Bachelorarbeit

Automatisierte Analyse von Obsidian Plugins zur Einschätzung der weiterführenden Operabilität

Verfasser:	Johannes Theiner
Matrikelnummer:	7010923
Prüfer:	Prof. Dr. Niels Streekmann
Zweitprüfer:	Dipl.-Ing. Hilke Fasse
Studiengang:	Informatik
Semester:	Wintersemester 2022
Abgabe:	28. April 2023

Ich, der/die Unterzeichnende, erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Quellenangaben und Zitate sind richtig und vollständig wiedergegeben und in den jeweiligen Kapiteln und im Literaturverzeichnis wiedergegeben. Die vorliegende Arbeit wurde nicht in dieser oder einer ähnlichen Form ganz oder in Teilen zur Erlangung eines akademischen Abschlussgrades oder einer anderen Prüfungsleistung eingereicht.

Mit ist bekannt, dass falsche Angaben im Zusammenhang mit dieser Erklärung strafrechtlich verfolgt werden können.

_____

Datum, Unterschrift

Inhaltsverzeichnis

1. Einleitung

In dieser Arbeit sollen verschiedene Metriken zu Community Plugins im Kontext von Obsidian erhoben und ausgewertet werden. Die einzelnen Metriken sollen überprüft werden auf ihre Eignung zur Erkennung von ungepflegten Erweiterungen, oder Erweiterungen die nicht mehr mit der aktuellen Obsidian Version kompatibel sind. Zusätzlich sollen Daten über die von Plugin Entwicklern genutzten internen Funktionen gesammelt werden, um die offizielle API zu verbessern und somit stabilere Erweiterungen in der Zukunft zu ermöglichen.

Diese Arbeit ist in mehrere Abschnitte unterteilt, im nächsten Kapitel werden die Grundlagen gelegt die in den nachfolgenden Kapiteln benötigt werden. Das dritte Kapitel erläutert die verschiedenen Ansätze aus der Forschung und welche davon in diesem Kontext verwendet werden. Das vierte Kapitel beleuchtet die Rahmenbedingungen, die während der Durchführung der einzelnen Analysen beachtet werden müssen, während das fünfte Kapitel die Implementation der einzelnen Analysen beschreibt. Die Auswertung der Analysen wird im sechsten Kapitel durchgeführt, das letzte Kapitel fast die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf weitere Entwicklungsmöglichkeiten.

Wenn nicht anderweitig ausgewiesen sind Daten und Grafiken innerhalb dieser Arbeit generiert worden am 19.02.2023.

1.1. Motivation

Obsidian bietet für Nutzer sehr viele Möglichkeiten die Anwendung den eigenen Wünschen anzupassen, etwa durch das Installieren oder das Programmieren eines Plugins[1]. Im Vergleich mit Konkurrenzprodukten bietet Obsidian eine viel größere Anzahl von Erweiterungen direkt innerhalb der Anwendung zum Herunterladen an.

Aktuell sind 954 Erweiterungen in der offiziellen Liste aufgenommen, und können somit direkt innerhalb der Anwendung heruntergeladen und installiert werden. Seit der Einführung von Community Plugins in Obsidian am 30. Oktober 2020[2] wurden nur wenige Einträge wieder entfernt. Entfernt wurde bisher nur auf Antrag des originalen Entwicklers oder in wenigen Fällen wurde die Wartung eines Plugins wegen Inaktivität an andere Entwickler übertragen, angestoßen von mehreren Mitgliedern der Community.

Erweiterungen die für eine lange Zeit nicht gepflegt werden, sorgen für eine schlechte Nutzererfahrung, weil Nutzer bei der Erkundung der Erweiterungen tendenziell auf mehr Fehlerhafte Plugins treffen, deren Probleme nicht behoben werden. Zusätzlich reflektiert dies schlecht auf das Grundprodukt, auch weil die Stabilität der Anwendung potenziell beeinträchtigt wird.

Aus diesem Grund sollen in dieser Arbeit Daten zu allen, aktuell in der offiziellen Liste eingetragenen Erweiterungen gesammelt werden, um mehrere Metriken zu erstellen, diese zu bewerten und veraltete oder inkompatible Plugins somit auszusortieren zu können.

[1] englisch für Zusatzmodul, die deutsche Obsidian Übersetzung verwendet *Erweiterung*
[2] `https://forum.obsidian.md/t/obsidian-release-v0-9-8/7857`

Innerhalb der Obsidian Community wurde dieses Thema bereits mehrfach von Nutzern ausführlich diskutiert, an mehreren dieser Diskussionen konnte der Autor dieser Arbeit bereits teilhaben[3]. Seit diese Diskussionen geführt wurden, wurde der Autor dieser Arbeit Teil des Obsidian Teams, und ist dort unter anderem für das erste Code-Review zuständig welches jedes neue Plugin durchlaufen muss. Die Entscheidung den Autor einzustellen ist unabhängig vom Thema dieser Arbeit getroffen worden.

[3] u.a. `https://forum.obsidian.md/t/curating-out-of-date-plugins/34569`

2. Grundlagen

In diesem Abschnitt sollen die Grundlagen für die verschiedenen Auswertungen besprochen werden. Zuerst wird das von allen Erweiterungen verwendete Versionskontrollsystem Git sowie der Anbieter GitHub betrachtet. Im Anschluss werden die hier benötigten Grundlagen der Programmiersprachen JavaScript und TypeScript gelegt. Im letzten Abschnitt werden die Grundlagen von Obsidian und der Entwicklung von Plugins für Obsidian erläutert.

2.1. Git

Git[4] ist ein verteiltes System zur Versionskontrolle von Daten, kurz VCS. Ein VCS ermöglicht es auf frühere Versionen von Daten zuzugreifen und diese zu vergleichen. Als verteiltes VCS ermöglicht Git es Änderungen mehrerer Entwickler zusammenzuführen, ohne einen zentralen Server zu benötigen[1, S. 10 ff.]. Zur Zusammenarbeit werden häufig zentrale Server oder Anbieter wie GitHub(siehe nächster Abschnitt) verwendet. Ein mit Git verwalteter Ordner wird als Repository bezeichnet.

Änderungen an einzelnen Dateien können von einem Entwickler zu einem Commit zusammengefasst werden. Diese Commits werden üblicherweise auf separate Server übertragen und somit für andere Entwickler bereitgestellt. Wird an mehreren Features oder Fehlerbehebungen gleichzeitig mit mehreren Entwicklern gearbeitet wird die Verwendung von Branches empfohlen. Hierbei werden Änderungslisten separat verwaltet, und können später zusammengeführt werden[1, S. 63].

2.2. GitHub

GitHub[5] ist der größte Anbieter von online Git Repositories, die Nutzung ist für die in dieser Arbeit relevanten Zwecke kostenlos. Integriert ist ein Issue-Tracker, bei dem Nutzer Fehler melden oder neue Features vorschlagen können.

Einzelnen Nutzern kann das Recht erteilt werden direkt, ohne Pull-Request Änderungen am Quellcode im Repository vorzunehmen. Diese Nutzer werden auch als Collaborator[6] bezeichnet. Andere Nutzer können Quellcodeänderungen durch die Erstellung eines Pull-Requests(PR) vorschlagen, dazu muss von diesen erst ein Fork erstellt werden. Diese Nutzer werden als von GitHub als Contributor[7] bezeichnet, sobald deren Beitrag im Haupt-Branch vorhanden ist.

Als Fork wird eine Kopie eines Repositories bezeichnet, ein solcher Fork kann entweder zu einem separaten Projekt heranwachsen, oder zum Beitragen am Quellcode des Originals verwendet werden [1, S. 171 ff.]. Nach einer Untersuchung von Jiang, Lo, He

[4] `https://git-scm.com`
[5] `https://github.com/`
[6] Deutsch: Kollaborateur oder Mitarbeiter
[7] Deutsch: Beitragender

u. a. erstellen 85,3% der Entwickler Forks um Fehler zu beheben, während 45,1% neue Features entwickeln wollen [2].

