Die Geschäftslogik regelt das Zuordnen, Speichern und Abfragen der Erklärungen. Dabei stellt die UI eine Benutzerschnittstelle für diese Geschäftslogik bereit. Die Geschäftslogik funktioniert grundsätzlich auch ohne die Benutzeroberfläche (UI). Die UI dient lediglich als Ergänzung und Unterstützung bei der Nutzung.

Um das alles besser gewährleisten zu können, ist es sinnvoll, die Geschäftslogik getrennt in einem C#-Package zu implementieren. Die gesamte grafische Benutzerschnittstelle wird als eine VS Code Extension umgesetzt.

3.1 Architektur der Interaktion von Package und Plugin

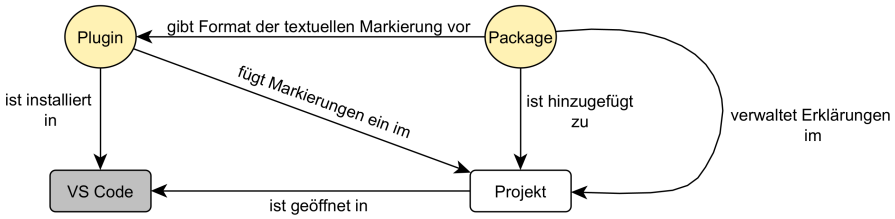


Abbildung 3.1: Grundlegende Architektur des Projektes dieser Arbeit

Abbildung 3.1 zeigt die entworfene Architektur. Die in dieser Arbeit zu entwickelnden Komponenten sind hier durch ihre umkreisten Namen dargestellt (Plugin und Package). Dabei stellt das in VS Code installierte Plugin eine Benutzerschnittstelle für das Markieren im Projekt bereit. Der Nutzer muss eine Codestelle durch Textauswahl visuell markieren, damit das Plugin anschließend eine vom Package vorgegebene textuelle Markierung einfügt. Das Package muss zunächst dem Projekt hinzugefügt werden, damit es hier Erklärungen speichern und abrufbereit halten kann. Beim

Durchlaufen einer markierten Codestelle während der Ausführung wird die entsprechende Erklärung abrufbereit gehalten.

3.2 Package-Konzept

Das C#-Package besteht aus zwei Klassen. Einer Klasse, die das Speichern bzw. Abrufen der aktuellen Use Cases und Erklärung auf Basis des Singleton Patterns umsetzt (die Verwaltungsklasse `Explainer`). Und einer AOP basierten, die der Verwaltungsklasse mitteilt, welchen Use Case das Programm gerade abarbeitet (die Benachrichtigungsklasse `EnteringUseCase`). Das dafür erstellte Klassendiagramm in Abbildung 3.2 schafft einen nachvollziehbaren Überblick über die geplante Struktur.

3.2.1 Struktur des Packages

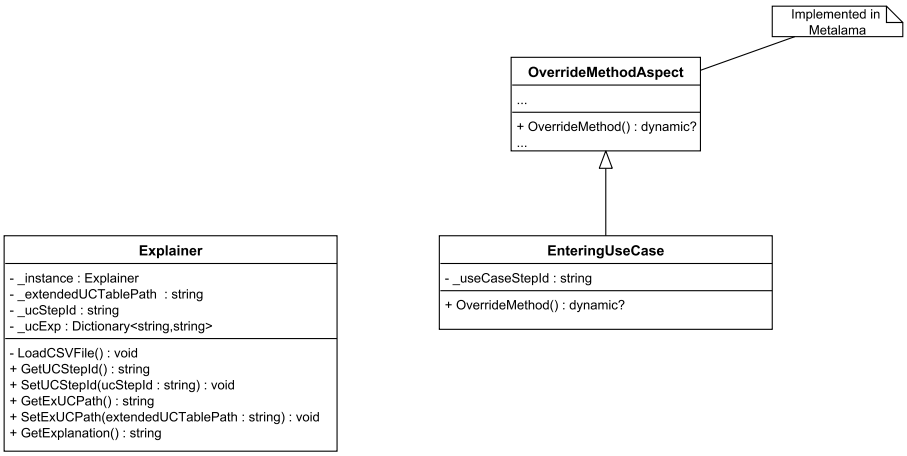


Abbildung 3.2: Verkürztes Klassendiagramm des Packages

Als Basis für die Benachrichtigungsklasse wird die `OverrideMethodAspect` Klasse aus dem Metalama Package genutzt. Diese Klasse überschreibt mithilfe der `OverrideMethod()` eine andere markierte Methode. Es wird also nicht der eigentliche Inhalt der Methode ausgeführt, sondern der Inhalt der `OverrideMethod()` Methode. Der ursprüngliche Inhalt der überschriebenen Methode kann aber an definierter Stelle wieder ausgeführt werden.

Durch die indirekte Oberklasse `Attribute` kann die `EnteringUseCase` Klasse wie ein normales C#-Attribut verwendet werden. Dadurch bietet sich nicht nur die Möglichkeit, Informationen deklarativ einer Codestelle bzw. einem Zielelement zuzuordnen, sondern es können auch Systeminformationen dieser Art an viele verschiedene Zielelemente gleichzeitig gebunden werden. [8][9] Die zu überschreibenden Funktionen werden in dem Package einfach mit `[EnteringUseCase]` markiert (ggf. kann über einen definierten Konstruktor noch der Use Case Schritt mitgegeben werden).

Die `Explainer` Klasse wird nach dem Singleton Pattern (vgl. Abschnitt 2.2) entworfen. Dadurch existiert nur eine Verwaltungsinstanz für die Erklärungen, welche global angesprochen werden kann. Die genauere Umsetzung des Patterns in C# wird im nächsten Kapitel in Abschnitt 4.1 diskutiert.

Das in den Grundlagen gezeigte Pattern wird noch um weitere Felder und Methoden ergänzt. Die Explainer Klasse muss zusätzlich noch speichern, wo die Use Case-Tabelle liegt (`_extendedUCTablePath`) und in welchem aktuellen Use Case Schritt sich die Anwendung gerade befindet bzw. zuletzt befand (`_ucStepId`). Zusätzlich soll die Zuordnung von Use Case Schritt Id und Erklärung gespeichert werden, damit diese nicht bei jeder Anfrage neu geladen werden muss (`_ucExp`). Für die Zuordnung wird ein Dictionary mit der Use Case Schritt Id als Schlüssel und den jeweiligen Erklärungen als Wert verwendet.

Mit den privaten Feldern kann über passende Getter und Setter Methoden gearbeitet werden. Es gibt zusätzlich intern eine Möglichkeit, die CSV-Datei mit den Erklärungen (nach) zu laden (`LoadCSVFile()`). Um eine Erklärung zum aktuell gespeicherten Use Case zu erhalten, muss die `GetExplanation()` Methode aufgerufen werden. Auf den Vorgang des Aufrufens wird im Abschnitt 4.2 genauer eingegangen.

3.2.2 Fehlerbehandlung

Benutzerdefinierte Exceptions gewährleisten neben einer angepassten, funktionsfähigen Fehlerbehandlung auch einen weiteren Erleichterungsschritt für Entwickler. Durch diese können (durch das Plugin auftretende) Fehler und ihre Quellen leichter identifiziert, behandelt und abgefangen bzw. vermieden werden. Diese sind in der Lage, von anderen Programmteilen unter anderem zur Detektion von vermeidbaren Fehlern verwendet zu werden. [33] Um aufzuzeigen, wie Fehlerbehandlung in dieser Proof-of-Concept gestaltet werden kann, wurden auf dieser Grundlage drei exemplarische benutzerdefinierte Exceptions erstellt. Diese sind in dem Klassendiagramm in Abbildung 3.3 dargestellt.

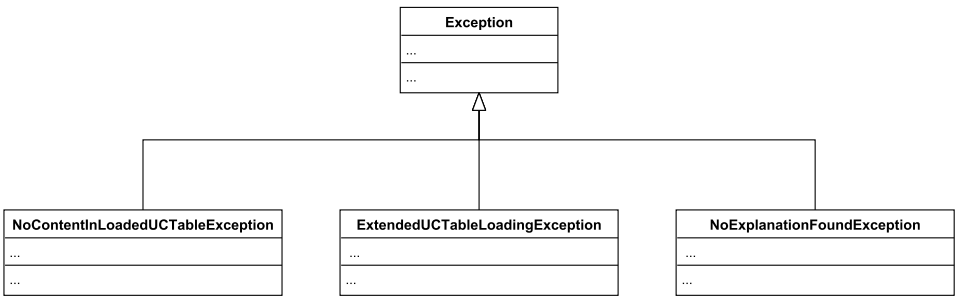


Abbildung 3.3: Verkürztes Klassendiagramm der Package-Exceptions

Dabei wird die `NoContentInLoadedUCTableException` geworfen, sobald die eingelesene Tabelle keine oder inkorrekt formatierte Daten enthält. Tritt ein anderweitiger Fehler beim Laden der Use Case-Tabelle auf, z. B. dass die zu lesende Datei gar nicht existiert, dann wird die `ExtendedUCTableLoadingException` geworfen. Die `NoExplanationFoundException` wird geworfen, falls zur Use Case Schritt Id keine Erklärung in der Tabelle gefunden werden kann. Um Fehler in einem realen Anwendungsszenario besser detektieren und behandeln zu können, sollten die ursprünglichen drei Ausnahmefälle weiter unterteilt und zusätzliche, spezifische Fehlerklassen hinzugefügt werden.

3.2.3 Beispielanwendung des Konzepts

Mit der Grafik 3.4 lässt sich der Kommunikationsfluss eines Beispielprogramms nachvollziehen. Das Objekt der `klasseA` aktualisiert die Daten von `klasseB`, indem es die Methode `UpdateData()` aufruft. Diese ruft intern `CalculateData()` auf, um neue Daten zu berechnen. Bei dieser Funktion wurde in der Planung Erklärungsbedarf festgestellt. Daher wurde für diese das Attribut `EnteringUseCase` festgelegt, um so beim Aufrufen die `OverideMethod()` Funktion auszulösen. Dadurch wird der angegebene Use Case Schritt im `Explainer` hinterlegt (`SetUCStepId(_useCaseStepId)`). Anschließend wird die Ausführung von `CalculateData()` mit `meta.Proceed()` fortgesetzt. Schließlich fordert `klasseA` eine Erklärung vom `Explainer` an, der den aktuellen Use Case überprüft und die entsprechende Erklärung zurückgibt.

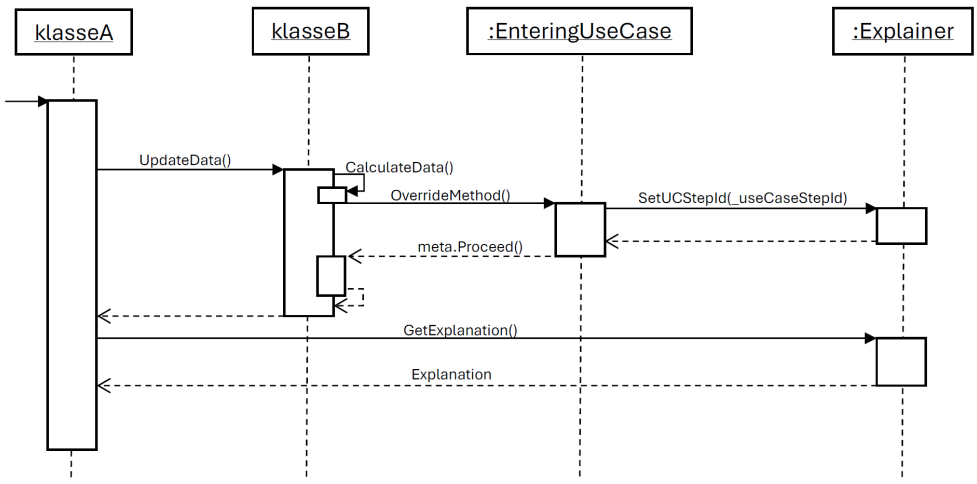


Abbildung 3.4: Sequenzdiagramm eines beispielhaften Programmablaufs

3.2.4 Erweiterungsideen für das Package

Wie bereits in Abschnitt 2.1 aufgezeigt, gibt es unterschiedliche Bereiche von Erklärungsbedarf. Dabei könnte es sein, dass Nutzer möglicherweise Erklärungen für unterschiedliche Bereiche anfordern könnten. Solche Anfragen können in unterschiedlichen Anwendungsfällen auftreten und müssten so je nach Bedarf eine andere Erklärung für den gleichen Schritt ausgeben. Daher könnte pro festgestelltem Bereich der aktuelle Use Case gespeichert und auch unterschiedliche Erklärungen ausgegeben werden. Das wäre hier möglich, indem der Typ von `_udId` auf eine Liste an strings oder ein Dictionary an strings abgeändert wird. In diesen würde dann die Id für einen Bereich gespeichert werden. Dabei müsste untersucht werden, anhand welcher Kriterien der geforderte Bereich korrekt identifiziert werden kann, um dem Nutzer auch die passende Erklärung bereitzustellen.

