

Benutzerfreundliche MicroPython Entwicklung in VS- Code

Bachelorarbeit FS26

Version: 1.0

Autor: Silvan Bollhalder
E-Mail: silvan.bollhalder@ost.ch

Autor: Quentin Weber
E-Mail: quentin.weber@ost.ch

Dozent: Prof. Dr. Markus Stolze
Experte: Christoph Amrein
Praxispartner: Sebastian Romero
Gegenleser: Dr. Arno Wagner
Studiengang: B.Sc. Informatik

Erstellt am: 16. Februar 2026

Letzte Änderung am: 11. Juni 2026

Abstract

MicroPython ermöglicht die Entwicklung von Software für Mikrocontroller mit der Programmiersprache Python und erfreut sich insbesondere im Bildungsbereich sowie bei Prototyping-Projekten grosser Beliebtheit. Die Entwicklung und Verwaltung von MicroPython-Anwendungen ist jedoch häufig mit der Nutzung verschiedener Werkzeuge und Kommandozeilenprogramme verbunden, was die Benutzerfreundlichkeit und Effizienz des Entwicklungsprozesses erheblich beeinträchtigen kann. Insbesondere für unerfahrene Nutzer stellt die Vielzahl an erforderlichen Tools sowie die fehlende Integration in eine einheitliche Entwicklungsumgebung eine erhebliche Einstiegshürde dar.

Ziel dieser Bachelorarbeit ist die Entwicklung einer benutzerfreundlichen Extension für Visual Studio Code, die zentrale Funktionen für die MicroPython-Entwicklung in einer integrierten Entwicklungsumgebung bündelt und zugänglich macht. Die entwickelte Extension richtet sich insbesondere an Arduino-basierte MicroPython-Boards und bietet Funktionen zur Verwaltung von Dateien direkt auf dem Zielgerät, zur Installation und Verwaltung von Bibliotheken sowie zur Herstellung und Überwachung von Geräteverbindungen. Darüber hinaus werden Funktionen zur Verbesserung der allgemeinen Entwicklererfahrung bereitgestellt, darunter Codeunterstützung für Bibliotheken und boardspezifische Funktionen sowie eine klar strukturierte, übersichtliche Benutzeroberfläche.

Im Rahmen der Arbeit wurden zunächst die Anforderungen an eine moderne MicroPython-Entwicklungsumgebung systematisch analysiert und ein entsprechendes Konzept erarbeitet. Die Umsetzung erfolgte mit TypeScript und der von Visual-Studio-Code angebotenen API für Extension-Entwicklung. Die entwickelte Lösung integriert bestehende MicroPython-Werkzeuge und erweitert diese um eine grafische Benutzeroberfläche, wodurch wiederkehrende Aufgaben vereinfacht und die Bedienung intuitiver gestaltet werden. Die technische Umsetzung umfasste dabei anspruchsvolle Aspekte wie die gleichzeitige Verwaltung mehrerer verbundener Boards, die zuverlässige serielle Kommunikation über USB sowie die nahtlose Integration der Board-Ausgabe in die VS Code-Oberfläche. Besondere Herausforderungen entstanden dabei durch Einschränkungen der seriellen Kommunikation mit den Mikrocontrollern.

Die Ergebnisse zeigen, dass durch die Konsolidierung der wichtigsten MicroPython-Funktionen in Visual Studio Code die Komplexität des Entwicklungsprozesses spürbar reduziert und die Benutzerfreundlichkeit deutlich verbessert werden kann. Die entwickelte Extension bietet robuste Gerätekommunikation, skalierbare Board-Unterstützung und einen professionellen Entwicklungs- sowie Veröffentlichungsprozess über den Visual Studio Code Marketplace. Dabei wurde nicht nur auf einen niederschweligen Einstieg für Einsteiger geachtet, sondern es wurde bewusst so konzipiert, dass die Extension auch für erfahrene Entwickler eine vollwertige und produktive Arbeitsumgebung darstellt.

Inhalt

Abstract	2
Inhalt	3
1 Ausgangslage	5
1.1 Hintergrund	5
1.2 Problemstellung	5
2 Aufgabenstellung / Ziel der Arbeit	6
2.1 Zielsetzung	6
2.2 Teilziele	6
2.3 Erwarteter Nutzen	6
3 Analyse und Konzeption	7
3.1 Analyse bestehender Lösungen	7
3.2 Identifizierte Probleme & Stolpersteine für Einsteiger	10
3.3 Abgeleitete UX-Anforderungen	11
3.4 Konzeption des Extension-Paket	13
4 Anwendungsübersicht	15
4.1 Domänenmodell	15
5 Anforderungen	17
5.1 Funktionale Anforderungen	17
5.2 Nicht-funktionale Anforderungen (URPS)	20
5.3 Nicht-Anforderungen	22
5.4 Definition MVP	23
6 Architektur	24
6.1 Technologie	24
6.2 Libraries und Frameworks	27
6.3 Auflistung verwendeten 3rd Party Libraries	28
6.4 Lizenz der Extension: GNU AGPL-3.0	29
6.5 Projekt Ordner Struktur	30
6.6 Wartung / Weiterentwicklung; Guide und Tools	31
7 UI / UX	32
7.1 Aufbau der Sidebar	32
7.2 Verbindungsverwaltung	33
7.3 Basis-Aktionen mit dem Board	33
7.4 Workspace Dateibaum	34
7.5 Mount Workspace	35
7.6 Board Dateibaum	36
7.7 Library Verwaltung	36
8 Implementation	38
8.1 Überblick	38
8.2 Board Status	39
8.3 Ein Board verbinden	39
8.4 Library Installieren	40

8.5	Mount	41
8.6	Board Dateien öffnen	42
8.7	Script ausführen & REPL.....	42
8.8	Workspace-Filetree	42
9	Erreichte Ziele / Resultate.....	43
9.1	Funktionale Anforderungen	43
9.2	Nicht-funktionale Anforderungen	43
9.3	Zusammenfassung.....	44
9.4	Vergleich mit bestehenden Extensions.....	45
9.5	Ausblick und Erweiterungsmöglichkeiten.....	46
10	Projektmanagement	48
10.1	Projektplan-SOLL	48
10.2	Projektplan-IST	49
10.3	Zeiterfassung pro Person	49
10.4	Vergleich & Diskussion Zeitplan	50
10.5	Risikomanagement.....	52
11	Qualitätsmanagement.....	58
11.1	Regeln.....	58
11.2	GitLab Pipeline	59
11.3	Deployment	59
11.4	Testkonzept.....	60
11.5	Code Patterns	71
11.6	Code Metriken.....	74
12	KI-Reflexion	76
12.1	Positive Aspekte.....	76
12.2	Herausforderungen	77
12.3	Fazit	77
13	Literaturverzeichnis	78
14	Verzeichnisse	79
14.1	Abbildungen	79
14.2	Tabellen	80
14.3	Glossar.....	81
15	Anhang.....	85
15.1	Aufgabenstellung.....	85
15.2	Usability Test Durchführung 1	88
15.3	Usability Test Szenarien Durchführung 2	91
15.4	Usability Test Resultat 1. Durchführung	93
15.5	Usability Test Resultat 2. Durchführung	97
15.6	Code Metriken.....	101

1 Ausgangslage

MicroPython ermöglicht einen Einstieg in die Mikrocontroller-Programmierung, indem es eine vereinfachte Python-Implementierung direkt auf dem Gerät ausführt.

1.1 Hintergrund

Mit der in Abbildung 1 zu sehenden Applikation Arduino Lab for MicroPython existiert eine bewusst einfach gehaltene Entwicklungsumgebung, die insbesondere für Einsteiger gut geeignet ist. MicroPython eignet sich besonders für Lehr- und Prototyping-Zwecke, da es eine schnellere und einfachere Entwicklung ermöglicht als C/C++ [1]. Es setzt jedoch eine geeignete Toolchain voraus, die den Einstieg nicht unnötig erschwert. Allerdings fehlen dort moderne Editorfunktionen wie Autovervollständigung oder KI-gestützte Code-Assistenten, die den Entwicklungsalltag erheblich erleichtern würden.

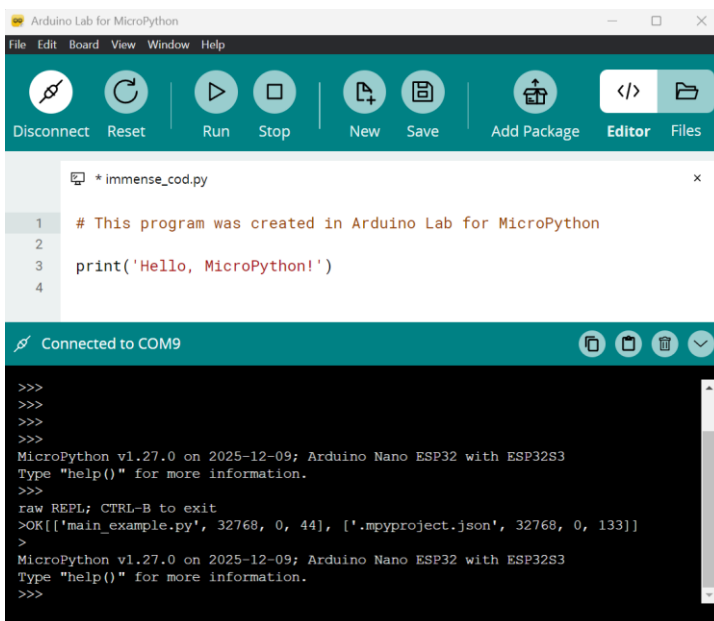


Abbildung 1 - Arduino Lab for MicroPython

Demgegenüber stehen bestehende VS-Code-Extensions für MicroPython, die zwar funktional umfangreicher sind, in der Praxis jedoch oft als komplex, fehleranfällig und wenig einsteigerfreundlich wahrgenommen werden. Es existiert damit eine Lücke zwischen der Einfachheit von Arduino Lab und der Mächtigkeit von VS-Code, die bisher keine bestehende Lösung überzeugend schliesst. Diese Problematik zeigt sich konkret im Modul Informatikseminar 1 des Studiengangs Digital Design an der OST, in welchem Studierende mit MicroPython und Mikrocontrollern in Berührung kommen und für das diese Extension primär entwickelt wird.

1.2 Problemstellung

Gerade für Einsteiger ohne Programmiererfahrung stellen Installation, Konfiguration und Bedienung bestehender VS-Code-Extensions häufig unnötige Hürden dar. Wer MicroPython lernen möchte, soll sich auf das Programmieren konzentrieren können – nicht auf das Einrichten von Entwicklungsumgebungen. Eine Lösung, die die intuitive Benutzerführung von Arduino Lab mit den modernen Editor- und KI-Funktionen von VS-Code verbindet, fehlt bisher.

2 Aufgabenstellung / Ziel der Arbeit

Die originale Aufgabenstellung befindet sich im Anhang.

2.1 Zielsetzung

Ziel dieser Bachelorarbeit ist die Konzeption und Umsetzung eines VS-Code-Extension-Pakets für MicroPython, das die klare und einfache Benutzerführung von Arduino Lab for MicroPython aufgreift und gleichzeitig die starken Editier- und KI-Funktionen von VS-Code nutzbar macht. Der Fokus liegt konsequent auf Einfachheit, Stabilität und guter Usability für Einsteiger. Um dies sicherzustellen, werden bestehende Extensions analysiert, sinnvolle Konzepte gezielt übernommen und die Lösung durch Usability-Tests mit Einsteigern iterativ überprüft und verbessert.

2.2 Teilziele

Die Arbeit gliedert sich in vier Teilbereiche. Zunächst werden bestehende VS-Code-Extensions für MicroPython analysiert, typische Stolpersteine für Einsteiger identifiziert und daraus UX- sowie Funktionsanforderungen abgeleitet. Auf dieser Basis wird ein reduziertes, stabiles Extension-Paket konzipiert und implementiert. Der Kernfunktionsumfang umfasst dabei das Hochladen, Ausführen, Bearbeiten und Löschen von Dateien auf dem Mikrocontroller, eine integrierte Konsole für Programmausgaben und Fehlermeldungen, Syntax-Highlighting für Python, Unterstützung von KI-Code-Assistenten wie GitHub Copilot sowie einen Soft-Reset des Boards. Je nach Fortschritt können zusätzliche Funktionen wie intelligente Autovervollständigung, Bibliotheksverwaltung oder die Erkennung nicht unterstützter Python-Bibliotheken umgesetzt werden. Abschliessend wird die Extension durch Usability-Tests mit Einsteigern evaluiert und mit bestehenden Lösungen verglichen.

2.3 Erwarteter Nutzen

Die Extension soll Einsteigern den Zugang zur MicroPython-Entwicklung in VS-Code deutlich erleichtern. Durch den Wegfall unnötiger Einrichtungsschritte und eine intuitive Benutzeroberfläche können sich Benutzer von Beginn an auf das eigentliche Programmieren konzentrieren. Gleichzeitig profitieren sie von den modernen Editorfunktionen von VS-Code, die Arduino Lab nicht bietet, insbesondere von Autovervollständigung und KI-gestützter Codeunterstützung.

3 Analyse und Konzeption

3.1 Analyse bestehender Lösungen

Für die Analyse wurden die meistgenutzten VS-Code-Extensions für MicroPython-Entwicklung aus dem Visual Studio Marketplace ausgewählt. Für die Tests wurde das Arduino Nano ESP32 verwendet, welches auch im Moodle Kurs Informatikseminar 1 verwendet wird. Die nachfolgende Tabelle gibt einen Überblick über die drei untersuchten Extensions:

Extension	Publisher	Installs	Marketplace-ID
RT-Thread MicroPython	RT-Thread	~62k	RT-Thread.rt-thread-micropython
MicroPython IDE for VSCode	Bao Phan	~85k	dphans.micropython-ide-vscode
Pymakr	Pycom	~215k	pycom.Pymakr

Tabelle 1 - Übersicht bestehende Extensions

3.1.1 RT-Thread MicroPython

Die Extension des chinesischen Herstellers RT-Thread bietet einen direkten Einstieg aus dem Marketplace ohne zusätzliche Installationsschritte. Die Konfiguration muss mit Auswahl des Ports und Angabe eines Projektpfades abgeschlossen werden. Das Ausführen von Python-Dateien auf dem Board gelingt schnell (auch über Shortcuts wie ALT+Q möglich), jedoch fehlt es an Informationen, welche Dateien sich aktuell auf dem Board befinden. Man ist ungenügend informiert, was den Zustand des Boards anbelangt. Die nach einer Projektsynchronisation angezeigte DEVICE FILES LIST spiegelt lediglich die synchronisierten lokalen Projektdaten wider und gibt keinen Aufschluss über den tatsächlichen Dateizustand des Boards.

Aktionen sind teils über Buttons in der unteren Toolbar, teils über das Rechtsklick-Kontextmenü erreichbar. Leider sind Aktionen nicht immer intuitiv oder haben Schreibfehler. Die Aktion «Download file/fodler to device» kann über das Rechtsklick Kontextmenü aufgerufen werden. Die Aktion macht aber nichts. Ansonsten erfüllt die Extension alle wichtigen Punkte für eine MicroPython Entwicklung, wie main.py Upload oder REPL-Konsole (Read Evaluate Print Loop), ausgenommen der Neustart Funktionalität. Es gibt weder Hard- noch Softreset.

Die Extension unterstützt laut Dokumentation zusätzlich Verbindungen über Netzwerk und USB, im Test wurde ausschliesslich die serielle Verbindung verwendet.

Zwar bewirbt RT-Thread MicroPython laut Dokumentation eine MicroPython-spezifische Autovervollständigung, diese setzt jedoch die Microsoft Python-Extension sowie ein korrekt im PATH registriertes Python voraus. In der durchgeführten Testumgebung war diese Funktionalität trotz erfüllter Voraussetzungen nicht verfügbar, weshalb die Bewertung auf Basis der tatsächlich erlebten Funktionalität erfolgte

Die Dokumentation ist grösstenteils auf Chinesisch verfasst und die englische Dokumentation, falls vorhanden, hat Fehler. Bemerkenswert ist jedoch, dass RT-Thread die beste Marketplace Bewertung hat von allen drei getesteten Extensions mit 4.5 Sternen.

3.1.2 MicroPython IDE for VSCode

Diese Extension ist mit 85k Downloads eine bekannte in der Community, ist jedoch die am schwierigsten einzurichtende der drei: Zusätzliche Python-Pakete müssen manuell via pip installiert werden, konkret rshell sowie adafruit-ampy, was für Einsteiger eine unnötige Hürde darstellt.

Erschwerend kommt hinzu, dass der serielle Port (z. B. COM9) manuell in der Konfiguration eingetragen werden muss, da keine automatische Port-Erkennung stattfindet. Gerade für Anfänger, die mit Begriffen wie COM-Port oder Gerätepfaden nicht vertraut sind, ist dies eine weitere Barriere.

Die Bedienung erfolgt über Commands, z.B. «>MicroPython: Run». Ganz unten rechts gibt es für den Command Run einen kleinen Button, welcher aber leicht zu übersehen ist.

Beim ersten Run Befehl zeigten sich weitere Mängel. Zuerst wurde der Port nicht zuverlässig erkannt und anschliessend wurde die ampy Library nicht gefunden. Nach mehreren Versuchen und Anpassungen des Python Environments wurde aufgegeben. Somit konnte die REPL-Funktionalität und der Datei-Upload leider nicht getestet werden.

Mit einer Marketplace-Bewertung von 2/5 Sternen ist sie die schlechteste der untersuchten Extensions.

3.1.3 Pymakr

Pymakr von Pycom verfügt über die grösste Community und wird auf GitHub und Reddit am häufigsten empfohlen. Das Look-and-Feel ist insgesamt am ausgereiftesten, die Buttons sind jedoch ein bisschen klein. Die Struktur des Sidepanels weist eine Schwäche auf: Das verbundene Board wird sowohl unter «Projects» als auch unter «Devices» mit denselben Steuerbefehlen angezeigt, was zu Verwirrung führt.

Der grösste funktionale Mangel ist das Fehlen einer Direktausführung einzelner Dateien. Man muss stets eine main.py erstellen, diese mit dem Board synchronisieren und danach einen Hard-Reset machen oder einen manuellen REPL-Befehl ausführen. Dieses Vorgehen ist für Anfänger wenig intuitiv und fehleranfällig. Alternativ könnte man eine REPL-Konsole öffnen und den geschriebenen Code hineinkopieren, was jedoch auch wenig anfangersfreundlich ist. Ansonsten hat Pymakr die gewünschten Funktionalitäten und hält, was es verspricht. Die Bedienung der Extension geschieht über Buttons und nicht über Commands, was besser ist für Einsteiger.

Die Bewertung des Marketplace scheint mit 2.5 Sternen nicht ganz fair zu sein, jedoch lag das an einem Bug von vor 3 Jahren, welcher sich viele negative Rezensionen einhandelte. Inzwischen wurde dieser behoben.

3.1.4 Bewertungsmatrix

Die nachfolgende Bewertungsmatrix basiert auf den praktischen Testerfahrungen der Autoren und spiegelt die subjektiv erlebte Benutzerfreundlichkeit wider, nicht eine vollständige Produktbeschreibung anhand der offiziellen Dokumentation.

Kriterium	RT-Thread MicroPython	MicroPython IDE VSCode	Pymakr
Installation & Einrichtung			
Einfachheit der Installation	4 / 5	1 / 5	4 / 5 *
Konfigurationsaufwand	3 / 5	2 / 5	3 / 5
Kernfunktionalität			
Datei-Upload & Ausführen	4 / 5	1 / 5	3 / 5
REPL-Konsole	4 / 5	1 / 5	4 / 5
Board Reset	1 / 5	1 / 5	5 / 5
Benutzerfreundlichkeit			
Konsistenz der UI-Elemente	3 / 5	2 / 5	3 / 5
Einsteigerfreundlichkeit	3 / 5	1 / 5	3 / 5
Stabilität & Wartung			
Stabilität im Betrieb	3 / 5	1 / 5	4 / 5
Community & Aktualität	3 / 5	1 / 5	4 / 5
Marketplace-Bewertung	4.5 / 5	2 / 5	2.5 / 5
Ø Durchschnitt	3.25 / 5	1.3 / 5	3.55 / 5

Tabelle 2 - Bewertungsmatrix bestehende Extensions

	Gut (4-5 / 5)
	Mittel (3 / 5)
	Schwach (1-2 / 5)

Tabelle 3 - Legende Bewertungsmatrix

*Im Nachhinein ist aufgefallen, dass für Pymakr Node.js installiert sein muss. Dies war auf dem Testrechner schon installiert und ist daher nicht aufgefallen. Darum ist die Bewertung in der Bewertungsmatrix 4/5 zu hoch und wäre zwischen 2-3 angebrachter.

3.2 Identifizierte Probleme & Stolpersteine für Einsteiger

Aus der Analyse der Extensions und der Bewertungsmatrix leiten sich allgemeine Stolpersteine und Probleme für Einsteiger ab.

3.2.1 Ersteinrichtung

Manche Extensions setzen voraus, dass Benutzer manuell Python-Pakete installieren oder serielle Port-Bezeichnungen wie «COM9» kennen und selbst eintragen muss. Für Einsteiger ohne Erfahrung mit der Kommandozeile oder Gerätepfaden stellt dies bereits eine Hürde dar, bevor die eigentliche Entwicklung beginnt. Diese Problematik ist auch im akademischen Umfeld bekannt: Ioannou et al. [2] berichten, dass Studierende beim Einstieg in MicroPython vor allem an der Einrichtung der Entwicklungsumgebung scheitern, nicht am Programmieren selbst.

3.2.2 Benutzeroberfläche

Buttons sind teilweise sehr klein, schwer zu finden oder über mehrere Orte verteilt, Toolbar, Kontextmenü und Seitenleiste, ohne erkennbare Logik. In einzelnen Fällen führen Aktionen schlicht zu keinem Ergebnis, ohne dass eine Fehlermeldung erscheint. Einsteiger wissen dann nicht, ob sie etwas falsch gemacht haben oder ob die Extension nicht funktioniert.

3.2.3 Board-Zustand

Es ist oft unklar, welche Dateien sich auf dem Gerät befinden, ob ein Upload erfolgreich war oder was gerade ausgeführt wird. Diese Intransparenz verunsichert besonders Anfänger.

3.2.4 Ausführen einzelner Dateien

Es ist nicht immer möglich Dateien direkt auszuführen. Stattdessen muss man umständlich eine main.py anlegen, synchronisieren und das Board manuell neu starten. Ein Workflow, der für schnelles Ausprobieren und Lernen ungeeignet ist.

3.2.5 Code Completion

Die Code Completion war leider in keiner der drei Extensions für MicroPython-spezifischen Code vorhanden. Die Stärken von Visual Studio Code sind vorhanden, also Intellisense, Copilot etc., jedoch ist es nur für Python spezifischen Code, wenn man z.B. pylance installiert hat. Für Einsteiger wäre es besser, wenn Unterstützung für MicroPython vorhanden ist.

3.2.6 Support

Alle drei untersuchten Extensions haben seit 2022 keine Updates mehr erhalten, was im Hinblick auf den langfristigen Einsatz in einem Lehrkontext ein Risikofaktor darstellt.

3.3 Abgeleitete UX-Anforderungen

Auf Basis der identifizierten Probleme wurde ein eigenes Interface-Konzept entwickelt. Ziel des Designs ist es, Einsteigern eine möglichst klare, visuell verständliche und geführte Arbeitsumgebung bereitzustellen. Dabei steht nicht die maximale Integration in das native VS-Code-Interface im Vordergrund, sondern eine möglichst intuitive Bedienbarkeit.

Das entwickelte UX-Design orientiert sich an einem Dashboard-Ansatz innerhalb der VS-Code-Seitenleiste. Alle zentralen Funktionen werden in klar abgegrenzten Modulen dargestellt, die logisch nach Aufgabenbereichen strukturiert sind. Dadurch wird verhindert, dass Funktionen über verschiedene Menüs, Kontextmenüs oder Toolbars verteilt sind.

3.3.1 Struktur der Benutzeroberfläche

Die Oberfläche ist in mehrere Funktionsbereiche unterteilt, die jeweils eine klar definierte Aufgabe erfüllen.

Connection

In Abbildung 2 ist das Konzept des Verbindungsmanagements dargestellt. Der obere Bereich zeigt den aktuellen Verbindungsstatus zum Board. Es wird angezeigt, welches Board verbunden ist sowie über welchen seriellen Port die Verbindung läuft. Ein farbiger Statusindikator signalisiert, ob das Board bereit ist oder nicht. Zusätzlich kann die Verbindung über einen Refresh-Button neu erkannt werden.

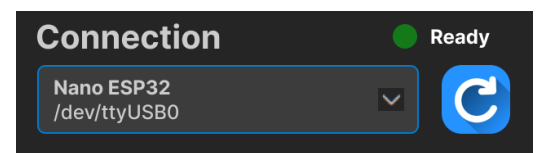


Abbildung 2 - UI Konzept Connection

Dieser Bereich adressiert direkt das Problem der intransparenten Board-Verbindung, das bei anderen Extensions häufig auftritt.

Workspace Files

Im Abschnitt Workspace (Abbildung 3) werden die Python-Dateien angezeigt, die sich im aktuellen Workspace befinden. Benutzer können hier eine Datei auswählen und anschliessend direkt auf das Board hochladen oder als neue main.py speichern.

Dies ermöglicht einen klaren Überblick über den lokalen Code und vereinfacht den Transfer auf das Board.

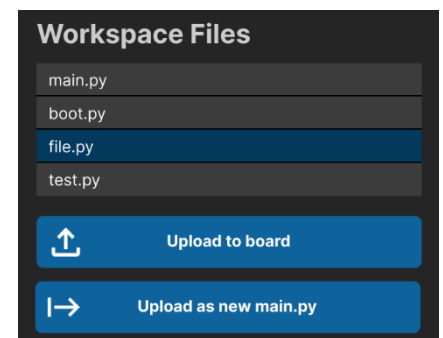


Abbildung 3 - UI Konzept Workspace

Board Actions

Der in Abbildung 4 abgebildete Bereich enthält die wichtigsten Aktionen für die Arbeit mit dem Board:

- Ausführen der aktuellen Datei
- Öffnen der REPL-Konsole
- Stoppen eines laufenden Skripts
- Soft Reset des Boards

Die Funktionen sind als grosse, klar beschriftete Buttons dargestellt. Damit wird das Problem sehr kleiner oder versteckter Bedienelemente vermieden, welches in anderen Extensions festgestellt wurde.

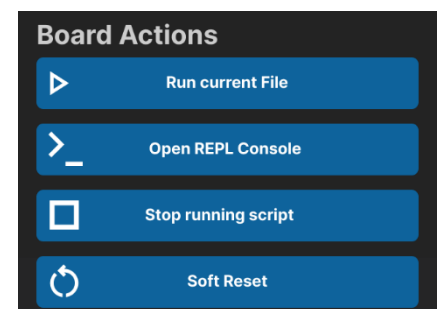


Abbildung 4 - UI Konzept Board Actions

Board Files

Ein wesentliches Merkmal des Designs ist die Visualisierung des Dateisystems auf dem Board, welches in Abbildung 5 dargestellt ist. In diesem Bereich werden alle Dateien angezeigt, die sich aktuell auf dem Gerät befinden.

Benutzer können Dateien herunterladen oder löschen. Dadurch entsteht eine hohe Transparenz darüber:

- welche Dateien auf dem Board existieren
- ob ein Upload erfolgreich war
- welche Datei aktuell verwendet wird

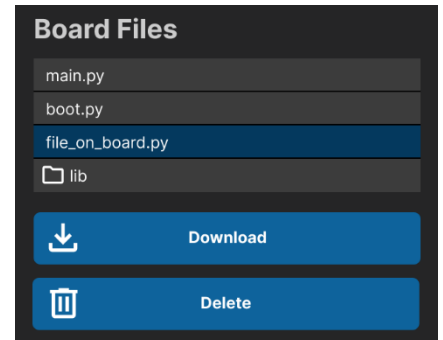


Abbildung 5 - UI Konzept Board Files

Libraries

Das Konzept in Abbildung 6 zeigt verfügbare MicroPython-Bibliotheken. Benutzer können nach Libraries suchen, Informationen anzeigen und diese direkt auf das Board installieren.

Darunter befindet sich eine Liste der bereits installierten Libraries, die bei Bedarf wieder deinstalliert werden können.

Diese Funktion reduziert die Notwendigkeit, Bibliotheken manuell über die Kommandozeile zu installieren.

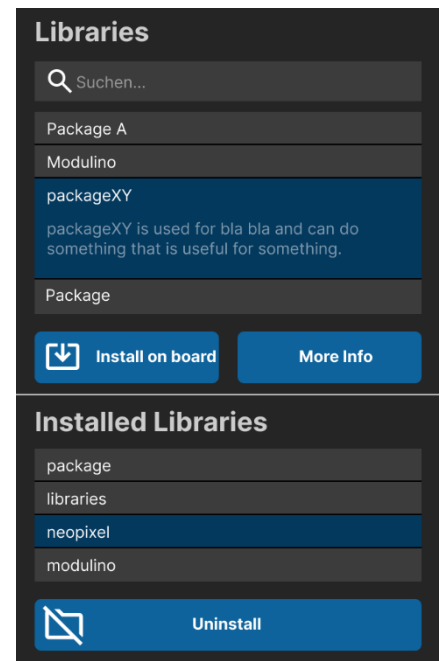


Abbildung 6 - UI Konzept Libraries

Editor Code Unterstützung

Der Abschnitt in Abbildung 7 konzentriert sich auf die Unterstützung im Editor. Hier können MicroPython Stubs für verschiedene Boards installiert oder überprüft werden. Diese Stub-Dateien ermöglichen eine bessere Code Completion und IntelliSense-Unterstützung innerhalb von VS Code.

Durch die Auswahl eines Boards können spezifische Stub-Dateien verwendet werden, wodurch ein Multiboard-Setup unterstützt wird.

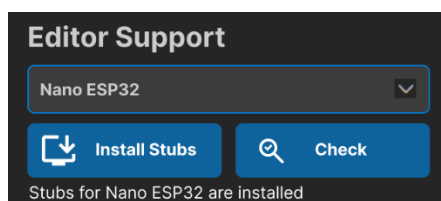


Abbildung 7 - UI Konzept Code Support

3.4 Konzeption des Extension-Paket

Aus den identifizierten Schwachstellen und den UX-Anforderungen lassen sich Leitprinzipien für ein Konzept der Extension ableiten. Im Vordergrund stehen dabei Einfachheit, Stabilität und intuitive Benutzerführung für Einsteiger.

3.4.1 Einfache Nutzerführung und UI

Arduino Lab for MicroPython soll als Referenz für die einfache Nutzerführung und das übersichtliche UI dienen. Gleichzeitig sollen die Editor- und KI-Funktionen des Visual Studio Code enthalten bleiben. Alle zentralen Funktionen sollen über eindeutig beschriftete, gut sichtbare Schaltflächen erreichbar sein. Das UI soll intuitiv und selbsterklärend sein. Da Buttons nativ in VS-Code sehr klein sind, wird das native Look-And-Feel in den Hintergrund gerückt, um maximale Intuition zu gewährleisten.