Sowohl bei Issues als auch bei Pull-Requests können Kommentare von anderen Nutzern hinterlassen werden. Diese können von Kollaborateuren geschlossen werden, auch ohne diese zu implementieren/anzunehmen. Durch das Markieren mit Labels können Issues und PRs in verschiedene Gruppen sortiert werden, je nach Einrichtung durch den Besitzer.

Zum Bereitstellen von nutzbaren Versionen der Software können diese als Release hochgeladen werden. In einem Release können Dateien für unterschiedliche Betriebssystem oder für ähnliche Zwecke zusammengefasst werden.

2.3. JavaScript

Während HTML und CSS für das Layout und das Design von Webseiten verantwortlich sind, ist JavaScript die am meisten verbreitete Programmiersprache für dynamische Inhalte. JavaScript ist Prototypenbasierte Sprache sowie eine dynamische Programmiersprache, so müssen Variablen und Methoden nicht deklariert werden, bevor diese aufgerufen werden können [3]. Auch ist JavaScript eine schwach typisierte Sprache, so ist das Ergebnis der Berechnung `42 + "1"` gleich dem String 421. Ist ein solches Verhalten vom Entwickler nicht gewünscht können unerwartete Laufzeitfehler auftreten [4].

Als Document Object Model(DOM) wird die API bezeichnet, die es ermöglicht die Repräsentation eines HTML oder XML Dokuments auszulesen und zu verändern [5]. JavaScript ist standardisiert als ECMAScript, diese Spezifikation wird verwendet für die Implementation der Laufzeitumgebung, etwa innerhalb eines Browsers. Je nachdem für welche Version des Standards entwickelt wurde, sind unterschiedliche Funktionalitäten und Sprachfeatures vorhanden [3].

Die serverseitige Ausführung von JavaScript Code kann zum Beispiel mit der NodeJS[8] Laufzeitumgebung umgesetzt werden. Mit Projekten wie Electron[9], welches auf dem Chromium[10] Browser sowie NodeJS basiert, können Anwendungen für den Browser entwickelt werden, welche auf HTML, CSS und JavaScript basieren. Für Mobilgeräte kann etwa mit Capacitor[11] eine Anwendung entwickelt werden.

2.4. TypeScript

TypeScript erweitert JavaScript um Typen und Klassen, welche bei der Umwandlung in JavaScript Code vom Compiler in entsprechenden JavaScript Code aufgelöst werden. Jedes gültige JavaScript Programm ist gleichzeitig ein gültiges TypeScript Programm, TypeScript überprüft nur zusätzlich während dem kompilieren auf übliche Probleme und meldet diese als Fehler [6].

[8] `https://nodejs.org`
[9] `https://electronjs.org`
[10] `https://www.chromium.org`
[11] `https://capacitorjs.com/`

2.5. Abstract Syntax Tree

Ein Abstract Syntax Tree(AST) repräsentiert den Quelltext eines Programs, ohne unnötige Syntax, in einer Baumstruktur. Verwendet werden ASTs innerhalb eines Compilers oder innerhalb einer Entwicklungsumgebung für die Bereitstellung von Vorschlägen und Warnungen. Auch zur Bestimmung der Testabdeckung sind ASTs geeignet. [7]

Der TypeScript Code `const foo: TFile[] = this.app.vault.getFiles();` wird als AST wie in Abbildung 1 repräsentiert.

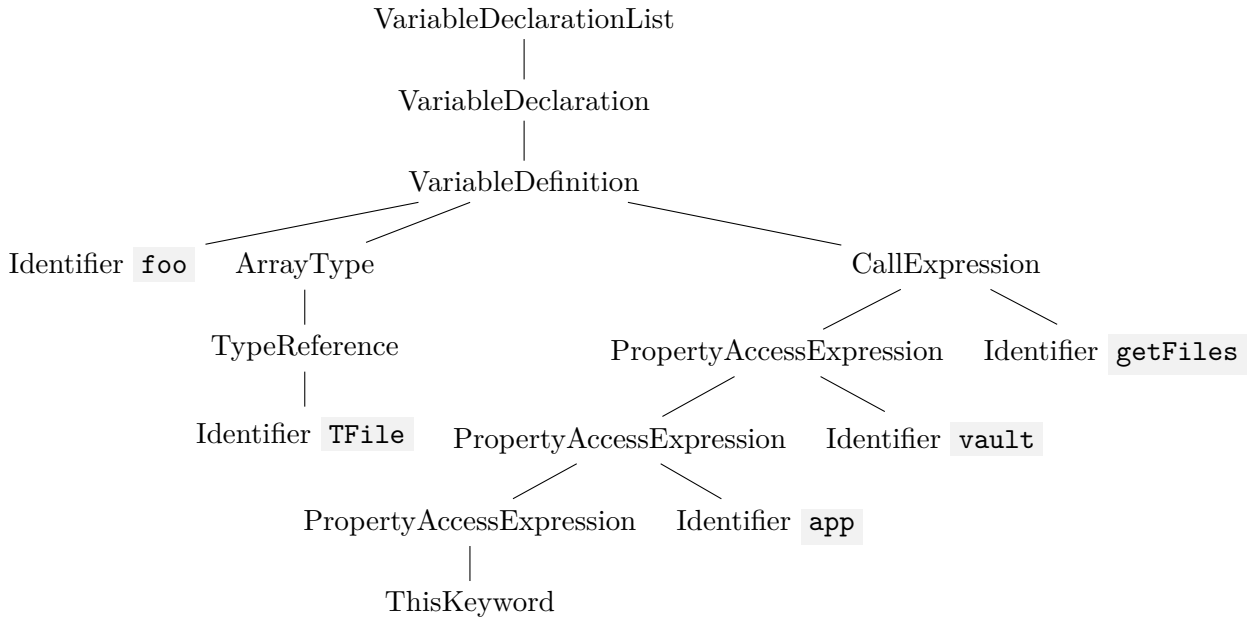


Abbildung 1: Repräsentation eines AST

2.6. Monkey-Patching

Der Begriff Monkey-Patching bezeichnet das Ändern von Quellcode während der Ausführung. In JavaScript und anderen Programmiersprachen ist es möglich definierte Funktionen durch eigene auszutauschen, und so die originale Funktionalität zu ändern. Weil innerhalb eines Monkey-Patches auch häufig interne Funktionalität aufgerufen werden muss, oder durch einen Monkey-Patch diese ausgetauscht werden kann, muss beim Einsatz vorsichtig vorgegangen werden.[8]

2.7. Obsidian

Obsidian[12] ist eine Anwendung zum Erstellen und Verwalten von Notizen. Andere bekannte Software in dieser Kategorie sind *Notion*, *Evernote*, *Microsoft OneNote*, *Roam Research*, *LogSeq* sowie *Vimwiki*.

[12] `https://obsidian.md/`

Gerade als Software zur Verwaltung eines *Personal Knowledge Management(PKM)* Systems erfreut sich Obsidian großer Beliebtheit. Ein solches System, oder auch *Second brain*, *Memex* oder *Digital garden* genannt wird verwenden um eigene Ideen, Wissen und andere wichtige Informationen abzuspeichern um diese später abrufen zu können[9, S. 4]. Dies kann entweder auf Papier basieren, wie der *Zettelkasten* von Niklas Luhmann, oder digital. Auch im Firmenumfeld wird Obsidian zur Verwaltung von Wissen und Informationen verwendet. Obsidian ist ebenfalls sehr beliebt bei Spielern von Tabletop Spielen wie *Dungeons & Dragons*.

