|  |  |
| --- | --- |
|  | Fachhochschul-Bachelorstudiengang  **SOFTWARE ENGINEERING**  A-4232 Hagenberg, Austria |

**Vergleich der Extension-APIs  
in VS Code und IntelliJ**

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science in Engineering

Eingereicht von

**Philipp Seiringer**

Begutachtet von Dr. Josef Pichler

Hagenberg, Februar 2024

**Erklärung**

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Die vorliegende, gedruckte Bachelorarbeit ist identisch zu dem elektronisch übermittelten Textdokument.

Datum Unterschrift

Kurzfassung

Abstract

Inhaltsverzeichnis

[1 Einleitung 1](#_Toc146741949)

[1.1 Motivation 1](#_Toc146741950)

[1.2 Ziel 1](#_Toc146741951)

[1.3 Aufbau 1](#_Toc146741952)

[2 Grundlagen der Plugin Entwicklung 1](#_Toc146741953)

[2.1 Entwicklungsumgebungen 1](#_Toc146741954)

[2.1.1 Visual Studio Code 1](#_Toc146741955)

[2.1.2 IntelliJ IDEA 1](#_Toc146741956)

[2.2 Programmiersprachen 1](#_Toc146741957)

[2.2.1 Typescript 1](#_Toc146741958)

[2.2.2 Java 1](#_Toc146741959)

[2.3 Aufbau der Plugin API 1](#_Toc146741960)

[2.3.1 Visual Studio Code 1](#_Toc146741961)

[2.3.2 IntelliJ IDEA 1](#_Toc146741962)

[2.4 Funktionalität der Plugin API 1](#_Toc146741963)

[2.4.1 Visual Studio Code 1](#_Toc146741964)

[2.4.2 IntelliJ IDEA 1](#_Toc146741965)

[3 Aufbau des Prototyps – RecentChangesPlugin 1](#_Toc146741966)

[3.1 1](#_Toc146741967)

[4 Entwicklung des Prototyps für Visual Studio Code 1](#_Toc146741968)

[4.1 Design 2](#_Toc146741969)

[4.2 Implementierung 2](#_Toc146741970)

[4.3 Tests 2](#_Toc146741971)

[4.4 (eventuell) CI/CD 2](#_Toc146741972)

[4.5 Publishing 2](#_Toc146741973)

[5 Entwicklung des Prototyps für IntelliJ IDEA 2](#_Toc146741974)

[5.1 Design 2](#_Toc146741975)

[5.2 Implementierung 2](#_Toc146741976)

[5.3 Tests 2](#_Toc146741977)

[5.4 (eventuell) CI/CD 2](#_Toc146741978)

[5.5 Publishing 2](#_Toc146741979)

[6 Bewertungskriterien 2](#_Toc146741980)

[6.1 Popularität der Entwicklungsumgebung 2](#_Toc146741981)

[6.2 Hardwareanforderungen (besserer Name -> vergleich von Geschwindigkeit, Speicherverbrauch usw von VSCode vs IJ) 2](#_Toc146741982)

[6.3 Feature Umfang 2](#_Toc146741983)

[6.4 Intuitivität der API (eventuell) 2](#_Toc146741984)

[6.5 Dokumentation der API 2](#_Toc146741985)

[6.6 Testbarkeit der Plugins 2](#_Toc146741986)

[6.7 Möglichkeiten des Publishings 2](#_Toc146741987)

[6.8 Installationsprozess des Plugins 2](#_Toc146741988)

[7 Vergleich der Kriterien 3](#_Toc146741989)

[7.1 Popularität der Entwicklungsumgebung 3](#_Toc146741990)

[7.1.1 Visual Studio Code 3](#_Toc146741991)

[7.1.2 IntelliJ IDEA 3](#_Toc146741992)

[7.1.3 (eventuell hier noch jeweils einen Abschnitt „Vergleich“) 3](#_Toc146741993)

[7.2 Hardwareanforderungen (besserer Name -> vergleich von Geschwindigkeit, Speicherverbrauch usw von VSCode vs IJ) 3](#_Toc146741994)

[7.2.1 Visual Studio Code 3](#_Toc146741995)

[7.2.2 IntelliJ IDEA 3](#_Toc146741996)

[7.3 Feature Umfang 3](#_Toc146741997)