In dem Zuge könnte ein Verlauf eingebaut werden. Dadurch bestände die Möglichkeit, auch vergangene Erklärungen zu erhalten. Dies könnte dann sinnvoll sein, wenn der Nutzer bei einer Erklärung merkt, dass nicht der aktuelle Schritt, sondern der letzte Schritt ein Verständnisproblem birgt. Dies könnte über ein neues Feld, z. B. einen größenbegrenzten Stack implementiert werden.

3.3 Plugin-Konzept

Das Plugin stützt sich für tatsächliche Codefunktionalitäten auf das C#-Package und stellt nur eine Bedienoberfläche dafür bereit. Dabei werden die Standardelemente der VS Code API genutzt. Zum einen wird ein weiterer Menüpunkt (Kommando) dem Kontextmenü hinzugefügt (mit Shortcut) und zum anderen wird die Benutzeroberfläche zur Auswahl der jeweiligen Use Cases realisiert. Das Plugin muss zum Ladezeitpunkt feststellen, ob das C#-Package installiert ist und den Nutzer im Fehlerfall benachrichtigen. Im Fall des Fehlens darf keine Funktionalität aktiviert werden, da so die Nutzungsgrundlage entfällt.

Für die UI Auswahl des Use Cases wird das QuickPick, wie es in Abbildung 3.5 zu erkennen ist, genutzt. Dabei werden die entsprechenden Use Case Namen aus der Tabelle geladen und können entweder durch Scrollen und Mausklick oder Betätigen der Enter-Taste ausgewählt werden. Durch Eingabe von Zeichen kann nach dem jeweiligen Namen aus der Liste gefiltert werden. Zusätzlich muss beim Versuch des Aktivierens des QuickPicks geprüft werden, ob eine entsprechende Use Case-Tabellen-Datei existiert. Wenn nicht, wird der Nutzer darauf hingewiesen und jegliches Öffnen der Auswahl blockiert. Sollte die Datei existieren, aber keine (lesbaren bzw. korrekt formatierten) Daten enthalten, wird der Nutzer darauf hingewiesen. Nach Auswahl des Use Cases wird an die im Editortab ausgewählte Stelle der Code-Datei das im Package definierte Attribut eingefügt.

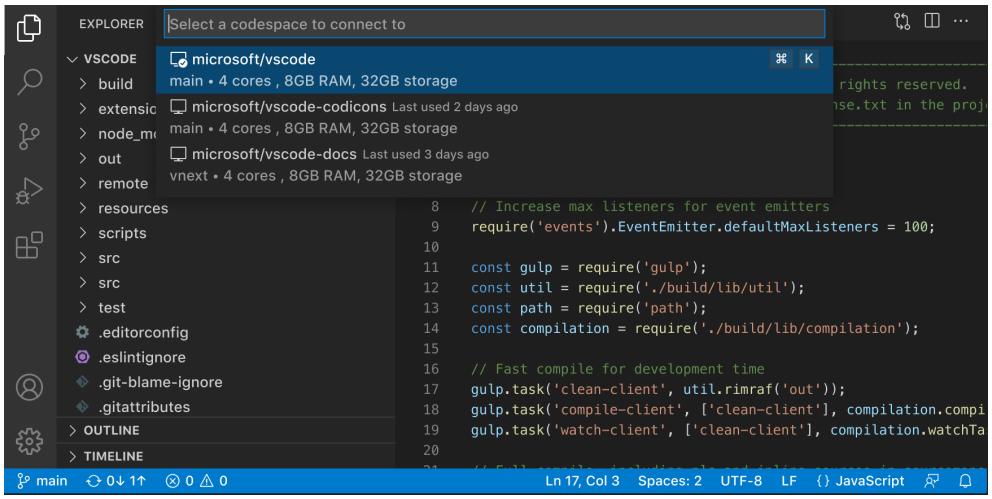


Abbildung 3.5: VS Code Quick Pick Anwendungsbeispiel, angepasst nach [34]

3.4 Übertragbarkeit auf andere Plattformen

Das entworfene Package und Plugin kann nur für C# verwendet werden. Jedoch ist es möglich aufgrund der Erkenntnisse aus den vorangegangenen Abschnitten eine Version für andere Programmiersprachen (und andere IDEs) zu entwickeln.

3.4.1 Übertragbarkeit des Packages

Das Wichtigste ist hierbei, dass die jeweilige Plattform eine Möglichkeit zur Anwendung von AOP anbietet. Unabhängig davon, ob die Implementierung als Paket, Bibliothek oder Framework zur Verfügung steht. Für die meisten der gängigen Programmiersprachen existieren solche Möglichkeiten (vgl. Java [19], Python [21]). Entsprechende Anpassungen sind abhängig von der spezifischen Implementierung von AOP. Das betrifft u. a. das Markieren von Funktionen. Insgesamt kann diese Arbeit, insbesondere Kapitel 3 und z. T. Kapitel 4 als Grundlage für eine solche Umsetzung genutzt werden.

3.4.2 Übertragbarkeit des Plugins

Das Plugin ist für VS Code ausgelegt. Es kann so erweitert werden, dass es die Möglichkeit gibt zwischen verschiedenen Programmiersprachen zu wählen. So könnte es in weiteren Kontexten genutzt werden. Zu beachten ist, es muss zuerst das Package entwickelt werden, damit die Art des Markierens klar ist. Sollte eine andere IDE bevorzugt werden, so kann sich an die vorgestellten Konzepte gehalten werden. Es kann jedoch sein, dass eine IDE manche Elemente aus der VS Code API in anderer Form oder gar nicht bereitstellt.

Kapitel 4

Implementierung

Nachdem im letzten Kapitel bereits die Konzepte von Package und Plugin erläutert worden sind, befasst sich folgendes Kapitel mit den Implementierungsdetails. Hierbei wird der Code nicht Zeile für Zeile durchgesprochen, sondern auf wichtige Implementierungsentscheidungen eingegangen. Daher wird im Nachfolgenden auf die Abbildung von Dateien oder gar der ganzen Codebasis verzichtet und sich auf konzeptuelle Codeausschnitte fokussiert. In den Codeausschnitten wird aus Platzgründen auf Codekommentare verzichtet, auch wenn diese im Original vorhanden sind.

4.1 Singleton

Nach Skeet [35] gibt es mindestens sechs unterschiedliche Möglichkeiten, das Singleton Pattern in C# umzusetzen. Da moderne Anwendungen oft mit Multithreading arbeiten, wurde im Rahmen dieser Arbeit für die Implementierung eine threadsichere Methode gewählt. Hier war vor allem eine Abwägung zwischen Laziness und Performanz entscheidend, daher ist die Wahl nicht auf jedes Projekt anwendbar. Es muss sich eingehend mit den unterschiedlichen Möglichkeiten und eigenen Anforderungen beschäftigt werden, um hier eine treffende Entscheidung zu fällen.

Es gibt eine sehr simple Methode, die lediglich Locks nutzt, um Threadsicherheit zu gewährleisten oder auch eine, die das sogenannte double-check Locking verwendet. Skeet stellt auch eine Methode vor, die mit dem in .NET 4.0 eingeführten Lazy<T> Typen arbeitet. Für das erstellte Package wurde hier die in Abbildung 4.1 aufgezeigte Möglichkeit gewählt. Diese ist besonders einfach gehalten. Aber sollten andere statische Mitglieder in dieser Klasse existieren, dann würde die Klasse auch beim ersten Aufruf dieser Instanziiert werden und somit wäre diese Methode nicht so „Lazy“ wie andere. In diesem Kontext ist dies irrelevant, da hier keine vollständige Laziness erforderlich ist. Die Klasseninitialisierung der Explainer Klasse ist schließlich nicht sonderlich zeitintensiv bzw. komplex. [35]

```
1  public sealed class Singleton
2  {
3      private static readonly Singleton _instance = new Singleton
           ();
4
5      static Singleton() { }
6
7      private Singleton() { }
8
9      public static Singleton Instance
10     {
11         get
12         {
13             return _instance;
14         }
15     }
16 }
```

Listing 4.1: Möglichkeit der Singletonimplementierung nach Skeet [35]

Ein genauerer Blick in den Code in Abbildung 4.1 ermöglicht es, nachzuvollziehen, warum diese Implementierung des Singleton für den gegebenen Kontext geeignet ist. In diesem Zusammenhang ist nochmal hervorzuheben, dass ein Ziel des Singleton die Sicherstellung der Existenz nur einer einzigen Instanz der Klasse ist (vgl. dazu Abschnitt 2.2).

In Zeile 1 verhindert der Zugriffsmodifizierer `sealed` das Vererben der Klasse. Dadurch wird sichergestellt, dass keine abgeleitete zweite Instanz existieren kann, was dem Grundgedanken des Singletons entspricht. Eine zweite Instanz würde nicht nur gegen den Singleton-Entwurf verstoßen, sondern könnte auch zu Konflikten und Irritation bei der Wahl der passenden Speicherklasse für Erklärungen führen. Der private Konstruktor in Zeile 7 verhindert Instanziierung der Klasse von außerhalb der Klasse. Die Klasse selbst stellt durch das statische private `_instance` Feld die einzige Instanz der Klasse bereit (Zeile 3). Dabei sorgt `readonly` dafür, dass auf kein anderes Objekt als die gesetzte Singleton Instanz verwiesen werden kann. Das Objekt selbst bzw. die Felder des Objektes können aber weiterhin verändert werden. Der Zugriff auf das Feld erfolgt ausschließlich kontrolliert über die statische `Instance()` Methode. Zuletzt wendet Skeet hier noch einen Kniff an. Durch den statischen Konstruktor wird die gesamte Klasse nicht als `beforefieldinit` markiert. Das ist wichtig, denn ansonsten könnte die Klasse zu beliebiger Zeit initialisiert werden. Durch die Beliebigkeit könnte die Klasse dann in mehreren Threads gleichzeitig initialisiert werden. Das hätte zur Folge, dass mehrere Instanzen und unterschiedliche Versionen der Speicherklasse existieren würden. [35][36][37]

4.2 Statische Methoden der Explainer Klasse

Die Hauptfunktionen der Klasse sind `static`, damit es nicht notwendig ist, zuerst den Wert der `_instance` Variable abzurufen, um darüber die Methoden aufzurufen. Dies vereinfacht das Arbeiten mit der Explainer Klasse. Dabei wird innerhalb dieser statischen Funktion die definierte `Instance()` Methode genutzt, um mit der einzigen Instanz der Klasse zu arbeiten. Im Normalfall muss ein Entwickler, der mit dem `Explainer` arbeitet, nie tatsächlich auf die `Explainer`-Instanz zugreifen. Es reicht hier die Wrapper für die jeweilige Funktion (die statischen Methoden) zu nutzen. Abgesehen von den Getter/Setter Methoden gehört dazu auch die `GetExplanation()` Methode. Diese liefert die aktuelle Erklärung oder wirft im Fehlerfall eine passende Exception (vgl. Abschnitt 3.2.2).

4.3 Plugin

Die Auswahl der Use Cases wurde wie in Abbildung 4.1 gezeigt mithilfe eines QuickPicks (vgl. Abschnitt 3.3) implementiert. Dabei liest die Erweiterung die Use Case Namen und die Erklärung aus einer CSV-Datei aus.

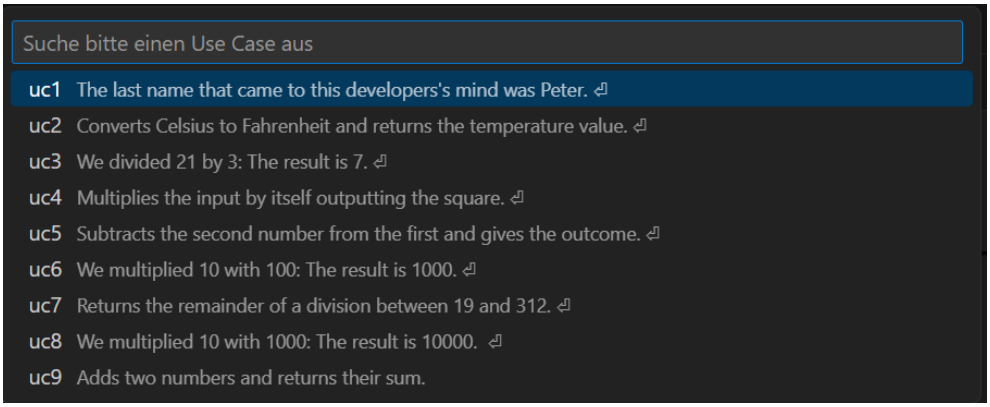


Abbildung 4.1: Auswahl der Use Cases

Fehlermeldungen werden dem Nutzer über Nachrichten der „Error“-Kategorie angezeigt und sind, sofern möglich, mit direkten Lösungsanweisungen versehen.