Mit der Entwicklung von Obsidian wurde 2020 während der ersten Corona Lockdowns von Shida Li und Erica Xu gestartet. Zuvor wurde bereits Dynalist[13] von diesen Entwicklern erstellt[10].

2.7.1. Philosophie

Teil der Philosophie von Obsidian ist die Erweiterbarkeit: **Make it super extensible**[14]. Änderungen am Design können mit kleineren CSS-Bausteinen vorgenommen werden, oder über Entwicklung eines Farbschemas[15]. Neue Funktionalitäten können durch die Entwicklung einer Erweiterung hinzugefügt werden[16]. Von anderen Nutzern erstellte Farbschemen und Erweiterungen können direkt innerhalb der Anwendung heruntergeladen und installiert werden.

Ein weiteres Grundprinzip ist: **Local-first and plain text**[14].Notizen werden lokal in der eigenen Orderstruktur in Klartext Dateien gespeichert, ohne eine Datenbank als Zwischenspeicher. Für Textnotizen wird dafür das Markdown Format verwendet[17]. Die Dateien des Canvas Features[18] werden als JSON formatiert gespeichert[19], die Spezifikation des Formats ist öffentlich verfügbar[20]. Notizen werden in einem vom Nutzer festgelegten Ort abgelegt, dieser Ordner wird als *Vault* bezeichnet.

We want you to own and control your data[21]: Obsidian funktioniert komplett lokal, auf dem Gerät des Nutzers. Eine Kommunikation mit externen Servern findet nur statt für die Bereitstellung von Updates, und zur Installation der bereits erwähnten Farbschemen und Erweiterungen.

2.7.2. Verwendete Technologien

Geschrieben ist Obsidian hauptsächlich in den Programmiersprachen *TypeScript* und *JavaScript*, sowie kleinen Anteilen von *C++*, *Java*, *Swift* und weiteren. Für die Ausführung

[13] `https://dynalist.io/`
[14] `https://obsidian.md/about`
[15] `https://help.obsidian.md/Advanced+topics/Customizing+CSS`
[16] `https://help.obsidian.md/Advanced+topics/Community+plugins`
[17] Genauer CommonMark (`https://commonmark.org`), Teile von GitHub Flavored Markdown (`https://github.github.com/gfm/`), sowie eigenen Erweiterungen.
[18] `https://obsidian.md/canvas`
[19] `https://json.org`
[20] `https://github.com/obsidianmd/obsidian-api/blob/master/canvas.d.ts`
[21] `https://help.obsidian.md/Obsidian/Obsidian#How+we're+different`

der Anwendung werden die Frameworks *Electron*[22] auf dem Desktop, sowie *Capacitor*[23] auf Android und iOS Geräten verwendet[11]. UI Frameworks wie *React*, *Vue* oder *Svelte* werden nicht verwendet.

2.7.3. Entwicklung von Plugins

Für Community Plugins ist es empfohlen diese in *TypeScript* zu entwickeln, da nur für TypeScrip eine entsprechende API Definition[24] vorhanden ist. Auch ein offizielles Beispiel Plugin ist nur für TypeScript verfügbar[25]. Andere Programmiersprachen können ebenfalls verwendet werden, solange diese entweder JavaScript Code als Ausgabeformat verwenden können, oder Quellcode als *WebAssembly* kompilieren können.

Jedes Plugin muss eine Hauptklasse bereitstellen die von der `Plugin` Klasse aus der Obsidian API erbt. Die Funktion `onload()` muss innerhalb dieser Klasse implementiert sein, und wird beim Laden des Plugins ausgeführt(Auflistung 1). Weitere Funktionen wie `onunload()` können bei Bedarf implementiert werden.

[22] `https://www.electronjs.org/`
[23] `https://capacitorjs.com/`
[24] `https://github.com/obsidianmd/obsidian-api`
[25] `https://github.com/obsidianmd/obsidian-sample-plugin`

```
import { App, Notice, Plugin } from 'obsidian';
import { SampleSettingTab, SimpleModal, MyPluginSettings } from
↪  './otherFile';

export default class MyPlugin extends Plugin {
    //Variable in der die Einstellungen gespeichert werden
        settings: MyPluginSettings;

        async onload() {
            //Einstellungen aus Datei laden
                await this.loadSettings();

        //Befehl hinzufügen, der ein Modal öffnet
                this.addCommand({
                        id: 'open-simple-modal',
                        name: 'Open simple modal',
                        callback: () => {
                                new SimpleModal(this.app).open();
                        }
                });

                //Einstellungs Tab initialisieren, damit Nutzer diese
                ↪  ändern können
                this.addSettingTab(new SampleSettingTab(this));
        }

        onunload() {}
        ...
}
```

Auflistung 1: Quellcodeauschnitt eines Beispiel-Plugins

2.7.4. Einreichung von Community-Plugins

Um in die offizielle Community-Plugin-Liste eingetragen zu werden, muss der Quellcode in einem Git Repository auf GitHub gehostet werden. Folgend muss ein Pull Request auf das offizielle Repository geöffnet werden[26], und in die `community-plugins.json` Datei ein Eintrag im folgenden Format hinzugefügt werden:

[26] `https://github.com/obsidianmd/obsidian-releases`

```
{
    "id": "obsidian-plantuml",
    "name": "PlantUML",
    "description": "Generate PlantUML diagrams",
    "author": "Johannes Theiner",
    "repo": "joethei/obsidian-plantuml"
},
```

Auflistung 2: Beispielhafter Eintrag eines Plugins

Die einzelnen Komponenten sind dabei wie folgt:

- `id` : Eindeutige ID
- `name` : Anzeigename
- `description` : Kurze Beschreibung
- `author` : Name(n) der/des Entwickler(s)
- `repo` : GitHub Repository ID

Nach einer Überprüfung des Plugin Quellcodes durch zwei Mitglieder des Obsidian Teams kann ein Plugin direkt innerhalb von Obsidian heruntergeladen werden. In diesem Code Review wird der Code auf offensichtliche Programmierfehler, übliche Fehlerquellen und fehlerhafte Verwendung der Obsidian API's überprüft. Nach dieser initialen Überprüfung werden keine weiteren Überprüfungen durchgeführt, nur im Falle der Meldung eines Verstoßes gegen die vorhanden Richtlinien, etwa im Fall einer kritischen Sicherheitslücke wird eine erneute Überprüfung durchgeführt.

2.7.5. Verteilung von Plugins und Updates

Beim Installieren und Aktualisieren von Plugins wird die im vorherigen Abschnitt erwähnte `community-plugins.json` Datei konsultiert. Jedes Plugin Repository muss im Hauptordner des Hauptbranches eine `manifest.json` Datei enthalten(Auflistung 3), in dieser wird die aktuellste verfügbare Version angegeben. Nur alphanumerische Zeichen und Punkte sind innerhalb der Versionsnummer zulässig. Von dem GitHub Release des gleichen Namens werden die vom Entwickler bereitgestellten `manifest.json` , `main.js` , sowie optional `styles.css` Dateien heruntergeladen und in den entsprechenden Ordner geladen.[12]

Die `manifest.json` hat dabei das folgende Format:

```
{
        "id": "obsidian-plantuml",
        "name": "PlantUML",
        "version": "1.6.6",
        "minAppVersion": "0.15.0",
        "description": "Render PlantUML Diagrams",
        "author": "Johannes Theiner",
        "authorUrl": "https://github.com/joethei/",
        "isDesktopOnly": false,
        "fundingUrl": "https://github.com/sponsors/joethei"
}
```

Auflistung 3: Beispielhaftes Plugin-Manifest

Die einzelnen Komponenten sind dabei wie folgt:

- `id` : Eindeutige ID
- `name` : Anzeigename
- `version` : Aktuelle Version des Plugins.
- `minAppVersion` : Niedrigste Obsidian Version in der das Plugin funktioniert.
- `description` : Kurze Beschreibung
- `author` : Name(n) der/des Entwickler(s)
- `authorUrl` : Webseite/GitHub Profil des/der Entwickler(s)
- `isDesktopOnly` : Plugin funktioniert nur auf dem Desktop, oder auch auf mobilen Endgeräten.
- `fundingUrl` : Link, unter dem Nutzer spenden können, bei mehreren Möglichkeiten kann ein alternatives Format verwendet werden.