3.4.2 Einfache Installation und Konfiguration

Die Extension soll ohne manuelle Vorbereitungsschritte funktionieren. Zusätzliche Installationen sollen automatisiert von der Extension übernommen werden. Auch die Konfiguration soll einfach sein. Keine manuellen Eingaben von Port. Wenn möglich soll auch die Erstellung eines Projekts, wie es in anderen Extensions der Fall ist, automatisiert werden.

3.4.3 Transparenz des Boards

Unter Transparenz wird die sichtbare Darstellung des aktuellen Zustands des Mikrocontrollers verstanden. Dazu gehören Informationen wie der Verbindungsstatus, die Verfügbarkeit des Serial Ports oder ob aktuell ein Programm ausgeführt wird. Diese Informationen erleichtern die Entwicklung und Fehlersuche erheblich.

Wichtig ist zudem die Transparenz des Dateisystems. Die Extension soll anzeigen, welche Dateien auf dem Board gespeichert sind. Diese Funktion fehlt bei vielen bestehenden Extensions und stellt daher ein wichtiges Unterscheidungsmerkmal dar.

3.4.4 Dateien direkt ausführen

Ein zentrales Ziel ist es, den Workflow für Einsteiger so kurz wie möglich zu halten. Einzelne Dateien sollen direkt ausgeführt werden können, ohne dass erst eine `main.py` angelegt und ein Reset ausgelöst werden muss. Das senkt die Hemmschwelle zum Experimentieren und fördert den Lernprozess. Dies kann erreicht werden, indem die aktuell ausgewählte Datei per REPL-Konsole direkt an das Board gestreamt wird. Dieser Ansatz wird unter anderem von Arduino Lab for MicroPython verwendet. Dabei ist jedoch zu beachten, dass das direkte Streaming nicht für alle Anwendungsfälle geeignet ist. Grössere Projekte mit mehreren Dateien können Einschränkungen aufweisen, da lokal bestehende Imports zunächst auf das Board übertragen werden müssen, damit die direkte Ausführung funktioniert. Für Lern- und Testzwecke bietet dieser Ansatz dennoch eine einfache und effiziente Möglichkeit, Code auszuführen.

3.4.5 Code-Unterstützung durch MicroPython-Stubs und IntelliSense

Damit die Extension MicroPython-spezifischen Code erkennen, vervollständigen und Vorschläge anzeigen kann, werden sogenannte Stubs verwendet. Dabei handelt es sich um Python-Interface-Dateien, die ausschliesslich zur Unterstützung der Entwicklungsumgebung dienen und keinen ausführbaren Code enthalten. Sie liefern stattdessen Typinformationen, Funktionssignaturen und Modulstrukturen, die von IntelliSense genutzt werden, um Codevervollständigung und Dokumentation im Editor zu ermöglichen.

Im Kontext von MicroPython sind diese Stubs besonders wichtig, da sie die API des jeweiligen Boards und der darauf verfügbaren Bibliotheken abbilden. Je nach verbundenem Mikrocontroller können unterschiedliche Stubs geladen werden, da sich die verfügbaren Module und Funktionen zwischen Boards unterscheiden können. Dadurch entsteht ein sogenanntes Multi-Board-Setup, bei dem die Code-Unterstützung dynamisch an die aktuell verwendete Hardware angepasst wird.

Für jede board-spezifische Bibliothek werden entsprechende Stubs benötigt, damit die Entwicklungsumgebung die verfügbaren Funktionen korrekt erkennen und Vorschläge passend zum jeweiligen Gerät liefern kann.

Auf den Einsatz eines eigenen Language Servers wird bewusst verzichtet, da dessen Entwicklung sehr komplex ist und den Rahmen dieser Arbeit deutlich überschreiten würde. Stattdessen wird auf die vorhandenen Mechanismen von VS Code IntelliSense in Kombination mit Stubs gesetzt, um eine einfache, aber effektive Code-Unterstützung zu gewährleisten.

4 Anwendungsübersicht

4.1 Domänenmodell

Das Domänenmodell beschreibt die Kernstruktur einer VS Code Extension zur Verwaltung von MicroPython-Boards. Es modelliert die Verbindung zu physischen Mikrocontrollern, deren Dateisystem, Status und installierten Libraries.

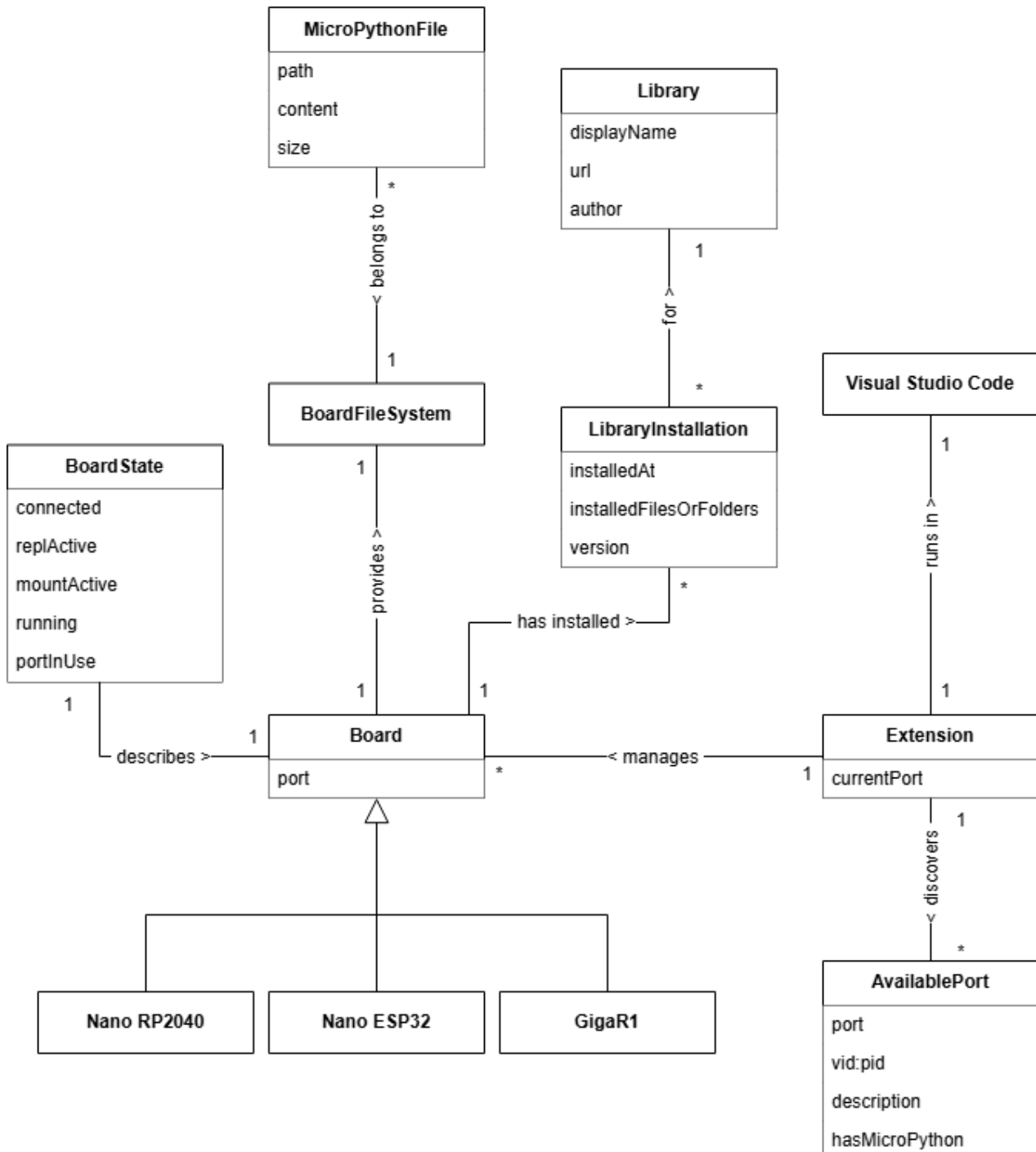


Abbildung 8 - Domänenmodell

4.1.1 Board

Board repräsentiert einen physischen Mikrocontroller, der über die Extension verwaltet wird. Jedes Board besitzt einen Serial Port, über den die Kommunikation zwischen Extension und Board erfolgt. Die unterstützten Arduino-Boards Arduino Nano RP2040 Connect, Arduino Nano ESP32 und Arduino GIGA R1 WiFi sind als Spezialisierungen von Board modelliert.

4.1.2 Extension

Die Extension übernimmt die Verbindungslogik und Verwaltung der Boards. Das Attribut `currentPort` repräsentiert den aktuell ausgewählten Port, mit welchem der Nutzer interagiert. Zusätzlich erkennt die Extension verfügbare Schnittstellen über `AvailablePort`, damit dem Nutzer mögliche Verbindungen angezeigt werden können. Dabei werden Informationen wie Port, VID:PID, Beschreibung und die Erkennung von MicroPython bereitgestellt.

4.1.3 Visual Studio Code

Visual Studio Code repräsentiert die Entwicklungsumgebung, in welcher die Extension ausgeführt wird. Die Entwicklungsumgebung stellt die für die Extension benötigten Schnittstellen und Funktionalitäten bereit und ermöglicht dadurch die Interaktion zwischen Nutzer, Extension und verbundenem Board.

4.1.4 Board Dateisystem

`BoardFileSystem` repräsentiert das Dateisystem eines Boards und dient als Verwaltungseinheit für sämtliche darauf gespeicherten Dateien. Eine einzelne Datei wird durch `MicroPythonFile` beschrieben. Die Attribute `path` und `size` beschreiben die Dateieigenschaften, während `content` den Inhalt der Datei enthält.

4.1.5 Board Status

`BoardState` kapselt den aktuellen Zustand eines Boards aus Sicht der Extension. Der Zustand beschreibt ausschliesslich Verbindungen und Aktivitäten, die durch die Extension selbst erkannt oder verwaltet werden. Falls ein Port beispielsweise durch eine externe Anwendung verwendet wird, kann dies nicht zuverlässig erkannt werden. Die Attribute `connected`, `replActive`, `mountActive`, `running` und `portInUse` definieren den aktuellen Zustand des Boards. Abhängig davon werden bestimmte Funktionen oder Interaktionen in der Benutzeroberfläche aktiviert, deaktiviert oder eingeschränkt dargestellt.

4.1.6 Libraries

`LibraryInstallation` beschreibt die Installation einer bestimmten Library auf einem Board. Neben einem Verweis auf die installierte Library werden Informationen wie Installationszeitpunkt, installierte Dateien oder Verzeichnisse sowie die verwendete Version gespeichert. Libraries sind von der Community entwickelte und veröffentlichte Softwarebibliotheken, welche die Entwicklung mit MicroPython vereinfachen. Sie ermöglichen beispielsweise die Kommunikation mit Sensoren oder anderen angeschlossenen Hardware-Komponenten. Die Klasse `Library` enthält dazu die Attribute `displayName`, `url` und `author`.

5 Anforderungen

Die Anforderungen werden nach dem FURPS-Modell strukturiert, das erstmals von Grady und Caswell beschrieben [3, p. 159] und später durch Rational Software (später IBM) zu FURPS+ erweitert wurde [4]. Die Anforderungen stellen sich aus den funktionalen Anforderungen und den nicht-funktionalen Anforderungen, welche in die Kategorien Usability, Reliability, Performance, Supportability und Compatibility unterteilt werden, zusammen. Jede Anforderung wird jeweils im Kapitel Qualitätsmanagement mit E2E-Tests überprüft. Zu jeder Anforderung gibt es einen zugehörigen E2E-Test.

5.1 Funktionale Anforderungen

5.1.1 Verbindungsverwaltung

Tabelle 4 - FA Verbindungsverwaltung

ID	E2E-Test	Beschreibung	Priorität
FA-V1	E2E-FA-V1	Das System soll erkennen, ob ein MicroPython-fähiges Gerät über eine serielle Schnittstelle angeschlossen ist.	1
FA-V2	E2E-FA-V2	Das System soll eine Verbindung zu einem erkannten Gerät herstellen können.	1
FA-V3	E2E-FA-V3	Das System soll eine bestehende Verbindung zum Gerät trennen können.	1
FA-V4	E2E-FA-V4	Das System soll bei Verbindungsabbruch den Benutzer informieren.	2

5.1.2 Dateiverwaltung

Tabelle 5 - FA Dateiverwaltung

ID	E2E-Test	Beschreibung	Priorität
FA-D1	E2E-FA-D1	Das System soll Dateien vom lokalen Rechner auf den Mikrocontroller übertragen können.	1
FA-D2	E2E-FA-D2	Das System soll Dateien vom Mikrocontroller auf den lokalen Rechner herunterladen können.	1
FA-D3	E2E-FA-D3	Das System soll Dateien auf dem Mikrocontroller löschen können.	1
FA-D4	E2E-FA-D4	Das System soll den Inhalt des Dateisystems auf dem Mikrocontroller anzeigen.	1
FA-D5	E2E-FA-D5	Das System soll einzelne Dateien direkt auf dem Mikrocontroller ausführen können.	1
FA-D6	E2E-FA-D6	Das System soll eine bestehende Datei auf dem Mikrocontroller als main.py festlegen können, sodass sie beim nächsten Board-Start automatisch ausgeführt wird.	2
FA-D7	E2E-FA-D7	Das System soll die Nutzung von mpremote mount unterstützen, sodass Dateien aus dem lokalen Workspace direkt vom Board verwendet werden können, ohne auf das Board kopiert zu werden.	1

5.1.3 Integrierte Konsole

Tabelle 6 - FA Integriere Konsole

ID	E2E-Test	Beschreibung	Priorität
FA-I1	E2E-FA-I1	Das System soll eine integrierte Konsole bereitstellen, die Programmausgaben des Mikrocontrollers anzeigt.	1
FA-I2	E2E-FA-I2	Das System soll Fehlermeldungen des Mikrocontrollers in der Konsole anzeigen.	1
FA-I3	E2E-FA-I3	Das System soll Benutzereingaben über die Konsole an den Mikrocontroller übermitteln können.	1
FA-I4	E2E-FA-I4	Es soll möglich sein zwei verschiedene Boards anzuhängen und gleichzeitig zwei Konsolen zu öffnen, um Verbindung zwischen den Boards testen zu können.	3

5.1.4 Reset

Tabelle 7 - FA Reset

ID	E2E-Test	Beschreibung	Priorität
FA-R1	E2E-FA-R1	Das System soll einen Soft-Reset des angeschlossenen Mikrocontrollers auslösen können.	1

5.1.5 Editor-Integration

Tabelle 8 - FA Editor-Integration

ID	E2E-Test	Beschreibung	Priorität
FA-E1	E2E-FA-E1	Das System soll Syntax-Highlighting für Python-Dateien bereitstellen.	1
FA-E2	E2E-FA-E2	Das System soll mit KI-Code-Assistenten wie GitHub Copilot kompatibel sein.	1
FA-E3	E2E-FA-E3	Das System soll MicroPython-spezifische Autovervollständigung für gängige Bibliotheken (z. B. machine) bereitstellen.	1
FA-E4	E2E-FA-E4	Das System soll den Benutzer auf die Verwendung von nicht unterstützten Python-Standardbibliotheken hinweisen.	2
FA-E5	E2E-FA-E5	Das System soll nach der Installation einer Bibliothek auf dem Board diese erkennen und direkt Autovervollständigung bieten.	1

5.1.6 Bibliotheksverwaltung

Tabelle 9 - FA Bibliotheksverwaltung

ID	E2E-Test	Beschreibung	Priorität
FA-B1	E2E-FA-B1	Das System soll die Installation von MicroPython-Bibliotheken direkt aus VS-Code heraus ermöglichen.	1

5.2 Nicht-funktionale Anforderungen (URPS)

5.2.1 Usability

Tabelle 10 - NFA Usability

ID	E2E-Test	Beschreibung
NFA-U1	E2E-U1	Die Extension soll ohne Vorkenntnisse in VS-Code-Extensions von Einsteigern installierbar sein.
NFA-U2	E2E-U2	Alle zentralen Funktionen sollen über klar beschriftete Schaltflächen oder Menüeinträge erreichbar sein, ohne dass die Kommandozeile verwendet werden muss.
NFA-U3	E2E-U3	Der aktuelle Verbindungsstatus zum Gerät soll jederzeit sichtbar in der Extension angezeigt werden.

5.2.2 Reliability

Tabelle 11 - NFA Reliability

ID	E2E-Test	Beschreibung
NFA-R1	E2E-R1	Die Extension soll bei einem Verbindungsabbruch zum Board nicht abstürzen, sondern einen stabilen Zustand beibehalten.
NFA-R2	E2E-R2	Bei einer Verbindungsunterbrechung während einer laufenden Übertragung soll die Extension den Vorgang abbrechen, den seriellen Port freigeben und den Nutzer mit einer Fehlermeldung über den Abbruch informieren.
NFA-R3	E2E-R3	Die Extension soll nach einem Neustart von VS Code automatisch in einen einsatzbereiten Zustand zurückkehren, ohne dass eine manuelle Konfiguration erforderlich ist. Falls ein Board angeschlossen ist, soll dieses erkannt und dem Nutzer zur Verbindung angeboten werden.

5.2.3 Performance

Tabelle 12 - NFA Performance

ID	E2E-Test	Beschreibung
NFA-P1	E2E-P1	Das Hochladen einer einzelnen kleinen Datei (< 10 kB) soll in unter 5 Sekunden abgeschlossen sein.
NFA-P2	E2E-P2	Die Verbindungsherstellung zu einem erkannten Gerät soll in unter 3 Sekunden erfolgen.
NFA-P3	E2E-P3	Ausgaben des Mikrocontrollers in der Konsole sollen ohne wahrnehmbare Verzögerung (< 200 ms) angezeigt werden.

5.2.4 Supportability

Tabelle 13 - NFA Supportability

ID	E2E-Test	Beschreibung
NFA-S1	E2E-S1	Die Extension soll ein README.md sowie eine MAINTAINER.md enthalten, welche zukünftigen Maintainern dabei helfen, die Extension zu warten und weiterzuentwickeln.

5.2.5 Compatibility

Tabelle 14 - NFA Compatibility

ID	E2E-Test	Beschreibung
NFA-C1	E2E-C1	Die Extension soll mit den unterstützten Boards (Nano ESP32, Nano RP2040 Connect, GIGA R1 WIFI) kompatibel sein.
NFA-C2	E2E-C2	Die Extension soll parallel zu anderen VS-Code-Extensions (z. B. GitHub Copilot, Pylance) ohne Konflikte betreibbar sein.
NFA-C3	E2E-C3	Die Extension soll mit der aktuellen stabilen MicroPython-Firmware der unterstützten Boards kompatibel sein.
NFA-C4	E2E-C4	Die Extension soll auf Windows und macOS lauffähig sein.

5.3 Nicht-Anforderungen

Die folgenden Punkte sind explizit nicht Bestandteil dieser Arbeit. Sie dienen zur Abgrenzung und verhindern eine unkontrollierte Erweiterung des Projektumfangs.

Tabelle 15 - Nicht Anforderungen

ID	Beschreibung
NA-1	Das System ist keine vollständige IDE. Es ersetzt nicht den VS-Code-Kern oder andere Editorfunktionen.
NA-2	Das System unterstützt nicht alle existierenden MicroPython-fähigen Boards; der Fokus liegt auf einer definierten Auswahl: Nano ESP32, Nano RP2040 Connect, GIGA R1 WIFI. Diese wurden getestet und verifiziert. Für andere Boards wie zum Beispiel das Raspberry PI Pico kann keine Funktionsgarantie gegeben werden, da diese nicht Teil der Evaluation waren.
NA-3	Das System optimiert nicht die MicroPython-Firmware selbst und nimmt keine Änderungen daran vor.

5.4 Definition MVP

Zur Umsetzung der Extension wird ein iteratives Vorgehen gewählt. Dabei wird ein Minimal Viable Product definiert, welches sich an den Prioritäten orientiert, welche bei den funktionalen Anforderungen verteilt wurden. Danach können Zusatzfunktionalitäten schrittweise hinzugefügt werden.

5.4.1 Stufe 1 - Minimum Viable Product (MVP)

Das MVP umfasst alle Anforderungen mit Priorität 1 und deckt damit die zentralen Nutzungsszenarien ab, die für den produktiven Einsatz zwingend erforderlich sind: vollständige Verbindungsverwaltung, vollständige Dateiverwaltung (Upload, Download, Löschen, Anzeige des Dateisystems sowie Ausführen einzelner Dateien), integrierte Konsole inklusive Benutzereingaben, Soft-Reset, Syntax-Highlighting und Copilot-Kompatibilität, MicroPython-spezifische Autovervollständigung samt Erkennung neu installierter Bibliotheken mit direkter Bibliotheksinstallation aus VS Code heraus, sowie die Einbindung des Board-Dateisystems via mpremote zur direkten Bearbeitung von Board-Dateien in VS Code. Sobald das MVP stabil läuft, bildet es die Grundlage für die geplanten Usability-Tests.

5.4.2 Stufe 2 - Erweiterungen

Nach Abschluss des MVP folgen die Priorität-2-Anforderungen: Benachrichtigung bei Verbindungsabbruch, Festlegen einer Datei als main.py sowie die Erkennung nicht unterstützter Python-Standardbibliotheken.

5.4.3 Stufe 3 - Optionale Features

Abschliessend wird, sofern Zeit und Umfang es erlauben, die Priorität-3-Anforderung umgesetzt: der parallele Betrieb zweier Boards mit je einer eigenen Konsole.

5.4.4 Vorzug von Optionalen Features

Im Verlauf der Entwicklung wurde das ursprüngliche Stufenmodell angepasst. Da die initiale Evaluation gezeigt hat, dass viele ursprünglich als optional eingestuft Funktionen technisch gut umsetzbar sind und die Usability-Tests bewusst erst zu einem späteren Zeitpunkt geplant wurden, wurden zahlreiche Priorität-2- und Priorität-3-Anforderungen in das MVP vorgezogen. Ein fortgeschrittener Funktionsstand zum Zeitpunkt der Usability-Tests ermöglicht aussagekräftigeres Feedback der Testpersonen.

6 Architektur

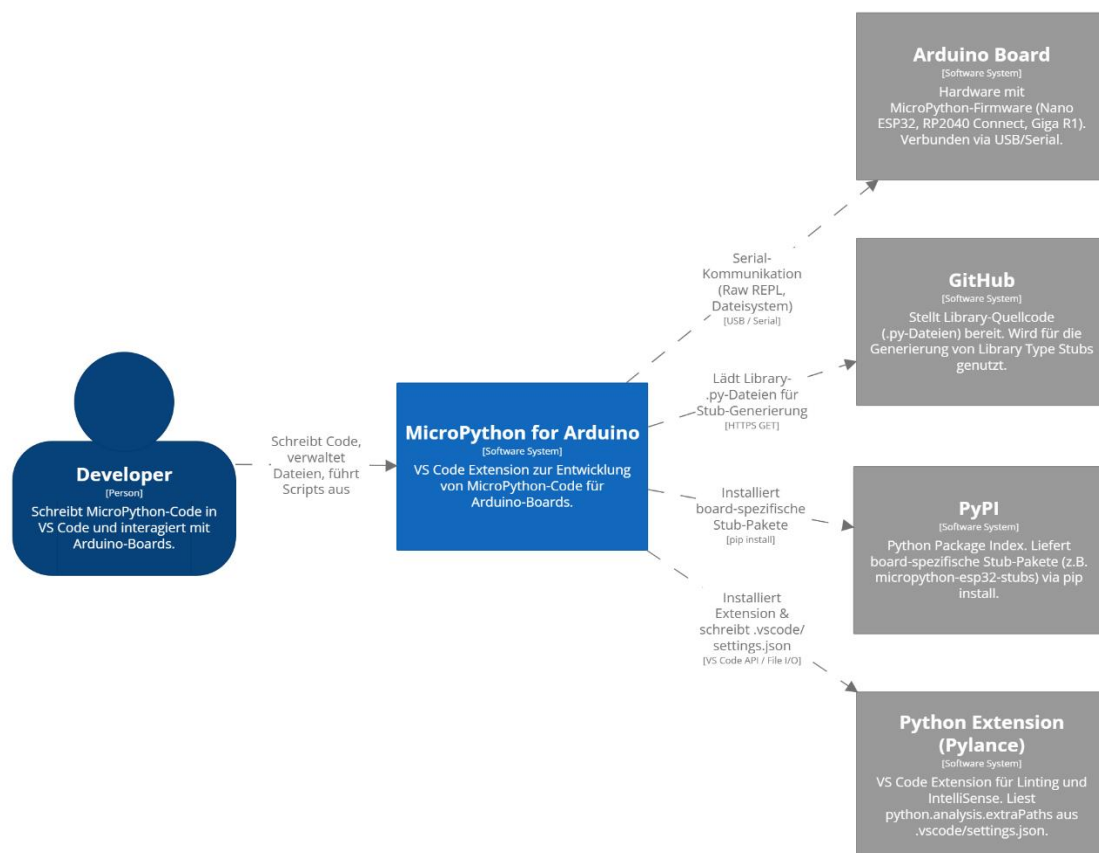
6.1 Technologie

Eine Visual Studio Code Extension wird typischerweise mit Typescript/Javascript und Node.js geschrieben. Man könnte Sprachen wie Rust, Go, C# oder Python verwenden, wenn man einen eigenen Language Server macht. Da für diese Arbeit kein eigener Language Server erstellt wird, ist die Wahl der Technologie somit klar bei Typescript/Javascript und Node.js.

6.1.1 C4 Diagramm

Um die Softwarearchitektur der entwickelten Anwendung strukturiert darzustellen, wurde das C4-Modell angewendet [5]. Dieses Modell ermöglicht eine hierarchische Visualisierung der Systemarchitektur in vier Ebenen (Context, Container, Component, Code), wobei für die Dokumentation die Ebenen L1 bis L3 relevant sind.

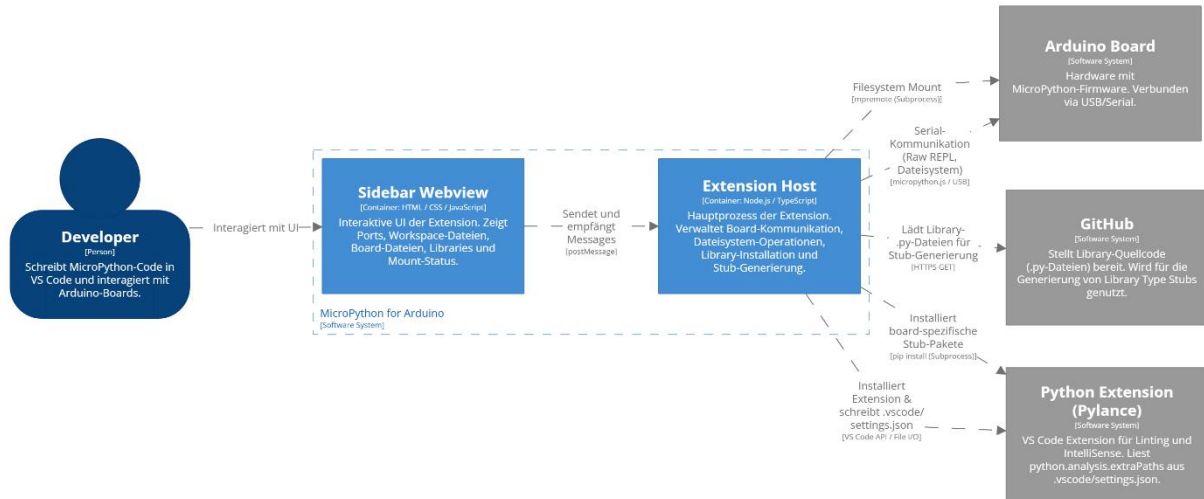
6.1.2 C4 – Level 1: Systemkontext



Level 1 – System Context: MicroPython for Arduino
Donnerstag, 28. Mai 2026 um 14:03 Mitteleuropäische Sommerzeit

Abbildung 9 - C4 Systemkontext

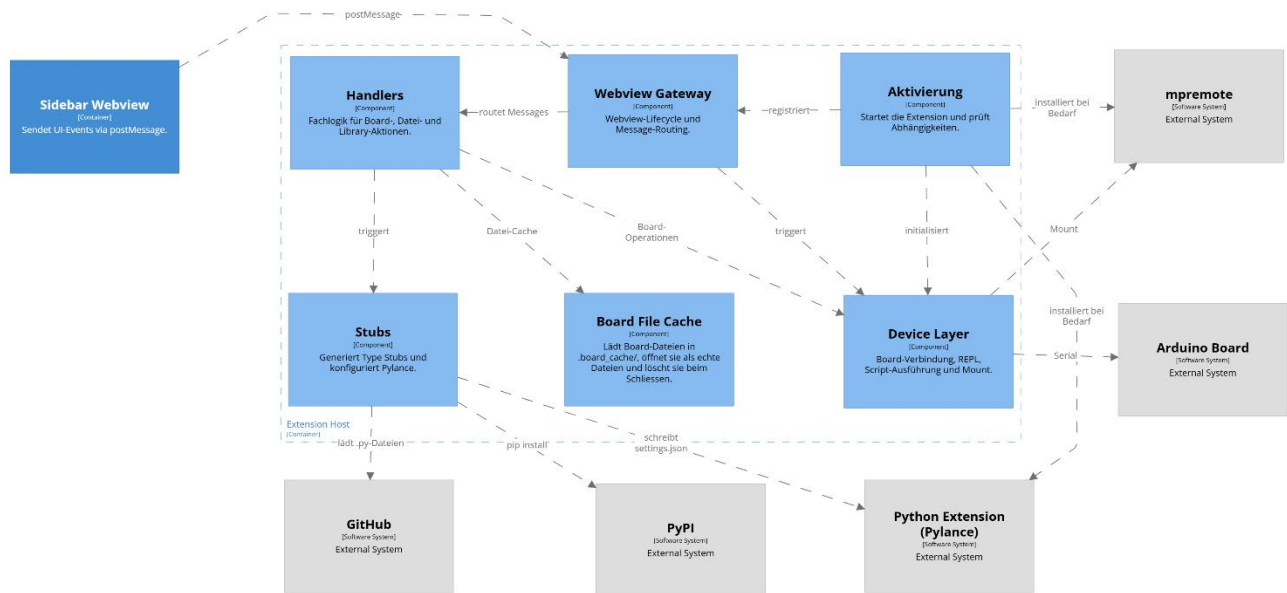
6.1.3 C4 – Level 2: Container



Level 2 – Container: MicroPython for Arduino
Donnerstag, 28. Mai 2026 um 14:27 Mitteleuropäische Sommerzeit

Abbildung 10 - C4 Container

6.1.4 C4 – Level 3: Komponente



Level 3 – Extension Host
Donnerstag, 26. Mai 2020 um 14:28 Mitteleuropäische Sommerzeit

Abbildung 11 - C4 Component

6.2 Libraries und Frameworks

6.2.1 TypeScript

Als Programmiersprache für die Extension wurde TypeScript eingesetzt. TypeScript ist eine typisierte Version von JavaScript, die von Microsoft entwickelt wird und zu standardkonformem JavaScript kompiliert wird. Für die Entwicklung von VS Code Extensions ist TypeScript die empfohlene Sprache, da die offizielle VS Code Extension API vollständig mit TypeScript-Typdefinitionen ausgeliefert wird [6]. Dies ermöglicht eine erstklassige IDE-Unterstützung, frühzeitige Fehlererkennung zur Compile-Zeit sowie eine bessere Wartbarkeit und Lesbarkeit des Codes gegenüber reinem JavaScript.