[7.3.1 Visual Studio Code 3](#_Toc146741998)

[7.3.2 IntelliJ IDEA 3](#_Toc146741999)

[7.4 Intuitivität der API (eventuell) 3](#_Toc146742000)

[7.4.1 Visual Studio Code 3](#_Toc146742001)

[7.4.2 IntelliJ IDEA 3](#_Toc146742002)

[7.5 Dokumentation der API 3](#_Toc146742003)

[7.5.1 Visual Studio Code 3](#_Toc146742004)

[7.5.2 IntelliJ IDEA 3](#_Toc146742005)

[7.6 Testbarkeit der Plugins 3](#_Toc146742006)

[7.6.1 Visual Studio Code 3](#_Toc146742007)

[7.6.2 IntelliJ IDEA 3](#_Toc146742008)

[7.7 Möglichkeiten des Publishings 3](#_Toc146742009)

[7.7.1 Visual Studio Code 4](#_Toc146742010)

[7.7.2 IntelliJ IDEA 4](#_Toc146742011)

[7.8 Installationsprozess des Plugins 4](#_Toc146742012)

[7.8.1 Visual Studio Code 4](#_Toc146742013)

[7.8.2 IntelliJ IDEA 4](#_Toc146742014)

[8 Conclusion 4](#_Toc146742015)

[9 Einleitung 1](#_Toc146742016)

[9.1 Motivation 1](#_Toc146742017)

[9.2 Ziel 1](#_Toc146742018)

[9.3 Aufbau der Arbeit 1](#_Toc146742019)

[10 Vergleich der Entwicklungsplattformen 2](#_Toc146742020)

[10.1 Visual Studio Code 2](#_Toc146742021)

[10.2 IntelliJ IDEA 2](#_Toc146742022)

[11 Vergleich der angebotenen Funktionalität und deren Anwendbarkeit 6](#_Toc146742023)

[11.1 Visual Studio Code 6](#_Toc146742024)

[11.1.1 Das Plugin Projekt 6](#_Toc146742025)

[11.1.2 Funktionalität 7](#_Toc146742026)

[11.2 IntelliJ IDEA 8](#_Toc146742027)

[11.2.1 Das Plugin Projekt 8](#_Toc146742028)

[11.2.2 Funktionalität 8](#_Toc146742029)

[11.2.3 IntelliJ IDEA Flora Plugin 9](#_Toc146742030)

[12 Codebeispiel: Recent Changes Plugin 11](#_Toc146742031)

[13 Vergleich der Code-Metriken 12](#_Toc146742032)

[14 Conclusion 13](#_Toc146742033)

1. Einleitung
   1. Motivation
   2. Ziel
   3. Aufbau
2. Grundlagen der Plugin Entwicklung
   1. Entwicklungsumgebungen
      1. Visual Studio Code
      2. IntelliJ IDEA
   2. Programmiersprachen
      1. Typescript
      2. Java
   3. Aufbau der Plugin API
      1. Visual Studio Code
      2. IntelliJ IDEA
   4. Funktionalität der Plugin API
      1. Visual Studio Code
      2. IntelliJ IDEA
3. Aufbau des Prototyps – RecentChangesPlugin
4. Entwicklung des Prototyps für Visual Studio Code
   1. Design
   2. Implementierung
   3. Tests
   4. (eventuell) CI/CD
   5. Publishing
5. Entwicklung des Prototyps für IntelliJ IDEA
   1. Design
   2. Implementierung
   3. Tests
   4. (eventuell) CI/CD
   5. Publishing
6. Bewertungskriterien
   1. Popularität der Entwicklungsumgebung
   2. Hardwareanforderungen (besserer Name -> vergleich von Geschwindigkeit, Speicherverbrauch usw von VSCode vs IJ)
   3. Feature Umfang
   4. Intuitivität der API (eventuell)
   5. Dokumentation der API
   6. Testbarkeit der Plugins
   7. Möglichkeiten des Publishings
   8. Installationsprozess des Plugins
7. Vergleich der Kriterien
   1. Popularität der Entwicklungsumgebung
      1. Visual Studio Code
      2. IntelliJ IDEA
      3. (eventuell hier noch jeweils einen Abschnitt „Vergleich“)
   2. Hardwareanforderungen (besserer Name -> vergleich von Geschwindigkeit, Speicherverbrauch usw von VSCode vs IJ)
      1. Visual Studio Code
      2. IntelliJ IDEA
   3. Feature Umfang
      1. Visual Studio Code
      2. IntelliJ IDEA
   4. Intuitivität der API (eventuell)
      1. Visual Studio Code
      2. IntelliJ IDEA
   5. Dokumentation der API
      1. Visual Studio Code
      2. IntelliJ IDEA
   6. Testbarkeit der Plugins
      1. Visual Studio Code
      2. IntelliJ IDEA
   7. Möglichkeiten des Publishings
      1. Visual Studio Code
      2. IntelliJ IDEA
   8. Installationsprozess des Plugins
      1. Visual Studio Code
      2. IntelliJ IDEA
8. Conclusion
9. Einleitung
   1. Motivation