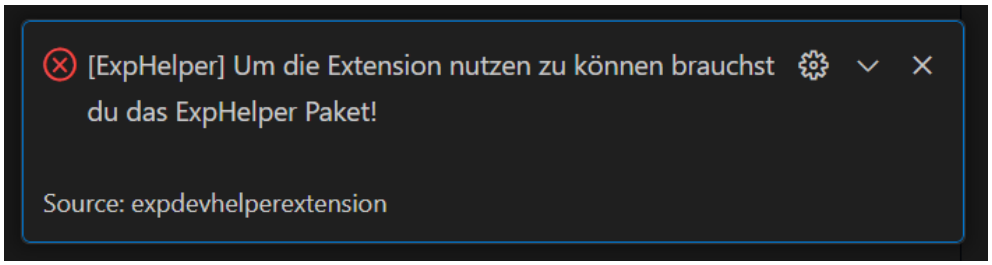


Abbildung 4.2: Fehlermeldung, falls das ExpHelper-Package nicht installiert worden ist.

4.4 Inhalt der erweiterten Use Case-Tabelle

Um sich in diesem Proof-of-Concept auf die Kernpunkte zu fokussieren, wird die Tabelle auf das Wesentliche reduziert. Dabei entfällt die Spalte des Erklärungsbedarfs. Diese kann in Zukunft wieder ergänzt und die Bedarfe an passender Stelle angezeigt werden.

Damit das Plugin und das Package die entsprechenden Daten aus der CSV-Datei auslesen kann, muss folgendes erfüllt sein:

- Die Datei muss sich im Projektordner befinden, die genaue Position innerhalb der Verzeichnisstruktur ist dabei irrelevant.
- Die Datei muss „myfile.csv“ heißen.
- Die Datei muss headless sein.
- Jede Zeile muss folgendes Format haben „Use Case Schritt Id, Erklärung“. Die Id eines Use Case Schrittes besteht dabei aus der Use Case Id und der Schrittnummer. Damit steht die Use Case Schritt Id 4.2, für Use Case 4 im Schritt 2.

Zusätzlich sollte darauf geachtet werden, dass die Erklärungen keine Kommata enthalten dürfen, sonst wird dies als neue Spalte und nicht als Teil der Erklärung angesehen. Um dies zu verhindern, könnte zukünftig ein anderes Zeichen als Komma zur Abtrennung verwendet oder Beginn und Ende der Erklärung separat markiert werden.

Kapitel 5

Nutzerstudie

Nach der vollständigen Konzeptionierung und Implementierung des Packages und Plugins wird nun geprüft, ob beides nutzbar ist. In den folgenden Abschnitten wird der Prozess dieser Prüfung beschrieben. Im weiteren Verlauf wird stellvertretend für Plugin und Package stellenweise der Begriff ***ExpHelper*** verwendet.

5.1 Forschungsfragen

Grundsätzlich gilt es hier herauszuarbeiten, ob die Verwendung von Plugin und Package Entwicklern einen signifikanten Vorteil dabei verschafft, Erklärungen in verschiedene Abschnitte einer Anwendung zu integrieren. Ein solcher Vorteil wird in einer schnelleren Umsetzung und einer geringeren mentalen Belastung des Entwicklers bei Integrierung der Erklärungen angenommen. Daraus ergeben sich folgende Fragen:

RQ1: Ermöglicht der *ExpHelper* sowohl die Speicherung als auch das Abrufen von Erklärungen bei Bedarf?

RQ2: Beschleunigt der *ExpHelper* den Prozess der Integrierung von Erklärungen?

RQ3: Reduziert die Anwendung des *ExpHelpers* die kognitive Belastung von Entwicklern während der Integrierung von Erklärungen?

5.2 Planung der Studie

Es wird zunächst eine entsprechende Testanwendung benötigt, um den Unterschied zwischen der Integrierung von Erklärungen mit und ohne *ExpHelper* festhalten zu können. In diese Anwendung sollen Erklärungen integriert werden. Aufgrund der Spezifikationen des *ExpHelpers* muss die

Testanwendung auf C# basieren und in VS Code erweiterbar sein. Der Aufbau dieser Testanwendung wird in Abschnitt 5.4 detailliert beschrieben.

Die Studie soll mit sieben Teilnehmern durchgeführt werden. Diese sind nicht zwingend Personen, die täglich Erklärungen implementieren (s. Abschnitt 5.3). Daher ist die subjektive Einschätzung des *ExpHelpers* hinsichtlich der Tauglichkeit zweitrangig. Gemessen wird hier die Geschwindigkeit des Abschlusses der gestellten Aufgabe und die empfundene mentale Belastung der Teilnehmer, anhand des NASA Task Load Index (TLX). Dieser misst die mentale Arbeitsbelastung auf Basis von sechs Kategorien. Dazu gehören mentale Anforderung, körperliche Anforderung, zeitliche Anforderung, Leistung, Anstrengung und Frustration. Die Teilnehmer füllen dazu einen Fragebogen aus, der diese verschiedenen Aspekte ihrer mentalen Belastung abfragt. Daraus wird dann ein Gesamtwert zur mentalen Belastung errechnet, der TLX-Score (vgl. Abschnitt 2.5). [32]

5.3 Teilnehmer

Von den insgesamt sieben Teilnehmern dieser Studie sind sechs männlich und eine weiblich. Die Teilnehmer sind zwischen 21 und 25 Jahren alt. Zwei der sieben Teilnehmer befinden sich aktuell im Bachelorstudium Informatik, einer im Bachelorstudium Wirtschaftswissenschaften. Drei der sieben Teilnehmer haben das Bachelorstudium Informatik abgeschlossen und eine Person befindet sich im dritten Ausbildungsjahr der Fachinformatik für Systemintegration (vgl. Anhang A.1).

Um an dieser Studie teilnehmen zu können, ist es notwendig, Erfahrungen im Programmieren zu besitzen. Es ist nicht notwendig bereits in C# programmiert zu haben, aber es sollen mindestens Erfahrungen in einer vergleichbaren objektorientierten Programmiersprache vorliegen. Dies ist erforderlich, damit die Teilnehmer grundlegend mit den Konzepten der Objektorientierung vertraut sind und so die gestellte Aufgabe lösen können. Da es sich um einen Proof-of-Concept handelt, ist die gestellte Aufgabe eher leicht und somit ist es nicht erforderlich Erfahrungen im Bereich Erklärbarkeit zu haben. Es werden keine Anforderungen an Alter, Geschlecht oder Bildungsgrad der Teilnehmer gestellt, außer, dass die Teilnehmer volljährig sein müssen.

5.4 Beispielanwendung für die Studie

Innerhalb dieser Anwendung können unterschiedliche Algorithmen ausgelöst werden. Außerdem kann zu beliebiger Zeit eine Erklärung angefragt werden, wodurch eine Erklärung zum zuletzt ausgeführten Algorithmus angezeigt wird. Damit die Teilnehmer das Plugin in einer möglichst realitätsnahen

Umgebung nutzen, wird eine Desktopanwendung mit aktuellen Techniken aus der C# und .NET Familie, in dem Fall dem WinAppSDK, eingesetzt.

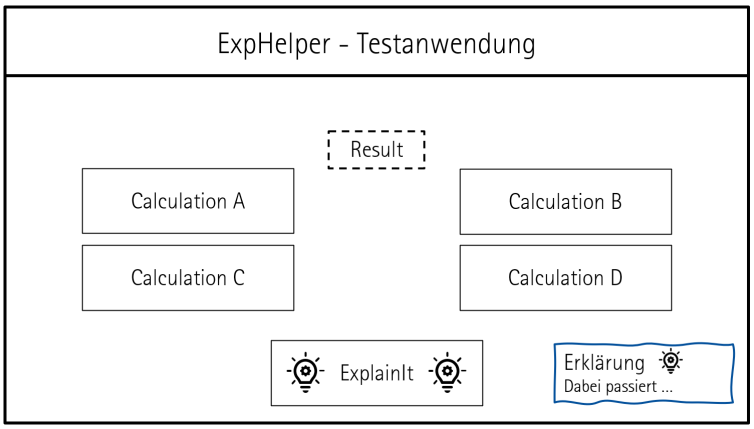


Abbildung 5.1: Mockup der Beispielanwendung

In der Abbildung 5.1 ist die grobe Designidee des erstellten Programms zu erkennen. Es soll vier Buttons beinhalten, welche bestimmte Berechnungen auslösen. Das ausgelöste Event zeigt dann an einer bestimmten Stelle das Ergebnis der Rechnung an. Der Nutzer weiß in diesen Fällen aber nicht, was gerechnet wurde. Sollte er sich das fragen, so kann er auf den fünften Button „Explain it“ klicken. Dies sorgt dafür, dass sich in der unteren rechten Ecke ein *TeachingTip* öffnet. In diesem soll dann die Erklärung für die letzte ausgeführte Aktion stehen, so kann geprüft werden, ob in dem Programm die richtige Erklärung angezeigt wird. Die Abbildung 5.2 zeigt das endgültige Design der Anwendung und bietet einen Einblick, wie die Benutzeroberfläche aussieht und wie die Interaktionen ablaufen werden.

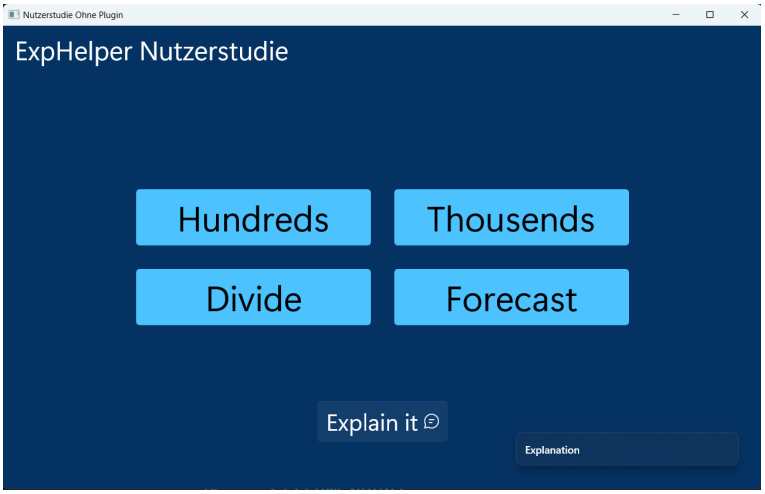


Abbildung 5.2: Abbildung des finalen Designs der Beispielanwendung

5.5 Ablauf der Studie

Die Abbildung 5.3 zeigt den Ablauf der Nutzerstudie. Vor Beginn der Teilnahme wird jeder Teilnehmer umfassend über die Studie und die erhobenen Daten informiert. Anschließend erfolgt eine schriftliche Bestätigung der Kenntnisnahme durch Unterschrift.

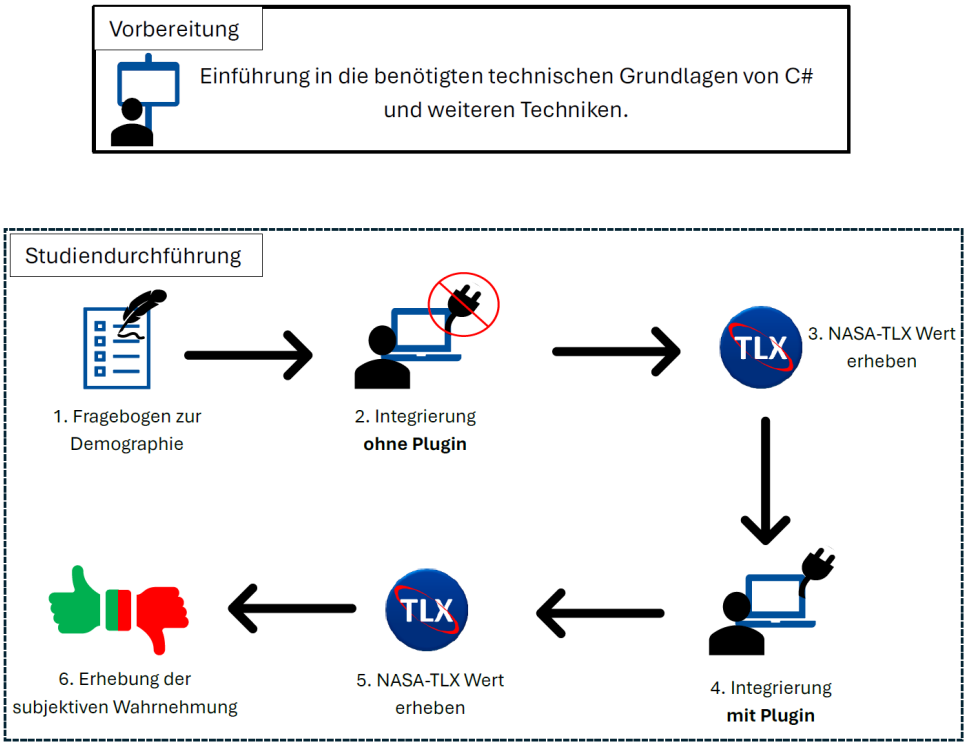


Abbildung 5.3: Ablauf der Nutzerstudie als Diagramm

Danach beginnt die Einführung. Hier werden zur Auffrischung die Umsetzung grundlegender Konzepte und Schlüsselworte in C# vorgestellt. So wird u. a. über `using`, `namespace` oder `for` gesprochen. Ist die Vorstellung abgeschlossen, so wird mit den in der Abbildung 5.3 dargestellten Schritten der Studiendurchführung begonnen.