6.2.2 HTML, CSS und JavaScript (Webview)

Für die Benutzeroberfläche der Extension wurde eine VS Code Webview verwendet. Webviews sind in VS Code das vorgesehene Mittel, um komplexere grafische Oberflächen darzustellen, die über die nativen VS Code UI-Komponenten hinausgehen. Die Webview wird als isoliertes HTML-Dokument gerendert und kommuniziert über ein Message-Passing-API mit dem Extension-Host. Die Oberfläche wurde mit reinem HTML, CSS und JavaScript ohne zusätzliche Frontend-Frameworks umgesetzt, da der Umfang der UI überschaubar ist und der Einsatz eines Frameworks wie React oder Vue den Komplexitäts- und Konfigurationsaufwand unnötig erhöht hätte. Diese Entscheidung hält die Webview leichtgewichtig und direkt in das Extension-Build integrierbar.

6.2.3 Pylance (Extension Dependency)

Im Kontext von VS Code stehen für Python-Sprachunterstützung primär Pylance und Pyright zur Verfügung, da beide als offizielle Microsoft-Erweiterungen nativ in die Python-Extension integriert sind. Weitere Language-Server wie pylsp oder Jedi Language Server existieren, erfordern jedoch zusätzliche Konfiguration und sind nicht Bestandteil der offiziellen Microsoft-Python-Toolchain. Als Language-Server-Extension für Python-Entwicklung in VS Code wurde Pylance eingesetzt. Pylance basiert intern auf Pyright, bietet jedoch zusätzliche IntelliSense-Features wie erweiterte Typinferenz, Auto-Import-Vorschläge sowie verbesserte Codevervollständigung. Da Pylance und Pyright nicht gleichzeitig installiert werden können, fiel die Wahl auf Pylance als Obermenge von Pyright, ohne dabei auf dessen Funktionalität verzichten zu müssen. Es wurden keine Nachteile gegenüber einer reinen Pyright-Installation festgestellt. Ein späterer Wechsel zu Pyright wäre bei Bedarf unkompliziert möglich.

6.2.4 micropython.js

micropython.js ist eine von Arduino entwickelte JavaScript-/Node.js-Bibliothek, die eine programmatische Schnittstelle zur MicroPython-REPL auf einem Mikrocontroller bereitstellt. Sie abstrahiert die serielle Kommunikation und ermöglicht es, MicroPython-Befehle direkt aus dem Extension-Backend heraus auszuführen, Dateien zu übertragen und den Gerätezustand abzufragen. Da Arduino die Bibliothek nicht im öffentlichen npm-Registry veröffentlicht hat, wird sie direkt über das GitHub-Repository von Arduino eingebunden.

6.2.5 mpremote

mpremote ist das offizielle Kommandozeilenwerkzeug der MicroPython-Community zur Interaktion mit MicroPython-Geräten über USB. Im Rahmen dieser Extension wurde ausschliesslich die Mount-Funktion von mpremote eingesetzt. Diese ermöglicht es, ein lokales Verzeichnis auf dem Entwicklungsrechner

direkt als Dateisystem auf dem Mikrocontroller einzubinden, sodass MicroPython-Skripte ohne explizite Dateiübertragung direkt vom Host-Dateisystem ausgeführt werden können. mpremote wurde aufgrund seiner offiziellen Unterstützung durch die MicroPython-Community und seiner breiten Kompatibilität mit MicroPython-fähiger Hardware gewählt.

6.2.6 upy-package

upy-package ist eine ebenfalls von Arduino entwickelte Bibliothek zur Verwaltung von MicroPython-Paketen auf dem Gerät. Sie übernimmt die Auflösung und Installation von Abhängigkeiten direkt auf dem Mikrocontroller und ergänzt damit micropython.js um ein Package-Management analog zu pip für reguläres Python. Auch diese Bibliothek wird direkt aus dem Arduino-GitHub-Repository bezogen.

6.3 Auflistung verwendeten 3rd Party Libraries

Tabelle 16 - 3rd Party Libraries

Library	Beschreibung	Lizenz
micropython.js	Node.js-Bibliothek zur programmatischen Kommunikation mit MicroPython-Geräten über die serielle Schnittstelle	AGPL-3.0
upy-package	Bibliothek zur Verwaltung und Installation von MicroPython-Paketen direkt auf dem Mikrocontroller	MPL-2.0
typescript	Typisierte Obermenge von JavaScript; Grundlage der gesamten Extension-Entwicklung für bessere Wartbarkeit und IDE-Unterstützung	Apache-2.0
eslint	Statisches Analyse-Tool zur Einhaltung von Code-Qualitätsrichtlinien und Erkennung von Fehlern	MIT
typescript-eslint	ESLint-Plugin und -Parser für TypeScript-spezifische Linting-Regeln	MIT
prettier	Opinionated Code-Formatter zur automatischen und einheitlichen Codeformatierung	MIT
jest	JavaScript-Testing-Framework für Unit- und Integrationstests	MIT
ts-jest	Jest-Transformer, der TypeScript-Dateien direkt ohne separaten Build-Schritt testen kann	MIT
jest-junit	Jest-Reporter, der Testergebnisse im JUnit-XML-Format ausgibt (z.B. für CI/CD-Pipelines)	MIT
jscpd	Copy-Paste-Detector zur Erkennung von Code-Duplikaten im Quellcode	MIT
@types/jest	TypeScript-Typdefinitionen für Jest	MIT
@types/js-yaml	TypeScript-Typdefinitionen für js-yaml	MIT
@types/mocha	TypeScript-Typdefinitionen für Mocha	MIT
@types/node	TypeScript-Typdefinitionen für die Node.js-Standardbibliothek	MIT
@types/vscode	TypeScript-Typdefinitionen für die VS Code Extension API	MIT
@vscode/test-cli	Offizielles CLI-Tool von Microsoft zum Ausführen von VS Code Extension Tests	MIT
@vscode/test-electron	Stellt eine Electron/VS Code-Instanz für Integrationstests bereit	MIT
Pylance	VS Code Language-Server-Extension für Python mit erweiterter IntelliSense-Unterstützung	proprietär (Microsoft)
mpremote	Offizielles MicroPython-CLI-Tool; in dieser Extension wird ausschliesslich die Mount-Funktion verwendet	MIT

6.4 Lizenz der Extension: GNU AGPL-3.0

Die Extension wird unter der GNU Affero General Public License Version 3 (AGPL-3.0) veröffentlicht. Diese Wahl ist keine freie Entscheidung, sondern eine lizenzrechtliche Konsequenz aus der Verwendung der Bibliothek micropython.js von Arduino, welche ebenfalls unter AGPL-3.0 steht.

micropython.js wird als Abhängigkeit eingebunden und beim Build-Prozess in das JavaScript-Bundle der Extension kompiliert. Dieses Bundle ist Bestandteil der vsix-Datei, die über den Visual Studio Marketplace verteilt wird. Da die Bibliothek somit aktiv im Sinne der Lizenz verteilt wird, greift das Copyleft-Prinzip der AGPL-3.0 vollumfänglich: Der gesamte Quellcode der Extension muss unter derselben Lizenz veröffentlicht werden.

Die AGPL-3.0 ist eine starke Copyleft-Lizenz. Sie erlaubt die freie Nutzung, Modifikation und Weitergabe der Software, auch für kommerzielle Zwecke, stellt dabei jedoch folgende Anforderungen:

- Der vollständige Quellcode muss bei jeder Distribution öffentlich zugänglich gemacht werden.
- Alle Modifikationen am lizenzierten Code müssen unter AGPL-3.0 weitergegeben werden.

6.4.1 Einschränkungen und warum sie kein Problem darstellen

Auf den ersten Blick erscheint die Copyleft-Pflicht restriktiv. Für die Extension ergeben sich in der Praxis jedoch keine Einschränkungen:

Der Quellcode ist bereits öffentlich. Die Extension ist von Beginn an als Open-Source-Projekt konzipiert. Der Quellcode liegt in einem öffentlichen GitHub-Repository.

Marketplace-Vertrieb ist AGPL-konform. Die AGPL verbietet weder die Veröffentlichung auf dem Visual Studio Marketplace noch die kostenlose Abgabe. Entscheidend ist allein, dass der Quellcode zugänglich bleibt, was durch das Repository gewährleistet ist.

Lizenzkompatibilität ist gegeben. Die weiteren Abhängigkeiten (MPL-2.0, MIT) sind mit AGPL-3.0 kompatibel. MPL-2.0-lizenzierte Dateien müssen lediglich als solche erkennbar bleiben, was durch die Projektstruktur sichergestellt ist.

Der **Netzwerkparagraph (§ 13)** erweitert die Copyleft-Pflicht auf Software, die Nutzer über ein Netzwerk verwenden, ohne dass eine klassische Distribution stattfindet. Für eine lokal installierte VS Code Extension ist dieser Paragraph nicht anwendbar.

Die AGPL-3.0 entspricht damit dem Open-Source-Charakter des Projekts und stellt unter den gegebenen Rahmenbedingungen keinerlei praktische Einschränkung dar.

6.5 Projekt Ordner Struktur

src	Quellcode der Extension
device	Board-Kommunikationsschicht
operation	Board-Operationen (Run, Stop, Reset, Dateien, Libraries)
ConnectionManager.ts	Serielle Ports auflisten, VID:PID-Erkennung, Port auswählen
DeviceManager.ts	Serialisiert Board-Operationen via withBoard()
ReplTerminal.ts	Interaktives REPL-Terminal in VS Code
stubs	Stub-Generierung für Pylance/Pyright
StubGenerator.ts	.pyi-Stubs aus Board-Libraries generieren
PylanceConfig.ts	pyrightconfig.json mit extraPaths befüllen
test	Tests
integration	Integrationstests (VS Code Extension Host)
unit	Unit-Tests (Jest)
types	Typen und Konstanten
messages.ts	Typisiertes Nachrichtenprotokoll (Webview ↔ Extension)
webview	VS Code Sidebar (Extension-Seite)
handlers	Ein Handler pro Feature-Domain
SidebarProvider.ts	Verdrahtet alle Handler, verwaltet globalState
WebviewGateway.ts	Typsicherer Nachrichtenrouter (Webview ↔ Extension)
extension.ts	Einstiegspunkt, registriert Provider & Commands
media	Frontend-Ressourcen (Vanilla JS, kein Bundler)
sidebar	Sidebar-Frontend
modules	JS-Module
filetree	Dateibaum-Komponente
genericTree.js	Wiederverwendbare Dateibaum-Komponente
boardFilesProvider.js	Daten-Provider für Board-Dateibaum
workspaceProvider.js	Daten-Provider für Workspace-Dateibaum
libraries.js	Library-Install/Uninstall UI
stubs.js	Stubs UI
sidebar.html	HTML-Template der Sidebar
sidebar.js	Messages Orchestrator
assets	Extension-Icons
out	Kompiliertes JavaScript (Build-Artefakt)

6.6 Wartung / Weiterentwicklung; Guide und Tools

Für die Wartung und Weiterentwicklung der Extension sind Node.js (Version 22.x) und npm sowie Python 3.x erforderlich. Wie die Entwicklungsumgebung eingerichtet und die Extension lokal gestartet wird, ist im MAINTAINER.md des Projektordners beschrieben. Dort sind ebenfalls alle Einstiegspunkte für häufige Änderungen dokumentiert, wie das Hinzufügen neuer Boards oder das Implementieren neuer Features. Während der Entwicklung wurde GitLab verwendet, die bestehende CI/CD-Pipeline führt bei jedem Push automatisch Tests, Linting und Typprüfung aus. Das Open Source Repository ist jedoch auf GitHub und hat einen identischen Workflow.

Die Veröffentlichung der Extension läuft automatisch über einen GitHub Workflow, welcher beim Erstellen eines Releases die Extension verpackt und auf den Visual Studio Code Marketplace veröffentlicht. Dabei wird mit einem Secret im GitHub gearbeitet. Das Secret names VSCE_PAT befindet sich unter Settings > Secrets and variables > Actions > Repository secrets und enthält einen Token, der das Managen der Extension auf dem Marketplace erlaubt. Dazu wird ein Microsoft Account mit Zugriff auf die Extension Publisher Organisation benötigt (sichtbar unter <https://marketplace.visualstudio.com/manage>). Der Token kann dann mit diesem Account unter <https://dev.azure.com/> > User Settings (oben rechts) > Personal access tokens > New Token wie folgt generiert werden:

- Name frei wählen
- Organization: All accessible organizations
- Expiration (maximal ein Jahr)
- Scopes (Custom defined) > Marketplace: Manage

Über den generierten Token erhält der GitHub Workflow die Berechtigung, die Extension zu veröffentlichen. Dieser Token muss entsprechend nach der Expiration erneut generiert und im GitHub Secret ersetzt werden.

7 UI / UX

Für die Extension wurde eine eigene Benutzeroberfläche entwickelt. Das Ziel bei der Entwicklung war es, eine möglichst intuitive und anfängerfreundliche Bedienung zu ermöglichen, sodass auch Nutzer ohne Vorkenntnisse in MicroPython oder eingebetteten Systemen schnell mit der Extension arbeiten können.

Anstelle der nativen VS-Code-Komponenten erstellt die Extension eine Webview-basierte Benutzeroberfläche. Dadurch kann das Erscheinungsbild zwar an den VS-Code-Stil angepasst werden, gleichzeitig stehen deutlich mehr Gestaltungsmöglichkeiten und Freiheiten bei der Umsetzung zur Verfügung.

Die Benutzeroberfläche fasst alle zentralen Funktionen der Extension an einem Ort zusammen. Dazu gehören die Verwaltung von Geräteverbindungen, das Ausführen von Skripten, das Öffnen der REPL-Konsole, die Verwaltung lokaler Projektdaten, die Anzeige der auf dem Mikrocontroller gespeicherten Dateien sowie die Installation und Deinstallation von Bibliotheken. Durch diese zentrale Anordnung müssen Nutzer nicht zwischen verschiedenen Menüs oder Terminals wechseln, was den Arbeitsablauf vereinfacht und die Benutzerfreundlichkeit erhöht.

In diesem Kapitel werden Aufbau und Funktionsweise der Benutzeroberfläche sowie deren Integration in die Entwicklungsumgebung Visual Studio Code beschrieben. Darüber hinaus werden die wesentlichen Designentscheidungen erläutert und dokumentiert, die während der Entwicklung getroffen wurden.

7.1 Aufbau der Sidebar

Die Sidebar ist in mehrere Bereiche gegliedert. Im oberen, permanent sichtbaren Bereich befinden sich die Verbindungsverwaltung sowie die verfügbaren Board-Aktionen. Darunter folgen die Darstellung des lokalen Workspaces und der Dateibaum des verbundenen Mikrocontrollers. Im unteren Bereich ist die Bibliotheksverwaltung mit der Funktion zur Installation zusätzlicher Bibliotheken sowie eine Übersicht der verfügbaren Code-Support-Funktionen angeordnet.

Um die Benutzeroberfläche übersichtlich zu halten, werden boardspezifische Funktionen zunächst ausgeblendet. Dann sieht die Sidebar zunächst wie in Abbildung 12 aus. Erst nach dem erfolgreichen Verbindungsaufbau mit einem Mikrocontroller werden die entsprechenden Bereiche und Aktionen sichtbar. Während Operationen auf dem Board ausgeführt werden, können bestimmte Funktionen temporär deaktiviert werden, um konkurrierende Zugriffe zu verhindern und einen konsistenten Zustand der Anwendung sicherzustellen.

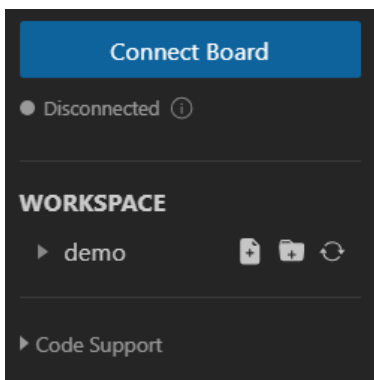


Abbildung 12 - UI ohne Verbindung

7.2 Verbindungsverwaltung

Zum Aufbau einer Verbindung kann die Schaltfläche «Connect Board» verwendet werden. Nach dem Anklicken öffnet sich, wie in Abbildung 13 zu erkennen, ein Menü, das alle erkannten und verfügbaren Boards auflistet. Sobald eine Verbindung hergestellt wurde, wird die Schaltfläche durch eine Tab-Ansicht (Abbildung 14) ersetzt.

Diese ermöglicht das gleichzeitige Verwalten mehrerer Board-Verbindungen. Über das Plus-Symbol können zusätzliche Boards hinzugefügt werden, während über die Tabs zwischen den verbundenen Geräten gewechselt werden kann. Das Schliessen-Symbol eines Tabs trennt die entsprechende Verbindung wieder.

Unterhalb der Tab-Ansicht werden der Name des aktuell ausgewählten Boards sowie dessen Status angezeigt. Neben der Statusanzeige befindet sich ein Informationssymbol, über das zusätzliche Details und Erläuterungen zum aktuellen Status abgerufen werden können. In Abbildung 15 wird der Status Running

Ursprünglich war die Verbindung durch eine Dropdown-Auswahl geplant, was nicht umgesetzt werden konnte, da die Anforderung mehrere Boards anzusteuern später dazukam. Die Tab-basierte Verwaltung ermöglicht den Wechsel zwischen mehreren verbundenen Mikrocontrollern, ohne bestehende Verbindungen unterbrechen zu müssen. Dadurch wird die Entwicklung auf verschiedenen Geräten zur selben Zeit ermöglicht. Die anfängliche Anzeige nur einem Button zum Verbinden eines Boards reduziert die Komplexität der Benutzeroberfläche und erleichtert insbesondere neuen Nutzern den Einstieg.

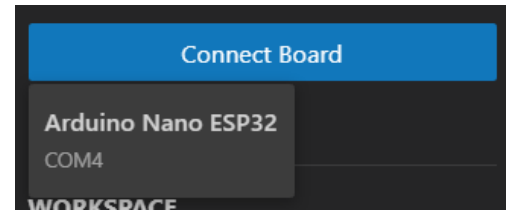


Abbildung 13 - UI Verbindung aufbauen

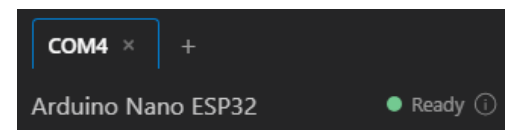


Abbildung 14 - UI Verbindung Tab Ansicht

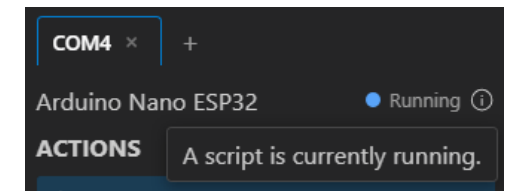


Abbildung 15 - Board Status mit Tooltip

7.3 Basis-Aktionen mit dem Board

Die Basis-Aktionen werden unterhalb der Verbindung dauerhaft im oberen Bereich der Oberfläche fixiert dargestellt. Jede Aktion ist mit einem passenden Icon sowie einer klar verständlichen Beschriftung versehen, um die Benutzerführung zu erleichtern. Die Aktionen sind in Abbildung 16 zu erkennen.

In Abbildung 17 ist eine komprimierte Version der Aktionen zu erkennen. Um den verfügbaren Platz effizienter zu nutzen, kann «icon-only» aktiviert werden. In diesem Modus werden die Buttons ausschliesslich durch ihre Icons dargestellt und in einer einzigen oder

maximal zwei Reihen angeordnet. Die zugehörigen Beschriftungen werden dabei als Tooltips angezeigt, sodass die Verständlichkeit der Funktionen weiterhin gewährleistet bleibt, ohne die Oberfläche visuell zu überladen.

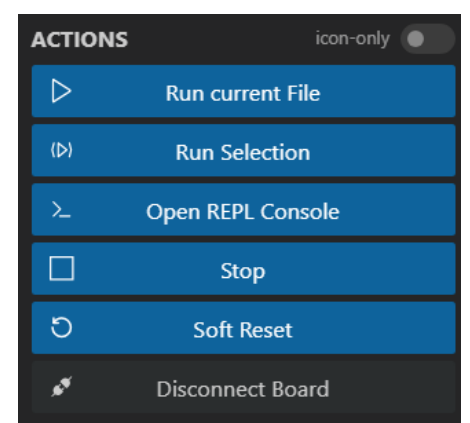


Abbildung 16 - UI Kontrollzentrum

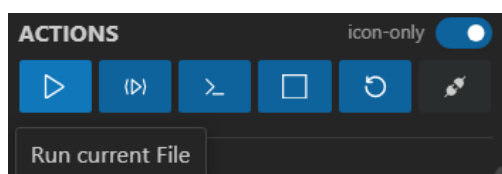
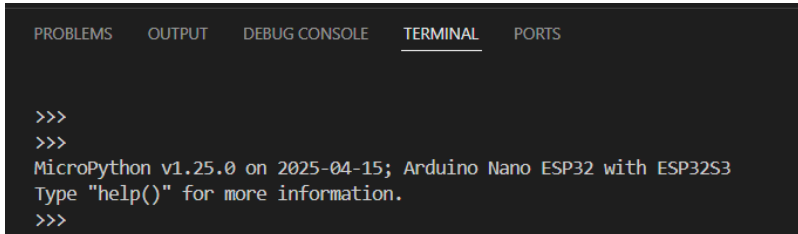


Abbildung 17 - UI Kontrollzentrum mit Icon Buttons

Run current File: Führt die aktuell im Editor geöffnete Python Datei aus.

Run Selection: Führt den selektierten Code im Editor aus.

Open REPL Console: Öffnet ein VS Code Terminal, indem die REPL sichtbar ist, wie in Abbildung 18.



```
>>>
>>>
MicroPython v1.25.0 on 2025-04-15; Arduino Nano ESP32 with ESP32S3
Type "help()" for more information.
>>>
```

Abbildung 18 - REPL Konsole

Stop: Stoppt ein Skript, dass gerade auf dem Board ausgeführt wird.

Soft Reset: Öffnet die REPL-Konsole und führt darin einen Neustart des Boards durch.

Disconnect Board: Bricht die Verbindung zum Board und schliesst den Tab.

7.4 Workspace Dateibaum

Der in Abbildung 19 dargestellte Bereich «Workspace» zeigt den aktuellen Projekt-Workspace. Über Inline-Buttons können Aktionen für das entsprechende Verzeichnis oder die Datei ausgeführt werden. Bei Verzeichnissen können neue Dateien und Ordner erstellt werden. Im Wurzelverzeichnis steht zusätzlich eine Aktualisierungsfunktion zur Verfügung, mit der der Workspace neu geladen werden kann. Unterordner können zusätzlich gelöscht sowie umbenannt werden. Bei Dateien stehen die Funktionen Upload, Umbenennen und Löschen zur Verfügung.

Durch das Anklicken von Dateien werden diese im Editor geöffnet, während Ordner entsprechend aufgeklappt werden. Zusätzlich unterstützt die Oberfläche das Verschieben von Dateien und Ordnern mittels Drag-and-Drop, um eine intuitive Strukturverwaltung zu ermöglichen.

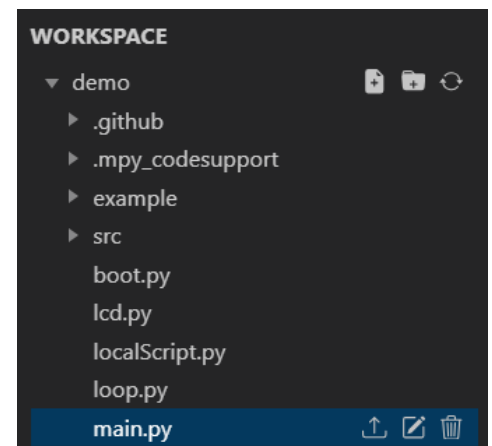


Abbildung 19 - UI Workspace

Damit der Workspace-Dateibaum so intuitiv wie möglich ist, wurde sich weitestgehend am VS Code Verhalten orientiert. Dauerhaft sichtbaren Buttons neben dem Wurzelverzeichnis orientieren sich stark an der Explorer-Ansicht von VS Code. Einfaches Anklicken einer Datei öffnet eine Vorschau und ein Doppelklick öffnet es permanent im Editor. Bei den anderen Aktionen wurde sich gegen ein Kontextmenü durch ein Rechtsklick und für die Inline Buttons entschieden. Diese sind für neue Nutzer schnell erkennbar durch das Auftauchen bei einem Mouseover über das entsprechende Element. Auch die selektierte Datei passt sich wie im VS Code Explorer der geöffneten Datei im Editor an.

Aktionen, die zusätzliche Benutzereingaben erfordern, werden über das VS Code QuickPick oder InputBox (Abbildung 20) abgefragt. Dazu gehören beispielsweise das Umbenennen (Eingabe des neuen Namens), das Erstellen neuer Dateien oder Ordner (Eingabe des Namens) sowie das Hochladen von Dateien (Auswahl des Zielordners).

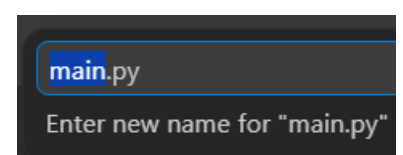


Abbildung 20 - Rename über Input Box

7.5 Mount Workspace

Unterhalb des Workspace-Dateibaums befindet sich die Funktion «Mount Workspace». Diese ermöglicht es, einen Ordner aus dem aktuellen Workspace temporär mit dem verbundenen Board zu synchronisieren. Zur Unterstützung der Benutzerführung befindet sich neben dem Button ein Informationssymbol. Beim Überfahren mit der Maus wird ein Tooltip angezeigt, der die Funktionalität des Mount-Vorgangs kurz erläutert, wie in Abbildung 21 zu erkennen.

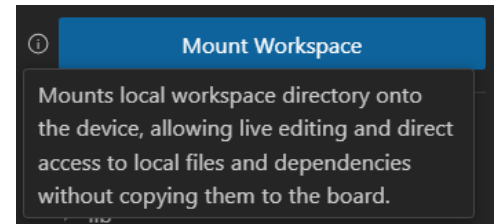


Abbildung 21 - Mount Button mit Tooltip

Nach dem Betätigen des Buttons wird über ein VS Code QuickPick zunächst der zu mountende Ordner ausgewählt. Anschliessend öffnet die Extension automatisch ein Terminal und führt den entsprechenden Mount-Befehl aus. Das geöffnete Terminal dient danach als sogenannte Mount REPL und stellt die zentrale Kommunikationsschnittstelle zum Board dar.

Während ein Workspace gemountet ist, können ausschliesslich Befehle über diese Mount REPL ausgeführt werden. Um Konflikte zwischen dem synchronisierten Dateisystem und direkten Boardzugriffen zu vermeiden, werden deshalb bestimmte Funktionen der Extension deaktiviert. Dazu gehören insbesondere der Board-Dateibaum sowie die Bibliotheksverwaltung. Die deaktivierten Funktionen werden teilweise, wie in Abbildung 22, mit einem Tooltip versehen. Dieser weist darauf hin, dass der aktive Mount zuvor beendet werden muss.