SoftwareentwicklerInnen arbeiten täglich mit verschiedensten Werkzeugen und Entwicklungsumgebungen, sogenannten IDEs (=Integrated Development Environment). Diese Plattformen bieten teils sehr unterschiedliche Funktionalitäten, die die Softwareentwicklung erleichtern sollen. Dabei bieten sie Unterstützung für verschiedenste Programmiersprachen und Technologien und binden zahlreiche Werkzeuge für spezifische Anwendungsfälle ein. Aufgrund des immer rascher werdenden Entstehens von neuen Technologien bieten mehr und mehr IDEs Möglichkeiten zur Entwicklung von eigenen Plugins, welche dann auch an andere EntwicklerInnen bereitgestellt werden können. So können in kürzester Zeit neue Technologien unterstützt werden und EntwicklerInnen haben selbst die Macht darüber zu entscheiden welche Plugins sie nutzen möchten und welche nicht.

Vor der Entwicklung solcher Plugins ist es wichtig zu entscheiden für welche IDE das Plugin erstellt werden soll. Dabei spielen Aspekte wie zum Beispiel die Einfachheit und Flexibilität in der Entwicklung, der Umfang an angebotener Funktionalität, die Möglichkeit die Nutzerinteraktion und somit die User Experience zu steuern und viele weitere eine Rolle. Diese Bachelorarbeit versucht in diesen Bereichen einen Überblick zu schaffen und vergleicht hierfür die Plugin Entwicklung in zwei der momentan beliebtesten IDEs, Visual Studio Code und IntelliJ IDEA. Durch den Vergleich der beiden Produkte und dem Herausarbeiten und Aufbereiten der Unterschiede wird es anderen EntwicklerInnen erleichtert diese Entscheidung zu treffen.

* 1. Ziel
  2. Aufbau der Arbeit

1. Vergleich der Entwicklungsplattformen
   1. Visual Studio Code

Die erste offizielle Version von Visual Studio Code, häufig abgekürzt auch als VS Code, wurde am 14. April 2016 von Microsoft veröffentlicht. Die Idee hinter VS Code war einen möglichst einfachen Code Editor anzubieten, welcher nur die wichtigsten und besten Funktionen für EntwicklerInnen beinhaltete. Es hob sich somit von anderen IDEs wie der Visual Studio Reihe von Microsoft ab, da es ein sehr leichtgewichtiger Editor war, welcher trotzdem mit einer großen Menge an Programmiersprachen arbeiten konnte und für diese auch Microsofts code completion namens „IntelliSense“ unterstützte. Weiters war Visual Studio Code das erste Produkt der Visual Studio Familie welches Cross-Plattform Entwicklung auf Windows, Linux und OSX unterstützte.

Aus den Stack Overflow developer surveys der vergangenen Jahre kann der rasche Aufstieg von VS Code beobachtet werden. Während es im Jahr 2016 nur von etwa 7,2 Prozent der EntwicklerInnen genutzt wurde, war es zwei Jahre später bereits (wenn auch knapp) das meistgenutzte IDE mit 34,9%. In der aktuellsten Umfrage von 2023 war es der klare Sieger und wurde vom 73,71% der Abstimmenden aktiv genutzt.

Ein Grund für diesen Erfolg mag vermutlich die Möglichkeit zur Entwicklung von Plugins sein. Durch die direkte Einbindung des Visual Studio Marketplace in VS Code bildete sich über die Jahre eine große Community die eine enorme Anzahl von Plugins entwickelt, verbessert und betreut.