1. **Fragebogen zur Demografie**: Hierbei beantworten die Teilnehmer den mit Microsoft Forms erstellten Fragebogen zu demografischen Angaben wie Alter, der höchsten beruflichen Qualifikation und generellen Kenntnissen zum Programmieren. Hierbei soll die eigene Programmiererfahrung eingeschätzt und generell bewertet werden (s. Anhang A.1). Die durch die nachfolgenden Aufgaben erhaltenen Werte werden mit Teilen dieser Angaben anonymisiert in Bezug gesetzt.

2. **Integrierung ohne Plugin**: Hier sollen Erklärungen ohne das Plugin

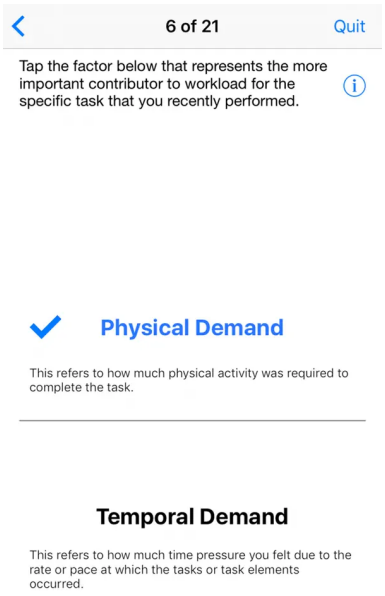


Abbildung 5.4: Paarweises Vergleichen

Abbildung 5.5: Ausgeprägtheit des Faktors bewerten

in die Anwendung integriert werden. Die Studienteilnehmer erhalten ein Aufgabenblatt (s. Anhang A.2). Sie haben so viel Zeit, wie sie zum Lesen der Aufgabenstellung benötigen. Sobald sie mit der Bearbeitung der Aufgabe beginnen, haben sie zehn Minuten Zeit, die Aufgaben zu bearbeiten. Innerhalb von zehn Minuten müssen die Teilnehmer die in Abschnitt 5.4 beschriebene Anwendung um Erklärungen erweitern. Dazu verschaffen sie sich zunächst einen Überblick über zwei Codedateien und entwickeln dann ein rudimentäres System, das die Speicherung und den Abruf von Erklärungen ermöglicht.

3. **NASA-TLX Wert erheben**: Nach Abschluss der vorherigen Aufgabe müssen die Studienteilnehmer auf einem mobilen Endgerät in der offizielle TLX App der NASA einen Fragebogen ausfüllen. Die Teilnehmer wählen zunächst zwischen einigen Faktoren im paarweisen Vergleich den passenden Faktor aus (s. Abbildung 5.4) und müssen danach die ausgewählten Aspekte in Bezug auf ihre Ausgeprägtheit bewerten (niedrig bis hoch) (s. Abbildung 5.5).

   Die Teilnehmer werden aufgefordert, die Aufgabe an sich zu bewerten, aber nicht die Komplexität des genutzten Frameworks / der Programmiersprache.

4. **Integrierung mit Plugin**: Hier sollen Erklärungen mithilfe des

Plugin in die Anwendung integriert werden. Die Studienteilnehmer erhalten ein Aufgabenblatt (s. Anhang A.2). Sie haben so viel Zeit, wie sie zum Lesen der Aufgabenstellung benötigen. Sobald sie mit der Bearbeitung der Aufgabe beginnen, haben sie zehn Minuten Zeit, die Aufgaben zu erfüllen. Innerhalb von zehn Minuten müssen die Teilnehmer die in Abschnitt 5.4 beschriebene Anwendung um Erklärungen mithilfe des *ExpHelpers* erweitern. Dafür müssen sie die Anleitung des Plugins lesen, sich einen Überblick über zwei Codedateien verschaffen und dann mithilfe des Plugins Erklärungen integrieren.

5. **NASA-TLX ausfüllen**: Nach Abschluss der vorherigen Aufgabe müssen die Studienteilnehmer wieder auf einem mobilen Endgerät in der offizielle TLX App der NASA einen Fragebogen ausfüllen. Die Prozedur bleibt die Gleiche wie in Schritt 3 beschrieben.

6. **Erhebung der subjektiven Wahrnehmung**: Den Teilnehmenden wird eine Microsoft Forms Umfrage vorgelegt. Diese umfasst eine grundsätzliche Bewertung bzw. Einschätzung der Nützlichkeit und Tauglichkeit mithilfe von sechs Fragen. Diese müssen auf einer Likert Scala in fünf Stufen von „stimme nicht zu“ bis „stimme zu“ bewertet werden (s. Anhang A.3). Es gibt die Möglichkeit, sich bei den jeweiligen Fragen zu enthalten.

5.5.1 Begründung der Aktivitätsreihenfolge

Wie im vorher beschriebenen Ablauf zu erkennen, wird in der Nutzerstudie erst ohne und anschließend mit dem Plugin gearbeitet. Dies ermöglicht den Teilnehmern, ein Verständnis dafür zu entwickeln, wie die Aufgaben ohne Unterstützung des Plugins bewältigt werden müssen. Auf diese Weise können sie besser beurteilen, welchen Mehrwert das Plugin bietet und wie es die Aufgaben erleichtert. In Abschnitt 6.3 wird zu dem möglichen Einfluss auf das Ergebnis genauer Stellung genommen.

5.5.2 Pilotstudie

Um sicherzustellen, dass der Ablauf (siehe Abbildung 5.3) vollständig durchführbar ist und die gestellte Aufgabe die Teilnehmer gemäß Anforderungsprofil (Abschnitt 5.3) nicht überfordert, wurde eine Pilotstudie mit einem Teilnehmer durchgeführt. Diese Vorstudie diente dazu, potenzielle Probleme frühzeitig zu identifizieren und entsprechende Anpassungen vorzunehmen. Diese Pilotstudie ergab, dass die Schwierigkeit und der Ablauf passend gewählt wurden. Zur Verbesserung des Verständnisses wurden kleine Veränderung an Formulierungen der Aufgabenstellungen vorgenommen.

5.6 Datenerhebung

Innerhalb der Studie wird die Zeit jedes Teilnehmers, die zur Lösung der Aufgaben benötigt wird, gemessen (Schritt 2, Schritt 4). Zusätzlich wird nach der Erfüllung jeder Aufgabe über den NASA-TLX Fragebogen der TLX-Score erhoben (Schritt 3, Schritt 5). Um die Lösungen eingehend auf Korrektheit und thread-safety prüfen zu können, werden diese gespeichert.

Zusätzlich wird zu Beginn ein Fragebogen in Microsoft Forms über demografische Daten und einer Selbsteinschätzung der eigenen Programmierfähigkeit ausgefüllt. Diese wird vom Teilnehmer auf einer Ordinalskala zwischen (keine) und (sehr gut) eingeordnet (vgl. Anhang A.1). Am Ende jedes Durchlaufs wird ein Fragebogen ausgefüllt, um eine subjektive Wahrnehmung des *ExpHelper* zu erheben. Auch hier ordnet der Teilnehmer die Wahrnehmung auf einer fünfstufigen Ordnialskala ein, von (Ich stimme nicht zu) bis (Ich stimme zu).

5.7 Datenanalyse

Die erfassten Zeiten und TLX-Scores wurden mittels deskriptiver Statistik analysiert und visuell aufbereitet, um generelle Tendenzen der Unterschiede zwischen den beiden Aufgaben zu identifizieren. Die Unterschiede der TLX-Scores werden daraufhin mithilfe des t-tests auf statistisch Signifikanz geprüft.

Die erfüllte Aufgabe wird im Nachgang noch auf Korrektheit und Thread-Sicherheit überprüft. Dabei wird Korrektheit hier folgendermaßen ausgelegt: Solange die richtigen Erklärungen im Tooltip angezeigt werden, gilt die Aufgabe als korrekt. Innerhalb der zweiten Aufgabe ist thread-safety durch den *ExpHelper* bereits erfüllt.

Kapitel 6

Auswertung und Analyse

Nachfolgend werden die Ergebnisse der Nutzerstudie vorgestellt, analysiert und interpretiert. Dabei wird auch auf mögliche Einschränkungen für die Validität eingegangen. In der Ergebnissicherung und Auswertung werden Zeitangaben z. T. verkürzt durch eine Angabe des Formats **Minute:Sekunde** getätigt, also z. B. 20 Minuten und 40 Sekunden als 20:40. Andere Auslegungen dieses Formats werden explizit gekennzeichnet.

6.1 Ergebnisse

Die Zeiten, welche die Teilnehmer zum Absolvieren der gestellten Aufgaben (A1 - ohne Plugin und A2 - mit Plugin) benötigt haben, sind in der Tabelle 6.1 dargestellt. Zusätzlich können aus dieser Tabelle auch die TLX-Scores der Teilnehmer abgelesen werden. Diese Werte wurden nach Abschluss der jeweiligen Aufgabe erhoben.

Ohne Plugin (A1) beträgt die minimale Zeit für den Abschluss 6:40 und die maximale Zeit 10:00, mit Plugin (A2) liegt die minimale Zeit bei 01:40 und die maximale Zeit bei 05:10. Die TLX-Scores für die erste Aufgabe reichen von 48,33 bis hin zu 81,33 und für die zweite Aufgabe von 10,33 bis 61,67.
Die Tabelle 6.2 zeigt die arithmetischen Mittel und Mediane der Zeiten und TLX-Scores. Dabei liegt das arithmetische Mittel für das Abschließen der ersten Aufgabe bei 9:05 und bei der zweiten Aufgabe bei 4:07. Für die Aufgabe ohne Plugin liegt der TLX-Score-Mittelwert bei 59,67 und für die Aufgabe mit Plugin bei 33,81.

Der Median für das Abschließen der ersten Aufgabe liegt bei 9:51 und bei der zweiten Aufgabe bei 4:25. Für die Aufgabe ohne Plugin liegt der Median des TLX-Scores bei 55,67 und für die mit Plugin bei 32,67.

Die Verteilung bzw. Streuung der in der Studie aufgenommen Werte werden als Boxplots in Abbildung 6.1 und 6.2 für die Zeiten und TLX-Scores visualisiert.

Teilnehmer	Zeit A1 (in Minuten)	TLX-Scores A1	Zeit A2 (in Minuten)	TLX-Scores A2
1	07:49	50,00	04:25	21
2	09:59	49,33	04:51	32,67
3	06:40	48,33	01:40	34,33
4	09:21	55,67	05:10	61,67
5	10:00	81,33	04:31	15,33
6	09:57	74,00	03:58	10,33
7	09:51	59,00	04:11	61,33

Tabelle 6.1: Wertetabelle der aufgenommenen Zeiten und TLX-Scores je Teilnehmer

Art des Durschnitts	Zeit A1 (in Minuten)	TLX A1	Zeit A2 (in Minuten)	TLX A2
Arithmetisches Mittel	09:05	59,67	04:07	33,81
Median	09:51	55,67	04:25	32,67

Tabelle 6.2: Wertetabelle der Durchschnitte von Zeiten und TLX-Scores

Dabei zeigt die Abbildung 6.1 die Verteilung für aufgenommene Zeitwerte. Bei Betrachtung des ersten Boxplots fällt auf, dass sich die gesamten Werte zwischen dem untersten Punkt der Antenne (6:40) und dem obersten Punkt der Antenne (10:00) befinden. Die Spannweite (Streubreite) des Boxplots beträgt insgesamt 4:20. Die Box umschließt einen Bereich der Größe 1:23 zwischen 8:35 (1. Quartil) und 9:58 (3. Quartil). Insgesamt ist die Verteilung, die durch diesen Boxplot visualisiert wird, asymmetrisch und stark linksschief. Der zweite Boxplot (Mit Plugin) hat zwischen dem untersten (3:58) und obersten Punkt der Antenne (5:10) eine Spannweite (Streubreite) von 1:12. Der Wert 1:40 ist hier ein Ausreißer. Die Box umschließt die Werte zwischen 4:04 (1. Quartil) und 4:41 (3. Quartil) und damit einen Bereich von 0:37. Die hier visualisierte Verteilung ist asymmetrisch und sehr leicht rechtsschief.