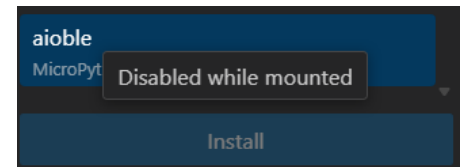


Abbildung 22 - Disabled Tooltip bei Mount

Einige häufig benötigte Aktionen bleiben weiterhin verfügbar. Dazu zählen das Ausführen der aktuell geöffneten Datei (Run current File), das Ausführen einer Codeauswahl (Run Selection), ein Soft Reset des Boards sowie das Beenden laufender Programme (Stop). Diese Aktionen werden während eines aktiven Mounts direkt über die Mount REPL ausgeführt. In Abbildung 23 ist zu erkennen, wie das main.py über die Mount REPL ausgeführt wurde.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\weber\Documents\BA\demo> mpremove connect COM4 mount "c:\Users\weber\Documents\BA\demo" + repl
Use Ctrl-J or Ctrl-x to exit this shell
>
MicroPython v1.25.0 on 2025-04-15; Arduino Nano ESP32 with ESP32S3
Type "help()" for more information.
>>>
paste mode; Ctrl-C to cancel, Ctrl-D to finish
=== exec(open("main.py").read())
main.py executes...
result: 42
>>>

```

Abbildung 23 - Mount REPL

Die Funktion ermöglicht eine effiziente Entwicklung, da Änderungen an Dateien unmittelbar innerhalb des gemounteten Workspace vorgenommen werden können, ohne diese manuell auf das Board übertragen zu müssen. Gleichzeitig verhindern die deaktivierten Funktionen mögliche Inkonsistenzen und tragen zu einer klaren sowie fehlerarmen Benutzerführung bei. Leider wird das Lesen des Terminal-Inhalts nicht unterstützt. Dadurch ist es nicht möglich den Status des Boards während des aktiven Mounts zu unterscheiden und es kann nicht erkannt werden, ob gerade etwas auf dem Board läuft oder ob die REPL bereit für eine Eingabe ist (>>>). Dadurch sind immer alle möglichen Buttons aktiv, auch wenn ihr Input möglicherweise ignoriert wird. VS Code bietet zwar eine Möglichkeit das Terminal zu lesen, jedoch wird bei Verwendung die Veröffentlichung der Extension über den Marketplace nicht angeboten.

7.6 Board Dateibaum

Der Board-Dateibaum orientiert sich sowohl hinsichtlich seines Erscheinungsbildes als auch seiner Bedienung weitgehend am Workspace-Dateibaum. Dadurch profitieren Benutzer von einer konsistenten Benutzeroberfläche und können bereits bekannte Interaktionsmuster wiederverwenden.

Die verfügbaren Aktionen unterscheiden sich jedoch teilweise von denen des Workspace-Dateibaums. Dateien auf dem Board können direkt ausgeführt werden. Zudem steht anstelle einer Upload-Funktion eine Download-Funktion zur Verfügung, mit der Dateien vom Board in die lokale Entwicklungsumgebung übertragen werden können.

Da das Laden von Dateien vom Board im Vergleich zu lokalen Dateien eine höhere Latenz aufweist, wurde ein zweistufiges Öffnungskonzept implementiert. Bei einem einfachen Klick wird geprüft, ob bereits eine temporäre lokale Kopie der Datei vorhanden ist. Ist dies der Fall, wird diese direkt im Editor geöffnet. Ein Doppelklick führt dieselbe Prüfung durch, lädt die Datei jedoch zusätzlich vom Board herunter, falls noch keine lokale Kopie existiert.

Die heruntergeladenen Dateien werden in einem versteckten Verzeichnis innerhalb des Workspace gespeichert, das im Workspace-Dateibaum nicht angezeigt wird.

Um diese Dateien eindeutig von regulären Workspace-Dateien zu unterscheiden, werden sie im Editor mit einer abweichenden Farbkennzeichnung dargestellt und zusätzlich mit Symbolen versehen, die ihren temporären sowie heruntergeladenen Zustand visualisieren. Dies ist in Abbildung 24 zu erkennen.

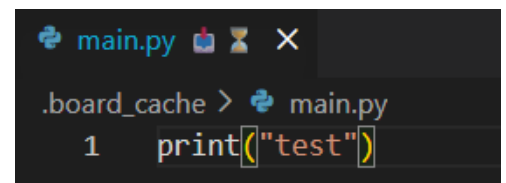


Abbildung 24 - Boarddatei im Editor

Wird eine heruntergeladene Datei über die Tastenkombination «Ctrl+S» gespeichert, überträgt die Extension die Änderungen automatisch zurück an die ursprüngliche Position auf dem Board. Dadurch können Dateien direkt im gewohnten VS-Code-Editor bearbeitet werden, ohne dass ein manueller Down- und Upload erforderlich ist.

Während der Evaluation zeigte sich, dass die Möglichkeit zum Öffnen von Dateien per Doppelklick von Benutzern nur selten entdeckt wurde. Aus diesem Grund wird beim Überfahren einer Python-Datei mit der Maus innerhalb des Tooltips ein zusätzlicher Hinweis «doubleclick to open» angezeigt. Dadurch wird die Auffindbarkeit der Funktion verbessert und die Benutzerfreundlichkeit erhöht. Während der Workspace gemounted ist, wird der Board Dateibaum mit einer Nachricht ersetzt, dass er während des Mounts nicht genutzt werden kann.

7.7 Library Verwaltung

Der Bereich «Libraries» dient zur Installation, Deinstallation und Verwaltung von MicroPython-Bibliotheken sowie des zugehörigen Code Supports. Im oberen Bereich werden sämtliche auf dem Board im Verzeichnis «/lib» installierten Bibliotheken angezeigt. Diese können, wie in Abbildung 25 ersichtlich, selektiert und direkt deinstalliert werden.

Darunter befinden sich zwei ausklappbare Bereiche, welche die Funktionen zur Bibliotheksinstallation und zur Verwaltung des Code Supports zusammenfassen.

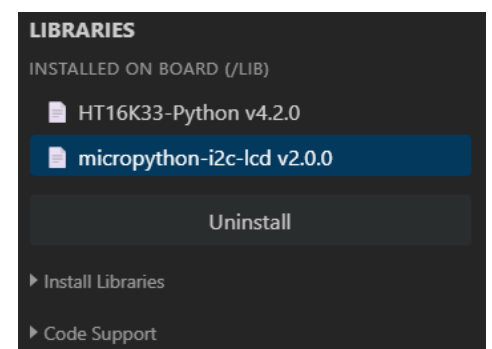


Abbildung 25 - UI Libraries

7.7.1 Library installieren

Wird «Install Libraries» ausgeklappt, wird das in Abbildung 26 abgebildete UI angezeigt. Es steht eine Auswahl an MicroPython-Bibliotheken zur Verfügung, die direkt auf dem verbundenen Board installiert werden können. Alternativ kann eine Bibliothek auch über die Angabe einer GitHub-URL installiert werden.

Während des Installationsvorgangs werden Statusmeldungen und Protokolleinträge im VS-Code-Output-Channel angezeigt. Dadurch kann der Fortschritt der Installation jederzeit nachvollzogen werden.

Nach erfolgreicher Installation werden die Python-Dateien der Bibliothek automatisch heruntergeladen und in den Workspace integriert. Dadurch stehen dem Entwickler Funktionen wie Autovervollständigung, Typinformationen und weitere Code-Support-Funktionen unmittelbar zur Verfügung. Falls die Bibliothek eine README-Datei enthält, wird diese zusätzlich automatisch als Vorschau im Editor geöffnet.

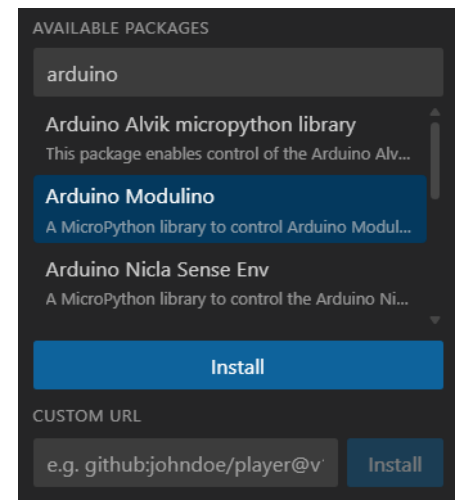


Abbildung 26 - UI Library installieren

7.7.2 Code Support

Der Bereich «Code Support» (Abbildung 27) bietet eine Übersicht darüber, für welche Bibliotheken im aktuellen Workspace Unterstützung durch den Editor verfügbar ist. Dabei werden sowohl Bibliotheken berücksichtigt, die auf dem Board installiert sind, als auch Bibliotheken, die bereits lokal im Workspace vorhanden sind.

Für lokal verfügbare Bibliotheken kann der Code Support individuell aktiviert oder deaktiviert werden. Dadurch können Entwickler gezielt festlegen, welche Bibliotheken bei der Codeanalyse und Autovervollständigung berücksichtigt werden sollen.

Zusätzlich steht eine Funktion zur Verfügung, mit der der Code Support für sämtliche auf dem Board installierten Bibliotheken neu generiert werden kann. Dies ermöglicht eine einfache Aktualisierung der Entwicklungsumgebung, beispielsweise nach dem Öffnen eines neuen Workspace.

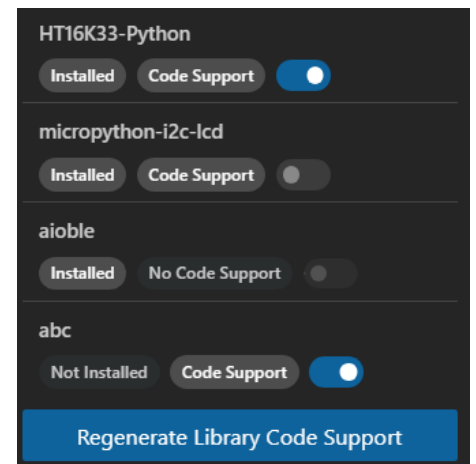


Abbildung 27 - UI Code Support

Die Aufteilung in die Bereiche Bibliotheksverwaltung und Code Support reduziert die Komplexität der Oberfläche und trennt die Installation von Bibliotheken klar von der Konfiguration der Entwicklungsunterstützung. Durch die automatische Bereitstellung des Code Supports nach einer Installation entfällt zudem ein zusätzlicher manueller Arbeitsschritt, wodurch die Benutzerfreundlichkeit verbessert und der Einstieg in die Entwicklung mit MicroPython erleichtert wird.

8 Implementation

8.1 Überblick

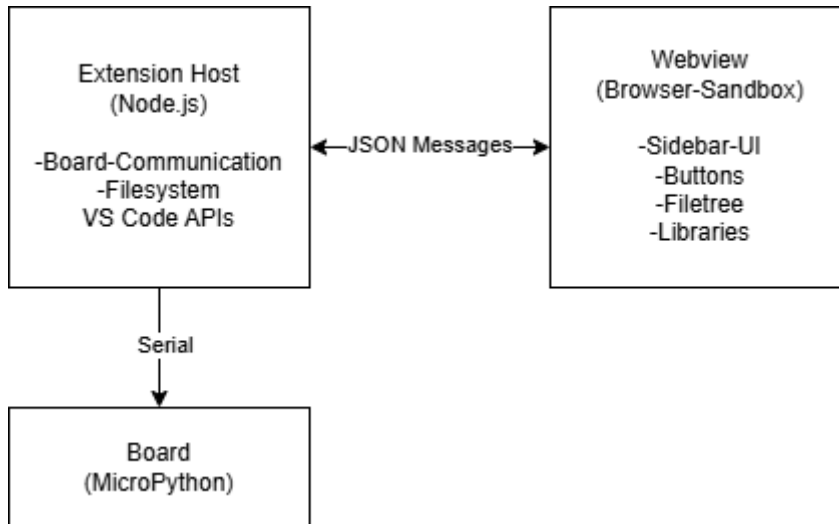


Abbildung 28 - Überblick Implementation

Die Extension besteht aus zwei isolierten Teilen, die in Abbildung 28 dargestellt werden: dem Extension Host (Node.js) und der Webview (Browser-Sandbox). Da diese nicht direkt aufeinander zugreifen können, kommunizieren sie ausschliesslich über JSON-Nachrichten. Der Extension Host hat Zugriff auf VS Code APIs, das Dateisystem und die serielle Board-Verbindung. Für die serielle Kommunikation wird die Library micropython.js verwendet. Die Webview ist für die Darstellung der Sidebar-UI zuständig.

Nachrichtentypen (Webview → Extension):

Tabelle 17 - Messagetypes Webview zu Extension

Kategorie	Beispiele
Verbindung	getPorts, openPort, closePort
Board-Aktionen	runFile, stopFile, softReset
Workspace-Dateien	ws_openFile, ws_delete, ws_move
Board-Dateien	bf_openFile, bf_delete, bf_move
Mount	toggleMount
Libraries	installLibrary, getLibraries
Code-Support	activateCodeSupport, generateStubs

Nachrichtentypen (Extension → Webview):

Tabelle 18 - Messagetypes Extension zu Webview

Kategorie	Beispiele
Initialisierung	init
Board-Status	boardState, ports, disconnected
Workspace-Baum	workspaceFiles, ws_nodeCreated
Board-Baum	boardFiles, bf_nodeDeleted
Libraries	installedLibraries, installResult
Code-Support	codeSupport

8.2 Board Status

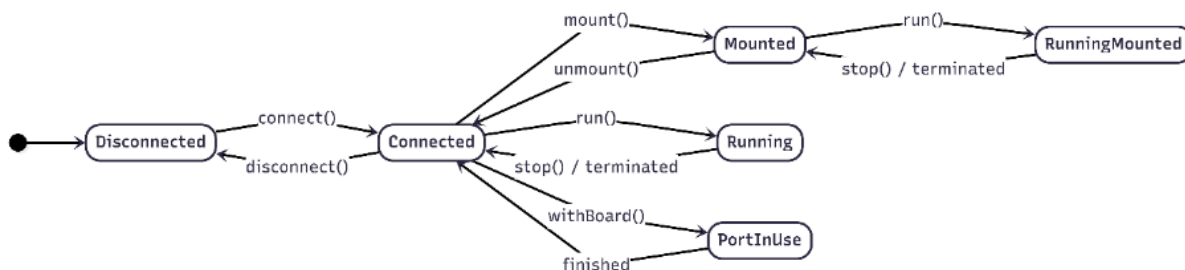


Abbildung 29 - Board Statediagramm

Abbildung 29 zeigt ein State-Diagramm, welches den Boardstatus darstellt. Der Boardstatus wird von der Extension bei jeder Änderung an die Webview gesendet. Dabei gibt es 4 verschiedene Busy States in denen sich das Board befinden kann. Mount, Running, PortInUse und REPL. RunningMounted kann durch Einschränkungen seitens VS Code nicht erkannt werden. REPL ist nicht im State-Diagramm zu erkennen, da die REPL-Konsole durch andere Aktionen einfach geschlossen wird und somit keine Blockaden verursacht. Andere Aktionen sind je nach Boardstatus nicht verfügbar.

8.3 Ein Board verbinden

Wenn der User einen Port in der Webview (Sidebar) auswählt, erstellt der Extension Host mit dem ConnectionManager eine DeviceManager-Instanz für diesen Port. Die Extension lädt anschliessend die Board-Dateien und installierten Libraries vom Board und zeigt sie in der Sidebar an. Im Hintergrund werden automatisch Board-spezifische Code-Support-Pakete installiert.

Einstiegspunkt/Änderungen: Um ein neues Board hinzuzufügen muss in `src/device/ConnectionManager.ts` in der `KNOWN_BOARDS`-Konstante ein neuer VID:PID Eintrag gemacht werden. Das ist die Vendor ID und Produkt ID des Boards. Zudem kann da ein Name und ein zugehöriges Stub Paket definiert werden.

Da der serielle Port eine exklusive Ressource ist, auf die jeweils nur eine Operation gleichzeitig zugreifen kann, läuft jede Board-Operation über das `withBoard()`-Pattern im `DeviceManager`. Dieses stellt sicher, dass keine zwei Operationen den Port gleichzeitig öffnen. Jede neue Board-Operation muss zwingend über dieses Pattern implementiert werden, andernfalls riskiert sie einen Konflikt mit einer bereits laufenden Operation.

8.4 Library Installieren

Der User wählt eine Library aus der Sidebar. Die Extension ruft den externen `upy-package`-Prozess auf, der die Library direkt auf das Board installiert. Danach wird ein Manifest (`/lib/manifest.json`) auf dem Board aktualisiert, das alle installierten Libraries verwaltet. Die Sidebar zeigt anschliessend die aktualisierte Liste. Das `manifest.json` dient zur Verwaltung von Libraries.

Einstiegspunkt/Änderungen: Änderungen an der Library-Installation oder dem Manifest sind in `src/device/operation/LibraryOperations.ts` vorzunehmen.

Ein Problem war ursprünglich, dass `upy-package` alle Dateien einer Library in einen Ordner schiebt. Dabei gehen alle Informationen bezüglich, welche Files zu welcher Library gehören, verloren. Spätestens beim Deinstallieren stellt dies ein Problem dar. Mit dem `manifest.json` wird dieses Problem gelöst.