Src:

<https://en.wikipedia.org/wiki/Visual_Studio_Code>

<https://de.wikipedia.org/wiki/Visual_Studio_Code>

<https://web.archive.org/web/20151009211114/http://blogs.msdn.com/b/vscode/archive/2015/04/29/announcing-visual-studio-code-preview.aspx>

<https://shiftmag.dev/vs-code-171/>

* 1. IntelliJ IDEA

IntelliJ IDEA wurde erstmals im Januar 2001 von dem Unternehmen JetBrains veröffentlicht. Im Gegensatz zu Visual Studio Code handelt es sich bei IntelliJ um ein IDE welches speziell auf die Entwicklung von Programmen in den Programmiersprachen Java und Kotlin ausgelegt ist. IntelliJ IDEA wird in einer frei zu verwendenden, open source „Community Edition“, sowie in einer kommerziellen Form als „IntelliJ IDEA Ultimate“ angeboten.

Aufgrund der Spezialisierung auf Java und Kotlin Projekte unterstützt die IntelliJ Community Edition nur eine relativ kleine Auswahl an Sprachen, Frameworks und Build Tools. Während IntelliJ IDEA Ultimate den Umfang an Features schon deutlich erweitert, bietet JetBrains auch noch weitere (kommerzielle) IDEs an. Diese sind alle für unterschiedliche Programmiersprachen oder Sprachfamilien ausgelegt. Einige der bekanntesten sind dabei CLion für die Sprachen C und C++, Rider für die .NET Sprachen, PhpStorm für PHP, WebStorm für JavaScript und viele weitere. Zum aktuellen Zeitpunkt sind es insgesamt elf verschiedene IDEs die von JetBrains angeboten werden und die alle auf der IntelliJ Platform basieren. Das bedeutet nicht nur, dass sich all diese IDEs in der Verwendung und im Aussehen sehr ähnlich sind, sondern auch, dass ein Plugin, welches für die allgemeine IntelliJ Platform entworfen wurde, relativ problemlos auch für mehrere IDEs dieser Form veröffentlicht werden kann.

Im Gegensatz zu Visual Studio Code ist IntelliJ ein eher schwergewichtiger Editor, der sehr viel Funktionalität schon von Grund auf eingebaut hat. Die EntwicklerInnen sind hier nicht so stark auf Plugins angewiesen. Dies lässt sich auch durch die Anzahl von Plugins erkennen, die auf dem JetBrains Marketplace angeboten werden. Für die IntelliJ Platform gibt es aktuell etwas über 7500 Plugins die in die IDE integriert werden können. Für Visual Studio Code sind es hingegen inzwischen fast 51000 Plugins.

Zum aktuellen Zeitpunkt bietet IntelliJ in den unterschiedlichen Versionen folgende features:

Sprachen

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Frameworks

Ein Bild, das Text, Screenshot, Dokument enthält.

Automatisch generierte Beschreibung

Build Tools

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

TODO irgendwas mit andere IDEs von Jetbrains

Src:

<https://en.wikipedia.org/wiki/IntelliJ_IDEA>

<https://www.jetbrains.com/products/compare/?product=idea-ce&product=idea>

1. Vergleich der angebotenen Funktionalität und deren Anwendbarkeit
   1. Visual Studio Code
      1. Das Plugin Projekt
         1. Setup eines Plugin Projekts

Zum Aufsetzen eines neuen Projektes kann bei VS Code ein einfaches Generator Programm verwendet werden, welches die Ordnerstruktur den Angaben entsprechend aufbaut. Funktionieren tut dies über die beiden npm Packages „Yeoman“ und „VS Code Extension Generator“, welche mit dem Befehl „npm install -g yo generator-code“ installiert werden können. Der Generator wird dann mit „yo code“ gestartet. Daraufhin können verschiedene Templates für das Projekt gewählt werden und Angaben zum Projektnamen und ähnlichen Parametern getroffen werden.