Weder die Bereiche der Boxen, noch die Streuungsbereiche beider Boxplots überschneiden sich. Die Spannweite des ersten Boxplots ist ungefähr viermal so groß, wie die des Zweiten und der Boxbereich des ersten ist fast dreifach so groß, wie der des Zweiten. Insgesamt liegen die Werte der zweiten Verteilung deutlich dichter beieinander, als die der ersten Verteilung.

Die Abbildung 6.2 zeigt die Verteilung für die aufgenommenen TLX-Scores. Bei Betrachtung des ersten Boxplots fällt auf, dass sich die gesamten Werte zwischen dem untersten Punkt der Antenne (48,33) und dem obersten Punkt Antenne (81,33) befinden. Die Spannweite (Streubreite) des Boxplots beträgt insgesamt 33. Die Box umschließt einen Bereich der Größe 16,83 zwischen 49,67 (1. Quartil) und 66,5 (3. Quartil). Insgesamt ist die Verteilung, die

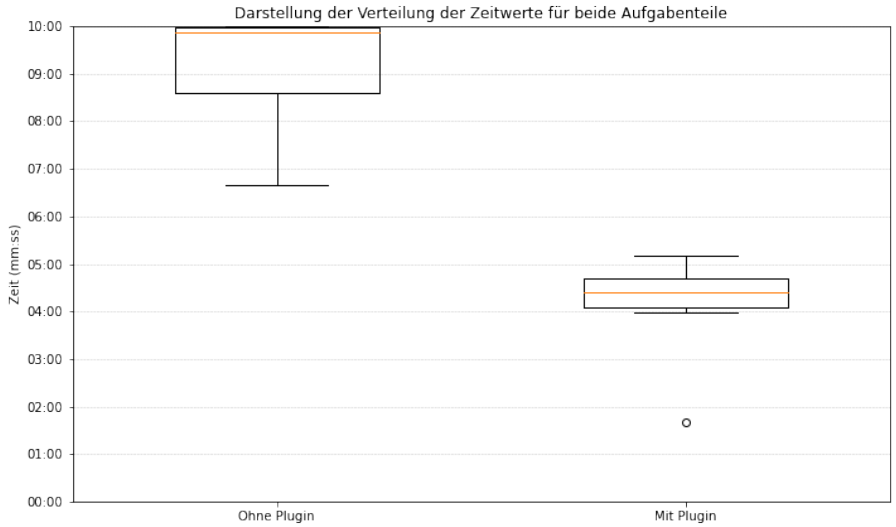


Abbildung 6.1: Boxplot über die aufgenommenen Zeitwerte, die in Tabelle 6.1 dargestellt sind.

durch diesen Boxplot visualisiert wird, asymmetrisch und stark rechtschief. Der zweite Boxplot (Mit Plugin) hat zwischen dem untersten (10,33) und obersten Punkt der Antenne (61,33) eine Spannweite (Streubreite) von 51. Die Box umschließt die Werte zwischen 18,17 (1. Quartil) und 47,83 (3. Quartil) und damit einen Bereich von 29,66. Die hier visualisierte Verteilung ist asymmetrisch und sehr leicht rechtsschief.

Die Bereiche der Boxen beider Boxplots überschneiden sich nicht, wobei die zweite Box vollständig unter dem gesamten ersten Boxplot liegt. Allerdings überschneidet sich die erste Box mit der Antenne der zweiten Box. Der Median des ersten Boxplots ist dennoch mehr als anderthalbmal so groß wie der des zweiten. Die Spannweite des zweiten Boxplots ist ungefähr doppelt so groß wie die des ersten, und auch der Boxbereich des zweiten Boxplots ist mehr als doppelt so groß. Insgesamt liegen die Werte der ersten Verteilung enger beieinander als die der zweiten Verteilung.

Zusätzlich zu den Zeiten und den TLX Werten, wurde am Ende jeder Aufgabe bewertet, ob diese Aufgabe korrekt gelöst wurde und ob die Implementierung thread-safe ist. Die Daten dazu sind in der Abbildung 6.3 und der Abbildung 6.4 dargestellt.

Von allen sieben Studienteilnehmern konnte nur in einem Fall keine korrekte Lösung für die erste Aufgabe (ohne Plugin) implementiert werden. Alle sieben Studienteilnehmer konnten eine korrekte Lösung für die zweite Aufgabe (mit Plugin) implementieren. Keiner der sieben Teilnehmer konnte ohne Plugin eine Variante implementieren, die thread-safe ist. Durch das Plugin ist eine thread-safe Implementierung automatisch gegeben, solange die Implementierung korrekt vorgenommen wurde.

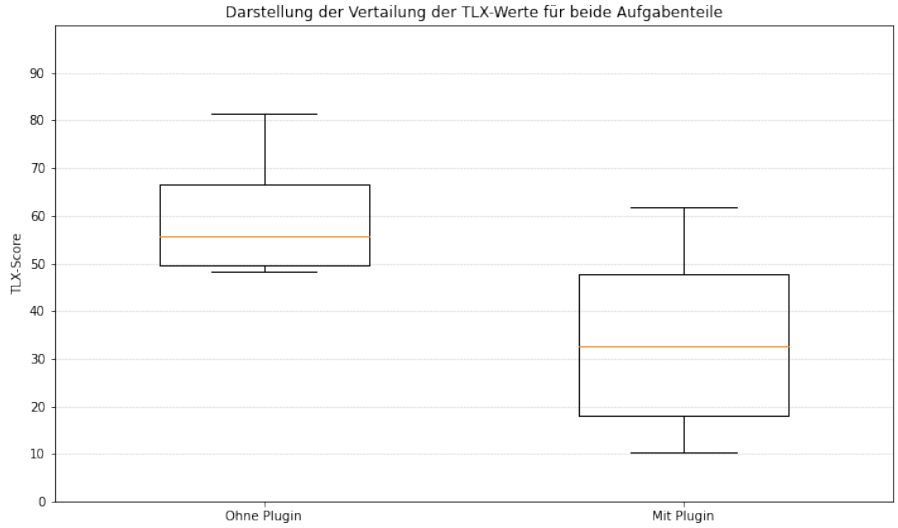


Abbildung 6.2: Boxplot über die aufgenommenen TLX-Werte, die in Tabelle 6.1 dargestellt sind.

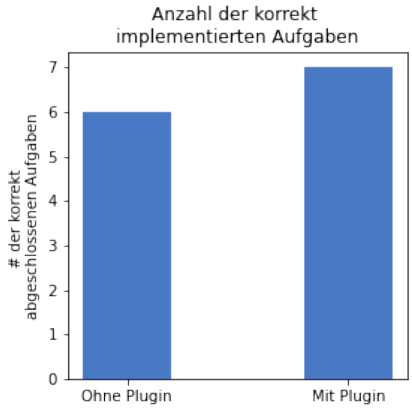


Abbildung 6.3: Vergleich der Anzahl der korrekt implementierten Aufgaben

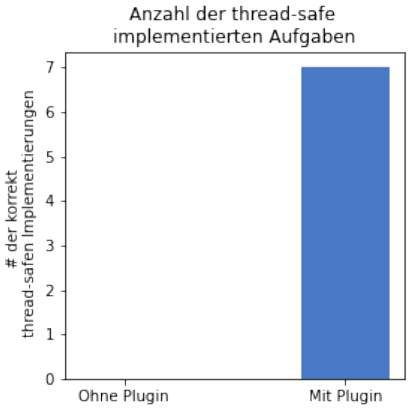


Abbildung 6.4: Vergleich der Anzahl der thread-safe implementierten Aufgaben

6.1.1 Signifikanztest

Anhand der Diagramme wird deutlich, dass sich die aufgenommenen Werte der Zeit und TLX-Scores zwischen den beiden gestellten Aufgaben der Nutzerstudie unterscheiden. Fraglich ist, ob dieser Unterschied signifikant ist. Der signifikante Unterschied wird mithilfe eines auf kleine Sample-Mengen ausgelegten statistischen Tests nachgewiesen. In diesem Fall dem t-test. [38][39]
Die Wertepaare, die aus beiden Mengen gebildet werden, stammen immer von der gleichen Person, nur die Aufgabe unterscheidet sich leicht. Daher muss ein abhängiger (dependent) t-test angewandt werden. Zusätzlich wird nur nach einer signifikanten Steigerung gefragt (directional hypothesis) und es sollen zwei Mengen miteinander verglichen werden. Insgesamt wird daher ein dependent paired sample one-tailed t-test durchgeführt. Um diesen anwenden zu können, wurden zuvor die Voraussetzungen dafür geprüft (s. Anhang B). Das Signifikanzniveau α wurde auf 5% festgelegt.

Dieser t-test wird mithilfe der SciPy Bibliothek und der passenden Funktion `t-test` durchgeführt. [40]

```
t_stat, p_value = stats.ttest_rel(scores_with_plugin,
    scores_without_plugin, alternative='less')
```

Für die TLX-Scores ergibt sich ein p-Wert von 0,0285, also 2,85% und 2,85 ist kleiner als α (5). Damit ergibt sich ein signifikant geringerer Median der TLX-Werte mit Plugin, mit einer 97,15% Chance, dass die Ergebnisse (und damit auch dieser Schluss) nicht durch Zufall zu Stande gekommen sind.
Die Berechnung für die Zeitwerte ergibt hier ein p-Wert von 0,000003, also 0,0003% und 0,0003 ist kleiner als das Signifikanzniveau 5. Es liegt ein signifikant geringerer Median der Zeitwerte mit Plugin vor, mit einer 99,9997% Chance, dass die Ergebnisse (und damit auch dieser Schluss) nicht durch Zufall zu Stande gekommen sind.

6.2 Beantwortung der Forschungsfragen

Die Fragen RQ2 und RQ3 können hier nur bedingt im Rahmen der Stichprobe beantwortet werden. Der geringe Umfang der Studie sorgt dafür, dass für diese Fragen nur eine Tendenz abgeleitet werden kann. Mithilfe der genannten Ergebnisse können die Forschungsfragen folgendermaßen beantwortet werden:

6.2.1 RQ1 - Ermöglicht der *ExpHelper* sowohl die Speicherung als auch das Abrufen von Erklärungen bei Bedarf?

Der *ExpHelper* ermöglicht sowohl Speicherung als auch Abrufen von Erklärungen. Eine Evidenz dafür zeigt das Diagramm in Abbildung 6.3.

Demnach konnten alle Teilnehmer die zweite Aufgabe vollständig und korrekt abschließen. Es konnten also alle Erklärungen für die Funktionen richtig gespeichert und auf Anfrage abgerufen bzw. angezeigt werden.

Zusätzlich kann die Antwort durch die Erkenntnisse aus Abbildung 6.4 erweitert werden. Die Art der Implementierung sorgt dafür, dass sie auch in einer Multithreading Umgebung angewendet werden kann.

6.2.2 RQ2 - Beschleunigt der *ExpHelper* den Prozess der Integrierung von Erklärungen?

Die Tendenz ist hier sehr eindeutig, wie in dem Diagramm in Abbildung 6.1 und Tabelle 6.2 zu erkennen ist. Dabei sind die Mittelwerte der ersten Aufgabe (ohne Plugin) mehr als doppelt so hoch wie in der zweiten Aufgabe (mit Plugin). Im Vergleich 09:05 zu 04:07 im arithmetischen Mittel und 09:51 zu 04:25 im Median. Im Rahmen der durchgeführten Studie kann nach Abschnitt 6.1.1 sogar ein signifikanter Unterschied nachgewiesen werden.

6.2.3 RQ3 - Reduziert die Anwendung des *ExpHelpers* die kognitive Belastung von Entwicklern während der Integrierung von Erklärungen?

Anhand der aufgenommenen TLX-Scores konnte eine positive Tendenz für die Reduktion gezeigt werden, wie in dem Diagramm in Abbildung 6.2 und Tabelle 6.2 zu erkennen ist. Dabei sind die Mittelwerte der ersten Aufgabe (ohne Plugin) weniger als doppelt so hoch wie in der zweiten Aufgabe (mit Plugin). Im Vergleich 59,67 zu 33,81 beim arithmetischen Mittel und 55,67 zu 32,67 im Median. Während der niedrigere Score im Bereich der niedrigen Belastung liegt, fällt der höhere Score deutlich in den Bereich der mittleren an der Grenze zur hohen Belastung (vgl. Abschnitt 2.5). Im Rahmen der durchgeführten Studie kann nach Abschnitt 6.1.1 sogar ein signifikanter Unterschied nachgewiesen werden. Ein Studienteilnehmer, der das Plugin als größere Last empfunden hat, gab folgendes sinngemäß als Grund an:

- Dem Teilnehmer war unklar, wie das Plugin und das Package tatsächlich funktionieren. Aber er wollte den Code bzw. die Prinzipien dahinter verstehen. Hierbei hat der Teilnehmer angegeben, es lieber selbst programmieren zu wollen, anstatt ein Paket zu nutzen. Dadurch könnte er den Code dahinter genauer verstehen.