```
{
  "packages":{
    "Arduino Modulino":{
      "url":"https://github.com/arduino/arduino-modulino-mpy",
      "displayName":"Arduino Modulino",
      "installedAt":"2026-04-10",
      "files":[
        "ism6dsox.mpy",
        "micropython_hs3003",
        "modulino"
      ],
      "version":"1.1.0"
    },
    "zlib":{
      "url":"",
      "displayName":"zlib",
      "installedAt":"2026-04-10",
      "files":[
        "zlib.mpy"
      ],
      "version":"1.0.0"
    }
  }
}
```

8.4.1 Code Support

Wenn der User eine Library installiert oder den Code Support regenerieren will, liest die Extension das Manifest vom Board, um zu wissen welche Libraries installiert sind. Die Quellcode-Dateien dieser Libraries werden von GitHub heruntergeladen und lokal im Workspace unter `.mpy_codesupport/` gespeichert. Abschliessend wird die Pylance-Konfiguration angepasst, sodass Autocomplete und Typprüfung auf den Board-Libraries funktionieren. In der Pylance-Konfiguration steht der Pfad, wo die Extension die Files findet pro Library.

Für die Board spezifischen Stubs sind ebenfalls Pfade in der Pylance-Konfiguration vorhanden. Diese Stubs werden beim Verbinden eines Boards mit pip installiert.

Einstiegspunkt/Änderungen: Die GitHub-Quellcode-Generierung ist in `src/stubs/StubGenerator.ts`, die Pylance-Konfiguration in `src/stubs/PylanceConfig.ts` zu finden.

Für den Library Code Support werden nicht wie ursprünglich angedacht Stubs verwendet. Stattdessen werden die Quellcode-Dateien direkt in den Workspace kopiert. Nach ersten Versuchen mit Stubs, sind einige Schwächen aufgefallen:

- Return Type Erkennung
 - Stubs: nur wenn im Methodenheader angegeben
 - Python: auch wenn aus Return Statement ersichtlich
- Docstrings werden nicht immer zuverlässig angezeigt

Durch den Fakt, dass die Quellcode-Dateien diese Schwächen nicht haben und den zusätzlichen Aufwand ersparen die Stub-Dateien zu generieren, wurde sich bewusst dagegen entschieden Stubs für Library Code Support zu verwenden. Zusätzlich ist es so für den Nutzer einfach möglich die Implementation der Libraries zu studieren, wenn er mit Ctrl+Klick auf eine Methode oder Klasse klickt.

8.5 Mount

Mount bindet den lokalen Workspace direkt auf dem Board ein, sodass Scripts ohne vorherigen Upload ausgeführt werden können. Beim Aktivieren wird `main.py` auf dem Board temporär umbenannt, um Konflikte zu vermeiden. Mount läuft über `mpremote` in einem separaten VS Code Terminal. Beim Deaktivieren wird `main.py` wieder zurückbenannt. `Mpremove` wird automatisch beim Installieren der Extension über pip installiert.

Während Mount aktiv ist gelten folgende Einschränkungen wegen serieller Exklusivität:

- Board-Dateioperationen (Dateien lesen, schreiben, löschen) sind deaktiviert
- Library-Installation ist deaktiviert
- VS Code darf nicht geschlossen werden, solange Mount aktiv ist. Der `mpremote`-Prozess läuft weiter im Hintergrund und belegt den seriellen Port. Beim nächsten Öffnen von VS Code ist der Port blockiert und kann erst nach manuellem Ctrl+X im Terminal wieder verwendet werden.
- Mount muss über den Toggle-Button oder Ctrl+X im Terminal deaktiviert werden. Wird das Terminal gewaltsam geschlossen, versucht die Extension Mount automatisch sauber zu beenden.

Einstiegspunkt/Änderungen: Das Mount-Verhalten ist in `src/device/MountManager.ts` und `src/device/operation/ActivateMountOperation.ts` implementiert.

8.6 Board Dateien öffnen

Board-Dateien werden beim Öffnen in den lokalen Ordner `.board_cache/` heruntergeladen und als normale Dateien in VS Code geöffnet. So funktionieren Pylance und Autocomplete auch auf Board-Dateien. Änderungen werden mit `Ctrl+S` direkt auf das Board hochgeladen. Beim Schliessen des Editor-Tabs wird die lokale Kopie automatisch gelöscht.

Einstiegspunkt/Änderungen: Der gesamte Board-Datei-Cache ist in `src/webview/handlers/BoardFileSystemProvider.ts` implementiert.

8.7 Script ausführen & REPL

Der User kann Scripts direkt vom Workspace oder vom Board aus starten. Der Output wird live in einem Terminal angezeigt. Über Stop wird ein laufendes Script unterbrochen, Soft Reset startet MicroPython auf dem Board neu. Das interaktive REPL öffnet ein VS Code Terminal mit direktem Zugriff auf die Python-Konsole des Boards.

Beim Ausführen von Scripts wird zwischen drei Fällen unterschieden:

- **Workspace Datei ohne Mount:** Der Quellcode der Datei wird lokal gelesen und direkt über das Raw-REPL-Protokoll als String an das Board gesendet.
- **Workspace Datei mit aktivem Mount:** Die Datei wird über das aktive Mount-REPL mittels Paste Mode ausgeführt (`exec(open("relativer/Pfad").read())`). Liegt die Datei ausserhalb des gemounteten Folders wird der Code direkt als Code-Block gesendet via Paste Mode.
- **Boarddatei:** Ein kurzes Python-Snippet wird über das Raw REPL gesendet, das die Datei direkt vom Dateisystem des Boards liest und ausführt (`exec(open(path).read(), globals())`).

Einstiegspunkt/Änderungen: Die Script-Ausführung ist in `src/device/operation/RunFileOperation.ts` und `src/device/operation/RunCodeOperation.ts` zu finden. Das REPL ist in `src/device/ReplTerminal.ts` implementiert.

8.8 Workspace-Filetree

Der Workspace-Dateibaum wird durch rekursives Einlesen des lokalen Dateisystems mit der Node.js-Funktion `fs.readdirSync` aufgebaut. Folgende Verzeichnisse werden dabei grundsätzlich ausgeblendet: `node_modules`, `.git`, `__pycache__`, `dist`, `out`, `.vscode` und `.board_cache`. Das Resultat ist ein FileNode-Baum, bei dem die id jedes Knotens dem absoluten Dateipfad auf dem Dateisystem entspricht. Dieser Pfad wird direkt an alle Dateioperationen (Öffnen, Umbenennen, Löschen, Verschieben, Hochladen) weitergegeben. Verzeichnisse werden vor Dateien angezeigt; innerhalb jeder Ebene erfolgt die Sortierung alphabetisch.

Einstiegspunkt/Änderungen: `src/webview/handlers/WorkspaceFileHandler.ts`, Funktion `buildWorkspaceTree`.

9 Erreichte Ziele / Resultate

9.1 Funktionale Anforderungen

9.1.1 Nicht erreichte Anforderungen

Von den 22 funktionalen Anforderungen konnte eine Anforderung nicht vollständig erfüllt werden: FA-E4 sieht vor, dass die Extension den Benutzer auf die Verwendung von nicht unterstützten Python-Standardbibliotheken hinweist. Die gewählte Stub-basierte Architektur bietet hierfür keine geeignete Grundlage, da Stubs lediglich Typinformationen für vorhandene MicroPython-Module bereitstellen, jedoch keine Mechanismen zur Erkennung von Bibliotheken bieten, die in MicroPython schlicht nicht existieren. Eine Umsetzung wäre theoretisch über eine statisch gepflegte Negativliste bekannter Standard-Python-Bibliotheken denkbar, was jedoch einen fragilen und wartungsintensiven Ansatz darstellt. Ein eigener Language Server hingegen könnte diesen Filter elegant abbilden, würde aber den Rahmen dieser Arbeit deutlich überschreiten. Da die Anforderung mit Priorität 2 eingestuft war, wurde sie bewusst zurückgestellt.

9.1.2 Erreichte Anforderungen

Alle übrigen 21 funktionalen Anforderungen wurden erfolgreich umgesetzt und durch die entsprechenden E2E-Tests verifiziert. Im Bereich der Verbindungsverwaltung (FA-V1 bis FA-V4) erkennt die Extension angeschlossene MicroPython-Boards automatisch, stellt Verbindungen her und trennt sie sauber, und informiert den Benutzer bei einem Verbindungsabbruch. Die vollständige Dateiverwaltung (FA-D1 bis FA-D7) umfasst Upload, Download, Löschen, Anzeige des Board-Dateisystems, direktes Ausführen von Dateien, Festlegen einer Datei als main.py sowie die Mount-Funktionalität via mpremote. Die integrierte Konsole (FA-I1 bis FA-I4) zeigt Programmausgaben und Fehlermeldungen in Echtzeit an, ermöglicht Benutzereingaben und unterstützt den gleichzeitigen Betrieb zweier Boards mit je einer eigenen Konsole. Soft-Reset (FA-R1), Syntax-Highlighting, Copilot-Kompatibilität, MicroPython-spezifische Autovervollständigung sowie die Bibliotheksinstallation direkt aus VS Code (FA-E1 bis FA-E3, FA-E5, FA-B1) sind ebenfalls vollständig implementiert und funktionsfähig.

9.2 Nicht-funktionale Anforderungen

9.2.1 Nicht erreichte Anforderungen

Eine nicht-funktionale Anforderung wurde teilweise eingeschränkt erfüllt: Bei NFA-R2 wurde festgestellt, dass die Upload-Fehlermeldung nach einem Verbindungsabbruch während einer laufenden Übertragung erst bei einem Neustart von VS Code verschwindet. Das Verhalten ist damit nicht vollständig wie spezifiziert, stellt jedoch keine kritische Beeinträchtigung des stabilen Zustands der Extension dar.

9.2.2 Erreichte Anforderungen

12 von 13 nicht-funktionalen Anforderungen wurden erfüllt. Die Extension ist ohne Vorkenntnisse installierbar (NFA-U1), alle zentralen Funktionen sind über die grafische Oberfläche erreichbar (NFA-U2), und der Verbindungsstatus ist jederzeit sichtbar (NFA-U3). Die Reliabilitätsanforderungen NFA-R1 und NFA-R3 sind erfüllt: Die Extension stürzt bei Verbindungsabbrüchen nicht ab und kehrt nach einem VS-Code-Neustart automatisch in einen einsatzbereiten Zustand zurück. Alle Performance-Anforderungen

(NFA-P1 bis NFA-P3) wurden eingehalten; Datei-Uploads unter 10 kB werden in unter fünf Sekunden abgeschlossen, der Verbindungsaufbau erfolgt in unter drei Sekunden, und Konsolenausgaben erscheinen ohne wahrnehmbare Verzögerung. Die Supportability-Anforderung NFA-S1 ist durch das vorhandene README.md sowie die MAINTAINER.md erfüllt. Die Kompatibilitätsanforderungen NFA-C1 bis NFA-C4 wurden auf allen drei unterstützten Boards (Nano ESP32, Nano RP2040 Connect, GIGA R1 WIFI), auf Windows und macOS sowie im Parallelbetrieb mit Pylance und GitHub Copilot erfolgreich verifiziert.

9.3 Zusammenfassung

Insgesamt wurden 21 von 22 funktionalen Anforderungen und 12 von 13 nicht-funktionalen Anforderungen vollständig erfüllt. Die einzige nicht umgesetzte funktionale Anforderung (FA-E4) betrifft die Erkennung nicht unterstützter Python-Standardbibliotheken und war mit Priorität 2 eingestuft. Die eingeschränkt erfüllte Reliabilitätsanforderung (NFA-R2) betrifft ein kosmetisches Problem mit einer Fehlermeldung und beeinträchtigt die Kernfunktionalität der Extension nicht.

Die Extension erfüllt damit alle als Priorität 1 definierten Kernfunktionen vollständig und bietet darüber hinaus sämtliche optionalen Erweiterungen der Stufen 2 und 3, einschliesslich des gleichzeitigen Betriebs mehrerer Boards. Die zwei Iterationen der Usability-Tests haben gezeigt, dass die Benutzeroberfläche von Einsteigern ohne Vorkenntnisse in MicroPython grundsätzlich als intuitiv und verständlich wahrgenommen wird. Identifizierte Usability-Probleme wurden iterativ behoben. Durch die Veröffentlichung auf dem Visual Studio Code Marketplace steht die Extension der breiteren MicroPython-Community zur Verfügung.

9.4 Vergleich mit bestehenden Extensions

Die entwickelte Extension wird nachfolgend in der bereits bekannten Bewertungsmatrix mit den drei zu Anfang analysierten Extensions verglichen. Die Bewertung der eigenen Extension stellt eine subjektive Selbsteinschätzung der Autoren dar und ist nicht durch unabhängige Tests Dritter validiert.

Kriterium	RT-Thread MicroPython	MicroPython IDE VSCode	Pymakr	MicroPython for Arduino
Installation & Einrichtung				
Einfachheit der Installation	4 / 5	1 / 5	4 / 5	4.5/5
Konfigurationsaufwand	3 / 5	2 / 5	3 / 5	5/5
Kernfunktionalität				
Datei-Upload & Ausführen	4 / 5	1 / 5	3 / 5	4.5/5
REPL-Konsole	4 / 5	1 / 5	4 / 5	5/5
Board Reset	1 / 5	1 / 5	5 / 5	5/5
Benutzerfreundlichkeit				
Konsistenz der UI- Elemente	3 / 5	2 / 5	3 / 5	4/5
Einsteigerfreundlichkeit	3 / 5	1 / 5	3 / 5	5/5
Stabilität & Wartung				
Stabilität im Betrieb	3 / 5	1 / 5	4 / 5	4/5
Community & Aktualität	3 / 5	1 / 5	4 / 5	3.5/5
Marketplace-Bewertung	4.5 / 5	2 / 5	2.5 / 5	NA/5
Ø Durchschnitt	3.25 / 5	1.3 / 5	3.55 / 5	4.5

Tabelle 19 - Bewertungsmatrix bestehende und entwickelte Extensions

	Gut (4-5 / 5)
	Mittel (3 / 5)
	Schwach (1-2 / 5)

Tabelle 20 - Legende Bewertungsmatrix

Wie in der Matrix zu sehen, wurde eine Extension entwickelt, welche die anderen übertrifft. Alle Bewertungspunkte sind im grünen Bereich, ausgenommen Community, da die Extension noch niemand kennt.

9.5 Ausblick und Erweiterungsmöglichkeiten

In diesem Kapitel sind sowohl konzeptionelle Erweiterungen als auch bekannte Verbesserungspotenziale beschrieben, die im Rahmen dieser Arbeit nicht mehr umgesetzt werden konnten.

9.5.1 Erweiterungen

Native Dateioperationen im Workspace

Der in der Webview implementierte Workspace-Dateibaum bietet aktuell grundlegende Funktionalität zur Dateiverwaltung. Eine sinnvolle Weiterentwicklung wäre die Angleichung des Verhaltens an den nativen VS Code Explorer: Dazu gehören kontextsensitive Rechtsklick-Menüs, Inline-Umbenennung von Dateien und Ordnern sowie einen gemeinsamen Kontext für aufgeklappte Ordner. Eine solche Anpassung würde die Nutzungserfahrung erheblich verbessern und die Lernkurve für bestehende VS Code-Nutzer reduzieren.

Commands und Tastenkürzel

Aktuell sind alle Funktionen ausschliesslich über die Webview-Oberfläche erreichbar. Eine wertvolle Erweiterung wäre die Registrierung von VS Code Commands für Aktionen, die kein grafisches Interface erfordern. So könnte beispielsweise ein Tastenkürzel definiert werden, das die aktuell im Editor geöffnete Datei direkt auf dem Board ausführt, ohne dass die Webview geöffnet sein muss. Dies würde insbesondere erfahrene Nutzer ansprechen und den Workflow beim iterativen Entwickeln und Testen von MicroPython-Skripten deutlich beschleunigen. Die Umsetzung erfolgt über Commands von VS Code und sollte keine grossen Implementationsschwierigkeiten bieten.

Debugging

Die aktuelle Version der Extension bietet keine dedizierte Debugging-Funktionalität. Die Fehlersuche erfolgt primär über die serielle Konsole sowie die Auswertung von Ausgaben und Fehlermeldungen des MicroPython-Interpreters. Dies ist für kleinere Projekte ausreichend, kann jedoch bei komplexeren Anwendungen die Analyse von Fehlern erschweren.

Die Integration einer Debugging-Funktionalität wurde im Rahmen dieser Arbeit nicht berücksichtigt, stellt jedoch eine vielversprechende Erweiterungsmöglichkeit für zukünftige Versionen der Extension dar.

9.5.2 Bekannte Verbesserungspotenziale

Verhalten beim Schliessen von VS Code mit aktivem Mount

Wird VS Code geschlossen, während ein Board über einen aktiven Mount-Prozess eingebunden ist, kommt es zu einem Fehlerzustand des Boards. Das Problem liegt in der Reihenfolge der Teardown-Operationen von VS Code: Terminals werden beendet, bevor die deactivate-Funktion der Extension aufgerufen wird. Da der Mount-Prozess im Terminal läuft, ist es nicht möglich, den Mount geordnet zu beenden und das Board in einen konsistenten Zustand zu versetzen.

Eine programmatische Lösung konnte im Rahmen dieser Arbeit nicht gefunden werden, da VS Code keine API zur Verfügung stellt, die ein geordnetes Herunterfahren vor dem Terminal-Kill ermöglicht.

Abhängigkeitsverwaltung von Libraries

Die aktuelle Implementierung der Library-Verwaltung weist ein bekanntes Problem bei der Behandlung geteilter Abhängigkeiten auf. Wird eine Library installiert, werden neu hinzugekommene Dateien und Verzeichnisse erfasst und im manifest.json unter dem Eintrag der jeweiligen Library gespeichert. Teilen zwei Libraries eine gemeinsame Abhängigkeit, beispielsweise lib1 und lib2, die beide lib3 benötigen, so wird lib3 nur im Eintrag der zuerst installierten Library geführt. Bei einer späteren **Deinstallation** von lib1 wird lib3 vom Board entfernt, obwohl lib2 weiterhin darauf angewiesen ist.

Ursache dieses Verhaltens ist der Diff-basierte Ansatz zur Erkennung neu installierter Dateien: Da lib2 beim zweiten Installationsvorgang bereits vorhanden ist, wird es nicht mehr als neu erkannt und dem Eintrag von lib3 nicht zugeordnet.

Eine korrekte Lösung erfordert ein referenzzählendes Abhängigkeitsmodell, das für jede Datei auf dem Board festhält, von welchen Libraries sie benötigt wird. Eine Deinstallation darf eine Datei nur dann entfernen, wenn keine weitere installierte Library mehr auf sie verweist.

Darüber hinaus besteht ein verwandtes Problem beim **Code-Support**:

Die automatische Generierung von Code-Support für installierte Libraries weist derzeit eine Einschränkung bei indirekten Abhängigkeiten auf. Installiert eine Library weitere Libraries als Abhängigkeiten, wird für diese automatisch installierten Libraries kein Code-Support erzeugt.

Beispielsweise installiert lib1 während der Installation zusätzlich die Abhängigkeit lib3. Die Extension generiert anschliessend den Code-Support für lib1, indem die Metadaten des zugehörigen GitHub-Repositories ausgewertet werden. Für lib3 liegen jedoch keine direkten Informationen vor, da die Library lediglich als Abhängigkeit installiert wurde und weder ein eigener GitHub-Link noch weitere Metadaten bekannt sind. Entsprechend wird kein Code-Support für lib3 erstellt.

Versucht der Benutzer später, lib3 explizit für die Code-Support-Generierung nachzuinstallieren, tritt ein weiteres Problem auf: Da die Dateien von lib3 bereits auf dem Board vorhanden sind, erkennt der aktuelle Diff-basierte Installationsmechanismus keine neuen Dateien oder Verzeichnisse. Folglich wird kein Eintrag im manifest.json erzeugt, wodurch die nachträgliche Generierung des Code-Supports ebenfalls ausbleibt.

Ursache des Problems ist, dass die aktuelle Implementierung installierte Dateien ausschliesslich anhand von Änderungen im Dateisystem erkennt und keine Informationen über die tatsächlichen Abhängigkeiten einer Library speichert. Eine langfristige Lösung würde eine explizite Verwaltung von Library-Abhängigkeiten erfordern, sodass auch indirekt installierte Libraries im Manifest erfasst und für die Generierung von Code-Support berücksichtigt werden können.

Unterstützung von Windows ARM64

Die Extension unterstützt Windows ARM64 derzeit nicht vollständig. Auf entsprechenden Systemen schlägt die Installation fehl. Als Workaround kann die x64-Version von Visual Studio Code verwendet werden. Ursache ist eine indirekte Abhängigkeit der Bibliotheken upy-package und micropython.js von der Bibliothek serialport. Für die Plattform win32-arm64 stehen aktuell keine kompatiblen Binärdateien zur Verfügung. Tests mit neueren Versionen von serialport ermöglichen zwar eine erfolgreiche Installation der Extension, die Kommunikation mit dem Board und damit die Kernfunktionalitäten der Extension bleiben jedoch nicht funktionsfähig. Eine native Unterstützung von Windows ARM64 ist daher zum aktuellen Zeitpunkt nicht gewährleistet.

10 Projektmanagement

Für das Projektmanagement wurde Jira eingesetzt. Das Tool diente sowohl zur Verwaltung der Anforderungen und Aufgaben mittels Tickets als auch zur Erfassung der im Projekt geleisteten Arbeitszeiten. Sämtliche Arbeitspakete wurden als Tickets erfasst und während der Umsetzung kontinuierlich gepflegt, wodurch der Fortschritt des Projekts jederzeit nachvollziehbar war.

Zur detaillierteren Auswertung der erfassten Arbeitszeiten wurde zusätzlich das Jira-Plugin Worklog360 verwendet. Dieses ermöglicht eine erweiterte Analyse der Zeiterfassung und bietet verschiedene Auswertungsmöglichkeiten, die über die Standardfunktionen von Jira hinausgehen. Dadurch konnten die aufgewendeten Zeiten für einzelne Projektphasen und Tätigkeiten genauer ausgewertet und dokumentiert werden.

10.1 Projektplan-SOLL

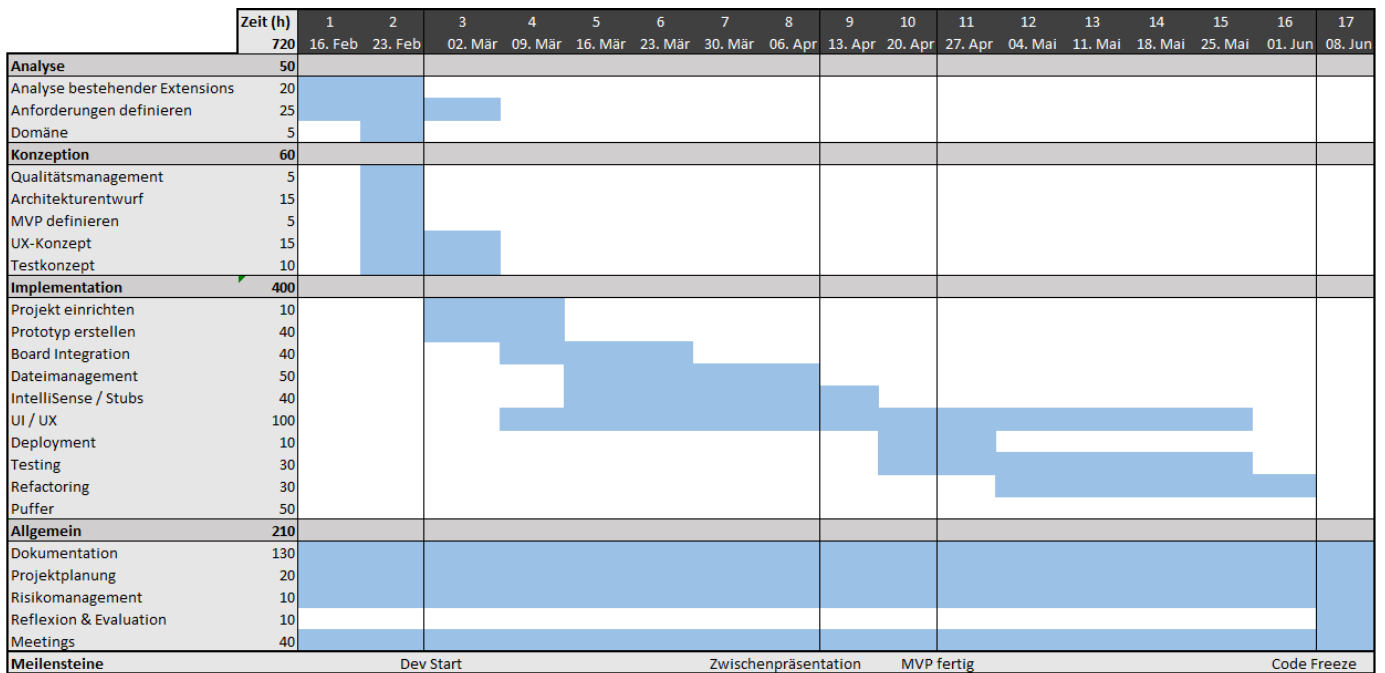


Abbildung 30 - Projektplan-SOLL

Zusätzlich relevante Daten:

Kickoff: 16.02.2026

Abgabe: 12.06.2026

Zwischenpräsentation: 16.04.2026

Abschlusspräsentation: 18.06.2026

10.2 Projektplan-IST

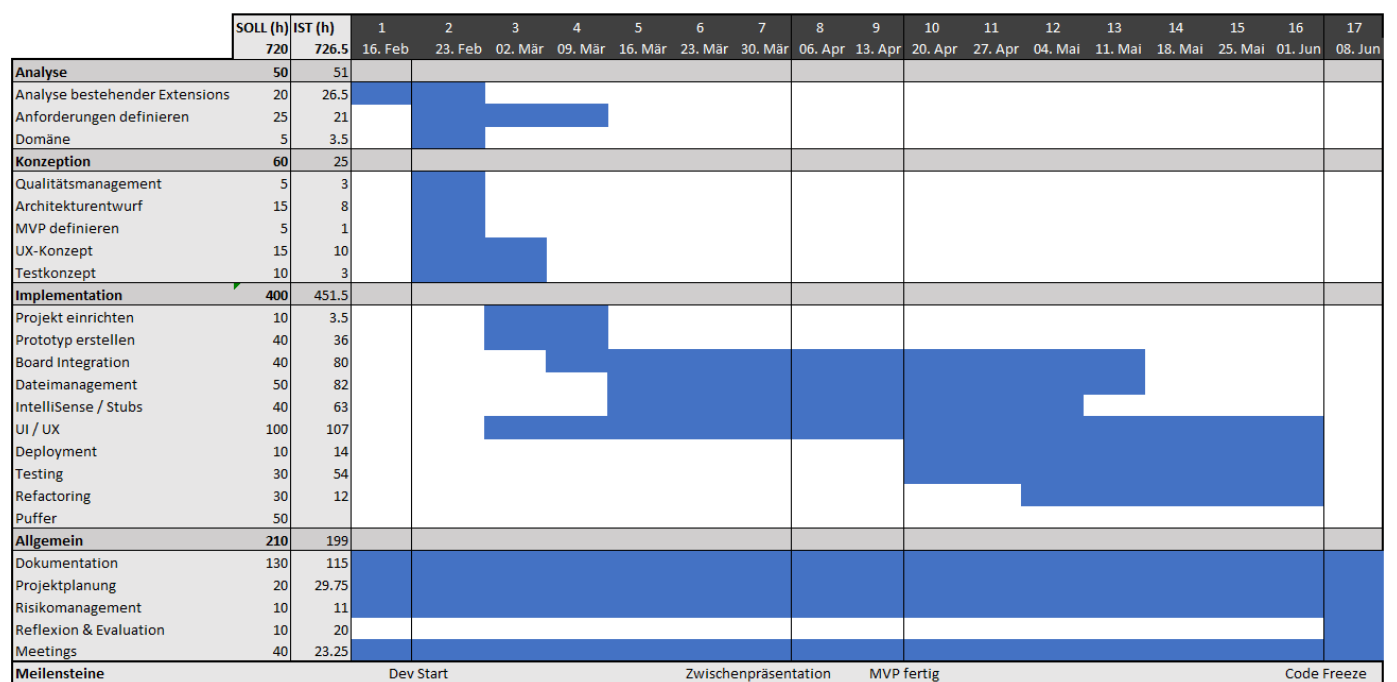


Abbildung 31 - Projektplan-IST

10.3 Zeiterfassung pro Person

Tabelle 21 - Stunden pro Woche

Woche	Quentin	Silvan
1	6	25.5
2	21	27.5
3	21.5	21
4	22.5	23.5
5	35.25	19.75
6	13.5	24.5
7	24	16.5
8	21	25
9	24	18
10	26	16.5
11	24	23
12	17	17
13	17	19
14	24.5	15
15	16	15
16	28.5	23
17	25	30
Total	366.75	359.75

10.4 Vergleich & Diskussion Zeitplan

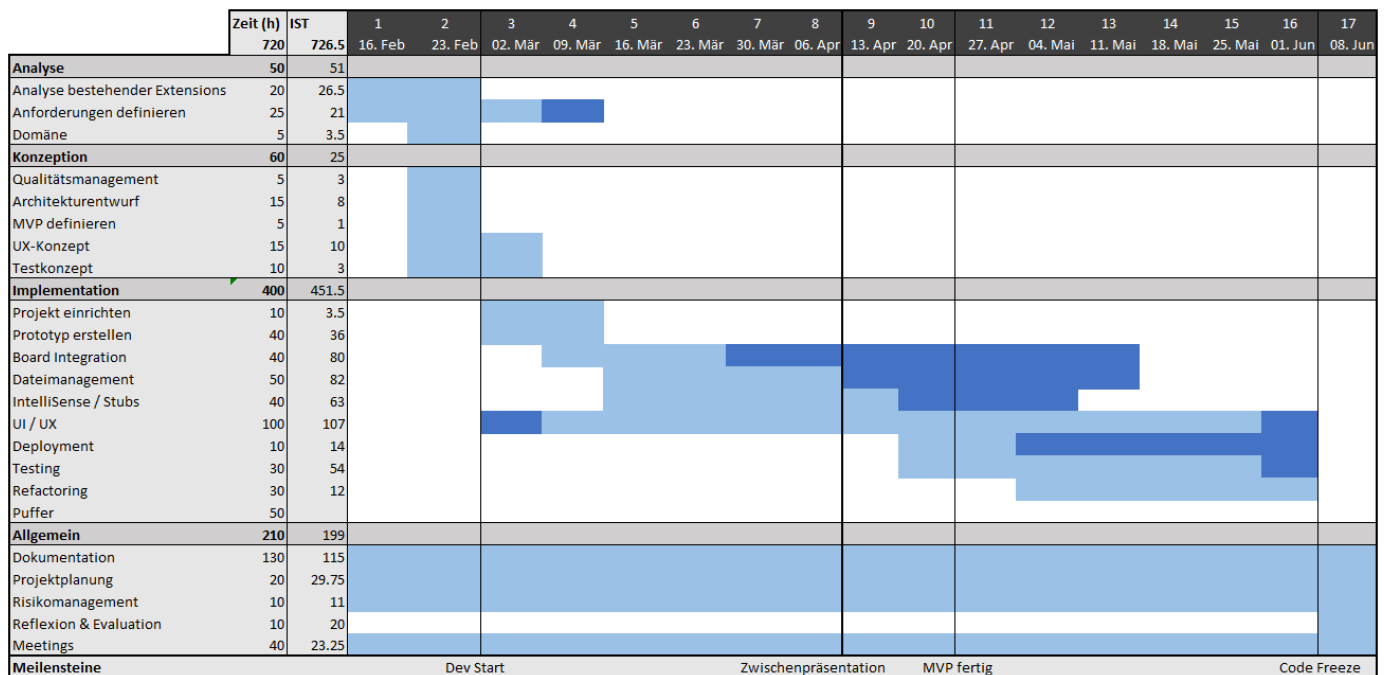


Abbildung 32 - Vergleich Projektplan IST - SOLL

Die Gegenüberstellung von Soll- und Ist-Zeitplan zeigt insgesamt eine solide Projektplanung, wobei sich in einzelnen Bereichen deutliche, aber nachvollziehbare Abweichungen ergeben haben. Die Gesamtprojektdauer von 726.5 Stunden lag knapp über der geplanten Zeit von 720 Stunden, was einer Abweichung von unter 1% entspricht und auf eine realistische Einschätzung des Gesamtumfangs hindeutet.

Die grössten relativen Abweichungen zeigen sich in der Konzeptionsphase: Mit einem Ist-Wert von 25 Stunden gegenüber 60 geplanten Stunden wurde hier deutlich weniger Zeit aufgewendet als vorgesehen. Es war durch die Aufgabenstellung bereits vorgegeben, dass eine VS Code Extension entwickelt werden soll, womit grundlegende Architekturentscheidungen bezüglich Plattform, Technologie und einige Architekturentscheidungen von vornherein feststanden und nicht erarbeitet werden mussten.

Die grössten Mehraufwände entstanden in der Implementierungsphase, welche mit 451.5 Stunden die geplanten 400 Stunden um mehr als 10% überschritt. Besonders auffällig sind dabei die Board Integration (80h statt 40h) sowie das Dateimanagement (82h statt 50h). Diese Überschreitungen sind massgeblich darauf zurückzuführen, dass im Projektverlauf entschieden wurde, das optionale Feature zur gleichzeitigen Verbindung mehrerer Boards zu implementieren. Da dieses Feature beide Bereiche grundlegend beeinflusst, wurden die entsprechenden Phasen nicht abgeschlossen, sondern bis zur Fertigstellung dieser Erweiterung offengehalten. Auch das Testing überschritt mit 54 Stunden den Sollwert von 30 Stunden erheblich, was auf den iterativen Charakter der Usability-Tests mit zwei Testdurchgängen zurückzuführen ist. Diese benötigten mehr Vor- und Nachbereitungszeit als erwartet.

Demgegenüber konnten im Bereich Allgemein mit 199 gegenüber 210 geplanten Stunden leichte Einsparungen erzielt werden. Insbesondere die Meetings (23.25h statt 40h) verliefen effizienter als geplant, da viele Abstimmungen kompakt gehalten werden konnten.

Der Puffer von 50 geplanten Stunden wurde nicht separat ausgewiesen, floss aber de facto in die Mehraufwände der Implementierungsphase ein und erfüllte damit seinen Zweck als Risikoreserve.

Insgesamt demonstriert die minimale Gesamtabweichung eine erfolgreiche Projektplanung und -durchführung. Die wesentlichen Meilensteine Dev Start, Zwischenpräsentation, MVP fertig und Code Freeze wurden im vorgesehenen zeitlichen Rahmen erreicht. Mehraufwände in technisch anspruchsvollen Implementierungsbereichen konnten durch Einsparungen in anderen Phasen und Puffer weitgehend kompensiert werden.

10.5 Risikomanagement

In diesem Kapitel sind die Risiken des Projekts erfasst, dabei wurden zu Beginn des Projekts 5 mögliche Risiken gefunden und eingeschätzt. Zusätzlich ist ein Log vorhanden, indem beschrieben ist, wie sich das Risiko durch Implementierungen oder neue Erkenntnisse verändert hat. Dabei werden zu jeder Änderung die Wahrscheinlichkeit (W) und der Schaden (S) des Risikos mit einem Wert von 0 (unmöglich, kein Schaden) bis 10 (Risiko eingetroffen, sehr grosser Schaden) bewertet.

10.5.1 Risiken Zusammenfassung

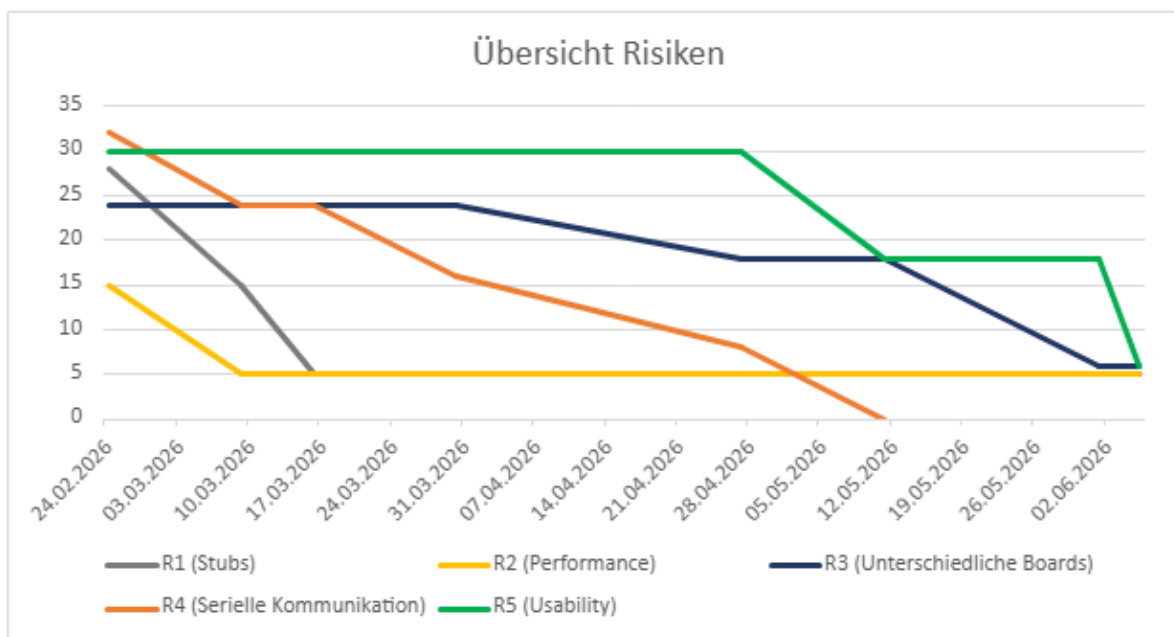


Abbildung 33 - Verlauf Risiken

Wie im Diagramm in Abbildung 33 ersichtlich, konnte nur eines der fünf definierten Risiken vollständig eliminiert werden. Insgesamt wurde die Risikobewertung jedoch von 129 auf 22 Risikopunkte reduziert. Die verbleibenden Restrisiken setzen sich aus den folgenden Punkten zusammen:

R1 (Unzureichende Stubs) besteht weiterhin, da mithilfe von Stubs nicht erkannt werden kann, ob Standard-Python-Bibliotheken verwendet werden, die in MicroPython nicht verfügbar sind. Diese Anforderung wird jedoch als weniger kritisch eingestuft und wird daher weggelassen.

R2 (Performance) kann nicht vollständig ausgeschlossen werden. Insbesondere sehr grosse Dateien, die auf das Board übertragen werden, sowie umfangreiche Skripte, die über die Mount-Konsole ausgeführt werden, können zu Performanceproblemen führen. Auf diese Szenarien hat die Extension jedoch nur begrenzten Einfluss.

R3 (Unterschiedliche Boards) ist grundsätzlich vollständig abgeschlossen. Die Entwicklung wurde mit zwei der drei vorgesehenen Boards durchgeführt. Das dritte Board wurde vom Praxispartner getestet und als funktionsfähig bestätigt. Weitere könnten funktionieren, es wird aber nichts garantiert.

Bei **R5 (Usability)** konnten zahlreiche Verbesserungen identifiziert und umgesetzt werden. Dennoch ist davon auszugehen, dass in einzelnen Anwendungsfällen weiterhin Herausforderungen bei der Bedienung auftreten können. Die Benutzerfreundlichkeit kann daher auch zukünftig weiter optimiert werden.

10.5.2 R1: Stubs sind nicht ausreichend

Es ist geplant, dass Stubs für die Auto Completion verwendet werden. Stubs werden gegenüber einem eigenen Language Server bevorzugt, da der Implementierungsaufwand deutlich geringer ist. Es besteht jedoch das Risiko, dass die Stubs nicht alle Anforderungen erfüllen können.

Risikolevel zu Beginn: Mittel

Risikolevel aktuell: Niedrig

Auswirkung: Wenn die Stubs nicht ausreichen, muss entweder auf die kritische Anforderung verzichtet werden oder ein Language Server implementiert werden, was einen grossen Zeitaufwand bedeutet.

Massnahmen: Die Anforderungen werden so früh wie möglich geprüft, ob sie mit Stubs umsetzbar sind. Es soll aber zusätzlich Puffer eingeplant werden, für den Fall, dass ein eigener Language Server verwendet werden muss.

Tabelle 22 - Risikolog Stubs

24.02.2026	Das Risiko wurde erfasst, als wir uns für Stubs und gegen einen eigenen Language Server entschieden haben.	W: 4 S: 7
09.03.2026	Es wurde ein Prototyp erstellt, welcher mit Stubs arbeitet und die ersten Versuche waren erfolgreich. spezifische Board-Libraries wurden von IntelliSense erkannt und bei einem Hover über die Codestelle wurden die entsprechende Info angezeigt. Offen ist noch, wie es für Sensor-Libraries gemacht werden kann.	W: 3 S: 5
16.03.2026	Die Stubs der Sensor-Libraries können über das GitHub Repository generiert werden. Das ermöglicht Autovervollständigung für nur auf dem Board installierte Libraries.	W: 1 S: 5

10.5.3 R2: Performance bei Datenübertragung

Die Extension sollte die Datenübertragung in einer Geschwindigkeit erledigen, die für die MicroPython Boards üblich ist. Der Nutzer sollte nicht das Gefühl haben, dass etwas fehlgeschlagen ist.

Risikolevel zu Beginn: Niedrig

Risikolevel aktuell: Niedrig

Auswirkung: Schlechte Performance beeinträchtigt die Benutzererfahrung und gefährdet die Akzeptanz der Extension. Zufriedene Nutzer sind entscheidend für eine langlebige Lösung.

Massnahmen: Die Performance soll konkurrenzfähig sein, kann jedoch sehr Zeitintensiv sein. Sollte die Performance problematisch werden, werden Optimierungen wie Fortschrittsanzeigen oder Übertragungsoptimierungen geprüft.

Tabelle 23 - Risikolog Performance

24.02.2026	Das Risiko wird als niedrig eingeschätzt, da die Übertragungsgeschwindigkeit primär durch die Hardware und das Protokoll bestimmt wird.	W: 3 S: 5
09.03.2026	Beim Erstellen des Prototyps wurden einige Hauptfunktionen implementiert und getestet. Dabei wurden keinerlei Probleme mit der Performance festgestellt.	W: 1 S: 5

10.5.4 R3: Unterschiedliches Verhalten von Boards und Versionen

Die Extension soll drei verschiedene Arduino Boards (Nano RP2040, Nano ESP32, GigaR1) unterstützen, die jeweils unterschiedliche MicroPython Versionen und boardspezifische Bibliotheken verwenden. Es besteht das Risiko, dass Features, die auf einem Board funktionieren, auf einem anderen Board abweichendes Verhalten zeigen oder nicht verfügbar sind.

Risikolevel zu Beginn: Mittel

Risikolevel aktuell: Niedrig

Auswirkung: Inkonsistentes Verhalten zwischen den Boards beeinträchtigt die Benutzererfahrung und gefährdet die Verlässlichkeit der Extension. Im schlimmsten Fall müssen boardspezifische Sonderlösungen implementiert werden, was den Entwicklungsaufwand erheblich erhöht.

Massnahmen: Es wird früh definiert, welche MicroPython Versionen offiziell unterstützt werden, um die Anzahl der zu testenden Kombinationen zu begrenzen. Jedes Feature wird auf allen drei Boards getestet, bevor es als abgeschlossen gilt.

Tabelle 24 - Risikolog Unterschiedliche Boards

24.02.2026	Das Risiko wurde erfasst, da drei Boards mit unterschiedlichen Chips und Firmware-Versionen unterstützt werden sollen. Die Kombination aus unterschiedlichen Boards und MicroPython Versionen macht eine vollständige Testabdeckung aufwändig.	W: 4 S: 6
27.04.2026	Alle Funktionen konnten ohne boardspezifische Anpassungen, sowohl für das Nano ESP32 als auch das RP2040 Connect, umgesetzt werden. Noch offen ist, ob auch das Giga R1 ohne Probleme funktioniert.	W: 3 S: 6
01.06.2026	Sebastian Romero hat die Extension getestet. Unter anderem auch mit dem Giga R1, wobei er keine Fehler gefunden hat. Daher sind die als unterstützt angegebenen Boards voll funktionsfähig und es ist davon auszugehen, dass auch weitere Boards funktionieren.	W: 1 S: 6

10.5.5 R4: Probleme bei serieller Kommunikation

Die Kommunikation zwischen der Extension und den Boards erfolgt über eine serielle Verbindung via REPL. Diese Kommunikation ist fehleranfällig und kann je nach Betriebssystem, Treiber-Version oder gleichzeitig geöffneten Anwendungen unterschiedlich reagieren. Es besteht das Risiko, dass Verbindungsabbrüche oder Timing-Probleme die Kernfunktionalität der Extension beeinträchtigen.

Risikolevel zu Beginn: Hoch

Risikolevel aktuell: Behoben

Auswirkung: Eine instabile serielle Kommunikation gefährdet die Grundfunktionalität der Extension, da alle Kernfeatures wie Dateiübertragung, Code Execution und Board-Erkennung auf einer stabilen Verbindung basieren. Für Anfänger sind unerklärliche Verbindungsfehler besonders frustrierend und können zur Ablehnung der Extension führen.

Massnahmen: Die serielle Kommunikation wird früh im Projekt als Prototyp auf allen drei Betriebssystemen (Windows, macOS, Linux) getestet. Verbindungsfehler sollen mit verständlichen Fehlermeldungen abgefangen werden, sodass der Nutzer nachvollziehen kann was schiefgelaufen ist.

Tabelle 25 - Risikolog serielle Kommunikation

24.02.2026	Das Risiko wurde erfasst, da die serielle Kommunikation die technische Basis der gesamten Extension bildet. Besonders das unterschiedliche Verhalten auf Windows gegenüber macOS und Linux wurde als potenzielle Fehlerquelle identifiziert.	W: 4 S: 8
09.03.2026	Beim Erstellen des Prototyps, wurden keine Stolpersteine mit der seriellen Kommunikation entdeckt. Die Verbindung zum Board konnte ohne Probleme hergestellt werden.	W: 3 S: 8
30.03.2026	Die Machbarkeit die REPL zweier Boards gleichzeitig zu öffnen, wurde erfolgreich getestet. Jedoch ist die Architektur der Extension nicht darauf aufgelegt und es müssten einiges umgebaut werden, um dies in unserer Extension zu ermöglichen.	W: 2 S: 8
27.04.2026	Es wurden alle Funktionalitäten eingebaut. Es ist nun ebenfalls möglich mehrere Boards zur gleichen Zeit anzusteuern.	W: 1 S: 8
11.05.2026	Die Usability-Tests wurden auf Windows, macOS und Linux durchgeführt. Auf allen Betriebssystemen gab es keine Probleme mit der seriellen Kommunikation.	W: 0 S: 8

10.5.6 R5: Mangelnde Benutzerfreundlichkeit durch unklare Interaktion

Es besteht das Risiko, das die Benutzeroberfläche nicht intuitiv verständlich ist und Nutzer Schwierigkeiten haben, Funktionen zu finden, korrekt zuzuordnen und zu bedienen. Ursachen können unter anderem unklare Navigation, fehlleitende Bezeichnungen oder fehlendes Feedback sein.

Risikolevel zu Beginn: Hoch

Risikolevel aktuell: Niedrig

Auswirkung: Wenn die Interaktionen, die durch die Extension angeboten werden, nicht intuitiv sind, haben neue Nutzer eine erhöhte Fehlerquote bei der Bedienung. Dadurch brauchen sie länger, um Aufgaben umzusetzen oder brechen die Nutzung ab und suchen nach anderen Extensions. Dies führt zu einer geringen Zufriedenheit und Akzeptanz der Anwendung in der MicroPython Community.

Massnahmen: Die Benutzeroberfläche soll konsistent und klar gestaltet werden. Dabei sollen verständliche Begriffe und Icons verwendet werden. Aktionen sollen sichtbar ausgeführt werden. Um die Verständlichkeit zu testen und optimieren werden iterativ Usability-Tests für echtes Nutzerfeedback durchgeführt.

Tabelle 26 - Risikolog Benutzerfreundlichkeit

24.02.2026	Das Risiko wurde erfasst, da die Nutzerfreundlichkeit einen grossen Einfluss auf den Erfolg der Extension hat. Dadurch müssen Interaktionen klar verständlich und intuitiv gestaltet werden.	W: 5 S: 6
11.05.2026	Es wurde die erste Iteration an Usability-Tests durchgeführt. Das UI war grundsätzlich verständlich, es konnten aber noch einiges an Verbesserungsmöglichkeiten erkannt werden. Diese werden in der nächsten Iteration getestet.	W: 3 S: 6
05.06.2026	Auch die zweite Iteration an Usability-Tests wurde durchgeführt. Es konnten weitere kleine Verbesserungen am UI gemacht werden, welche die erste Bedienung erleichtern. Zusätzlich wurde die Extension von Sebastian Romero getestet und wir konnten Feedback von einem erfahrenen MicroPython Entwickler umsetzen.	W: 1 S: 6

11 Qualitätsmanagement

11.1 Regeln

In diesem Kapitel sind die Definitionen zu finden, die während der Entwicklung eingehalten werden. Dabei geht es um die Definition of Ready, Definition of Done, Code Guidelines und Regeln zur Verwendung von GitLab.

11.1.1 Definition of Ready

Ein Ticket kann begonnen werden, sobald es:

- Im Sprint Backlog liegt
- Mit Story Points geschätzt wurde
- Keine offenen Abhängigkeiten hat

11.1.2 Definition of Done

Ein Ticket kann abgeschlossen werden, sobald es:

- Die angegebenen Aufgaben erfüllt
- Tests dafür geschrieben wurden (sofern möglich)
- Die GitLab Pipeline durchläuft (Tests und ESLint)
- Die Anwendung fehlerfrei läuft
- Die benötigte Zeit auf dem Ticket erfasst wurde
- Folge-Tickets erstellt wurden

11.1.3 Code Reviews

Bei jedem Merge eines Feature-Branches in den main-Branch wird der Code jeweils von der anderen Person überprüft. Dadurch wird sichergestellt, dass alle Änderungen von einer zweiten Person kontrolliert werden. Der Fokus der Reviews liegt auf der Codequalität, der Lesbarkeit, der Einhaltung der Projektstruktur sowie auf möglichen Verbesserungen.

11.1.4 Code Guidelines

Als Basis wurden die Google TS Richtlinien genommen. [7] Daraus leiten sich unsere eigenen Richtlinien ab:

- Vermeide doppelten Code - Funktionen sollten wiederverwendbar gestaltet werden.
- Behandle Fehlerfälle immer angemessen.
- Halte dich an die bestehenden Code-Konventionen des Projekts.
- Schreibe Tests für deinen Code.
- Vermeide den Typ any
- Namenskonventionen:
 - Variablen und Funktionen: camelCase
 - Klassen: PascalCase
 - Dateien und Ordner: kebab-case
- JSDoc wird erstellt für Komponenten und grössere Funktionen

11.1.5 Versionskontrolle (GitLab)

- Es soll kein Code gepusht werden, der nicht funktioniert oder die Pipeline nicht besteht.
- Sollte die Pipeline fehlschlagen, ist dies so schnell wie möglich zu korrigieren.