* + - 1. Aufbau eines Plugins

Beim Generieren eines einfachen Plugin Projekts in der Sprache TypeScript mit dem oben genannten Generator entsteht folgende Ordnerstruktur:

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte BeschreibungFür die Plugin Entwicklung am relevantesten sind hierbei die beiden markierten Dateien „package.json“ und „extension.ts“.

package.json wird als das „Extension Manifest“ bezeichnet und dient der Konfiguration des Plugins. Es enthält die Definitionen für ActivationEvents, Commands, Views, Menus, Keybindings und vieles mehr.

extension.ts ist die Datei die den eigentlichen Plugin-Code enthält. Sie enthält eine „activate“ Funktion die bei der ersten Aktivierung des Plugins ausgeführt wird. In den meisten Fällen wird in dieser Datei allerdings nur die Initialisierung aller für das Plugin nötigen Komponenten und zum Beispiel die Registrierung für verschiedene Events erledigt. Die eigentliche Logik befindet sich dann in den hier initialisierten Klassen. Bei der Aktivierung wird auch ein Objekt der „vscode.ExtensionContext“ Klasse übergeben. Über dieses Objekt kann auf die Schnittstellen der Extension API zugegriffen werden. Wann genau diese Aktivierung passiert, kann wiederum in der package.json Datei festgelegt werden.

* + 1. Funktionalität
       1. Ausführbarer Code (Commands)

Bei Visual Studio Code gibt es die Möglichkeit Plugin-definierten Code sozusagen „auf Befehl“ auszuführen. Diese Funktion eines Plugins wird passenderweise als „Command“ bezeichnet.

Um in einem Plugin einen Command einzubauen, muss dieser in der package.json Datei definiert werden. Dabei ist mindestens eine eindeutige Bezeichnung und ein Titel welcher später angezeigt wird festzulegen. Zusätzlich kann auch eine Kategorie, ein Icon, eine Kurzbezeichnung und eine Bedingung zu der der Command aktiv ist bestimmt werden. Welche der vorgenommenen Einstellungen für die Darstellung des Commands genutzt werden hängt unter anderem von dem Menü ab in dem der Command angezeigt wird.

Welcher Code ausgeführt wird kann bei der Aktivierung der Extension gesetzt werden. Dabei kann für jeden (zuvor definierten) Command über die vscode.commands.registerCommand() oder die vscode.commands.registerTextEditorCommand() Funktion ein Callback angegeben. Dieses Callback wird dann ausgeführt wird sobald der Command aufgerufen wird. Die register Funktionen retournieren ein Disposable Objekt welches beim ExtensionContext bekannt gegeben werden muss. Dieser kümmert sich dann um das Disposen der Commands wenn die Erweiterung deaktiviert werden sollte. /\*TODO evtl Ref Design Patterns GoF\*/

* + - 1. Persistente Speicherung (Data Storage, Settings, Secrets)
      2. Code Completion
      3. Benutzerdefinierte Sprachen
      4. Nutzerinteraktion

Um ein Plugin auch interaktiv gestalten zu können, bietet Visual Studio Code viele verschiedene Schnittstellen zum User an.

Die wahrscheinlich einfachste wurde bereits oben im Abschnitt Commands beschrieben. Der Nutzer kann also einen Befehl absetzen und ein bestimmter Algorithmus wird ausgeführt.

Um den Zugriff auf Commands etwas einfacher und intuitiver zu gestalten, können Keybindings definiert werden. Man legt dafür eine Tastenkombination fest welche, wenn gedrückt, den Command absetzt. Noch einfacher wird es für den User wenn das Plugin einen Menüeintrag bereitstellt. Solche Menüs können an vielen verschiedenen Stellen im IDE eingehängt werden. Gängige Positionen hierfür sind zum Beispiel die Titelleiste des Editors, verschiedene Kontext (Rechtsklick) Menüs, der Dialog für das Anlegen einer neuen Datei, die Titelleiste einer bestimmten View oder ein beliebiges von dem Plugin beigesteuertes Submenü in der Menüleiste.