Sollte ein Plugin Updates veröffentlichen, die die Datenintegrität der Nutzer gefährden, kann eine Version oder das ganze Plugin für alle Nutzer deaktiviert werden. Dies kann durch einen Eintrag in die `community-plugin-deprecation.json`[27] Datei erreicht werden. Bei Verstoß gegen die Community Plugin Richtlinien kann ein Plugin auch vollständig ausgeschlossen werden.

2.7.6. Legacy Editor

Mit Einführung vom sogenannten *Live Preview mode* im Dezember 2021[28] wurde unter anderem auch die verwendete Editor-Bibliothek CodeMirror[29] auf die Version 6 aktualisiert. Dazu musste auch in Obsidian intern viel Code umgeschrieben werden, weil die Architektur von CodeMirror vom dessen Entwicklern komplett neu gedacht wurde. Zu-

[27] `https://github.com/obsidianmd/obsidian-releases/blob/master/community-plugin-deprecation.json`
[28] `https://forum.obsidian.md/t/obsidian-release-v0-13-0-insider-build/26919`
[29] `https://codemirror.net`

vor wurde Version 5 von CodeMirror verwendet.[30] CodeMirror 5 und CodeMirror 6 sind nicht kompatibel.

Community-Plugins können ebenfalls Teile von CodeMirror verwenden. Plugins die nur Code für CodeMirror 5 enthalten funktionieren nur, wenn in den Obsidian Einstellungen der *Legacy-Editor* aktiviert ist. Entsprechend funktionieren Plugins die CodeMirror 6 verwenden, nur wenn diese Einstellung deaktiviert ist.

In der Zukunft soll der *Legacy-Editor* entfernt werden, entsprechend sollen auch alle Plugins entfernt werden, die von diesem abhängen.

[30] `https://codemirror.net/5/`

3. Konzept

In diesem Kapitel soll der aktuelle Stand der Forschung zum Pflegezustand von Open-Source-Projekten gezeigt werden, und warum diese nicht mehr gepflegt werden. Auch soll erläutert werden wie in dieser Arbeit die ungepflegten und inkompatiblen Erweiterungen erkannt werden können.

3.1. Warum Richtlinien aufstellen?

Nach Lehman's Gesetz des ständigen Wandels muss Software ständig gewartet werden, sonst wird diese über die Zeit nicht zufriedenstellend werden, und somit absterben [13].

Nach Coelho, Valente, Milen u. a. können Software Projekte in mehrere Aktivitätsklassen eingeteilt werden:

Under maintenance: Das Projekt wird aktiv gepflegt, neue Features sind in Arbeit.
Deprecated: Das Projekt ist überholt, die Entwickler planen keine Änderungen am Quellcode mehr vorzunehmen.
Finished: Nach Ansicht der Entwickler ist das Projekt fertiggestellt. Nur Fehlerbehebungen sind noch geplant, sollten Fehler gemeldet werden.
Stalled: Die Entwicklung ist ins Stocken geraten, zum Beispiel aufgrund anderer Prioritäten der Entwickler.[14]

In dieser Arbeit sollen Erweiterungen erkannt werden, die als Deprecated oder Stalled klassifiziert sind. Erweiterungen die als Finished klassifiziert können ebenfalls erkannt werden, und müssen daher manuell ausgewertet werden.

Die in dieser Arbeit erstellten Analysen und Richtlinien sollen dafür verwendet werden die entsprechend klassifizierten Plugins aus der offiziellen Erweiterungsliste auszusortieren. Auch sollen Hilfestellungen und Benachrichtigungen für Entwickler bereitgestellt werden, damit diese nicht von Änderungen an der API überrascht werden.

3.2. Wieso werden viele Open-Source Projekte vernachlässigt oder aufgegeben?

Nach einer Untersuchung von 104 Projekten durch Coelho und Valente, schlagen die meisten Projekte fehl, weil diese entweder von einem Konkurrenzprojekt verdrängt werden, veraltet sind, keine Zeit für, oder kein Interesse an der Weiterentwicklung besteht[15]. Geben genug Entwickler die Arbeit an einem Projekt auf ist dieses als aufgegeben angesehen werden.

Wie stark gefährdet ein Projekt für dies ist, kann anhand des sogenannten Bus Faktor's geschätzt werden, dieser gibt an wie viele Entwickler außer Gefecht gesetzt werden dürfen, bevor das Projekt ins Chaos stürzt [16]. So steigt nach Samoladas, Angelis und Stamelos mit jedem Entwickler die Überlebensfähigkeit eines Projekts um 15,8% [17, S. 14].

3.3. Wieso geben Entwickler die Arbeit an Open-Source Projekten auf?

Nach Miller, Widder, Kästner u. a. geben viele Open-Source Entwickler aufgrund beruflicher Umstände die Mitarbeit an OSS Projekten auf, etwa aufgrund eines Jobwechsels zu einer Firma, die Mitarbeit an Open-Source-Software nicht unterstützt, oder durch fehlende Zeit aufgrund einer Arbeits-/Lebensumstellung. Auch Interessensverlust und fehlende Unterstützung von anderen Entwicklern wurden in der Umfrage zwischen 151 Befragten erwähnt [18, S. 6].

3.4. Wie können ungepflegte Open-Source Projekte erkannt werden?

Eine gerne von Nutzern vorgeschlagene Metrik ist die Aktivität, so wird gerne angenommen das ein Projekt, welches seit mehreren Monaten keine Updates erhalten hat, nicht mehr funktionstüchtig ist. Auch in der Literatur(z.B. [19]) wird diese Metrik teilweise verwendet. Die Festsetzung eines Zeitrahmens für eine solche Metrik ist nicht trivial und Projekte können auch mit Aktivität als ungepflegt eingestuft werden[20, S. 1]. Auch ein Projekt ohne Aktivität kann als gepflegt gezählt werden, etwa wenn die Funktionalität fertiggestellt ist, Fehler aber nur sehr selten gemeldet aber schnell behoben werden.

Eine Arbeit von Coelho, Valente, Silva u. a. verwendet historische Daten zu Commits, Issues, Pull Requests und bewertet diese mit einem Machine-Learning Model. Die Anzahl von vergebenen Sternen, Hauptentwicklern[31] und die Anzahl der Zeilen hat keinen nennenswerten Einfluss. Projekte mit mehr Mitwirkenden werden tendenziell besser und häufiger gepflegt [20, S. 7 f.].

Für den Erfolg eines Open-Source Projektes werden häufig gewisse Praktiken empfohlen, etwa das Vorhandensein einer Lizenz, Verhaltensregeln und Vorlagen zur Erstellung von Issues und Pull Requests. Nach Coelho, Valente, Milen u. a. haben diese einen unerheblichen Einfluss auf den Pflegezustand eines Projekts. Die Verwendung von Continuous Integration, Leitlinien für Beiträge oder für Erstbeitragende empfohlene Issues haben einen kleinen Einfluss [14, S. 19].