- Erstelle immer einen neuen Branch, ausser Änderungen am main-Branch wurden explizit im Team abgesprochen.

11.2 GitLab Pipeline

Die CI/CD-Pipeline wird verwendet, um Qualitätsmessungen automatisiert durchzuführen. Dadurch erhält der Entwickler ein direktes Feedback, nachdem er seine Änderungen pusht.

Die Pipeline enthält die drei Stages build, test und metrics. In der ersten Stage werden die Abhängigkeiten (dependencies) des Projektes installiert. Diese werden von den anderen Stages benötigt.

11.2.1 Unit-Tests

In der zweiten Stage test werden die Unit Tests ausgeführt. Diese sollen die Logik der Applikation testen. Damit überprüft werden kann, ob genügend Tests geschrieben wurden, gibt die Pipeline einen Coverage-Report aus, der angibt, zu welchem Prozentsatz die einzelnen Klassen durch Tests abgedeckt sind.

11.2.2 JSCPD

In der dritten Stage wird die Qualität des Codes geprüft. Dabei wird zum einen der TypeScript Copy/Paste Detector (jscpd) verwendet. Dieser prüft den Code auf identische Codezeilen und gibt ebenfalls in der GitLab CI einen Report aus. Mit diesem Report können doppelte Codestellen erkannt und zentralisiert werden. Dadurch soll die Qualität des Codes verbessert werden.

11.2.3 ESLint

Zusätzlich läuft ESLint über den TypeScript Code und prüft den Code auf Unschönheiten. So können beispielsweise unbenutzte Variablen erkannt und entfernt werden. Falls ESLint mindestens eine Stelle findet, schlägt der Pipeline Job fehl und der Entwickler erhält eine entsprechende E-Mail. Dadurch können diese Unschönheiten schnell entfernt werden und der Code bleibt konsistent und gut wartbar.

11.3 Deployment

Da es sich um eine VS Code Extension handelt, unterscheidet sich das Deployment grundlegend von klassischen Web-Applikationen. Es gibt zwei relevante Szenarien: die lokale Entwicklungsinstallation und die öffentliche Veröffentlichung.

11.3.1 Lokale Installation (Entwicklung & Testing)

Während der Entwicklung wird die Extension direkt über VS-Code als Extension Development Host gestartet (F5). Für die Weitergabe an Testpersonen im Rahmen der Usability-Tests wird die Extension als .vsix-Paket gebündelt.

Das Paket kann manuell über "Install from VSIX..." in VS Code installiert werden. Dieses Vorgehen wird für alle Usability-Tests verwendet, um eine stabile, reproduzierbare Testumgebung sicherzustellen.

11.3.2 Veröffentlichung (VS-Code Marketplace)

Die Veröffentlichung der Extension erfolgt automatisiert über einen GitHub Workflow. Bei jedem Release wird der Package-Prozess ausgeführt und die Extension anschliessend automatisch im Visual Studio Marketplace publiziert. Hierfür wird das Werkzeug «vsce» verwendet sowie ein registrierter Publisher-Account bei Microsoft benötigt. Durch die Automatisierung des Veröffentlichungsprozesses wird sichergestellt, dass neue Versionen konsistent und ohne manuelle Eingriffe bereitgestellt werden können.

11.4 Testkonzept

Dieses Kapitel beschreibt die Teststrategie der entwickelten VS-Code-Extension. Es werden die eingesetzten Testarten, deren Zielsetzung sowie deren Beitrag zur Sicherstellung einer hohen Softwarequalität dargestellt.

11.4.1 Testpyramide

Die Teststrategie orientiert sich an der bekannten Testpyramide [8], die eine Gewichtung zugunsten schneller, isolierter Tests empfiehlt:

- **E2E (15%)**
Manuelles Testen der Workflows und NFRs
- **Integration (25%)**
Automatisierte Tests mit Dateisystem und VS-Code-API
- **Unit (60%)**
Automatisierte Tests für reine Kernlogik

11.4.2 Unit-Tests

Das Ziel der Unit-Tests ist die isolierte Überprüfung einzelner Funktionen und Klassen. Es werden die folgenden Komponenten durch Unit-Tests abgedeckt:

- Validierungsfunktionen
- Fehlerbehandlungslogik
- Command-Logik ohne externe Abhängigkeiten
- Logik zur Auswahl und Verwaltung des aktiven Boards
- Generierung und Verarbeitung von Dateien
- State-Management
- Edge-Cases

Externe Abhängigkeiten wie Dateisystemzugriffe, VS-Code-API und Boards werden mit Mocks simuliert und geprüft. Bei den Unit-Tests geht es darum die Kernlogik der Extension vollständig deterministisch und reproduzierbar zu testen.

11.4.3 Integration-Tests

Integration-Tests prüfen das Zusammenspiel der Extension-Komponenten innerhalb von VS-Code. Dabei werden VS-Code-API-Aufrufe und Hardware gemockt und Dateisystemzugriffe real getestet. Dabei werden folgende Punkte getestet:

- Zugriff auf Workspace-Ordner und Dateien
- Zusammenspiel von Commands und Dateiverarbeitung

11.4.4 Usability-Tests

Neben der Testpyramide werden auch Usability-Tests eingesetzt. Diese dienen zur Bewertung der Benutzerfreundlichkeit und Verständlichkeit der Extension. Zur Testgruppe gehören Personen mit Erfahrung im Bereich der Softwareentwicklung. Die Extension soll für jedermann zugänglich sein und dazu werden Aufgaben definiert, welche durch die Testpersonen absolviert werden. Durch das Verhalten der Testpersonen sollen Unklarheiten im UI und bei Fehlermeldungen erkannt werden. Die Testaufgaben werden zeitlich erfasst und dokumentiert. Zusätzlich werden die Fehlversuche und Rückfragen protokolliert, um die Benutzerfreundlichkeit quantitativ zu bewerten. Durch gezielte Verbesserungen soll die Effizienz der Arbeitsabläufe und die Lernkurve bei der Erstbenutzung verbessert werden.

Die genauen Ergebnisse und Testszenarien können im Anhang nachgelesen werden.

11.4.5 Usability-Tests Ergebnisse – Erste Iteration

An der ersten Iteration nahmen drei Testpersonen (P1–P3) mit unterschiedlichem Erfahrungshintergrund teil. Die Kenntnisse in VS Code, MicroPython und Programmierung variierten bewusst, um ein breites Einsteigerprofil abzudecken.

Erkannte Problemfälle:

- Doppelklick auf Board Datei: nicht intuitiv
- Während Mount aktiv, werden andere Funktionalitäten ausgegraut, war aber unklar weshalb
- Viele Primary Buttons überall verteilt, Mount wird schnell übersehen
- Unbekannte Funktionalitäten (Mount)

Usability Verbesserungen:

- Status des Boards wurde dauerhaft oben sichtbar gemacht
- Readme klarer beschreiben und mit Videos (Doppelklick Board Datei und Mount)
- Install Library wurde in zusammengeklapptes Menü gepackt (damit per default weniger Buttons)

Usability unabhängige Bugs:

- Probleme bei der Installation mit Python Pfad
- Upload einer temporären Board-Datei funktioniert nicht richtig, wenn Autosave nicht aktiv ist
- Auf MacOS gab es keine Tooltips
- Leere Connection Vorschläge auf MacOS

Im Abschlussgespräch wurde die Extension im Vergleich zu bestehenden Tools grundsätzlich positiv bewertet. Als häufigster Wunsch wurde eine Getting-Started-Dokumentation bzw. eine README mit Screenshots oder Videos genannt.

11.4.6 Usability-Tests Ergebnisse – Zweite Iteration

Die zweite Iteration wurde ebenfalls mit drei Testpersonen (P1–P3) durchgeführt. Im Vergleich zur ersten Iteration war das Profil homogener und deutlich einsteignäher: Alle drei Personen hatten kaum oder keine Vorkenntnisse in VS Code, MicroPython und Arduino. Lediglich P3 verfügte über solide Programmierkenntnisse. Zusätzlich wurde die Extension von Praxispartner Sebastian Romero getestet, wodurch weiteres Usability-Feedback einfluss.

Erkannte Problemfälle:

- Upload: Blick muss springen
- Informationssymbole (Board Status / Mount) in White Mode kaum erkennbar
- Unklare Benennungen (z.B. Add Connection ist nicht direkt klar, dass es um eine Verbindung mit dem Board geht)
- Readme enthielt 4 Videos, es wurde aber maximal eins konsumiert
- Doppelklick auf Board Datei (Nicht bekannt, wenn README nicht gelesen)

Usability Verbesserungen:

- Board Aktionen wurden dauerhaft oben sichtbar gemacht
- Tooltip für öffnen von Board Dateien
- Komprimierbare Version der Board Aktionen (Icon-only Buttons)
- Readme mit nur einem Setup Guide Video, Funktionalitäten Beschreibung reicht
- Umbenennen von unklaren Bezeichnungen und Entfernen von verwirrenden Beschreibungen
- Informationssymbole und Button-Icons immer in Vordergrundfarbe

Usability unabhängige Bugs:

- Installation mit Windows ARM64 nicht möglich
- Miniconda Installation führt zu Fehler

Im Abschlussgespräch wurde die Extension grundsätzlich positiv und als intuitiv bedienbar bewertet. Die Schwierigkeiten der Testpersonen waren häufig auf fehlendes Programmier- und Arduino-Grundlagenwissen zurückzuführen, nicht auf die Extension selbst. P3 schlug vor das Info-Icon links von den Buttons zu platzieren, da dies der natürlichen Leserichtung besser entspricht.

Auch Sebastian Romero konnte die Extension selbständig nutzen und hat uns zusätzliche Änderungsvorschläge gegeben, die ihm das Arbeiten mit der Extension erleichtern:

- Mounten eines Subfolders
- Ausführen von Dateien, die nicht im gemounteten Ordner sind
- Aktualisieren des Workspace Dateibaums, wenn durch KI oder im Explorer eine Datei erstellt oder gelöscht wurde
- Wenn auf den Reset Button auf dem Board gedrückt wird, soll die Verbindung getrennt werden

Auch diese Änderungen, konnten umgesetzt werden und führen zu einer besseren Usability, auch für erfahrene MicroPython Entwickler.

11.4.7 E2E-Tests

Das Ziel der E2E-Tests ist es eine vollständige Testung des Systems durchzuführen. Aufgrund der Hardware-Interaktion (Boards) werden E2E-Tests manuell durchgeführt und dokumentiert. Bei den E2E-Tests liegt die Verwendung des Boards sowie das Zusammenspiel mit VS-Code und dem Dateisystem im Mittelpunkt. Alle E2E-Tests werden vor Releases durchgespielt, um die Funktionalität der Anforderungen und das Erfüllen der NFR sicherzustellen.

11.4.8 End-to-End Tests Ergebnisse

Die folgenden End-to-End Tests sind manuell am realen System durchzuführen. Jeder Test referenziert eine funktionale oder nicht-funktionale Anforderung. Die Spalte «Testschritte» beschreibt die konkrete Durchführung; «Erwartetes Ergebnis» definiert das Akzeptanzkriterium.

11.4.8.1 Verbindungsverwaltung

Tabelle 27 - E2E Verbindungsverwaltung

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-FA-V1	FA-V1	VS Code geöffnet, MicroPython-Board via USB angeschlossen.	1. Extension aktivieren. 2. Verbindungspanel öffnen. 3. Verfügbare Geräte prüfen.	Das angeschlossene Gerät erscheint in der Geräteliste der Extension.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-V2	FA-V2	Extension aktiv, Gerät sichtbar in der Geräteliste.	1. Gerät in der Liste auswählen. 2. «Verbinden» klicken.	Statusleiste zeigt «Verbunden». Keine Fehlermeldung erscheint.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-V3	FA-V3	Aktive Verbindung zu einem Gerät vorhanden.	1. «Trennen» klicken.	Statusleiste zeigt «Nicht verbunden». Verbindung ist sauber beendet.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-V4	FA-V4	Aktive Verbindung zu einem Gerät vorhanden.	1. USB-Kabel während aktiver Verbindung abziehen.	Extension zeigt eine verständliche Fehlermeldung. Kein Absturz der Extension.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -

11.4.8.2 Dateiverwaltung

Tabelle 28 - E2E Dateiverwaltung

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-FA-D1	FA-D1	Verbindung aktiv. Lokale .py-Datei vorhanden.	1. Datei im Explorer auswählen. 2. «Auf Board	Datei erscheint im Dateisystem des Boards. Keine Fehlermeldung.	Durchführung: Silvan Bollhalder, 4.6.2026

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
			hochladen» klicken.		Bemerkung: -
E2E-FA-D2	FA-D2	Verbindung aktiv. Datei auf dem Board vorhanden.	1. Datei im Board-Explorer auswählen. 2. «Herunterladen» klicken. 3. Zielordner wählen.	Datei ist lokal im gewählten Ordner gespeichert und inhaltlich identisch.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-D3	FA-D3	Verbindung aktiv. Datei auf dem Board vorhanden.	1. Datei im Board-Explorer auswählen. 2. «Löschen» klicken. 3. Bestätigen.	Datei ist nicht mehr im Dateisystem des Boards vorhanden.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-D4	FA-D4	Verbindung aktiv. Board enthält mindestens eine Datei.	1. Board-Explorer öffnen.	Alle Dateien und Verzeichnisse auf dem Board werden korrekt aufgelistet.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-D5	FA-D5	Verbindung aktiv. Python-Datei auf dem Board vorhanden.	1. Datei im Board-Explorer auswählen. 2. «Ausführen» klicken.	Datei wird auf dem Board ausgeführt. Ausgabe erscheint in der Konsole.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-D6	FA-D6	Verbindung aktiv. Python-Datei auf dem Board vorhanden.	1. Datei auswählen. 2. «Als main.py festlegen» klicken.	Datei ist als main.py auf dem Board gespeichert und startet beim nächsten Reset.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: Datei muss umbenannt werden.
E2E-FA-D7	FA-D7	Verbindung aktiv. Lokale Dateien vorhanden.	1. Mount drücken. 2. Lokale Datei	Die gespeicherte Datei ist direkt auf dem Board aktualisiert.	Durchführung: Silvan Bollhalder, 4.6.2026

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
			öffnen und ändern.		Bemerkung: -

11.4.8.3 Integrierte Konsole

Tabelle 29 - E2E Integrierte Konsole

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-FA-I1	FA-I1	Verbindung aktiv. Programm gibt Text aus (print('Hello')).	1. Programm starten. 2. Konsolenpanel beobachten.	Ausgabe erscheint in Echtzeit in der integrierten Konsole.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-I2	FA-I2	Verbindung aktiv. Programm enthält einen Laufzeitfehler. (0 Division)	1. Fehlerhaftes Programm starten. 2. Konsolenpanel beobachten.	Fehlermeldung (Traceback) wird vollständig in der Konsole angezeigt.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-I3	FA-I3	Verbindung aktiv.	1. Konsolenpanel fokussieren. 2. Python-Ausdruck eingeben (1+1). 3. Enter drücken.	Ergebnis (2) wird in der Konsole ausgegeben.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-I4	FA-I4	Zwei unterstützte Boards angeschlossen.	1. Auf beiden Boards die Konsole öffnen. 2. Skript zur Nachrichtenübertragung ausführen und Konsolenausgaben beobachten	Nachrichten werden gesendet und empfangen. Jede REPL-Konsole zeigt nur die jeweiligen Ausgaben an.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: Übertragung nicht getestet, parallele Ausführung von Skripts ist jedoch möglich und korrekt

11.4.8.4 Reset

Tabelle 30 - E2E Reset

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-FA-R1	FA-R1	Verbindung aktiv. Programm läuft auf dem Board.	1. «Soft Reset» klicken.	Board wird neugestartet. Konsole zeigt den Reset-Hinweis. Verbindung bleibt bestehen.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -

11.4.8.5 Editor-Integration

Tabelle 31 - E2E Editor-Integration

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-FA-E1	FA-E1	VS Code geöffnet. Python-Datei (.py) geöffnet.	1. Python-Datei öffnen. 2. Syntax (Schlüsselwörter, Strings, Kommentare) prüfen.	Schlüsselwörter, Strings und Kommentare sind farblich hervorgehoben.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-E2	FA-E2	GitHub Copilot installiert und aktiv. Python-Datei geöffnet.	1. Code tippen. 2. Copilot-Vorschläge beobachten.	Copilot-Vorschläge erscheinen ohne Konflikte oder Fehlermeldungen durch die Extension.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-E3	FA-E3	Extension aktiv. Python-Datei geöffnet.	1. «import machine» eingeben. 2. «machine.» tippen und Autovervollständigung abwarten.	Board-spezifische Methoden der machine-Bibliothek werden als Vorschläge angezeigt.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-FA-E4	FA-E4	Extension aktiv. Python-Datei geöffnet.	1. «import os» eingeben (nicht unterstützt in MicroPython).	Extension zeigt eine Warnung, dass diese Bibliothek auf MicroPython nicht verfügbar ist.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: Funktioniert nicht

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-FA-E5	FA-E5	Board verbunden und dokumentierte Library installiert.	1. Library in neuer .py importieren 2. Funktionsnamen tippen und vervollständigen lassen.	Intellisense zeigt Autovervollständigung und Funktionsinfos	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -

11.4.8.6 Bibliotheksverwaltung

Tabelle 32 - E2E Bibliotheksverwaltung

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-FA-B1	FA-B1	Verbindung aktiv. Internetverbindung vorhanden.	1. Library-Panel öffnen. 2. Bibliothek suchen (z. B. urequests). 3. «Installieren» klicken.	Bibliothek wird auf dem Board installiert und erscheint im Dateisystem.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -

11.4.8.7 Usability

Tabelle 33 - E2E Usability

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-U1	NFA-U1	Frische VS-Code-Installation ohne Extension.	1. Installation der .vsix-Datei durchführen. 2. Zeit und Probleme protokollieren.	Extension kann ohne Hilfe in unter 5 Minuten installiert werden.	Durchführung: Usability Tests Bemerkung: -
E2E-U2	NFA-U2	Extension installiert und aktiv.	1. Kernfunktionen alle suchen 2. Keine Anleitung verwenden.	Alle Kernfunktionen (Verbinden, Datei hochladen, REPL öffnen, Reset, Trennen) werden von der Testperson ohne Dokumentation innerhalb von 5	Durchführung: Usability Tests Bemerkung: -

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
				Minuten über die UI gefunden.	
E2E-U3	NFA-U3	Extension aktiv.	1. Board anschliessen und verbinden. 2. Statusleiste beobachten. 3. Board trennen und Statusleiste beobachten.	Statusleiste wechselt innerhalb von 2 Sekunden auf «Verbunden» bzw. «Nicht verbunden».	Durchführung: Usability Tests Bemerkung: Eigener Status wurde eingebaut

11.4.8.8 Reliability

Tabelle 34 - E2E Reliability

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-R1	NFA-R1	Verbindung aktiv.	1. USB-Kabel während Betrieb abziehen. 2. Extension beobachten.	Extension stürzt nicht ab. Zustand bleibt stabil. Fehlermeldung wird angezeigt.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-R2	NFA-R2	Dateiübertragung läuft.	1. USB-Kabel während Übertragung abziehen.	Übertragung wird sauber abgebrochen. Benutzer wird informiert. Keine hängenden Prozesse.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: Upload Meldung verschwindet erst bei Neustart von VS Code
E2E-R3	NFA-R3	Extension konfiguriert, Board angeschlossen.	1. VS Code vollständig schliessen. 2. VS Code neu starten. 3. Extension prüfen.	Extension ist ohne manuelle Konfiguration direkt einsatzbereit.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -

11.4.8.9 Performance

Tabelle 35 - E2E Performance

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-P1	NFA-P1	Verbindung aktiv. Kleine .py-Datei (< 10 kB) bereit.	1. Datei hochladen. 2. Zeit vom Klick bis zur Bestätigung messen.	Upload dauert unter 5 Sekunden.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-P2	NFA-P2	Extension aktiv. Board angeschlossen.	1. «Verbinden» klicken. 2. Zeit bis zur Bestätigung messen.	Verbindungsaufbau dauert unter 3 Sekunden.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-P3	NFA-P3	Verbindung aktiv. Programm gibt kontinuierlich Daten aus.	1. Programm starten. 2. Ausgaben in der Konsole mit Stoppuhr beobachten.	Ausgaben erscheinen ohne wahrnehmbare Verzögerung (< 200 ms nach Ausgabe auf dem Board).	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -

11.4.8.10 Supportability

Tabelle 36 - E2E Supportability

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-S1	NFA-S1	VS-Code ohne Extension	Extension nach Anleitung installieren	Extension korrekt ohne weitere Hilfe installiert	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -

11.4.8.11 Compatibility

Tabelle 37 - E2E Compatibility

Test-ID	Ref.	Vorbedingung	Testschritte	Erwartetes Ergebnis	Ergebnis
E2E-C1	NFA-C1	Alle drei Boards mit MicroPython und Extension installiert	1. Erstes Board anschliessen und Verbindung herstellen. 2. Datei hochladen und ausführen. 3. Schritte 1–2 für die verbleibenden Boards wiederholen.	Alle drei Boards werden erkannt und die Datei wird auf jedem Board erfolgreich ausgeführt.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-C2	NFA-C2	VS-Code mit aktivierten Extensions GitHub Copilot und Pylance. Board angeschlossen.	1. Verbindung zum Board herstellen. 2. Alle Kernfunktionen der Extension durchführen. 3. Copilot-Vorschlag in einer Python-Datei auslösen.	Alle Funktionen der Extension arbeiten fehlerfrei; Copilot und andere Extensions bleiben ohne Konflikte voll funktionsfähig.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-C3	NFA-C3	Board mit aktueller MicroPython Version. Extension installiert	1. Verbindung zum Board herstellen. 2. Datei hochladen und ausführen. 3. Soft-Reset auslösen und Konsolenausgabe prüfen.	Alle Funktionen arbeiten korrekt mit der aktuellen Firmware zusammen.	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -
E2E-C4	NFA-C4	Windows und Mac System mit installierter Extension	Alle relevanten Funktionen der Extension testen auf beiden Systemen	Alle Funktionen arbeiten auf beiden Betriebssystemen fehlerfrei	Durchführung: Silvan Bollhalder, 4.6.2026 Bemerkung: -

11.5 Code Patterns

In diesem Kapitel sind die im Projekt verwendeten Code Patterns beschrieben und wie sie umgesetzt wurden.

11.5.1 Message-Gateway zwischen Webview und Extension

Die Anwendung besteht aus zwei getrennten Bereichen: der VS-Code-Extension selbst und der Webview-Oberfläche in der Sidebar. Diese beiden Bereiche können nicht direkt wie normale Objekte miteinander kommunizieren, sondern tauschen Nachrichten aus.

Dafür gibt es einen zentralen Nachrichtenvertrag. Darin ist festgelegt, welche Nachrichtentypen die Webview an die Extension senden darf und welche Antworten die Extension zurückgeben kann. Ein Beispiel wäre: Die Webview sendet eine Nachricht wie «lade die Board-Dateien», und die Extension antwortet mit einer Liste der gefundenen Dateien.

Das WebviewGateway übernimmt dabei die Rolle eines Verteilers. Es empfängt eingehende Nachrichten, schaut anhand des Nachrichtentyps nach, welcher Handler zuständig ist, und ruft diesen auf. Dadurch muss nicht jede Komponente selbst wissen, wie Nachrichten technisch verarbeitet werden. Die eigentliche Registrierung der verfügbaren Nachrichtentypen geschieht zentral im Sidebar-Provider.

Der Vorteil dieses Patterns ist, dass die Kommunikation zwischen Benutzeroberfläche und Extension strukturiert, typisiert und nachvollziehbar bleibt. Neue Funktionen können hinzugefügt werden, indem ein neuer Nachrichtentyp und ein passender Handler registriert werden.

11.5.2 Handler als UI-Orchestratoren

Die Handler übernehmen die Koordination zwischen Benutzeroberfläche, VS-Code-API und Board-Funktionen. Sie führen also nicht die eigentliche Low-Level-Kommunikation mit dem MicroPython-Board aus, sondern organisieren den Ablauf aus Sicht der Anwendung.

Ein Handler kümmert sich zum Beispiel darum, eine Eingabefeld anzuzeigen, eine Sicherheitsabfrage vor dem Löschen zu stellen, einen Fortschrittsdialog zu öffnen oder Fehlermeldungen anzuzeigen. Sobald alle nötigen Informationen vorhanden sind, delegiert er die eigentliche Aktion an den dafür zuständigen DeviceManager.

Dadurch entsteht eine klare Trennung: Die Handler enthalten UI-nahe Logik, während die Kommunikation mit dem Board an anderer Stelle gekapselt ist. Das macht den Code leichter verständlich, weil man auf Handler-Ebene den Benutzerablauf erkennt, ohne gleichzeitig alle technischen Details der Board-Kommunikation verstehen zu müssen.

11.5.3 DeviceManager als Fassade

Der DeviceManager stellt eine vereinfachte Schnittstelle für alle Aktionen auf einem verbundenen Board bereit. Nach aussen wirkt er wie ein zentrales Objekt, über das man Dateien lesen, Programme starten, einen Soft Reset ausführen oder den Mount-Zustand ändern kann.

Intern führt der DeviceManager diese Aktionen aber nicht vollständig selbst aus. Stattdessen leitet er sie an spezialisierte Operation-Klassen weiter. Diese Operation-Klassen enthalten die konkrete Logik für einzelne Aufgabenbereiche.

Das entspricht dem Fassaden-Pattern: Eine komplexe interne Struktur wird nach aussen über eine einfache, einheitliche Schnittstelle zugänglich gemacht. Andere Teile der Anwendung müssen dadurch nicht wissen, welche konkrete Operation-Klasse für welche Aufgabe verantwortlich ist. Sie sprechen nur mit dem DeviceManager.

11.5.4 Operation-Klassen mit State-Flags

Viele Aktionen auf dem Board dauern eine gewisse Zeit oder blockieren den seriellen Port. Deshalb verwalten die Operation-Klassen Zustandsinformationen wie `fileOpsActive`, `running` oder `mountActive`.

Diese Flags beschreiben, was gerade auf dem Board passiert. Wenn zum Beispiel eine Dateioperation läuft, wird `fileOpsActive` gesetzt. Wenn ein Programm ausgeführt wird, wird `running` gesetzt. Nach Abschluss der Aktion werden diese Flags wieder zurückgesetzt, häufig abgesichert durch `try/finally`, damit der Zustand auch bei Fehlern korrekt bereinigt wird.

Die Sidebar nutzt diese Zustände, um die Benutzeroberfläche passend zu reagieren. Während einer laufenden Operation können bestimmte Buttons deaktiviert werden, damit der Benutzer keine zweite Aktion startet, die mit der aktuellen kollidieren würde.

Dieses Pattern verbessert sowohl die Stabilität als auch die Benutzerführung. Der Benutzer sieht indirekt, wann das Board beschäftigt ist, und die Anwendung verhindert ungültige Parallelaktionen.

11.5.5 Exklusiver serieller Board-Zugriff

MicroPython-Boards werden über eine serielle Verbindung angesprochen. Diese Verbindung kann in der Regel nicht von mehreren Aktionen gleichzeitig verwendet werden. Deshalb gibt es mit `withBoard` ein Schutzpattern für exklusiven Zugriff.

Bevor eine Board-Operation ausgeführt wird, prüft `withBoard`, ob bereits eine andere Operation aktiv ist. Falls ja, wird die neue Operation abgebrochen oder verhindert. Falls die REPL-Konsole geöffnet ist, wird sie bei Bedarf geschlossen, damit die geplante Aktion den seriellen Port exklusiv nutzen kann. Danach wird die Verbindung frisch geöffnet, die Operation ausgeführt und die Verbindung anschliessend wieder geschlossen.

Dieses Pattern verhindert Port-Konflikte. Es stellt sicher, dass immer nur eine kontrollierte Operation zur gleichen Zeit mit dem Board kommuniziert. Gerade bei seriellen Geräten ist das wichtig, weil parallele Zugriffe nicht möglich sind.

11.5.6 Generischer FileTree mit Provider-Pattern

In der Sidebar gibt es Dateibäume für unterschiedliche Quellen: einerseits den lokalen Workspace, andererseits die Dateien auf dem Board. Beide Bäume sehen ähnlich aus und unterstützen ähnliche Aktionen wie Öffnen, Umbenennen, Löschen, Erstellen oder Drag-and-drop.

Statt zwei komplett getrennte Baum-Komponenten zu bauen, gibt es eine generische FileTree-Komponente. Diese kennt die allgemeine Darstellung und das Verhalten eines Dateibaums. Was sich unterscheidet, wird über sogenannte Provider bereitgestellt.

Der Workspace-Provider beschreibt, wie lokale Dateien geladen und bearbeitet werden. Der Board-Files-Provider beschreibt dasselbe für Dateien auf dem Board. Die generische Baum-Komponente muss dadurch nicht wissen, ob sie gerade lokale Dateien oder Board-Dateien darstellt. Sie fragt nur den Provider, welche Aktionen verfügbar sind und welche Nachricht bei einer Aktion gesendet werden soll.

Die gemeinsame UI-Logik bleibt an einer Stelle, während die Unterschiede sauber in Provider-Klassen ausgelagert sind. Dadurch wird Duplikation reduziert und neue Datei-Quellen könnten später leichter ergänzt werden.

11.6 Code Metriken

In diesem Kapitel werden die Code Metriken, welche im Projekt verwendet wurden ausgewertet. Dabei sind die vollständigen Analysen im Anhang zu finden.

11.6.1 Lines of Code & Anzahl Files

Die Analyse zeigt, dass die entwickelte VS-Code-Extension aus 85 Code-Dateien besteht und der Grossteil der Implementierung in TypeScript umgesetzt wurde. Ergänzend kommen JavaScript-Dateien für die Webview-Benutzeroberfläche sowie HTML-, CSS- und SVG-Dateien für die Darstellung der Oberfläche und das Logo zum Einsatz.

Die Verteilung der Codebasis zeigt, dass die Funktionalität auf viele einzelne Komponenten aufgeteilt wurde. Die grössten produktiven Dateien betreffen zentrale Funktionen der Extension. Dazu gehören insbesondere die Stub-Generierung, welche für die Bereitstellung von Codeunterstützung verwendet wird sowie die Verwaltung von Dateien sowohl auf dem Board als auch im lokalen Workspace. Ebenfalls zu den umfangreicheren Komponenten zählen die Operationen für Dateiübertragungen und Bibliotheksverwaltungen sowie die Verwaltung von Verbindungen zu angeschlossenen Geräten.

Ein weiterer auffälliger Bestandteil der Codebasis sind die Testdateien. Mehrere der grössten Dateien des Projekts sind Unit- oder Integrationstests für die wichtigsten Komponenten wie die Dateiverwaltung, die Bibliotheksverwaltung, die Gerätekommunikation oder die Stub-Generierung. Dies verdeutlicht den Fokus auf die Qualitätssicherung während der Entwicklung und erklärt gleichzeitig einen wesentlichen Teil des Gesamtumfangs der Codebasis.

Auffällig ist zudem, dass keine einzelne Datei einen unverhältnismässig grossen Umfang erreicht. Stattdessen verteilen sich die Implementierungen auf zahlreiche kleinere Module mit klar definierten Verantwortlichkeiten. Diese Struktur verbessert die Übersichtlichkeit des Projekts und erleichtert sowohl die Wartung als auch die zukünftige Erweiterung der Extension.

11.6.2 Testabdeckung

Insgesamt liegt die Testabdeckung des Projekts bei ca. 63 %. Dieser Wert zeigt eine solide, aber noch ausbaufähige Testbasis. Ein Teil der niedrigeren Abdeckung ist bewusst darauf zurückzuführen, dass bestimmte Klassen nur mit sehr hohem Mocking-Aufwand sinnvoll testbar wären. Insbesondere VS Code-nahe Komponenten sowie Board- und Gerätekommunikation wurden dabei teilweise aus Unit-Tests ausgelassen, da eine vollständige Isolation der Abhängigkeiten die Tests stark verfälschen oder unverhältnismässig komplex machen würde.

Der Coverage-Report zeigt eine insgesamt mittlere Testabdeckung des Projekts. Die Kernlogik ist bereits gut durch Unit-Tests und gezielte Integrationstests abgesichert. Insbesondere zentrale Operationen und fachliche Abläufe sind dadurch zuverlässig verifiziert und bilden die stabile Grundlage der Anwendung.

Im Gegensatz dazu ist die Abdeckung in Bereichen der VS Code-Integration sowie der Board- und Gerätekommunikation geringer. Diese Komponenten lassen sich im Rahmen von Unit-Tests nur eingeschränkt sinnvoll testen, da sie stark von externen APIs, VS Code und hardwarenahen Prozessen abhängen. Eine vollständige Abdeckung wäre hier nur durch umfangreiches Mocking möglich, was die Aussagekraft der Tests einschränken würde.

11.6.3 Code Komplexität

Strukturelle Komplexität

Die strukturelle Komplexität der Extension ergibt sich aus der VS Code Architektur. Die Trennung in Extension Host, Webview und das typisierte Nachrichtenprotokoll zwischen den beiden führt zu einer klaren Trennung zwischen Board-Kommunikation und Benutzeroberfläche. Diese Struktur verbessert die Wartbarkeit einzelner Module, da beispielsweise UI-Änderungen unabhängig von der Board-Logik vorgenommen werden können. Die Kommunikation über JSON-Nachrichten erhöht jedoch die Komplexität, da jede Interaktion als Nachrichtentyp definiert und auf beiden Seiten behandelt werden muss.

Logische Komplexität

Die logische Komplexität entsteht hauptsächlich durch die serielle Board-Kommunikation. Der exklusive Zugriff auf den seriellen Port, das Raw-REPL-Protokoll und die Koordination verschiedener Board-Operationen erfordern sorgfältige Fehlerbehandlung und Zustandsverwaltung. Hinzu kommt der Board-Datei-Cache, der den Lebenszyklus von Dateien zwischen Board, lokalem Dateisystem und VS Code Editor koordiniert.

Gesamtbewertung

Insgesamt weist die Extension eine mittlere Komplexität auf. Diese ist primär auf die gewählte VS Code Architektur und die Anforderungen der seriellen Board-Kommunikation zurückzuführen und nicht auf unnötig komplexe Implementierungen. Durch das typisierte Nachrichtenprotokoll, klar abgegrenzte Handler-Klassen und das withBoard()-Pattern bleibt der Code trotz der Komplexität gut wartbar und erweiterbar.

11.6.4 Code Duplikation

Zur Analyse von Codeduplikationen wurde das Werkzeug jscpd (JavaScript Copy Paste Detector) eingesetzt. Zunächst wurde der gesamte Quellcode inklusive der Jest-Testdateien untersucht. Dabei zeigte sich ein geringer Anteil an dupliziertem TypeScript Code von rund 4 %. In den anderen Dateien (JavaScript, HTML, CSS) wurden keine Duplikationen gefunden.

Da Testfälle aufgrund ähnlicher Teststrukturen häufig wiederkehrende Codeabschnitte enthalten, wurde zusätzlich eine Analyse ohne die Jest-Testdateien durchgeführt. In dieser Betrachtung reduzierte sich der Anteil der Duplikationen im TypeScript-Code auf rund 2 %.

Der Vergleich der beiden Analysen verdeutlicht, dass der überwiegende Teil der festgestellten Duplikationen aus den Testdateien stammt. Im produktiven Quellcode sind nur sehr wenige Wiederholungen vorhanden. Die Ergebnisse zeigen, dass die Codebasis weitgehend frei von redundanten Implementierungen ist. Dies wirkt sich positiv auf die Wartbarkeit und Erweiterbarkeit der Software aus, da Änderungen in der Regel nur an einer Stelle vorgenommen werden müssen.

12 KI-Reflexion

Im Rahmen dieser Bachelorarbeit wurden KI-gestützte Werkzeuge als unterstützende Hilfsmittel im Entwicklungs- und Dokumentationsprozess eingesetzt. Dabei kamen insbesondere ChatGPT von OpenAI sowie Claude von Anthropic zum Einsatz. Die Verwendung dieser Systeme diente ausschliesslich der Unterstützung bei der Ausarbeitung von Texten, der Strukturierung von Inhalten, der Ideenfindung sowie bei technischen Fragestellungen während der Entwicklung der MicroPython VS Code Extension.

12.1 Positive Aspekte

Während der Entwicklungsphase wurden KI-Systeme vor allem genutzt, um technische Konzepte zu diskutieren, Lösungsansätze für konkrete Probleme zu evaluieren und mögliche Implementierungsvarianten zu vergleichen. Insbesondere bei Fragestellungen rund um die VS Code API, TypeScript, Testing oder die Kommunikation mit Mikrocontrollern konnten die Systeme hilfreiche Denkanstösse liefern. Die in VS Code integrierten KI-Erweiterungen erwiesen sich dabei als besonders nützlich, da sie durch den direkten Zugriff auf den Projektkontext ein verbessertes Kontextverständnis ermöglichten und so zielgerichtete Vorschläge lieferten. Die vorgeschlagenen Lösungen wurden jedoch nie ungeprüft übernommen. Stattdessen wurden alle relevanten Vorschläge anhand der offiziellen Dokumentationen, eigener Tests und praktischer Implementierungen verifiziert. In mehreren Fällen erwiesen sich Vorschläge der KI als unvollständig oder nicht direkt auf das Projekt anwendbar, wodurch eine kritische Bewertung der Antworten notwendig war.

Auch bei der Erstellung der schriftlichen Arbeit wurden KI-Werkzeuge unterstützend eingesetzt. Dazu gehörten die Verbesserung von Formulierungen, die Umstrukturierung von Abschnitten sowie die sprachliche Überarbeitung einzelner Texte. Darüber hinaus leisteten KI-Systeme konkrete Unterstützung bei der technischen Dokumentation, etwa bei der Erstellung von Docstrings, der Ausarbeitung der README sowie der Maintainer-Dokumentation. Die inhaltliche Ausarbeitung, die fachlichen Entscheidungen sowie die Bewertung der Ergebnisse erfolgten jedoch eigenständig. Die KI diente dabei primär als Hilfsmittel zur Optimierung der Verständlichkeit und Lesbarkeit der Dokumentation.