Aus der Aussage geht hervor, dass der Teilnehmer grundsätzlich dem Nutzen von Paketen und nicht selbst programmiertem kritisch gegenübersteht oder zumindest so weit er diese nicht vollständig überblicken kann. Solche Erweiterungen scheinen also nicht grundsätzlich für alle Entwickler geeignet und diese Feststellung scheint somit nicht explizit abhängig vom hier beschriebenem Plugin zu sein.

6.3 Threats to Validity

Die Punkte zur Einschränkung der Validität werden im nachfolgenden Abschnitt nach Wohlin et al. [41] kategorisiert und beschrieben.

Construct Validty bezieht sich darauf, wie gut eine Studie das zugrundeliegende theoretische Konstrukt misst und die Ergebnisse auf dieses Konzept übertragen werden können. Diese Einschränkungen können sowohl durch das Design des Experiments als auch durch soziale Faktoren entstehen. Dazu gehören in dieser Studie:

- **Teilnehmerdemografie**: Die Auswahl der Teilnehmer könnte Einfluss auf das Ergebnis genommen haben. Alle befinden sich in einer ähnlichen Altersgruppe und sind hauptsächlich männlich. Zusätzlich haben die Teilnehmer alle mehr oder weniger den gleichen beruflichen Hintergrund (Studium der Informatik an derselben Einrichtung). Eine andere Verteilung in Alter, Geschlecht und Ausbildung bzw. Fähigkeiten könnte das Ergebnis der Studie beeinflussen.

- **Repräsentation der Zielgruppe**: Zusätzlich sind die Teilnehmer keine Personen, die sich in ihrem Alltag mit der Implementierung von Erklärbarkeit beschäftigen. Dadurch könnte es sein, dass die Ergebnisse mit Personen, die sich tatsächlich mit der Implementierung von Erklärbarkeit beschäftigen, anders ausfallen. Damit sind sie möglicherweise nicht repräsentativ für die Zielgruppe und auch das könnte Einfluss auf das Ergebnis und somit die Validität genommen haben. Um einen tatsächlichen Nachweis über einen Unterschied erbringen zu können, müsste eine weitere Studie mit deutlich mehr Teilnehmern in Bezug zur passenden Zielgruppe durchgeführt werden. Eine andere Auswahl an Teilnehmern war im Rahmen dieser Studie aus Gründen der Zugänglichkeit, der Kosten und zeitlicher Beschränkungen nicht realisierbar. Eine breitere Rekrutierung hätte den zeitlichen und logistischen Rahmen dieser Arbeit gesprengt.

- **Simulation des Arbeitsumfelds**: Im Normalfall würde dieses Plugin von Entwicklern mehrfach über einen längeren Zeitraum genutzt werden. In unterschiedlichen Szenarien, für unterschiedliche Programme. In dem Fall müsste hier an der Stelle eine Langzeitstudie durchgeführt werden, um Nützlichkeit exakt feststellen zu können, was aufgrund des zeitlichen Rahmens dieser Arbeit nicht möglich war.

Punkte in der Kategorie *Internal Validity* stehen in Zusammenhang mit dem Ausmaß, indem Ergebnisse der Studie auf kausale Einflüsse unabhängiger, nicht gemessener Faktoren zurückzuführen sind. Dazu zählen:

- **Reihenfolge der Aufgaben**: Es könnte sein, dass sich die Teilnehmer innerhalb der ersten Aufgabe an die Problemstellung an sich gewöhnt

haben, sodass sie sich leichter in der zweiten Aufgabe zurechtfanden. Dabei könnte sich der Zeitunterschied ändern, wenn die Reihenfolge getauscht wird. Ebenfalls könnte es sein, dass die Personen die mentale Belastung anders wahrnehmen. Zwar ist das Plugin eine Erleichterung, wodurch sich der Wert verbessern könnte. Zusätzlich mussten die Teilnehmer kurz davor eine mentale und zeitlich anstrengende Aufgabe lösen, was zu genereller Erschöpfung führen kann und sich somit negativ auf die Bewertung ausgewirkt haben könnte. Zur Entlastung der Teilnehmer hätten feste Pausen in den Durchläufen integriert werden können. Die Teilnehmer hatten die Möglichkeit, freiwillig Pausen während der Studie einzulegen, dies sollte die zeitliche Belastung senken. Diese Pausen waren also in dieser Studie nicht verpflichtend.

- **Studienaufbau / Umgebung**: Das aufgebaute Setup könnte ebenfalls die Ergebnisse beeinflusst haben. Dabei könnten die Ergebnisse in einer eigenen gewohnten Hard- und Softwareumgebung anders ausfallen. Insbesondere könnte es sein, dass die Teilnehmer sich innerhalb der ersten Aufgabe an die Umgebung gewöhnt haben, dadurch sicherer in Interaktion mit der Umgebung wurden und als Folge im Schnitt bessere Ergebnisse geliefert haben.

Die Kategorie der *Conclusion Validity* umfasst Faktoren, welche die Validität der Ergebnisse im Hinblick auf den Grad ihrer Belastbarkeit einschränken. Folgende Faktoren dieser Kategorie wurden hier identifiziert:

- **Repräsentativität**: Mit sieben Teilnehmern, war die Teilnehmeranzahl zu gering, um wirklich repräsentative Schlussfolgerungen ziehen zu können.

In der letzten Kategorie *External Validity* werden Punkte aufgelistet, die mit der Übertragbarkeit der Ergebnisse auf andere Situationen, Kontexte und Populationen zusammenhängen:

- **Kontextwechsel**: Die Möglichkeit, die Ergebnisse auf andere Kontexte und Gruppen zu übertragen, könnte dadurch begrenzt sein, dass die ausgewählte Gruppe nicht als repräsentativ betrachtet werden kann. Zusätzlich berücksichtigt die gestellte Aufgabe an sich keine vielfältigen Kontexte. Dies bedeutet, dass auch die externe Validität der Untersuchung potenziell eingeschränkt ist.

- **Zweckgebunden**: Zweck der Studie war es, einen Proof-of-Concept für das Plugin zu erbringen. Das bedeutet, es sollte hier hauptsächlich gezeigt werden, dass der *ExpHelper* nutzbar ist. Daher wurde keine umfassende Studie oder Untersuchung dazu durchgeführt, einen realen Kontext zu typischen Anwendungsfällen in der Realität zu erarbeiten.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Die Erklärungen anhand einer erweiterten Use Case-Tabelle manuell in eine entworfene Applikation zu integrieren, ist eine zeitintensive und fehleranfällige Aufgabe. In dieser Arbeit wurde sich damit beschäftigt, Entwickler dabei zu unterstützen, auf Basis von erweiterten Use Case-Tabellen [7] einer bestimmten Codestelle einen Use Case Schritt zuzuweisen. Konkret, sollte eine Möglichkeit erarbeitet werden, die Zuweisung direkt über eine Erweiterung in der Entwicklungsumgebung zu ermöglichen.

Um dieses Ziel zu erreichen musste zunächst ein Konzept erarbeitet werden. Dabei war es notwendig, die zentralen Punkte herauszuarbeiten, um zu klären, welche konkreten Handlungen die Entwickler mit dieser Erweiterung umsetzen können sollten: Stellen im Code sollen markiert und die Erklärungen aus der erweiterten UC Tabelle integriert werden.

Dazu wurde der *ExpHelper* entwickelt. Eine Kombination aus einer IDE-Erweiterung und einem Package, die es ermöglicht, die in einer erweiterten Use Case-Tabelle enthaltenen Erklärungen manuell den entsprechenden Codestellen zuzuordnen. Dafür wird eine solche Tabelle in Form einer CSV-Datei eingelesen. Das Package liefert dabei über AOP und dem Singleton-Entwurfsmuster die Möglichkeit, beliebige Funktionen aus dem Code über C#-Attribute zu markieren. Sobald diese Funktionen aufgerufen werden, stehen die passenden Erklärungen gespeichert und abrufbereit zur Verfügung. Die Speicherung und der Zugriff auf die Erklärungen wurden so umgesetzt, dass sie sicher in einer Umgebung mit mehreren Threads arbeiten (thread-safe). Dadurch bleibt der Inhalt konsistent, selbst wenn mehrere Threads gleichzeitig darauf zugreifen. Um das Markieren nutzerfreundlicher zu gestalten, wurde eine Visual Studio Code Extension erstellt, welche das Hinzufügen der Attribute über eine UI ermöglicht. Über eine kurze

Praxisstudie wurde die Nutzbarkeit und Funktionsfähigkeit des Plugins bestätigt.

Insgesamt kann zu den hier erarbeiten Konzepten des entwickelten *ExpHelpers* festgehalten werden, dass die Entwicklung einer Proof-of-Concept Lösung für das Problem in Anbetracht der Ergebnisse der Nutzerstudie gelungen ist. Damit wurde demonstriert, dass ein effektives, nutzbares und thread-safes Plugin zur Integrierung von Erklärungsbedarf in Codeabschnitte entwickelt werden kann. Dies erlaubt den Entwicklern, gemäß der ermittelten Tendenzen, Erklärungen schneller und mit geringerem mentalen Aufwand in ihren Code einzubinden. Durch die Vereinheitlichung werden zusätzlich die Wartbarkeit des Codes verbessert sowie die Konsistenz erhöht, was wiederum die Effizienz bei der Fehlersuche steigert und die Einarbeitungszeit für neue Entwickler verkürzt. Welchen genauen Effekt ein solches Plugin in der Praxis hat, also ob und inwiefern es eine tatsächliche Erleichterung im Alltag von Entwicklern darstellt, müsste aber in einer weiteren Arbeit geklärt werden.

7.2 Ausblick

Die Umsetzung der in dieser Arbeit vorgestellten Konzepte erfolgte in C# und für VS Code. Daher müsste in Zukunft noch erarbeitet werden, ob vergleichbare Konzepte auch für andere Programmiersprachen und andere IDEs nutzbar sind oder ob es hier für die Umsetzung andere bzw. bessere Methoden gibt.

Des Weiteren kann die Speicherung in diesem Plugin noch in mehrere Erklärbarkeitsbereiche aufgeteilt werden, sodass u. a. für UI und Logik gleichzeitig unterschiedliche Erklärungen abrufbereit hinterlegt werden können.

Letzten Endes könnte auch das aktuell in C# geschriebene Package erweitert werden. So könnte dafür gesorgt werden, dass ganze Klassen markiert werden könnten oder eben nur Teile aus einer Funktion. Auch das Plugin könnte in seiner Usability mehr für die Nutzenden optimiert werden.

Darüber hinaus lassen sich die gewonnenen Erkenntnisse und Resultate dieser Arbeit mit Triggern in Verbindung bringen. Da die Erklärungen jetzt abgespeichert und jederzeit abrufbereit zur Verfügung stehen, können diese automatisch durch einen Trigger ausgelöst und kontextgerecht an passender Stelle angezeigt werden. Dieses Vorgehen eröffnet Möglichkeiten zur weiteren Optimierung der Benutzerinteraktion mit dem System.