3.5. Wie können inkompatible Erweiterungen erkannt werden?

Bevor vorher öffentliche API's entfernt werden, ist es empfohlen diese als veraltet zu markieren, um Entwickler genügend Vorwarnung zu geben. Methoden werden etwa als veraltet markiert, weil diese andere existierende Methoden duplizieren, nicht vollständig funktionieren oder nicht benötigt werden. Auch aufgrund von Änderungen des API-Designs oder durch Schwachstellen können Methoden als veraltet markiert werden [21]. Durch eine Analyse der von Plugins aufgerufenen Methoden der API kann der Einfluss der zu entfernenden Methoden bestimmt werden, und die Entwickler über die Entfernung benachrichtigt werden. Nach Entfernung der Methoden können Plugins entfernt werden, die nicht die empfohlenen Änderungen durchgeführt haben. Weil bisher keine Pläne existieren, als veraltet markierte Methoden aus der Obsidian API zu entfernen, werden in dieser Arbeit keine entsprechenden Analysen durchgeführt.

[31] Verantwortlich für mindestens 80% der Commits

Auch der Aufruf von internen API's kann zu Inkompatibilitäten führen, sollten diese geändert oder entfernt werden. So wird im Allgemeinen von der Benutzung von internen API's abgeraten, weil diese häufiger geändert werden, und aufgrund von nicht veröffentlichter Dokumentation unerwartete Nebeneffekte haben können. Bei einer Untersuchung von Eclipse Erweiterungen von Drittentwicklern wurde etwa festgestellt das in 49,8% der Fälle die aufgerufenen internen API's nicht mit der Folgeversion kompatibel sind. Bei Verwendung von offiziell dokumentieren Methoden ist dies in nur 3,3% der Fall [22]. Durch die relativ stabileren internen API's von Obsidian sind solche eindeutigen Werte nicht zu erwarten. Hier soll eine Analyse der von Plugins aufgerufenen internen API's Aufschlüsse bieten.

Für bereits geplannte Änderungen wie die Entfernung des Legacy Editors (Abschnitt 2.7.6) und die Abschaltung der ES5 Kompatibilität (Abschnitt 2.3) sollen weitere spezifische Analysen erstellt werden, um den betroffenen Plugin Entwicklern eine rechtzeitige Vorwarnung zu geben, und Plugins entfernen zu können, bei denen nicht auf diese Vorwarnung reagiert wurde.

4. Rahmenbedingungen

Die Auswertung der existierenden Erweiterungen für Obsidian innerhalb dieser Arbeit ist unter der Einhaltung verschiedener Bedingungen erfolgt. In diesem Abschnitt wird betrachtet, warum keine Auswertung von Nutzungsdaten, historischen Daten oder Erweiterungen, die nicht in TypeScript entwickelt wurden, durchgeführt wird.

4.1. Telemetrie

Telemetrie, aus dem griechischen: Fernmessung, fasst die Messung und Übertragung von Daten zusammen. Gesammelte Daten müssen nicht am Messort verarbeitet werden, so werden etwa Wetterdaten zentral verarbeitet, aber von vielen kleinen Wetterstationen gesammelt.

In der Softwareentwicklung wird Telemetrie zu unterschiedlichen Zwecken verwendet. Zum einen zur Diagnose von Fehlern, so werden Fehlermeldungen und die Einstellungen der Software & Hardware automatisch gesammelt und ausgewertet. Weiterhin können per Telemetrie gesammelte Daten auch zur Analyse des Nutzerverhaltens verwendet werden, um etwa die Nutzungsfreundlichkeit der Software zu verbessern. Zur Unterscheidung zwischen Menschen und Bots kann Telemetrie ebenfalls verwendet werden.[23]

Von Datenschützern wird die Verwendung von Telemetrie in Software kritisiert, auch weil in vielen Fällen mehr Daten gesammelt werden als für den eigentlichen Zweck benötigt. Auch sind Datenschutz affine Nutzer Anwendungen mit Telemetrie kritisch gegenüber eingestellt. etwa weil Telemetrie häufig standardmässig aktiviert ist, und teilweise nur mit externen Mittel vollständig deaktiviert werden kann.[32]

Ein Obsidian Vault enthält potenziell sehr private Informationen eines Nutzers, entsprechend ist Teil der Philosophie von Obsidian ist, wie bereits in Abschnitt 2.7 erwähnt, das Nutzer ihre eigenen Daten besitzen, daher wird innerhalb Obsidian keine Telemetrie verwendet. Für die Sammlung von Anwendungsdaten in einem solchen Kontext muss ein großes Vertrauen zu den Entwicklern der Anwendung, und den Betreibern der Plattform die gesammelte Daten verarbeitet, existieren. Ansonsten kooperieren viele Nutzer nicht mit einer Datensammlung[25], aus diesem Grund ist in Obsidian keine Telemetrie eingebaut.

Erweiterungen können zur Bereitstellung von Funktionalität mit externen Servern kommunizieren. Informationen über den Benutzer oder dessen Gerät dürfen aber ebenfalls nicht innerhalb einer Erweiterung gesammelt werden. Eine Überprüfung, ob ein Plugin Daten sammelt, kann durch Quellcode Analysen und Auswertung der verwendeten Abhängigkeiten, durchgeführt werden. Dies ist nicht Bestandteil dieser Arbeit.

So können in dieser Arbeit keine Daten über aktuell von Nutzern verwendete Plugins gesammelt werden. Auch für Plugin-Entwickler sind keine entsprechenden Daten verfügbar, da auch hier keine Daten gesammelt werden dürfen. Würde Telemetrie in Obsidian vorhanden sein, könnten der zweite Teil dieser Arbeit, das Untersuchen aller Erweiterungen auf Inkompatibilität mit der Anwendung, mit der Auswertung von gesammelten

[32] Siehe z.B. Windows 10 und das quelloffenene Konkurenzprodukt Zettlr [24]

Fehlermeldungen ersetzt werden, wie etwa durchgeführt von Lenarduzzi, Stan, Taibi u. a. für Android Apps [26].

4.2. Historische Daten

Die in dieser Arbeit erstellten Analysen sind nur Momentaufnahmen, ein Vergleich mit Ergebnissen vorheriger Durchläufe der einzelnen Analysen findet nicht statt.

Obsidian ermöglicht seit dem 30. Oktober 2020 die Entwicklung von Community Plugins, zuvor konnte bereits seit Juni 2020 Erweiterungen mit dem Volcano Projekt[33] Erweiterungen für Obsidian entwickelt werden. Durch diesen kurzen Zeitraum, das Durchschnittsalter aller Plugins Repositories ist 439 Tage, wobei viele nur einen einzigen Zweck erfüllen, und somit nicht viel Pflege benötigen. Vermutet wurde, das aus diesem Grund rein aktivitätsbasierte Metriken nicht die gewünschten Ergebnisse liefern können. Auch sind diese historischen Daten nicht immer verfügbar(vgl.[27, S. 25])

So wurde entschieden in dieser Arbeit nur Metriken zu betrachten die auf aktuell vorhandenen Daten basieren. Auch aufgrund des zeitlich begrenzten Rahmens dieser Arbeit werden keine älteren Ergebnisse mit neuen verglichen.

4.3. Quellcodeauswertung

Obsidian Erweiterungen können in einer Vielzahl von Programmiersprachen entwickelt werden, aufgrund der Dominanz von TypeScript für die Entwicklung von Obsidian Plugins wird hier nur TypeScript Quellcode analysiert. Die Implementation für weitere Programmiersprachen wird auch aus Zeitgründen übersprungen. Da JavaScript, und somit auch TypeScript, eine sehr dynamische Programmiersprache ist, sind die hier erhobenen Daten als nicht vollständig zu betrachten.

Eine Auflistung der bisher für die Plugin-Entwicklung verwendeten Sprachen ist in Tabelle 1 abgebildet. Hier ist zu beachten, das eine Programmiersprache nicht im Plugin Quellcode selbst verwendet werden muss, sondern auch als Hilfsskript oder für die Konfiguration verwendet werden kann. So ist der große Anteil von JavaScript einzelnen Konfigurationsdateien geschuldet, die in der Standardvorlage bereits enthalten sind.