Ein weiterer wesentlicher Einsatzbereich war die Erstellung eines frühen Prototyps der Extension. Zu Beginn des Projekts wurde mithilfe von Claude Code in VS Code ein erster funktionsfähiger Demonstrator entwickelt, der dem Praxispartner und dem betreuenden Dozenten vorgeführt werden konnte. Dieser Prototyp ermöglichte es, die grundlegende Idee frühzeitig greifbar zu machen und wertvolles Feedback einzuholen. Im weiteren Projektverlauf zeigte sich jedoch, dass der generierte Code strukturelle Schwächen aufwies: Es entstanden umfangreiche, monolithische Dateien – sogenannte God-Files –, die kaum wartbar waren und eine saubere Weiterentwicklung verhindert hätten. Aus diesem Grund wurde der Prototyp bewusst verworfen und die Extension von Grund auf neu und sauber strukturiert entwickelt. Dieser Schritt verdeutlicht eine grundlegende Limitation des KI-gestützten Rapid Prototypings: Es eignet sich gut für die schnelle Demonstration von Konzepten, ersetzt jedoch nicht die sorgfältige, auf Wartbarkeit ausgerichtete Softwarearchitektur.

Ein weiterer Nutzen ergab sich bei der Reflexion von Architekturentscheidungen und bei der Identifikation möglicher Erweiterungen oder Risiken des Projekts. Durch den dialogorientierten Austausch konnten verschiedene Perspektiven auf bestimmte Fragestellungen betrachtet werden. Dies unterstützte den Entscheidungsprozess, ersetzte jedoch nicht die eigene Analyse und Bewertung.

12.2 Herausforderungen

Die Nutzung von KI brachte neben den Vorteilen auch Herausforderungen mit sich. Die Qualität der Antworten hing stark von der Formulierung der Eingaben ab. Zudem konnten technische Informationen veraltet oder fehlerhaft sein. Daher war eine kontinuierliche Überprüfung der generierten Inhalte erforderlich. Besonders bei projektspezifischen Fragestellungen oder bei der Verwendung aktueller Bibliotheken und APIs mussten die Antworten kritisch hinterfragt und mit weiteren Quellen abgeglichen werden.

12.3 Fazit

Insgesamt stellte der Einsatz von KI einen wertvollen Unterstützungsfaktor während der Bachelorarbeit dar. Die Werkzeuge ermöglichten eine effizientere Bearbeitung verschiedener Aufgaben und förderten die Auseinandersetzung mit technischen und fachlichen Fragestellungen. Gleichzeitig zeigte sich, dass KI-Systeme menschliches Fachwissen, praktische Erfahrung und kritisches Denken nicht ersetzen können. Die Verantwortung für sämtliche Entscheidungen, Implementierungen und Inhalte der Arbeit lag jederzeit bei den Autoren.

13 Literaturverzeichnis

- [1] I. Plaуска, A. Liutkevičius und Janavičiūtė, „Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller.“, *Electronics*, Bd. 12, Nr. 1, p. 143, 2023.
- [2] K. Ioannou, D. Karampatzakis, P. Amanatidis, V. Aggelopoulos und I. Karmiris, „Teaching Embedded Systems and IoT at the University using MicroPython and Raspberry Pi Pico,“ in *Proceedings of the 27th Pan-Hellenic Conference on Progress in Computing and Informatics (PCI '23)*, Athens, Greece, 2023.
- [3] R. B. Grady und D. L. Caswell, *Establishing a Company-Wide Program*, Prentice-Hall, 1987.
- [4] P. Eeles, „Capturing Architectural Requirements,“ IBM developerWorks, 2005. [Online]. Available: <https://web.archive.org/web/20201112020231/http://www.ibm.com/developerworks/rational/library/4706.html>. [Zugriff am 06 2026].
- [5] S. Brown, „The C4 Model: Visualizing Software Architecture,“ [Online]. Available: <https://www.oreilly.com/library/view/the-c4-model/9798341660113/>. [Zugriff am 06 2026].
- [6] Microsoft, „Your First Extension, Visual Studio Code Extension API,“ [Online]. Available: <https://code.visualstudio.com/api/get-started/your-first-extension>. [Zugriff am 06 2026].
- [7] Google, „Coding Styleguide Typescript,“ [Online]. Available: <https://google.github.io/styleguide/tsguide.html>. [Zugriff am 27 02 2026].
- [8] H. Vocke, „The Practical Test Pyramid,“ [Online]. Available: <https://martinfowler.com/articles/practical-test-pyramid.html#TheTestPyramid>. [Zugriff am 06 2026].

14 Verzeichnisse

14.1 Abbildungen

Abbildung 1 - Arduino Lab for MicroPython.....	5
Abbildung 2 - UI Konzept Connection	11
Abbildung 3 - UI Konzept Workspace	11
Abbildung 4 - UI Konzept Board Actions	11
Abbildung 5 - UI Konzept Board Files	12
Abbildung 6 - UI Konzept Libraries	12
Abbildung 7 - UI Konzept Code Support	12
Abbildung 8 - Domänenmodell.....	15
Abbildung 9 - C4 Systemkontext.....	24
Abbildung 10 - C4 Container.....	25
Abbildung 11 - C4 Component.....	26
Abbildung 12 - UI ohne Verbindung	32
Abbildung 13 - UI Verbindung aufbauen	33
Abbildung 14 - UI Verbindung Tab Ansicht	33
Abbildung 15 - Board Status mit Tooltip.....	33
Abbildung 16 - UI Kontrollzentrum	33
Abbildung 17 - UI Kontrollzentrum mit Icon Buttons	33
Abbildung 18 - REPL Konsole.....	34
Abbildung 19 - UI Workspace	34
Abbildung 20 - Rename über Input Box	34
Abbildung 21 - Mount Button mit Tooltip	35
Abbildung 22 - Disabled Tooltip bei Mount.....	35
Abbildung 23 - Mount REPL	35
Abbildung 24 - Boarddatei im Editor	36
Abbildung 25 - UI Libraries	36
Abbildung 26 - UI Library installieren	37
Abbildung 27 - UI Code Support	37
Abbildung 28 - Überblick Implementation.....	38
Abbildung 29 - Board Statediagramm	39
Abbildung 30 - Projektplan-SOLL.....	48
Abbildung 31 - Projektplan-IST	49
Abbildung 32 - Vergleich Projektplan IST - SOLL.....	50
Abbildung 33 - Verlauf Risiken.....	52

14.2 Tabellen

Tabelle 1 - Übersicht bestehende Extensions	7
Tabelle 2 - Bewertungsmatrix bestehende Extensions.....	9
Tabelle 3 - Legende Bewertungsmatrix.....	9
Tabelle 4 - FA Verbindungsverwaltung	17
Tabelle 5 - FA Dateiverwaltung.....	18
Tabelle 6 - FA Integriere Konsole	18
Tabelle 7 - FA Reset.....	19
Tabelle 8 - FA Editor-Integration	19
Tabelle 9 - FA Bibliotheksverwaltung	19
Tabelle 10 - NFA Usability	20
Tabelle 11 - NFA Reliability	20
Tabelle 12 - NFA Performance	21
Tabelle 13 - NFA Supportability	21
Tabelle 14 - NFA Compatibility	21
Tabelle 15 - Nicht Anforderungen	22
Tabelle 16 - 3rd Party Libraries.....	28
Tabelle 17 - Messagetypes Webview zu Extension	38
Tabelle 18 - Messagetypes Extension zu Webview	39
Tabelle 19 - Bewertungsmatrix bestehende und entwickelte Extensions	45
Tabelle 20 - Legende Bewertungsmatrix.....	45
Tabelle 21 - Stunden pro Woche	49
Tabelle 22 - Risikolog Stubs	53
Tabelle 23 - Risikolog Performance	54
Tabelle 24 - Risikolog Unterschiedliche Boards	55
Tabelle 25 - Risikolog serielle Kommunikation.....	56
Tabelle 26 - Risikolog Benutzerfreundlichkeit	57
Tabelle 27 - E2E Verbindungsverwaltung	63
Tabelle 28 - E2E Dateiverwaltung.....	63
Tabelle 29 - E2E Integrierte Konsole	65
Tabelle 30 - E2E Reset.....	66
Tabelle 31 - E2E Editor-Integration	66
Tabelle 32 - E2E Bibliotheksverwaltung	67
Tabelle 33 - E2E Usability.....	67
Tabelle 34 - E2E Reliability	68
Tabelle 35 - E2E Performance.....	69
Tabelle 36 - E2E Supportability.....	69
Tabelle 37 - E2E Compatibility.....	70

14.3 Glossar

Begriff	Definition
AGPL-3.0	Lizenz der Extension, erzwungen durch die Abhängigkeit micropython.js. Verpflichtet zur Offenlegung des Quellcodes bei jeder Distribution.
Arduino Lab for MicroPython	Einfache, offizielle Entwicklungsumgebung von Arduino für MicroPython. Dient als UX-Referenz für die Extension. Basiert auf Electron.
Arduino Nano ESP32	Unterstütztes Board auf Basis des ESP32-S3-Chips. Primäres Testboard im Kurs Informatikseminar 1.
Arduino Nano RP2040 Connect	Unterstütztes Board auf Basis des RP2040-Chips. Vollständig getestet und kompatibel.
Arduino GIGA R1 WIFI	Unterstütztes Board aus der STM-Reihe, getestet und bestätigt durch Praxispartner Sebastian Romero.
Autovervollständigung / IntelliSense	VS-Code-Funktion für kontextbezogene Codevorschläge. In der Extension via Pylance und MicroPython-Stubs realisiert.
Board	Physischer Mikrocontroller, der über die Extension verwaltet wird. Kommunikation erfolgt über einen seriellen USB-Port.
Board-Cache (.board_cache/)	Lokales Zwischenverzeichnis für vom Board heruntergeladene Dateien. Ermöglicht Bearbeitung im Editor, Dateien werden beim Schliessen des Tabs gelöscht.
BoardState	Datenstruktur mit Zustandsattributen des Boards (connected, replActive, mountActive, running, portInUse). Steuert aktive/deaktivierte UI-Elemente.
C4-Modell	Architekturvisualisierung in vier Ebenen (Context, Container, Component, Code). In der Arbeit werden Ebenen L1–L3 eingesetzt.
CI/CD-Pipeline	Automatisierte Pipeline (GitLab/GitHub) mit Tests, ESLint, Typprüfung und JSCPD. Veröffentlicht die Extension bei jedem Release automatisch im Marketplace.
COM-Port / Serieller Port	Schnittstelle zur USB-Verbindung mit dem Board (Windows: COM9, macOS/Linux: /dev/tty*). Die Extension erkennt verfügbare Ports automatisch.
Definition of Done	Abnahmekriterien für Tickets: Aufgaben erfüllt, Tests geschrieben, Pipeline grün, App läuft, Zeit erfasst, Folge-Tickets erstellt.
Definition of Ready	Startkriterien für Tickets: Im Sprint Backlog, Story Points geschätzt, keine offenen Abhängigkeiten.
DeviceManager	Fassaden-Klasse für alle Board-Operationen. Delegiert an spezialisierte Operation-Klassen und serialisiert Zugriffe via withBoard().

Domänenmodell	UML-Diagramm der Kernklassen: Board, Extension, BoardFileSystem, BoardState, Library, LibraryInstallation.
ESLint	Statisches Analyse-Tool für TypeScript. Prüft auf Stilprobleme, ein Fund lässt den Pipeline-Job fehlschlagen.
Extension Host	Node.js-Prozess der Extension mit Zugriff auf VS-Code-API, Dateisystem und serieller Verbindung. Kommuniziert mit der Webview via JSON.
FA (Funktionale Anforderung)	22 Anforderungen in den Kategorien Verbindung (FA-V), Dateiverwaltung (FA-D), Konsole (FA-I), Reset (FA-R), Editor (FA-E), Bibliotheken (FA-B). 21/22 erfüllt.
FURPS / URPS	Framework für nicht-funktionale Anforderungen: Usability, Reliability, Performance, Supportability, Compatibility.
GitHub Copilot	KI-Code-Assistent für VS Code. Die Extension ist vollständig kompatibel und verursacht keine Konflikte (FA-E2).
Google TypeScript Style Guide	Basis der Code-Richtlinien im Projekt
Handler	UI-Orchestrator-Klassen. Zeigen Dialoge und Eingabeboxen, delegieren die eigentliche Board-Aktion an den DeviceManager.
Jest	Testing-Framework für Unit- und Integrationstests. Erzeugt Coverage-Reports und JUnit-XML für die CI/CD-Pipeline.
Jira	Projektmanagement-Tool für Tickets und Zeiterfassung. Ergänzt durch das Plugin Worklog360 für detaillierte Auswertungen.
JSCPD (Copy-Paste-Detektor)	Werkzeug zur Erkennung von Code-Duplikaten.
manifest.json	JSON-Datei auf dem Board (/lib/manifest.json) mit Metadaten aller installierten Libraries. Ermöglicht korrekte Deinstallation.
MicroPython	Vereinfachte Python-Implementierung für Mikrocontroller. Geeignet für Lehr- und Prototyping-Zwecke, bietet einen Teil der Python-Standardbibliothek.
micropython.js	Node.js-Bibliothek von Arduino zur programmatischen REPL-Kommunikation. Direkt vom GitHub-Repository eingebunden (nicht über npm). Lizenz: AGPL-3.0.
MicroPython-Stubs (.pyi)	Python-Interface-Dateien mit Typinformationen und Funktionssignaturen. Ermöglichen IntelliSense für MicroPython-APIs ohne eigenen Language Server.
Mikrocontroller	Eingebettetes Computersystem für Steuerungsaufgaben. Im Projekt: Arduino Nano ESP32, Nano RP2040 Connect, GIGA R1 WIFI.
Mount / Mount-Funktion	Bindet einen lokalen Workspace-Ordner direkt als Board-Dateisystem ein. Skripte laufen ohne Upload. Dateioperationen und Bibliotheksverwaltung sind während des Mounts deaktiviert.

mpremote	Offizielles MicroPython-CLI-Tool. In der Extension nur für die Mount-Funktion genutzt. Wird bei Installation automatisch via pip installiert.
MVP (Minimum Viable Product)	Erste vollständig nutzbare Version der Extension mit allen Priorität-1-Anforderungen: Verbindung, Dateiverwaltung, Konsole, Reset, Autovervollständigung, Bibliotheksinstallation.
NFA (Nicht-funktionale Anforderung)	13 Anforderungen nach URPS: NFA-U (Usability), NFA-R (Reliability), NFA-P (Performance), NFA-S (Supportability), NFA-C (Compatibility). 12/13 erfüllt.
Node.js	JavaScript-Laufzeitumgebung des Extension Hosts. Benötigt Version 22.x für die Entwicklung der Extension.
Operation-Klassen	Spezialisierte Klassen für einzelne Board-Aktionen (z.B. RunFileOperation, LibraryOperations). Verwalten Zustandsflags und werden vom DeviceManager delegiert.
Provider-Pattern	Generische FileTree-Komponente mit austauschbaren Datenprovidern (WorkspaceProvider, BoardFilesProvider). Vermeidet Code-Duplikation zwischen den Dateibäumen.
Pylance	VS-Code-Extension von Microsoft für Python-Sprachunterstützung (basiert auf Pyright).
pyrightconfig.json	Konfigurationsdatei für Pylance mit Suchpfaden für Typinformationen (extraPaths). Wird von der Extension automatisch angepasst.
Raw-REPL-Modus	Betriebsmodus der MicroPython-REPL zur direkten Code-Übertragung.
REPL (Read Evaluate Print Loop)	Interaktive Python-Konsole auf dem Board. Die Extension öffnet sie in einem VS-Code-Terminal für direkte Eingaben.
Serieller Port / VID:PID	USB-Schnittstelle zum Board. VID (Vendor ID) und PID (Product ID) identifizieren das Gerät, bekannte Arduino-Boards werden automatisch erkannt.
Soft-Reset	Neustart des MicroPython-Interpreters (Ctrl+D). Setzt den Boardzustand zurück ohne Dateien zu löschen.
TypeScript	Typisierte Obermenge von JavaScript. Empfohlene Sprache für VS-Code-Extensions, ermöglicht Fehlererkennung zur Compile-Zeit
Usability-Tests	Zwei Iterationen mit je drei Testpersonen. Schweregrade: 1 = kosmetisch, 2 = erschwerend, 3 = blockierend. Erkannte Probleme wurden iterativ behoben.
upy-package	Arduino-Bibliothek zur Installation von MicroPython-Paketen auf dem Board. Löst Abhängigkeiten auf, analog zu pip.
vsce	Offizielles Microsoft-Tool zum Verpacken und Veröffentlichen von VS-Code-Extensions. Im

	GitHub-Workflow für automatisierte Publikation eingesetzt.
.vsix	Dateiformat für VS-Code-Extension-Pakete. Wird mit vsce erzeugt und kann manuell oder über den Marketplace installiert werden.
VS Code Marketplace	Offizielle Plattform für VS-Code-Extensions. Die entwickelte Extension ist dort öffentlich verfügbar.
VS Code Webview	Isoliertes HTML-Dokument innerhalb von VS Code für komplexere UIs. Kommuniziert mit dem Extension Host über ein Message-Passing-API.
WebviewGateway	Typsicherer Nachrichtenrouter zwischen Webview und Extension Host. Leitet Nachrichten anhand ihres Typs an den zuständigen Handler weiter.
withBoard()-Pattern	Schutzmuster für exklusiven seriellen Port-Zugriff. Prüft ob eine Aktion läuft, schliesst ggf. die REPL, führt die Operation aus und gibt den Port frei.
Workspace	Geöffneter Projektordner in VS Code. Die Extension zeigt ihn als Dateibaum in der Sidebar an und unterstützt Upload, Umbenennen, Löschen und Drag-and-Drop.

15 Anhang

15.1 Aufgabenstellung

Aufgabenstellung Bachelorarbeit FS26 Informatik OST «Benutzerfreundliche MicroPython- Entwicklung in VS Code»

1. Betreuer

Prof. Dr. Markus Stolze, OST, I3, markus.stolze@ost.ch

2. Praxispartner

Sebastian Romero, Sebastian Romero sebastian.romero@ost.ch <https://www.linkedin.com/in/romeroseb/>

3. Studierende

Silvan Bollhalder

Quentin Weber

4. Ausgangslage

MicroPython ermöglicht einen niederschweligen Einstieg in die Mikrocontroller-Programmierung. Mit *Arduino Lab for MicroPython* existiert eine bewusst einfach gehaltene Entwicklungsumgebung, die insbesondere für Einsteiger:innen gut geeignet ist. Allerdings fehlen dort moderne Editor-Funktionen wie Autovervollständigung oder KI-gestützte Code-Assistenten.

Demgegenüber stehen bestehende **VS-Code-Extensions für MicroPython**, die zwar funktional umfangreich sind, in der Praxis jedoch oft als **komplex, fehleranfällig und wenig einsteigerfreundlich** wahrgenommen werden. Gerade für Anfänger:innen stellen Installation, Konfiguration und Bedienung häufig unnötige Hürden dar.

5. Ziel der Arbeit

Ziel dieser Bachelorarbeit ist die **Konzeption und Umsetzung eines VS-Code-Extension-Pakets für MicroPython**, das:

- die **klare und einfache Benutzerführung** von *Arduino Lab for MicroPython* aufgreift,
- gleichzeitig aber die **starken Editier- und KI-Funktionen von VS Code** nutzbar macht.

Der Schwerpunkt der Implementierung liegt auf einer board-spezifischen, didaktisch reduzierten Code Completion für MicroPython, die typische Anfängerfehler verhindert und kontextgerechte APIVorschläge liefert.

Der Fokus liegt konsequent auf:

- **Einfachheit**
- **Stabilität**
- **guter Usability für Einsteiger:innen**

Um dies sicherzustellen, werden bestehende MicroPython-Extensions analysiert und sinnvolle Konzepte gezielt übernommen. Zusätzlich wird die Lösung durch **Usability-Tests mit Einsteiger:innen** iterativ überprüft und verbessert.

6. Aufgabenstellung

1. Analyse & Konzeption

- Analyse bestehender VS-Code-Extensions für MicroPython
- Identifikation typischer Probleme und Stolpersteine für Einsteiger:innen
- Ableitung klarer UX- und Funktionsanforderungen
- Konzeption eines reduzierten, stabilen Extension-Pakets

2. Implementierung (Kernfunktionen)

Die Extension soll mindestens folgende Funktionen bereitstellen:

- Upload, Ausführen, Bearbeiten und Löschen von MicroPython-Dateien auf dem Mikrocontroller
- Integrierte Konsole für Programmausgaben und Fehlermeldungen
- Syntax-Highlighting für Python
- Unterstützung von KI-Code-Assistenten (z. B. GitHub Copilot)
- Soft-Reset des Boards (Neustart)

3. Erweiterungen (optional / priorisierbar)

Je nach Umfang und Fortschritt können zusätzliche Funktionen umgesetzt werden, z. B.:

- Intelligente Autovervollständigung für MicroPython (inkl. spezifischer Libraries wie machine, network, ...)
- Erkennung nicht unterstützter Python-Bibliotheken
- Ausführen einzelner Dateien auf dem Mikrocontroller
- Festlegen der aktuellen Datei als main.py
- Installation von Libraries direkt aus VS Code (z. B. Arduino Package Index, upy-package)
- Mounten von Projektordnern via mpremote mit visueller Integration in VS Code
- Ansteuerung einer zusätzlichen seriellen Schnittstelle für den Datenaustausch mit dem HostSystem

(Ausgabe in separater Konsole)

4. Evaluation

- Durchführung von **Usability-Tests mit Einsteiger:innen**
- Evaluation der Verständlichkeit, Stabilität und Nutzerführung
- Vergleich mit bestehenden MicroPython-Extensions (qualitativ)

7. Abgrenzung

Nicht Bestandteil der Arbeit sind:

- Entwicklung einer vollständigen eigenen IDE
- Unterstützung aller existierenden MicroPython-Boards
- Tiefe Optimierung der MicroPython-Firmware selbst

Der Fokus liegt auf **Entwickler-Tooling und Benutzerfreundlichkeit**, nicht auf Embedded Firmware-Entwicklung.

8. Ergebnisse / Deliverables

Funktionsfähiges VS-Code-Extension-Paket für MicroPython

- Dokumentierte Architektur und UX-Entscheidungen
- Ergebnisse der Usability-Tests inkl. Verbesserungsmaßnahmen
- Quellcode inkl. Installations- und Nutzungsanleitung

9. Schlussbestimmungen

Im Weiteren gelten die Bedingungen zu den Informatik Bachelor- und Studienarbeiten entsprechend dem aktuellen «Leitfaden für Bachelor- und Studienarbeiten» des Bachelorstudiengangs Informatik an der OST

Das Kriterien-Raster zur Bewertung wird den Studierenden zusammen mit dieser Aufgabenstellung verteilt. Allfällige Anpassungen müssen bis Mitte der Arbeit einvernehmlich beschlossen werden.

Prof. Dr. Markus Stolze, 2026-02-15

(Dokument ohne Unterschrift)

15.2 Usability Test Durchführung 1

Usability Tests

MicroPython for Arduino - VS-Code Extension

Bachelorarbeit FS26 OST

Diese Tests dienen zur Evaluation der Benutzerfreundlichkeit der Extension. Jeder Test beschreibt ein Ziel, keine Schritt-für-Schritt-Anleitung.

Teilnehmerprofil

Name:

Datum:

(1 = keine Erfahrung, 5 = sehr viel Erfahrung):

Erfahrung mit VS-Code	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
Erfahrung mit MicroPython	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
Erfahrung mit Arduino	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
Erfahrung mit Programmierung	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Rolle des Testleiters

Der Testleiter beobachtet den Teilnehmer still und macht sich dabei Notizen. Er greift während der Tests nicht ein und gibt keine Hinweise zur Bedienung der Extension. Nach jedem Test kann er kurze Rückfragen stellen, um das Beobachtete besser zu verstehen. Falls der Teilnehmer gar nicht mehr weiterkommt, kann der Testleiter einen Hinweis geben, dies wird jedoch notiert.

Anleitung

Lies das Szenario durch und versuche das Ziel zu erreichen. Denk dabei laut und erzähle, was du siehst, was du versuchst und wie du dich dabei fühlst. Falls du nicht mehr weiterkommst, frage den Testleiter nach einem Hinweis. Nach Abschluss aller Tests findet ein kurzes Abschlussgespräch statt.

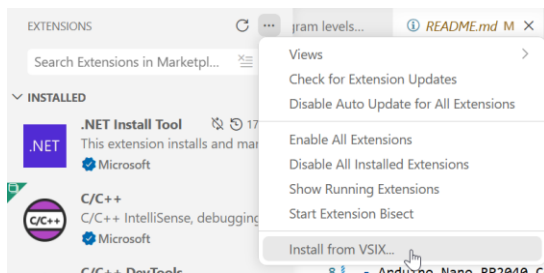
Voraussetzung

1. Visual Studio Code installiert
2. Python installiert
3. **Eines von drei unterstützten Boards mit installierter MicroPython Firmware vorhanden:**
 - Arduino Nano Esp32
 - Arduino Nano RP2040 Connect
 - Arduino Giga R1

Installation

Die folgenden Schritte dienen der einmaligen Installation der Extension und sind kein Bestandteil der Usability-Tests.

1. Öffne den mitgelieferten Ordner mit Visual Studio Code und wechsle zu Extensions.
2. Klicke auf das Dreipunkte-Menü und dann auf «Install from VSIX...»



3. Wähle die Datei micropython-for-arduino.vsix aus, welche mit den Usability Tests mitgeliefert wurden.

Test 1 - Script auf dem Board ausführen

Szenario:

Du hast die Datei `blink.py` erhalten und möchtest diese jetzt auf dem Board ausführen.

Test 2 - Datei auf dem Board bearbeiten und speichern

Szenario:

Main.py auf dem Board lässt die LED im 2-Sekunden-Takt blinken. Du möchtest, dass sie schneller blinkt, im 0.5 Sekunden-Takt.

Test 3 – Direkteingabe auf dem Board

Szenario:

Du möchtest schnell testen, ob dein Board noch reagiert, ohne ein ganzes Skript auszuführen.

Test 4 - Fehler beheben ohne Code zu ändern

Szenario:

Du führst `mounted-blink.py` auf dem Board aus und fragst dich, wieso es fehlschlägt. Du möchtest die Board LED zum Leuchten bringen, ohne den Code zu ändern.

Test 5 - OLED nutzen

Szenario:

Du möchtest Text auf dem beigelegten OLED-Display (SH1106) anzeigen. Bring das Display zum Laufen.

15.3 Usability Test Szenarien Durchführung 2

Usability Tests – Iteration 2

MicroPython for Arduino - VS-Code Extension

Bachelorarbeit FS26 OST

Anleitung und Installation ist identisch zu Iteration 1

Test 1 - Script auf dem Board ausführen

Szenario:

Du entwickelst ein einfaches Alarmsystem mit einer LED. Wenn der Alarm auslöst, soll das Board interne LED blinken. Dazu hast du die Datei blink.py erhalten und möchtest diese jetzt auf dem Board ausführen.

Test 2 - Datei auf dem Board bearbeiten und speichern

Szenario:

Beim Testen stellst du fest, dass die LED zu langsam blinkt, bei einem Alarm soll sie viel schneller aufleuchten, damit man sie sofort bemerkt. Ändere das Skript auf dem Board so, dass die LED im 0.5-Sekunden-Takt blinkt, und zwar so, dass die Änderung auch nach einem Neustart des Boards bestehen bleibt.

Test 3 - Direkteingabe auf dem Board

Szenario:

Es hat geregnet und dein Alarmsystem ist nass geworden. Nun möchtest du testen, ob die LED noch funktioniert. Für einen schnellen Test willst du kein Skript verwenden, sondern die LED direkt ansteuern.

Test 4 - Fehler beheben ohne Code zu ändern

Szenario:

Du hast eine Library entwickelt (library.py), welche die Leuchtfunktion übernimmt. Aus irgendeinem Grund schlägt das `blink_library_test.py` aber fehl, wieso funktioniert es nicht? Du möchtest die Board LED zum Leuchten bringen, ohne den Code zu ändern, oder das `library.py` auf das Board zu laden.

Test 5 - OLED nutzen

Szenario:

Nun möchtest du bei Auslösen des Alarms, direkt auf dem angehängten OLED anzeigen, dass der Alarm ausgelöst wurde. Dazu verwendest du das `OLED.py` Skript. Irgendwas scheint aber noch zu fehlen, damit das Skript funktioniert.

15.4 Usability Test Resultat 1. Durchführung

Testleiter-Protokoll

MicroPython for Arduino - VS-Code Extension | Bachelorarbeit FS26 OST

Teilnehmerprofil

	P1	P2	P3
VS-Code (1-5)	1	5	3
MicroPython (1-5)	2	1	2
Arduino (1-5)	4	2	2
Programmierung (1-5)	5	5	3

Beobachtungen pro Test

Schweregrad: 1 = kosmetisch, 2 = erschwerend, 3 = blockierend

Installation	P1	P2	P3	Notizen
Geschafft	Nein	Nein	Ja	
Schweregrad (1-3)	2	2	0	P1: Mpremove Installation fehlgeschlagen. Musste von Hand installiert werden mit P2: Python Installation nicht richtig erkannt, gab Fehler, danach lief es ohne Probleme

Test 1 – Script ausführen	P1	P2	P3	Notizen
Geschafft	Ja	Ja	ja	
Schweregrad (1-3)	0	1	0	P2: Tab auf der Seite nicht gefunden

Test 2 – Datei bearbeiten	P1	P2	P3	Notizen
Geschafft	Nein	Ja	Ja	
Schweregrad (1-3)	3	1	1	P1: Extension meldet Upload erfolgreich zu Unrecht. Upload funktioniert nicht. Button für Board-Datei im Editor öffnen fehlt P2: Meldung nicht gesehen P3: run current file statt Upload. Nach Hinweis, dass File auf Board noch nicht upgedatet wurde: Save Button gefunden

Test 3 – Direkteingabe	P1	P2	P3	Notizen
Geschafft	Ja	Ja	Ja	
Schweregrad (1-3)	0	0	1	P3: Aufgabe unklar (REPL geöffnet aber war nicht sicher, ob dies korrekt ist)

Test 4 – Fehler beheben	P1	P2	P3	Notizen
Geschafft	Teilweise	Ja	Ja	
Schweregrad (1-3)	1	1	1	P1: Es ist nicht klar, was mount macht. Nach Hinweis war die Aufgabe jedoch einfach gelöst P2: wollte zuerst library.py hochladen, zum unmounten board ausgesteckt. P3: hat zuerst library.py hochgeladen, nach längerem suchen mount Funktion gefunden.

Test 5 – OLED nutzen	P1	P2	P3	Notizen
Geschafft	Ja	Ja	Ja	
Schweregrad (1-3)	2	2	1	P1: Install Library war ausgeblendet, weil der Mount noch aktiv war. Klar nach Hinweis. P2: Hat Library installiert und ein Skript laufenlassen, Probleme bei korrekten Pin für I2C P3: Über AI herausgefunden, dass Library installiert werden muss. README.md und Generierter Code von AI war schlecht und deshalb verwirrend, wodurch etwas Unterstützung gebraucht wurde.

Auswertung - Probleme über alle Testpersonen

Problem	P1	P2	P3	Häufigkeit	Schweregrad	Massnahme
Mount nicht klar	X			1	2	Readme.md
Library disabled während mount	X			1	2	Tooltip
Board Upload funktioniert nicht	X	X		2	3	Bug fix
Board-File im Editor öffnen Button fehlt	X			1	2	Button hinzufügen?
Extension Tab nicht gefunden		X		1	1	Readme.md
I2C Pin Nummer nicht klar	X	X		2	2	Pin Mapper
Keine Tooltips (mac)			X	1	2	Tooltips hinzufügen

Connection Dropdown Element ohne text (Mac bluetooth)			X	1	2	Nicht-Boards nicht anzeigen
REPL unklar			X	1	1	Readme.md

Abschlussgespräch

Person	Feedback
P1	Keine Library installieren, weil Library gemounted war, ist hindernd. Ein Tooltip wäre gut. Getting Started Dokumentation. Readme.md wäre gut Insgesamt sehr gut, im Vergleich mit anderen Tools, welche die Testperson schon verwendet hat.
P2	Screenshots in Readme Code Completion / Autocomplete war zufriedenstellend
P3	Zu viele Primary Buttons, werden leicht übersehen (mount). War verwirrt, dass Boardfile tree bei library install nicht upgedatet wurde.

15.5 Usability Test Resultat 2. Durchführung

Testleiter-Protokoll – Iteration 2

MicroPython for Arduino - VS-Code Extension | Bachelorarbeit FS26 OST

Teilnehmerprofil

	P1	P2	P3
VS-Code (1-5)	1	1	5
MicroPython (1-5)	1	1	1
Arduino (1-5)	1	1	1
Programmierung (1-5)	1	1	5

Beobachtungen pro Test

Schweregrad: 1 = kosmetisch, 2 = erschwerend, 3 = blockierend

Installation	P1	P2	P3	Notizen
Geschafft	Nein	Ja	Nein	
Schweregrad (1-3)	3	0	2	P1: Sidebar Tab zeigt nichts an (Win32 ARM64) P2: Default Interpreter Path war auf Conda

Test 1 – Script ausführen	P1	P2	P3	Notizen
Geschafft	Ja	Ja	Ja	
Schweregrad (1-3)	0	0	1	P3: Auf MacOS werden manchmal leere Board Einträge gezeigt

Test 2 – Datei bearbeiten	P1	P2	P3	Notizen
Geschafft	Ja	Ja	Ja	
Schweregrad (1-3)	2	2	0	P1: Doppelklick auf Boardfile nicht gemacht P2: Augen mussten viel springen: Unten links File hochladen, dann oben in der Mitte Ordner auswählen, dann Erfolgsmeldung unten rechts.

Test 3 – Direkteingabe	P1	P2	P3	Notizen
Geschafft	Ja	Ja	Ja	
Schweregrad (1-3)	1	2	0	P1: Verständnisproblem / fehlendes Knowhow (Was ist REPL, es wurde durch Ausschlussverfahren gefunden, aber nicht was damit gemacht werden kann) P2: Ebenfalls Verständnisproblem was REPL ist. Nach Erklärung von Testleiter verstanden.

Test 4 – Fehler beheben	P1	P2	P3	Notizen
Geschafft	Ja	Ja	Ja	
Schweregrad (1-3)	1	3	2	P2: Erst nach ausführlicher Erklärung von Skripting und Libraries durch Testleiter möglich. Zudem unklar wieso Dateien nach Unmount noch auf Board verbleiben. Bei Unmount wurde erwartet, dass das Skript wieder fehlschlägt, was es aber nicht tut, da die Library im Speicher des Boards verbleibt, bis ein Reboot ausgeführt wird. P3: Es war nicht klar was mount macht und das I war nicht ersichtlich. Da im LightMode gearbeitet wurde.

Test 5 – OLED nutzen	P1	P2	P3	Notizen
Geschafft	Ja	Nein	Ja	
Schweregrad (1-3)	2	3	1	P1: Kleiner Tipp Library, helfen beim Programmieren -> wie kommen suggestions etc. P2: Aufklappbares Menü Available Packages nicht gefunden. Zudem zu wenig Erfahrung mit Programmieren, um das Skript zu vervollständigen. P3: Code Support Skipped

Auswertung - Probleme über alle Testpersonen

Problem	P1	P2	P3	Häufigkeit	Schweregrad	Massnahme
Win ARM64 nicht unterstützt	X			1	Hoch	Nutzer muss Visual Studio Code x64 installieren
Download, nicht ordner bestätigt	X			1	niedrig	Wenn nur ein Ordner, automatisch wählen
Doppelklick boardfile öffnen	X			1	mittel	Beim einmaligen Klicken etwas anzeigen, falls nicht offen (ist aber sehr nervig über Zeit)
Prozess von Upload Boardfile ist zu		X		1	Niedrig	Prozess in einer Meldung

verteilt auf dem Bildschirm						in der Mitte des Bildschirms abhandeln
Aufklappbares Menü Available Packages nicht gefunden		X		1	Mittel	Einzelne Abschnitte visuell klarer Abtrennen
Leerer Boardeintrag			X	1	Niedrig	
Mount nicht klar			X	1	Mittel	
Code Support Generation hat nicht funktioniert			X	1	Mittel	

Abschlussgespräch

Person	Feedback
P1	Tool war grundsätzlich hilfreich und verständlich. Der Grund für das nicht verstehen von Aufgaben war immer das fehlende Wissen zu Arduino und Programmieren. Nach kurzen Erklärungen war die Aufgabe mit der Extension lösbar.
P2	Tool war interessant und die Buttons waren intuitiv. Jedoch zu wenig Verständnis des Programmierens im Allgemeinen, sodass es sehr schwer war, die Aufgaben allein zu lösen. Nach Hinweis des Testleiters jedoch möglich.
P3	Mount war nicht klar. Macht eigentlich aber Sinn. Findet es grundsätzlich gut. Oben die grossen Buttons könnten zuklappbar sein bzw. der Text entfernt werden. Das (i) könnte Links von Buttons sein, weil die Augenführung besser ist. Wenn man Files splittet im Editor, dann eines schliesst, ist das verbleibende auch als deleted markiert.

15.6 Code Metriken

15.6.1 Test Coverage Report

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	63.41	53.66	56.63	63.63	
src	0	0	0	0	
extension.ts	0	0	0	0	1-161
src/device	60.97	43.39	64.7	61.86	
BoardStateManager.ts	94.73	44.44	100	94.73	63
ConnectionManager.ts	68.91	85	50	71.42	191-198, 247-289, 311-316
DeviceManager.ts	67.36	23.07	76.47	67.39	70, 73-108, 174-180, 242-258
MountManager.ts	83.11	57.14	90.47	82.66	22-23, 80-86, 124, 130, 142, 152, 162, 171, 182
ReplTerminal.ts	93.87	85.71	85.71	93.87	49, 90-91
VenvManager.ts	0	0	0	0	1-208
manifest.ts	87.5	100	100	87.5	16
src/device/operation	56.43	45.45	53.19	56.88	
ActivateMountOperation.ts	100	100	100	100	
BoardFileOperations.ts	49.61	33.33	53.84	49.6	14-69, 95, 122, 225-273, 282, 286, 322-356
FetchBoardFilesOperation.ts	96.55	84.61	87.5	96.55	88
LibraryOperations.ts	56.38	46.42	52.17	57.14	64-117, 174-201, 224-236
ReadManifestOperation.ts	50	100	0	66.66	10
RunCodeOperation.ts	100	100	100	100	
RunFileOperation.ts	100	100	100	100	
ScriptRunner.ts	53.62	48.83	42.85	54.41	16-20, 46-56, 77, 102-124, 135-136, 142-147, 169-195, 220-221, 226-231
SoftResetOperation.ts	100	100	100	100	
StopRunOperation.ts	100	100	100	100	
packageRegistry.ts	11.32	0	0	11.53	35, 42-49, 66-141
src/stubs	75.1	60.21	62.96	75.1	
PylanceConfig.ts	51.42	35.29	46.15	51.42	13-19, 27-35, 46-87
StubGenerator.ts	85.27	65.78	78.57	85.27	71-72, 91-92, 98, 182-189, 232-254, 282, 289, 306
src/webview	42.02	65.38	29.87	41.39	
SidebarProvider.ts	0	0	0	0	1-336
WebviewGateway.ts	100	100	100	100	
utils.ts	89.25	78.57	95	88.98	74-77, 130-131, 148-149, 155-158, 220-221, 297
src/webview/handlers	81.22	65.9	77.41	81.13	
BoardActionHandler.ts	90.76	78.57	88.88	90.76	51-58, 72, 112
BoardFileHandler.ts	67.59	45	69.23	67.59	33, 51-84, 280, 287-294, 312-360
BoardFileSystemProvider.ts	85.07	70.27	68.75	85.07	61-62, 80, 93, 156, 185-186, 215, 223, 279, 293-302, 331-334, 338, 362, 375, 382, 388
CodeSupportHandler.ts	80.7	78.94	50	80.7	22-39, 114
LibraryHandler.ts	90.9	83.33	83.33	90.66	27-31, 77-80, 187
MountHandler.ts	100	100	100	100	
StubHandler.ts	88.46	81.81	80	88.23	44, 111, 122-132
WorkspaceFileHandler.ts	71.31	53.12	92.85	71.31	51, 96-104, 111, 118, 138, 144-151, 168, 186, 190-193, 200, 216, 220-223, 228, 277-300

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
extension.ts	0	0	0	0	1-161
BoardStateManager.ts	94.73	44.44	100	94.73	63
ConnectionManager.ts	68.91	85	50	71.42	191-198, 247-289, 311-316
DeviceManager.ts	67.36	23.07	76.47	67.39	70, 73-108, 174-180, 242-258
MountManager.ts	83.11	57.14	90.47	82.66	22-23, 80-86, 124, 130, 142, 152, 162, 171, 182
ReplTerminal.ts	93.87	85.71	85.71	93.87	49, 90-91
VenvManager.ts	0	0	0	0	1-208
manifest.ts	87.5	100	100	87.5	16
ActivateMountOperation.ts	100	100	100	100	
BoardFileOperations.ts	49.61	33.33	53.84	49.6	14-69, 95, 122, 225-273, 282, 286, 322-356
FetchBoardFilesOperation.ts	96.55	84.61	87.5	96.55	88
LibraryOperations.ts	56.38	46.42	52.17	57.14	64-117, 174-201, 224-236
ReadManifestOperation.ts	50	100	0	66.66	10

RunCodeOperation.ts	100	100	100	100	
RunFileOperation.ts	100	100	100	100	
ScriptRunner.ts	53.62	48.83	42.85	54.41	16-20, 46-56, 77, 102-124, 135-136, 142-147, 169-195, 220-221, 226-231
SoftResetOperation.ts	100	100	100	100	
StopRunOperation.ts	100	100	100	100	
packageRegistry.ts	11.32	0	0	11.53	35, 42-49, 66-141
PylanceConfig.ts	51.42	35.29	46.15	51.42	13-19, 27-35, 46-87
StubGenerator.ts	85.27	65.78	78.57	85.27	71-72, 91-92, 98, 182-189, 232-254, 282, 289, 306
SidebarProvider.ts	0	0	0	0	1-336
WebviewGateway.ts	100	100	100	100	
utils.ts	89.25	78.57	95	88.98	74-77, 130-131, 148-149, 155-158, 220-221, 297
BoardActionHandler.ts	90.76	78.57	88.88	90.76	51-58, 72, 112
BoardFileHandler.ts	67.59	45	69.23	67.59	33, 51-84, 280, 287-294, 312-360
BoardFileSystemProvider.ts	85.07	70.27	68.75	85.07	61-62, 80, 93, 156, 185-186, 215, 223, 279, 293-302, 331-334, 338, 362, 375, 382, 388
CodeSupportHandler.ts	80.7	78.94	50	80.7	22-39, 114
LibraryHandler.ts	90.9	83.33	83.33	90.66	27-31, 77-80, 187
MountHandler.ts	100	100	100	100	
StubHandler.ts	88.46	81.81	80	88.23	44, 111, 122-132
WorkspaceFileHandler.ts	71.31	53.12	92.85	71.31	51, 96-104, 111, 118, 138, 144-151, 168, 186, 190-193, 200, 216, 220-223, 228, 277-300

15.6.2 Duplikationen

Duplikationen mit Jest Test Dateien:

```
jscpd ./src ./media
```

Format	Files analyzed	Total lines	Total tokens	Clones found	Duplicated lines	Duplicated tokens
typescript	64	12286	104306	40	464 (3.78%)	4161 (3.99%)
javascript	17	2370	17526	0	0 (0%)	0 (0%)
markup	2	220	2949	0	0 (0%)	0 (0%)
css	1	770	4017	0	0 (0%)	0 (0%)
Total:	84	15646	128798	40	464 (2.97%)	4161 (3.23%)

Duplikationen ohne Jest Test Dateien:

```
jscpd ./src ./media --ignore "**/*.test.ts"
```

Format	Files analyzed	Total lines	Total tokens	Clones found	Duplicated lines	Duplicated tokens
typescript	36	6080	49218	6	112 (1.84%)	978 (1.99%)
javascript	17	2370	17526	0	0 (0%)	0 (0%)
markup	2	220	2949	0	0 (0%)	0 (0%)
css	1	770	4017	0	0 (0%)	0 (0%)
Total:	56	9440	73710	6	112 (1.19%)	978 (1.33%)

15.6.3 Lines of Code

Language	Files	Blank	Comment	Code
TypeScript	64	1821	946	9583
JavaScript	17	287	324	1776
CSS	2	285	156	1581
HTML	1	12	0	163
SVG	1	9	7	31
SUM	85	2414	1433	13134

Language	Filename	Blank	Comment	Code
TypeScript	.\src\test\unit\handlers\LibraryHandler.test.ts	70	10	397
TypeScript	.\src\stubs\StubGenerator.ts	40	21	370
TypeScript	.\src\test\unit\handlers\BoardFileHandler.test.ts	74	11	323
TypeScript	.\src\webview\handlers\BoardFileSystemProvider.ts	39	49	302
TypeScript	.\src\webview\handlers\BoardFileHandler.ts	36	30	298
JavaScript	.\media\sidebar\modules\filetree\genericTree.js	52	84	291
TypeScript	.\src\webview\handlers\WorkspaceFileHandler.ts	22	40	291
TypeScript	.\src\device\operation\BoardFileOperations.ts	41	33	283
TypeScript	.\src\test\unit\handlers\StubHandler.test.ts	46	1	269
TypeScript	.\src\test\unit\handlers\BoardActionHandler.test.ts	79	11	264
TypeScript	.\src\test\unit\device\ConnectionManager.test.ts	34	0	263
TypeScript	.\src\test\unit\handlers\WorkspaceFileHandler.test.ts	72	29	262
TypeScript	.\src\test\unit\device\operations\LibraryOperations.test.ts	37	13	257
TypeScript	.\src\test\unit\device\operations\FetchBoardFilesOperation.test.ts	30	10	249
TypeScript	.\src\test\integration\StubGenerator.test.ts	76	6	242
TypeScript	.\src\test\unit\device\ReplTerminal.test.ts	50	0	240
TypeScript	.\src\test\unit\utils.test.ts	57	0	238
TypeScript	.\src\device\operation\LibraryOperations.ts	36	14	237
TypeScript	.\src\device\ConnectionManager.ts	25	63	230
TypeScript	.\src\device\DeviceManager.ts	19	18	229
TypeScript	.\src\device\operation\ScriptRunner.ts	30	20	228
JavaScript	.\media\sidebar\sidebar.js	33	0	211
TypeScript	.\src\test\unit\device\operations\DeviceManagerOperations.test.ts	54	1	209
TypeScript	.\src\webview\utils.ts	43	67	205
JavaScript	.\media\sidebar\modules\libraries.js	27	3	182
TypeScript	.\src\device\VenvManager.ts	16	14	179
TypeScript	.\src\test\unit\handlers\CodeSupportHandler.test.ts	32	7	173
TypeScript	.\src\webview\handlers\LibraryHandler.ts	18	13	173
HTML	.\media\sidebar\sidebar.html	12	0	163
TypeScript	.\src\test\unit\handlers\BoardFileSystemProvider.test.ts	54	9	163
JavaScript	.\media\sidebar\modules\filetree\dnd.js	25	45	161
JavaScript	.\media\sidebar\modules\tabs.js	35	31	159
TypeScript	.\src\test\unit\device\operations\BoardFileOperations.test.ts	50	9	154
TypeScript	.\src\device\MountManager.ts	30	27	152
TypeScript	.\src\extension.ts	13	2	146
TypeScript	.\src\test\integration\BoardFileSystemProvider.test.ts	34	4	143
TypeScript	.\src\webview\SidebarProvider.ts	26	176	136
TypeScript	.\src\stubs\PylanceConfig.ts	17	16	126
TypeScript	.\src\types\messages.ts	8	14	126
TypeScript	.\src\test\unit\handlers\MountHandler.test.ts	44	0	122
TypeScript	.\src\webview\handlers\BoardActionHandler.ts	18	21	122
TypeScript	.\src\device\operation\packageRegistry.ts	18	7	120
TypeScript	.\src\test\unit\device\MountManager.test.ts	62	0	116
JavaScript	.\media\sidebar\modules\boardState.js	18	12	115
TypeScript	.\src\webview\handlers\StubHandler.ts	17	9	108
TypeScript	.\src\test\unit\device\DeviceManager.test.ts	31	0	107
TypeScript	.\src\webview\handlers\CodeSupportHandler.ts	12	13	101
JavaScript	.\media\sidebar\modules\boardActions.js	15	0	100

TypeScript	.\src\test\unit\device\BoardStateManager.test.ts	31	2	98
JavaScript	.\media\sidebar\modules\filetree\actions.js	7	3	96
TypeScript	.\src\device\ReplTerminal.ts	15	9	94
TypeScript	.\src\test\unit\device\operations\RunFileOperation.test.ts	24	3	93
JavaScript	.\media\sidebar\modules\filetree\boardFilesProvider.js	12	38	91
JavaScript	.\media\sidebar\modules\tooltip.js	20	19	91
TypeScript	.\src\test\integration\PylanceConfig.test.ts	29	2	89
JavaScript	.\media\sidebar\modules\codesupport.js	16	8	88
TypeScript	.\src\test\unit\WebviewGateway.test.ts	22	2	84
JavaScript	.\media\sidebar\modules\filetree\treeHelpers.js	6	42	81
TypeScript	.\src\device\operation\FetchBoardFilesOperation.ts	10	6	79
TypeScript	.\src\test\unit\device\operations\SoftResetOperation.test.ts	10	5	71
TypeScript	.\src\test\unit\device\operations\StopRunOperation.test.ts	29	7	71
TypeScript	.\src\test\unit\device\operations\ActivateMountOperation.test.ts	19	2	69
TypeScript	.\src\test\unit\device\ScriptRunner.test.ts	19	2	65
JavaScript	.\media\sidebar\modules\filetree\workspaceProvider.js	11	33	64
TypeScript	.\src\test\unit\device\operations\RunCodeOperation.test.ts	22	4	62
TypeScript	.\src\webview\handlers\MountHandler.ts	9	9	59
TypeScript	.\src\device\BoardStateManager.ts	10	9	51
TypeScript	.\src\types\micropython-js.d.ts	27	30	48
SVG	.\media\icon.svg	9	7	31
TypeScript	.\src\device\operation\ActivateMountOperation.ts	3	6	31
TypeScript	.\src\device\operation\RunFileOperation.ts	4	6	27
TypeScript	.\src\device\operation\StopRunOperation.ts	2	5	26
TypeScript	.\src\webview\WebviewGateway.ts	7	12	26
JavaScript	.\media\sidebar\modules\stubs.js	5	0	21
TypeScript	.\src\device\operation\RunCodeOperation.ts	5	5	20
TypeScript	.\src\device\manifest.ts	3	6	19
TypeScript	.\src\device\operation\SoftResetOperation.ts	3	5	17
TypeScript	.\src\types\constants.ts	3	15	16
JavaScript	.\media\sidebar\modules\filetree\constants.js	2	3	12
JavaScript	.\media\sidebar\modules\mount.js	3	0	12
TypeScript	.\src\device\operation\ReadManifestOperation.ts	1	3	8
TypeScript	.\src\types\boardState.ts	4	3	7
JavaScript	.\media\sidebar\modules\vscode.js	0	3	1
SUM		2414	1433	13134