Um den NutzerInnen auch Feedback über die Ausführung des Plugin Codes zu geben ist in VS Code für drei allgemeine Anwendungsfälle vorgesorgt. Um den NutzerInnen eine kurze Rückmeldung zu geben können am besten Notifications genutzt werden. Diese zeigen eine Kurze Nachricht an, welche im Stil einer Information, einer Warnung oder einer Error Meldung dargestellt werden kann. Um einen längeren Fluss von Ausgaben (wie zum Beispiel Log-Nachrichten des Plugins) anzuzeigen können Output Channels genutzt werden. An diese können Textzeilen nach und nach angehängt werden und sie werden dem User dann in einem Terminalartigen Fenster präsentiert. In vielen fällen reicht es schon als Feedback einen einfachen Ladebalken anzuzeigen. So kann dem User klar gemacht werden, dass das Plugin immer noch arbeitet und noch kein Fehler aufgetreten ist. Für diesen Anwendungsfall kann die Progress API genutzt werden.

Natürlich braucht es während der Ausführung von Plugin Code häufig auch Eingaben der Nutzerinnen. Die einfachste Form bieten hier der Quick Pick Dialog und der File Picker Dialog. Mit Quick Pick können vom User eine Reihe von Eingaben verlangt werden. Dabei ist es möglich fixe Optionen vorzugeben, den User selbst etwas eingeben zu lassen, oder den Input per zusätzlichem Code zu validieren. Mithilfe des File Pickers können per Dialog auch Ordner oder Dateien im Dateisystem ausgewählt werden.

Sollten diese Features immer noch nicht reichen, gibt es auch noch die Möglichkeit eigene Views in VS Code anzuzeigen und damit die sogenannte „Workbench“ zu erweitern.

* + - 1. Themes
      2. Debugger
      3. Unit Tests für den Plugin Code
  1. IntelliJ IDEA
     1. Das Plugin Projekt
        1. Setup eines Plugin Projekts
        2. Aufbau eines Plugins
     2. Funktionalität

Ausführbarer Code (Actions)

Persistente Speicherung (Data Storage, Settings, Secrets)

Event Listener

Services

Extension Points

Code Completion

Benutzerdefinierte Sprachen

Themes

Debugger?

Unit Tests für den Plugin Code

* + 1. IntelliJ IDEA Flora Plugin

In der Plugin Dokumentation von JetBrains wird zu Beginn empfohlen sich noch einmal gründlich zu überlegen, ob man für die von einem gewünschte Funktionalität wirklich ein vollwertiges Plugin benötigt. Häufig kommt es nämlich vor dass nur bestimmte kleine Tasks innerhalb des IDEs automatisiert werden sollen. Hierfür schlägt JetBrains einige leichtgewichtige Alternativen vor. Eine nennenswerte Alternative ist das „Flora Plugin“ für das IntelliJ IDEA.

Flora kann über die Einstellungen des IntelliJ IDEA im Abschnitt „Plugins“ installiert werden.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Das Plugin sucht dann in den geöffneten Projektverzeichnissen nach ausführbaren JavaScript oder Kotlin Script „micro plugin“ Dateien. Diese müssen sich in einem Ordner namens „.plugins“ befinden und auf „.plugin.js“ oder „.plugin.kts“ enden.

Innerhalb diese Plugin Dateien kann über die Variable „ide“ auf die angebotene Schnittstelle zugegriffen werden. Diese erlaubt es unter anderem Actions, Keyboard Shortcuts, Services und ToolWindows zu erstellen.

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte Beschreibung

Flora Plugins bieten sich vor allem dann an, wenn eine projektspezifische Aufgabe automatisiert werden soll. Hier sind vor Allem die Leichtgewichtigkeit der Plugins und die Schnelle, mit der ein einfaches Plugin entwickelt werden kann, von großem Vorteil. Weiters spricht für diesen Anwendungsfall, dass der Plugin Code direkt im Projektordner abgelegt wird und somit auch in einem Version Control System wie Git mit abgelegt werden kann.

1. Codebeispiel: Recent Changes Plugin
2. Vergleich der Code-Metriken
3. Conclusion

Literaturverzeichnis

<https://code.visualstudio.com/api>

<https://github.com/microsoft/vscode-extension-samples>

<https://plugins.jetbrains.com/docs/intellij/developing-plugins.html>

<https://github.com/JetBrains/intellij-sdk-code-samples/>

<https://github.com/JetBrains/intellij-community/tree/master>

<https://github.com/google/diff-match-patch>

<https://medium.com/cacher-app/building-code-editor-plugins-a-comparison-83b5c21657fe>

<https://survey.stackoverflow.co/2023/>

<https://insights.stackoverflow.com/survey>