[33] https://github.com/kognise/volcano

Programmiersprache	**Anzahl der Plugins**
TypeScript	822
JavaScript	812
Python	12
Rust	5
Ruby	2
Elm	1
Nix	1
Lua	1
Clojure	1
F#	1
Emacs Lisp	1

Tabelle 1: Verteilung der verwendeten Programmiersprachen

5. Realisierung

Die einzelnen Analysen verwenden Daten aus unterschiedlichen Quellen, die meisten Daten stammen von der durch GitHub bereitgestellten API, weitere Daten werden durch die Auswertung von Quellcode gesammelt. In diesem Kapitel soll gezeigt werden wie aus diesen unterschiedlichen Quellen die benötigten Daten gesammelt werden. In Abbildung 2 werden schemenhaft die einzelnen Komponenten dargestellt.

Im ersten Abschnitt wird die Abfrage von Daten aus der GitHub API besprochen, im zweiten wird die Sammlung von Metadaten aus Git, und im dritten Abschnitt die Quellcodeanalyse besprochen. Der vierte Abschnitt betrachtet die Implementation eines simplen Smoketests. Der letzte Abschnitt zeigt wie die Auswertung dieser Daten umgesetzt ist.

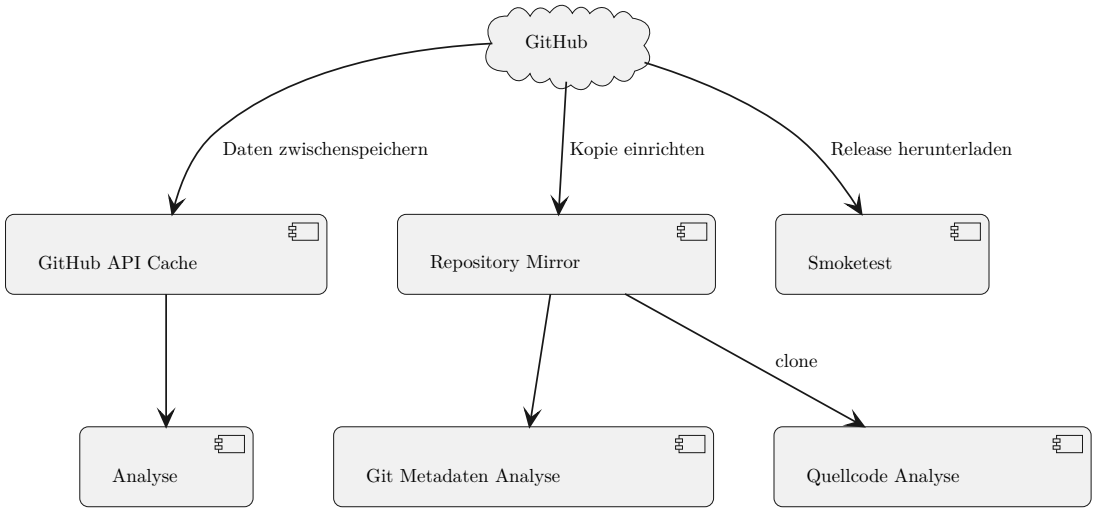


Abbildung 2: Verschiedene Popularitätsmetriken

5.1. Abfrage von Daten aus der GitHub REST API

Um die GitHub Ratenwertgrenze von 5000 Anfragen[34] pro Stunde nach Möglichkeit nicht zu überschreiten wurde ein kleiner Server in der Programmiersprache Go[35] entwickelt. Dieser Server nimmt HTTP Anfragen an und stellt entsprechende Anfragen an die GitHub Rest API[36].

Die von der GitHub API erhalten Daten werden von diesem Server leicht umgewandelt, da viele Daten enthalten sind, die hier für die Analysen nicht benötigt werden (Beispiel der vollständigen Rückgabe der GitHub API im digitalen Anhang in der `sample_GH_API.json` Datei oder in der GitHub Dokumentation[37]. Ein Beispiel der Rückgabe der eigenen API

[34] `https://docs.github.com/en/developers/apps/building-github-apps/rate-limits-for-github-apps#default-server-to-server-rate-limits-for-githubcom`
[35] `https://go.dev`
[36] `https://docs.github.com/en/rest`
[37] `https://docs.github.com/de/rest/issues/issues?apiVersion=2022-11-28#get-an-issue`

ist in Auflistung 4 dargestellt und enthält nur von den verschiedenen Analysen benötigte Daten.

So enthält die Rückgabe der GitHub API beispielsweise für jeden Nutzer Links zum Nutzerprofil, verwendetem Avatar, eigene Repositories, die mit Sternen versehene Repositories und weiteres. Für die Analysen in dieser Arbeit ist bei Nutzern, nur der Nutzername und in welchem Verhältnis dieser Nutzer zum entsprechenden Repository steht, relevant.

```
[
  {
    "title": "Found a bug",
    "url": "https://github.com/octocat/Hello-World/issues/1347",
    "user": "github_user",
    "association": "NONE",
    "assignees": ["octocat"],
    "description": "I'm having a problem with this.",
    "labels": ["bug"],
    "created": "2011-04-22T13:33:48Z",
    "closed": "2011-06-22T13:33:48Z",
    "comments": [
      {
        "user": "octocat",
        "message": "Will be fixed in v42",
        "association": "CONTRIBUTOR",
        "created": "2011-05-22T13:33:48Z",
        "updated": "2011-05-22T13:33:48Z"
      },
    ]
  },
]
```

Auflistung 4: Beispielhafte Rückgabe der eigenen API

Zusätzlich werden vom entwickelten Cache Server unter Umständen die Ergebnisse mehrerer Anfragen an die GitHub API kombiniert. Die GitHub API gibt zum Beispiel nicht alle Issues eines Repositories zurück, sondern teilt diese in Seiten ein, eine Seite enthält bis zu 30 Issues und benötigt eine eigene Anfrage[38].

Bestimmte Analyseskripte[39] werden mit eingebauten Verzögerungen durchgeführt, so wird nach Erhalt der Daten eines Plugins einige Sekunden gewartet bevor die Daten des

[38] `https://docs.github.com/en/rest/guides/using-pagination-in-the-rest-api?apiVersion=2022-11-28`

[39] Issue, Pull Request und Commit Analyse

nächsten Plugins angefragt werden. Dies ist nötig, weil hier mehrere Anfragen an die GitHub API kombiniert werden, und sonst die Ratenwertgrenze überschritten würde.

Die erstellten Rückgaben werden von der eigenen API bis zu 24 Stunden zwischengespeichert, um im Falle einer erneuten Anfrage zurückgegeben zu geben, ohne erneut Anfragen an die GitHub API stellen zu müssen. Im späteren produktiven Einsatz ist dies nicht relevant, da die Analysen in größeren Zeitabständen durchgeführt werden, während der Entwicklung der Analyseskripte aber vorteilhaft.

Die einzelnen Analyseskripte sind in Python[40] geschrieben und fragen die jeweils benötigten Daten per HTTP vom eigenen Cache Server ab. Ein Beispiel eines Analyseskripts ist in Auflistung 5 zu sehen, in diesem Skript wird die Rückgabe auf bestimmte Schlüsselwörter untersucht, und die gefundenen Sätze zur manuellen Auswertung in eine CSV Datei gespeichert.