Anhang A

Dokumente der Nutzerstudie

A.1 Fragebogen zur Demographie

Der Fragebogen wurde mithilfe von Microsoft Forms erstellt und von einzelnen Teilnehmer über die Forms Oberfläche beantwortet. Die Beantwortung der nachfolgenden Fragen wurde vor dem Lösen Aufgaben vorgenommen:

Abbildung A.1: Erste Frage des Fragebogens zur Demographie

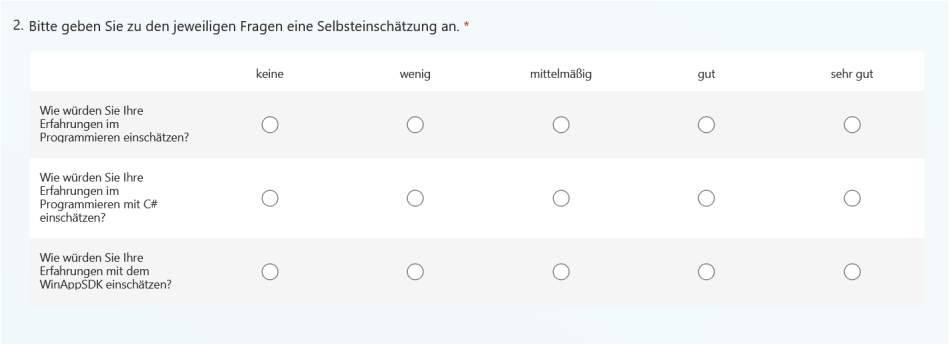


Abbildung A.2: Zweite Frage des Fragebogens zur Demographie

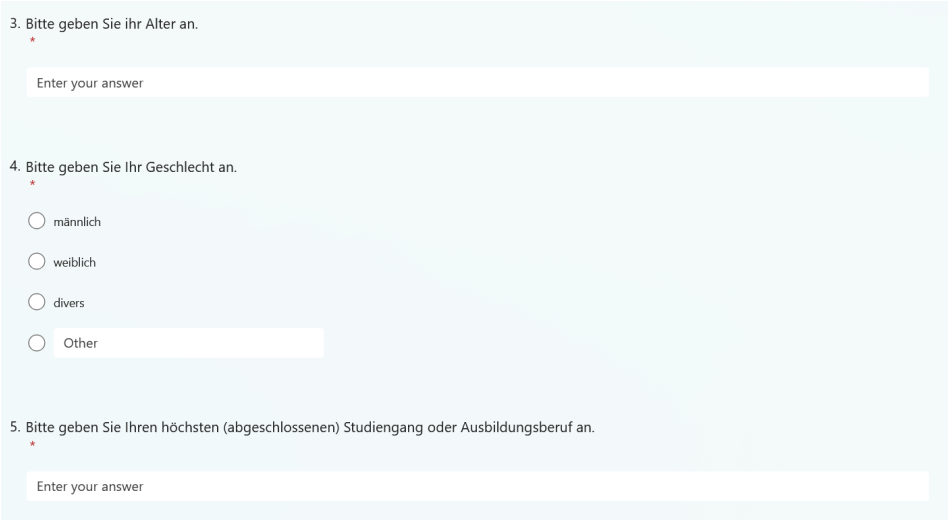


Abbildung A.3: Dritte bis fünfte Frage des Fragebogens zur Demographie

A.1.1 Ergebnisse des Fragebogens

Antwort	Anzahl
Weniger als 1 Jahr	0
Weniger als 3 Jahre	2
Weniger als 5 Jahre	3
Mindestens 5 Jahre	0
Mindestens 10 Jahre	2

Tabelle A.1: Vor wie vielen Jahren haben Sie mit dem Programmieren begonnen?

Antwort	Anzahl
Weniger als 1 Jahr	1
Weniger als 3 Jahre	2
Weniger als 5 Jahre	2
Mindestens 5 Jahre	1
Mindestens 10 Jahre	1

Tabelle A.2: Seit wie vielen Jahren programmieren Sie aktiv?

Antwort	Anzahl
Weniger als 1 Jahr	5
Weniger als 3 Jahre	1
Weniger als 5 Jahre	1
Mindestens 5 Jahre	0
Mindestens 10 Jahre	0

Tabelle A.3: Seit wie vielen Jahren arbeiten Sie bereits mit C#?

Antwort	Anzahl
Weniger als 1 Jahr	5
Weniger als 3 Jahre	1
Weniger als 5 Jahre	1
Mindestens 5 Jahre	0
Mindestens 10 Jahre	0

Tabelle A.4: Seit wie vielen Jahren arbeiten Sie bereits mit C#?

Antwort	Anzahl
Keine	0
Wenig	1
Mittelmäßig	3
Gut	2
Sehr gut	1

Tabelle A.5: Wie würden Sie Ihre Erfahrungen im Programmieren einschätzen?

Antwort	Anzahl
Keine	2
Wenig	4
Mittelmäßig	1
Gut	0
Sehr gut	0

Tabelle A.6: Wie würden Sie Ihre Erfahrungen im Programmieren mit C# einschätzen?

Antwort	Anzahl
Keine	6
Wenig	0
Mittelmäßig	1
Gut	0
Sehr gut	0

Tabelle A.7: Wie würden Sie Ihre Erfahrungen mit dem WinAppSDK einschätzen?

Alter	Anzahl
21	1
22	1
23	1
24	3
25	1

Tabelle A.8: Bitte geben Sie Ihr Alter an.

Geschlecht	Anzahl
männlich	6
weiblich	1
divers	0

Tabelle A.9: Bitte geben Sie Ihr Geschlecht an.

Abschluss	Anzahl
Bachelor of Science (Informatik)	3
Bachelor of Science (Informatik), laufend	2
Fachinformatiker für Systemintegration (laufend)	1
Bachelor of Science (Wirtschaftswissenschaften, laufend	1

Tabelle A.10: Bitte geben Sie Ihren höchsten (abgeschlossenen) Studiengang oder Ausbildungsberuf an.

A.2 Aufgabenstellungen

Auf den nachfolgenden beiden Seiten sind die ausgehändigten Aufgabenblätter der Studie dargestellt.

Nutzerstudie – Begleitzettel
Ohne Plugin

Innerhalb dieser Studie sollen Sie ein vorhandenes Programm erweitern. Die mit dem WinAppSDK erstellte App stellt vier Buttons bereit, die verschiedene Berechnungen auslösen. Sie sollen für die vier vorhandenen Funktionen in der **Calculations** Klasse Erklärungen (BusinessLogic/Calculations.cs) implementieren. Eine CSV-Datei mit Erklärungen wurde bereits erstellt. Dafür sollen Sie folgende Zuordnung vornehmen:

1. HundredsButton -> We multiplied 10 with 100: The result is 1000.
2. ThousendsButton -> We multiplied 10 with 1000: The result is 10000.
3. DivideButton -> We divided 21 by 3: The result is 7.
4. ForecastButton -> The last name that came to this developers's mind was Peter.

Versuchen Sie innerhalb von 10 Minuten ein rudimentäres System zu implementieren, dass durch einen Klick auf einen Button nach Aufruf einer der vier Funktion eine passende Erklärung ausgibt.

Achten Sie darauf, dass das Speichern der Erklärung durch den Funktionsaufruf gestartet werden muss und nicht durch das Klicken vom Button.

Betrachten Sie die geöffnete **MainWindow.xaml.cs** und **Calculation.cs** eingehend. Binden Sie die Erklärungen am besten „hardcoded“ in den Code ein. Sie können die Erklärungstexte aus der geöffneten CSV-Datei entnehmen. Die Erklärung soll angezeigt werden, sobald die explainItButton_Click Funktion ausgeführt wird, achten Sie hier auf die hinweisenden Kommentare im Code.

Hinweis 1: Durch die Eingabe von **dotnet run** können Sie das Projekt im in VS Code integrierten Terminal kompilieren und starten. Um einen Laufzeitversuch ihrer Implementierung zu starten, führen Sie diesen Befehl aus, wählen einen der Berechnungsknöpfe und danach den „Explain it“ Button an.

Hinweis 2: Nutzen Sie zum Planen / Erstellen gerne das bereitgelegte blanko Papier und die geöffnete C#-Dokumentation und C#-Tutorial Seite.

Hinweis 3: Sie müssen keine explizite Fehlerbehandlung implementieren.

Hinweis 4: Fragen Sie bei Problemen direkt nach und nutzen Sie zur Lösung des Problems weder weitere Internetrecherche noch eine LLM zur Hilfe.

Abbildung A.4: Aufgabenstellung Aufgabe 1 (ohne Plugin)

Nutzerstudie – Begleitzettel
Mit Plugin

Innerhalb dieser Studie sollen Sie ein vorhandenes Programm erweitern. Die mit dem WinAppSDK erstellte App stellt vier Buttons bereit, die verschiedene Berechnungen auslösen. Sie sollen für die vier vorhandenen Funktionen in der **Calculations** Klasse Erklärungen (BusinessLogic/Calculations.cs) implementieren. Eine CSV-Datei mit Erklärungen wurde bereits erstellt. Dafür sollen Sie folgende Zuordnung vornehmen:

1. HundredsButton -> We multiplied 10 with 100: The result is 1000.
2. ThousendsButton -> We multiplied 10 with 1000: The result is 10000.
3. DivideButton -> We divided 21 by 3: The result is 7.
4. ForecastButton -> The last name that came to this developers's mind was Peter.

Versuchen Sie innerhalb von 10 Minuten **mit Hilfe des Plugins** ein rudimentäres System zu implementieren, dass durch einen Klick auf einen Button nach Aufruf einer der vier Funktion eine passende Erklärung ausgibt.

Achten Sie darauf, dass das Speichern der Erklärung durch den Funktionsaufruf gestartet werden muss und nicht durch das Klicken vom Button.

Betrachten Sie die geöffnete **MainWindow.xaml.cs** und **Calculation.cs** eingehend. Beachten Sie auch die **geöffnete Beschreibungsseite des Plugins**. Die Erklärung soll angezeigt werden, sobald die explainItButton_Click Funktion ausgeführt wird, achten Sie hier auf die hinweisenden Kommentare im Code.

divide

Hinweis 1: Durch die Eingabe von **dotnet run** können Sie das Projekt im in VS Code integrierten Terminal kompilieren und starten. Um einen Laufzeitversuch ihrer Implementierung zu starten, führen Sie diesen Befehl aus, wählen einen der Berechnungsknöpfe und danach den „Explain it“ Button an.

Hinweis 2: Denken Sie auch daran, die gespeicherte Erklärung an entsprechender Stelle abzufragen (**Explainer.GetExplanation()**) und als Erklärung zu setzen.

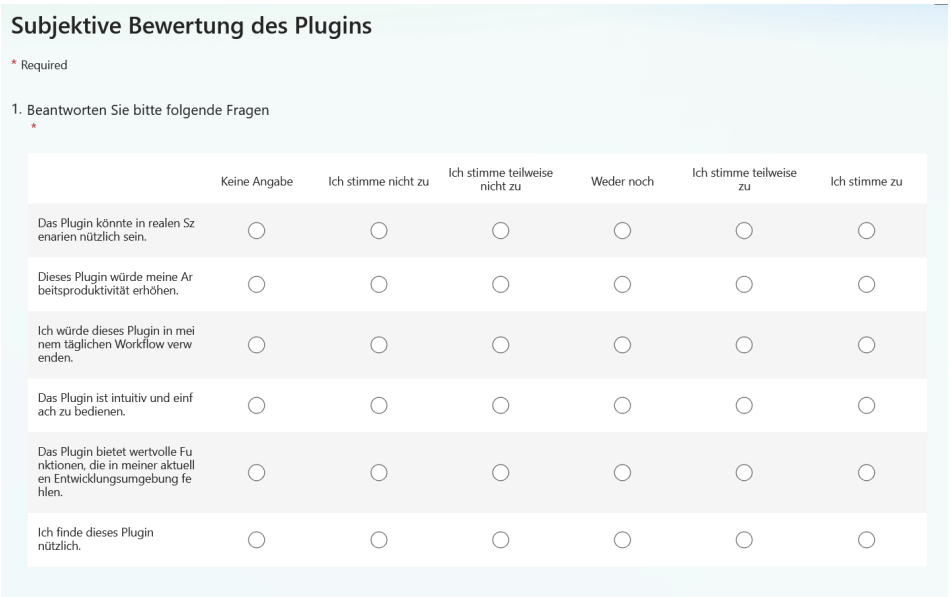
Hinweis 3: Sie müssen keine explizite Fehlerbehandlung implementieren.

Hinweis 4: Fragen Sie bei Problemen direkt nach und nutzen Sie zur Lösung des Problems weder weitere Internetrecherche noch eine LLM zur Hilfe.

Abbildung A.5: Aufgabenstellung Aufgabe 2 (mit Plugin)

A.3 Fragebogen zur Erhebung der subjektiven Wahrnehmung

Der Fragebogen wurde mithilfe von Microsoft Forms erstellt und von einzelnen Teilnehmer über die Forms Oberfläche beantwortet. Die Beantwortung der nachfolgenden Fragen wurde nach dem Lösen der Aufgaben vorgenommen:



Subjektive Bewertung des Plugins

* Required

1. Beantworten Sie bitte folgende Fragen *

	Keine Angabe	Ich stimme nicht zu	Ich stimme teilweise nicht zu	Weder noch	Ich stimme teilweise zu	Ich stimme zu
Das Plugin könnte in realen Szenarien nützlich sein.	○	○	○	○	○	○
Dieses Plugin würde meine Arbeitsproduktivität erhöhen.	○	○	○	○	○	○
Ich würde dieses Plugin in meinem täglichen Workflow verwenden.	○	○	○	○	○	○
Das Plugin ist intuitiv und einfach zu bedienen.	○	○	○	○	○	○
Das Plugin bietet wertvolle Funktionen, die in meiner aktuellen Entwicklungsumgebung fehlen.	○	○	○	○	○	○
Ich finde dieses Plugin nützlich.	○	○	○	○	○	○

Abbildung A.6: Fragebogen zur Erhebung der subjektiven Wahrnehmung

A.3.1 Ergebnisse des Fragebogens

Antwort	Anzahl
Ich stimme zu	5
Ich stimme teilweise zu	1
Weder noch	0
Ich stimme teilweise nicht zu	0
Ich stimme nicht zu	0
Keine Angabe	1

Tabelle A.11: Das Plugin könnte in realen Szenarien nützlich sein.