[40] `https://www.python.org`

```
deprecatedStrings = ['unmaintained', 'maintained', 'maintainer',
↪  'legacy editor']

if __name__ == '__main__':

    plugins = requests.get(BASE_URL + "plugins").json()
    frameData = list()

    for plugin in plugins:
        id = plugin['id']
        print(id)

        result = requests.get(BASE_URL + "plugin/" + id +
        ↪  "/readme").text.lower()

        sentences = result.split('.')
        for sentence in sentences:
            Wenn eines der Wörter im Satz enthalten
            if any(string in sentence for string in deprecatedStrings):
                frameData.append({
                    'id': id,
                    'sentence': sentence
                })

    # in CSV speichern zur manuellen Auswertung
    dataframe = pandas.DataFrame(frameData, columns=['id', 'sentence'])
    dataframe.fillna(value=0)
    dataframe.to_csv('output/csv/readme.csv')
```

Auflistung 5: Quellcodeausschnitt aus dem Analyse Skript des README Inhalt's

Im Anschluss werden die gesammelten Daten mit pandas[41] verarbeitet und die Ergebnise als weitere CSV Datei gespeichert, Bei Bedarf werden diese Ergebnisse mit Seaborn[42] graphisch dargestellt. Die so generierten Grafiken werden für die manuelle Betrachtung auf einem Server hochgeladen. Die einzelnen Analysen werden wöchentlich durchgeführt, nach Ende dieser Arbeit mit einer geringeren, noch nicht festgelegten, Frequenz.

[41] https://pandas.pydata.org
[42] https://seaborn.pydata.org

5.2. Git Metadaten

Die zur Spiegelung verwendete Software Gitea bietet eine eigene REST API an, über die alle Commits innerhalb eines Repositories angefragt werden können.

5.3. Analyse des Plugin Quellcodes

Um die den Entwicklern zugänglichen Statistiken nicht zu sehr zu verfälschen ist auf einer *Gitea*[43] Instanz eine Spiegelung der einzelnen Git Repositories eingerichtet. Dieser Spiegelserver wird durch das tägliche automatische Ausführen des `mirror.py` Skriptes um neue Repository Konfigurationen erweitert, wenn neue Erweiterungen vom Obsidian Team akzeptiert wurden. Als Nebeneffekt kann diese Repository-Kopie für das Nachvollziehen von gewissen destruktiven Aktivitäten der Entwickler verwendet werden. Etwa kann der Quellcode auch nach Löschen des originalen Repositories eingesehen werden.

Für die Durchführung dieser Analysen wird der aktuelle Quellcode der einzelnen Erweiterungen heruntergeladen, dazu wird das entsprechende Git Repository gecloned. Im Anschluss werden alle Abhängigkeiten durch einen Aufruf des `npm install` Befehls heruntergeladen [44]. Auch hier ist ein Spiegelserver im Einsatz, um die Server der *npm Registry*[45] nicht unnötig zu belasten.

Für eine einfacherere Automatisierung und verlässliche Ergebnisse werden alle benötigten Abhängigkeiten und der Plugin Quellcode bei jedem Durchgang erneut von den Spiegelservern geladen, und diese nicht von vorheriger Durchläufen wiederverwendet. Diese Vorbereitung nimmt mehrere Stunden in Anspruch. Das Herunterladen der Abhängigkeiten ist nötig damit die verwendete `TypeChecker` API Ergebnisse liefern kann.

Zur Analyse wird wie in Auflistung 6 dargestellt der Quellcode mithilfe der TypeScript Compiler-API geladen um den resultierenden AST[46] analysieren zu können. Nur wenn die für TypeScript Projekte empfohlene Konfigurationsdatei `tsconfig.json` existiert, wird der Quellcode analysiert.

[43] `https://gitea.io`
[44] Der *Node package manager*(NPM) ist der am häufigsten verwendete Packet Manager in JavaScript und TypeScript Projekten
[45] `https://npmjs.com`
[46] siehe Abschnitt 2.5

```
for (const repo of repos) {
    //TypeScript Konfigurationsdatei finden
    const configFile = ts.findConfigFile('cloned/' + repo,
    ↪  ts.sys.fileExists, "tsconfig.json");
    //wenn Datei nicht exisitiert ist dies kein TypeScript Projekt,
    ↪  Überprüfung überspringen
    if (!configFile) continue;

    //Konfiguration laden & Projekt initialisieren
    const config = ts.readConfigFile(configFile, ts.sys.readFile);
    const sourcePath = path.join(process.cwd(), "cloned", repo);
    const {options, fileNames, errors} =
    ↪  ts.parseJsonConfigFileContent(config, ts.sys, sourcePath);
    const program = ts.createProgram({options, rootNames: fileNames,
    ↪  configFileParsingDiagnostics: errors});

    //Quellcode analysieren
    for (const sourceFile of program.getSourceFiles()) {
        visitNode(sourceFile, program, repo);
    }
    ...
}
```

Auflistung 6: Initialisieren der Quellcodeanalyse

5.3.1. Offiziell deklarierte Funktionen

In der offiziellen Typendefinition `obsidian.d.ts`[47] ausgewiesene Funktionen können während der Entwicklung eines Plugins direkt importiert und aufgerufen werden. Weil diese offiziell dokumentiert sind, können diese Methodenaufrufe am einfachsten erkannt werden.

Zu Überprüfung, ob eine Funktion in der Obsidian API deklariert ist, kann die `getSymbolAtLocation()` Funktion der TypeScript Compiler API verwendet werden, diese wird unter anderem auch von Code Editoren und IDE's zur Bereitstellung der ''Zur Definition springen'' Funktionalität verwendet. Gibt diese eine Deklaration zurück wird der Dateipfad der Deklarationsdatei ausgelesen, und geprüft, ob dieser dem Pfad an dem NPM standardmäßig die Obsidian API Deklaration ablegt, entspricht.

In Auflistung 7 ist dies ausschnittweise dargestellt.. Für die spätere Auswertung wird die Funktionsdeklaration zusammen mit der Position der einzelnen Aufrufe in einer JSON

[47] `https://github.com/obsidianmd/obsidian-api/blob/master/obsidian.d.ts`

Datei gespeichert. Ein Beispiel ist in Auflistung 8 zu sehen. Dabei wird unter jeder Deklaration alle in Plugins gefundene Aufrufe dieser Funktion/Variable aufgelistet.

```
const symbol = program.getTypeChecker().getSymbolAtLocation(child);
if (symbol && symbol.valueDeclaration) {
    if (symbol.valueDeclaration.getSourceFile()
    .fileName.includes("node_modules/obsidian/obsidian.d.ts")) {
        const key = symbol.valueDeclaration.getText();
        const value: CallDeclaration = {
            plugin: repo,
            file: child.getSourceFile().fileName.split(repo + "/")[1],
            pos: child.pos
        }
        //value in Datei schreiben
        ...
    }
}
```

Auflistung 7: Quellcodeauschnitt der alle verwendeten Methoden/Variablen der offiziellen API ausliest

```
{
  "addChild(child: MarkdownRenderChild): void;": [
    {
      "plugin": "<plugin-id>",
      "file": "main.ts",
      "pos": 781
    },
}
```

Auflistung 8: Ausschnitt aus der Rückgabe der einzelnen Quellcodeanalysen

5.3.2. Manuell deklarierte Funktionen

Ist eine Funktion oder Variable nicht offiziell deklariert, kann im eigenen Code die Definition erweitert werden, und die definierten Methoden und Variablen im Anschluss wie gewohnt aufgerufen werden. Ein Beispiel einer solchen Definition ist in Auflistung 9 dargestellt.

```
declare module "obsidian" {
        interface App {
                plugins: PluginManager;
        }

        interface PluginManager {
                manifests: Record<string, PluginManifest>;
                enablePlugin(id: string): Promise<boolean>;
                disablePlugin(id: string): void;
        }
}
```

Auflistung 9: Beispielhafte Moduldeklaration

Dazu werden die einzelnen manuellen Moduldeklarationen ausgewertet, wenn diese für das ''obsidian'' Modul gedacht sind. In Auflistung 10 ist ein Teil dieser Implementation dargestellt. Eine Erkennung wie häufig diese innerhalb von Plugin Quellcode aufgerufen werden ist nicht möglich, da von der bereits im vorherigen Abschnitt erwähnten `getSymbolAtLocation()` Funktion diese nicht erkannt werden.

```
if (ts.isModuleDeclaration(node)) {
    node.forEachChild(child => {
        if (ts.isStringLiteral(child)) {
            if (child && child.getSourceFile() && child.getText() ===
            ↪ '"obsidian"') {
                node.forEachChild(child => {
                   moduleBlock(child, repo)
                });
            }
        }
    });
}
```

Auflistung 10: Auschnitt aus der Auswertung von Moduldeklarationen

5.3.3. Ignorierte Funktionsaufrufe

Durch die dynamische Natur von JavaScript kann auf Funktionen und Variablen zugegriffen werden, auch wenn diese nicht definiert sind. Zur Laufzeit wird ein Fehler erzeugt, sollte die Funktion/Variable nicht vorhanden sein. Wird eine nicht existierende Funktion/Variable einer in TypeScript definierten Klasse aufgerufen die, schlägt das transpilie-

ren mit einer Fehlermeldung fehl. Um diese Fehlermeldungen zu unterdrücken, kann von Entwicklern entweder ein Kommentar mit `//@ts-ignore` oder `//@ts-expect-error` vor die entsprechende Zeile gesetzt werden.