Antwort	Anzahl
Ich stimme zu	2
Ich stimme teilweise zu	3
Weder noch	1
Ich stimme teilweise nicht zu	1
Ich stimme nicht zu	0
Keine Angabe	0

Tabelle A.12: Dieses Plugin würde meine Arbeitsproduktivität erhöhen

Antwort	Anzahl
Ich stimme zu	2
Ich stimme teilweise zu	3
Weder noch	0
Ich stimme teilweise nicht zu	1
Ich stimme nicht zu	0
Keine Angabe	1

Tabelle A.13: Ich würde dieses Plugin in meinem täglichen Workflow verwenden

Antwort	Anzahl
Ich stimme zu	5
Ich stimme teilweise zu	2
Weder noch	0
Ich stimme teilweise nicht zu	0
Ich stimme nicht zu	0
Keine Angabe	0

Tabelle A.14: Das Plugin ist intuitiv und einfach zu bedienen.

Antwort	Anzahl
Ich stimme zu	1
Ich stimme teilweise zu	1
Weder noch	0
Ich stimme teilweise nicht zu	0
Ich stimme nicht zu	2
Keine Angabe	3

Tabelle A.15: Das Plugin bietet wertvolle Funktionen, die in meiner aktuellen Entwicklungsumgebung fehlen

Antwort	Anzahl
Ich stimme zu	5
Ich stimme teilweise zu	1
Weder noch	0
Ich stimme teilweise nicht zu	0
Ich stimme nicht zu	1
Keine Angabe	0

Tabelle A.16: Ich finde dieses Plugin nützlich

Anhang B

Erfüllung der Voraussetzungen für den t-test

In unserem Fall konnte der dependent paire-sample one-tailed t-test durchgeführt werden, da folgende Voraussetzungen nach Howell [42] und Wachsmuth [39] galten:

1. **Abhängige Wertemengen**: Die entsprechenden Wertepaare aus beiden Mengen müssen voneinander abhängig sein. Dies gilt, da in dieser Studie beide Paare aus von unterschiedlichen Aufgaben, aber von derselben Person stammen.

2. **Normalverteilung der Differenzen**: Dafür wurde der Shapiro-Wilk Test auf beiden Differenzmengen für Zeit und TLX ausgeführt. Das Ergebnis ergab, dass es keine signifikante Abweichung von der Normalverteilung gab. Somit ist auch dies erfüllt. [43][44]

3. **Intervall- oder Rational-Skala**: Gestoppte Zeiten liegen auf der Rationalskala. Bei Zeitmessungen gibt es einen absoluten Nullpunkt (0 Sekunden bedeutet keine verstrichene Zeit). Die Verhältnisse sind sinnvoll: 20 Sekunden sind doppelt so viel Zeit wie 10 Sekunden.

   NASA TLX-Werte werden typischerweise auf einer Intervallskala betrachtet, weil die Abstände zwischen den Werten interpretierbar sind und es keinen natürlichen Nullpunkt gibt.

4. **Varianzen nicht unterschiedlich**: Es muss geprüft werden, ob die Varianzen nicht zu unterschiedlich sind. Diese sollten sich ähnlich sein. Um nachweisen zu können, dass die Varianzen nicht unterschiedlich sind, wird Levenes Test durchgeführt. Das Ergebnis ergab, dass es keine signifikanten Unterschiede in den Varianzen beider Mengen (Zeit und TLX-Scores) gab. [45]

5. **Unabhängigkeit der Paare**: Die einzelnen Paare sind unabhängig voneinander, da es keinen Kontakt zwischen den einzelnen Teilnehmern oder sonstige Beeinflussung der Teilnehmer untereinander gegeben hat. Die Studiendurchläufe wurden unabhängig voneinander durchgeführt.

Da die Punkte wie beschrieben galten, konnte der t-test angewandt werden.

Abbildungsverzeichnis

Literaturverzeichnis

[1] M. Felderer, R. Reussner, und B. Rumpe, "Software Engineering und Software-Engineering-Forschung im Zeitalter der Digitalisierung," *Informatik Spektrum*, Bd. 44, Nr. 2, S. 82–94, 2021.

[2] H. Deters, J. Droste, M. Obaidi, und K. Schneider, "How explainable is your system? Towards a quality model for explainability," in *Requirements Engineering: Foundation for Software Quality*, Reihe Lecture Notes in Computer Science, D. Mendez und A. Moreira, Eds. Springer Nature Switzerland and Imprint Springer, 2024, S. 3–19.

[3] M. A. Köhl, K. Baum, M. Langer, D. Oster, T. Speith, und D. Bohlender, "Explainability as a non-functional requirement," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 2019, S. 363–368.

[4] L. Chazette, W. Brunotte, und T. Speith, "Exploring explainability: A definition, a model, and a knowledge catalogue," Reihe 2021 IEEE 29th International Requirements Engineering Conference (RE), 2021, S. 197–208.

[5] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2001.

[6] U. Hammerschall und G. Beneken, *Software Requirements*. Pearson Deutschland, 2013.

[7] D. Behrens, "Extending use case tables with explainability needs." Bachelorarbeit, Leibniz Universität Hannover, 2024.

[8] B. Wagner und Microsoft, "Definieren und Lesen von benutzerdefinierten Attributen." [Online]. Verfügbar: https://learn.microsoft.com/de-de/dotnet/csharp/advanced-topics/reflection-and-attributes/attribute-tutorial. (Zugriff am 24.10.2024)

[9] Microsoft, "Attribute Klasse." [Online]. Verfügbar: https://learn.microsoft.com/de-de/dotnet/api/system.attribute?view=net-8.0. (Zugriff am 24.10.2024)

[10] J. Droste, H. Deters, M. Obaidi, und K. Schneider, "Explanations in Everyday Software Systems: Towards a Taxonomy for Explainability Needs," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, Jun. 2024, S. 55–66.

[11] M. Unterbusch, M. Sadeghi, J. Fischbach, M. Obaidi, und A. Vogelsang, "Explanation Needs in App Reviews: Taxonomy and Automated Detection," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 2023, S. 102–111.

[12] E. Gamma, R. Helm, R. Johnson, und J. Vlissides, *Design Patterns: Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*, 1st ed. Frechen: mitp-Verl., 2015, S. 17–30, 123, 172–174.

[13] J. F. Dooley und V. A. Kazakova, *Software Development, Design, and Coding: With Patterns, Debugging, Unit Testing, and Refactoring*, 3rd ed. Berkeley, CA: Apress and Imprint Apress, 2024, S. 278–281.

[14] P. Japikse. Visual Studio Toolbox, "Designmuster: Singleton." Veröffentlicht am 08.08.2017. Verfügbar: https://learn.microsoft.com/de-de/shows/visual-studio-toolbox/design-patterns-singleton. (Zugriff am 24.08.2024)

[15] B. Wagner, P. Kulikov, R. Jaeschke, J. Skeet, und Y. Victor. C# standard specification - C# 8 draft specification, "Classes: Static and instance members." Veröffentlicht am 07.02.2024. [Online]. Verfügbar: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/classes#1538-static-and-instance-members. (Zugriff am 25.08.2024)

[16] O. Musch, *Design patterns mit Java: Eine Einführung*. Wiesbaden und Heidelberg: Springer Vieweg, 2021, S. 23–27.

[17] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, und J. Irwin, "Aspect-oriented programming," in *ECOOP'97 — Object-Oriented Programming*, Reihe Lecture Notes in Computer Science, G. Goos, J. Hartmanis, J. van Leeuwen, M. Akşit, und S. Matsuoka, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, Bd. 1241, S. 220–242.

[18] M. D. Groves, *AOP in .NET*. Manning, 2013.

[19] P. Späth, I. Cosmina, R. Harrop, und C. Schaefer, "Spring AOP," in *Pro Spring 6 with Kotlin*, P. Späth, I. Cosmina, R. Harrop, und C. Schaefer, Eds. Berkeley, CA: Apress, 2023, S. 189–270.

[20] C. Colombo und G. J. Pace, Eds., *Runtime Verification*. Cham: Springer International Publishing, 2022.

[21] O. Spinczyk und pure-systems GmbH, "Documentation: AspectC++ Language Reference." [Online]. Verfügbar: https://www.aspectc.org/doc/ac-languageref.pdf

[22] ionel, "Aspectlib-1.4.2." [Online]. Verfügbar: https://python-aspectlib.readthedocs.io/en/latest/. (Zugriff am 17.09.2024)

[23] SharpCrafters s.r.o, "Metalama Documentation." [Online]. Verfügbar: https://doc.postsharp.net/metalama. (Zugriff am 17.09.2024)

[24] Amazon, "Was ist eine IDE?" [Online]. Verfügbar: https://aws.amazon.com/de/what-is/ide/. (Zugriff am 31.08.2024)

[25] P. Fischer, *Lexikon der Informatik (German Edition)*, 15th ed. Dordrecht: Springer, 2011, S. 681.

[26] A. D. Sole, *Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux*, 3rd ed. Berkeley, CA: Apress and Imprint Apress, 2023, S. 1–4, 45–46.

[27] Microsoft, "Extension API." [Online]. Verfügbar: https://code.visualstudio.com/api. (Zugriff am 31.08.2024)

[28] Microsoft, "Extension Capabilities Overview." [Online]. Verfügbar: https://code.visualstudio.com/api/extension-capabilities/overview. (Zugriff am 31.08.2024)

[29] Microsoft, "Publishing Extensions." [Online]. Verfügbar: https://code.visualstudio.com/api/references/vscode-api. (Zugriff am 31.08.2024)

[30] Microsoft, "vscode/vsce." [Online]. Verfügbar: https://github.com/microsoft/vscode-vsce. (Zugriff am 31.08.2024)

[31] Microsoft, "Publishing Extensions." [Online]. Verfügbar: https://code.visualstudio.com/api/working-with-extensions/publishing-extension. (Zugriff am 31.08.2024)

[32] Human Performance Research Group (NASA Ames Research Center), "NASA Task Load Index (TLX): v. 1.0," 1986. [Online]. Verfügbar: https://ntrs.nasa.gov/api/citations/20000021488/downloads/20000021488.pdf

[33] C. Ullenboom, *Java ist auch eine Insel: Einführung, Ausbildung, Praxis*, 16th ed., Reihe Rheinwerk Computing. Bonn: Rheinwerk Verlag, 2023, S. 577–642.

[34] Microsoft, "Quick Picks." [Online]. Verfügbar: https://code.visualstudio.com/api/ux-guidelines/quick-picks. (Zugriff am 20.09.2024)

[35] J. Skeet. [Online]. Verfügbar: https://csharpindepth.com/articles/singleton#cctor. (Zugriff am 20.09.2024)

[36] J. Skeet. [Online]. Verfügbar: https://csharpindepth.com/articles/BeforeFieldInit. (Zugriff am 20.09.2024)

[37] B. Wagner und Microsoft, "Static Constructors (C Programming Guide)." [Online]. Verfügbar: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-constructors. (Zugriff am 20.09.2024)

[38] B. Rasch, M. Friese, W. Hofmann, und E. Naumann, *Der t-Test*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 33–79. [Online]. Verfügbar: https://doi.org/10.1007/978-3-662-43524-3_3

[39] H. Wachsmuth, "Introduction to Natural Language Processing — Part V: Basics of Empirical Methods." [Online]. Verfügbar: https://www.ai.uni-hannover.de/fileadmin/ai/teaching/inlp-24s/part05-empirical-methods.pdf. (Zugriff am 21.11.2024)

[40] The SciPy community, "SciPy API: Statistical Functions (scipy.stats): ttest_rel." [Online]. Verfügbar: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html. (Zugriff am 21.11.2024)

[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, und A. Wesslen, *Experimentation in software engineering*. Heidelberg and New York and Dordrecht and London: Springer, 2012.

[42] D. C. Howell, *Statistical methods for psychology*, 8th ed. Belmont, Calif.: Wadsworth Cengage Learning, 2013.

[43] The SciPy community, "SciPy API: Statistical Functions (scipy.stats): shapiro." [Online]. Verfügbar: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html. (Zugriff am 21.11.2024)

[44] C. Starbuck, *The Fundamentals of People Analytics: With Applications in R*, 1st ed. Cham: Springer International Publishing and Imprint Springer, 2023, S. 157–158.

[45] The SciPy community, "SciPy API: Statistical Functions (scipy.stats): levene." [Online]. Verfügbar: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.levene.html. (Zugriff am 21.11.2024)