In Auflistung 11 werden alle Kommentare im Quellcode ausgelesen, sind die entsprechenden Schlüsselwörter vorhanden, wird die nachfolgende Zeile in eine Datei zur manuellen Auswertung gespeichert.

Eine Unterscheidung zwischen Obsidian internen Methoden und Methoden aus anderen Quellen ist nicht möglich, dies kann nur durch eine manuelle Auswertung durchgeführt werden. Eine Durchsuchung der Rückgaben dieser Analyse auf bestimmte Schlüsselwörter kann dies nicht erreichen, weil die Anzahl der Obsidian internen Funktionen zu groß ist.

```
1  const commentRanges = ts.getLeadingCommentRanges(
2              node.getSourceFile().getFullText(),
3              node.getFullStart());
4          if (commentRanges?.length) {
5              for (const commentRange of commentRanges) {
6                  //Kommentar auslesen
7                  const comment = node.getSourceFile().getFullText()
8                      .slice(commentRange.pos, commentRange.end);
9                  if (comment.includes("@ts-ignore") ||
10                     comment.includes("@ts-expect-error")) {
11                     const line = node.getSourceFile()
12                     .getLineAndCharacterOfPosition(commentRange.pos);
13                     //Dem Kommentar nachfolgende Zeile auslesen
14                     const commented =
                       ↪  node.getSourceFile().getFullText()
15                         .split('\n')[line.line + 1].trim();
16
17                     //Kommentar in Datei schreiben
18                     ...
19                 }
20             }
21         }
```

Auflistung 11: Quellcodeausschnitt der alle Kommentare in Quellcode ausliest und auf bestimmte Stichwörter untersucht

Alternativ kann nach `any` ge-casted werden, entsprechender Code sieht wie folgt aus: `(this.app as any).commands`. In Auflistung 12 wird ein Abschnitt der Implementation dieses Scans gezeigt.

Auch hier kann nicht erkannt werden, ob eine Funktion von Obsidian oder eine Funktion anderen Ursprungs aufgerufen wird, diese Überprüfung muss ebenfalls manuell durchgeführt werden.

```
if (ts.isAsExpression(node)) {
        node.forEachChild(child => {
            if (child.kind === SyntaxKind.AnyKeyword) {
                const code = node.parent?.parent;
                if (code) {
                    const value: CallDeclaration = {
                        plugin: repo,
                        file: node.getSourceFile().fileName.split(repo
                        ↪  + "/")[1],
                        pos: code.pos,
                    }
                    //value in Datei speichern
                }
            }
        });
```

Auflistung 12: Quellcodeauschnitt der `as any` Überprüfung

5.3.4. Monkey-Patches

Aus der Erfahrung des Autors wird im Obsidian Umfeld für diesen Zweck meistens die *monkey-around* Bibliothek[48] verwendet. Andere Bibliotheken oder manuelle Ansätze werden in dieser Arbeit nicht betrachtet, da eine automatische Erkennung nicht möglich ist. Erkannt werden die Codestellen, an denen Monkey Patching verwendet, hier nur aufgrund der Methodennamen der bereits erwähnten Bibliothek. In Auflistung 13 ist ein Beispielhafter Monkey-Patch dargestellt[49], Auflistung 14 zeigt einen Ausschnitt der Implementation dieses Scans.

[48] `https://github.com/pjeby/monkey-around`
[49] Vollständige Implementation in `https://github.com/joethei/obsidian-key-promoter/blob/master/src/main.ts`

```
this.uninstallCommand = around(this.app.commands, {
            executeCommandById(oldMethod: Function) {
                return dedupe("key-promoter", oldMethod, function
                ↪  (...args: any) {
                    const result = oldMethod && oldMethod.apply(this,
                    ↪  args);
                    //code...
                    return result;
                })
            }
        });
```

Auflistung 13: Beispielhafter Monkey-Patch

```
if (symbol.getName() === "around" || symbol.getName() === "dedupe") {
    let key = "";
    identifier.parent?.forEachChild(child => {
        if (ts.isPropertyAccessExpression(child) &&
        ↪  child.getSourceFile())
            key += child.getText();
        if (ts.isObjectLiteralExpression(child)) {
            child.forEachChild(child => {
               if (ts.isPropertyAssignment(child) ||
               ↪  ts.isMethodDeclaration(child)) {
                    child.forEachChild(child => {
                       if (ts.isIdentifier(child)) {
                            const newKey = key + "." + child.getText();
                            //in Datei speichern
                       }
                    });
               }
            });
        }
    });

}
```

Auflistung 14: Quellcodeauschnitt der Implementation des Monkey-Patch Scans

5.4. Smoketest

Für diesen Test werden alle Fehlermeldungen die während des Startprozesses erzeugt werden analysiert.

Dazu wird die aktuellste Version eines Plugins in den dafür vorgesehenen Ordner kopiert, und ein fast leerer Test Vault gestartet. In diesem Test Vault ist ein eigens entwickeltes Plugin installiert, welches das geladene Plugin initialisiert und alle Fehlermeldungen abfängt, um diese in Textdateien zu speichern. Ein Ausschnitt aus der Implementation dieses Plugins ist in Auflistung 15 zu sehen.

```
1  //write all calls to `console.error("error message here")` to file.
2  console.error = (...args) => {
3          this.writeToFile("error", args);
4  }
5
6  //write JS errors to file
7  window.onerror = (err) => {
8          this.writeToFile("win_error", err);
9  }
10
11 //write unhandled promise rejections to file
12 window.onunhandledrejection = (err: PromiseRejectionEvent) => {
13         this.writeToFile("promise_rejection", err.reason.message);
14 }
15
16 //write uncaught exceptions to file
17 process.on('uncaughtExceptionMonitor', (err) => {
18         this.writeToFile("exception", err.message);
19 });
```

Auflistung 15: Quellcodeausschnitt des Smoketest Plugins

Nach 10 Sekunden wird dieser Test Vault wieder geschlossen und die erstellten Textdateien zur manuellen Auswertung in eine separate Ordnerstruktur kopiert, und der verwendete Test Vault zurückgesetzt. Ein Ausschnitt aus der Implementation dieses Skripts ist in Auflistung 16 zu sehen.

```
1  for plugin in plugins:
2      id = plugin['id']
3      print(id)
4
5      # Plugin in Ordner kopieren
6      shutil.copytree("releases/" + id, PLUGIN_PATH + id)
7
```

```
8       # Smoketest Vault in Obsidian öffnen, nach 10 Sekunden wieder
        ↪  schließen
9       os.system("open obsidian://vault/smoketest")
10      time.sleep(10)
11      os.system("open obsidian://close")
12
13      # Fehlermeldungen zur Auswertung in entsprechenden Ordner kopieren
14      copy_file("error.md", id)
15      copy_file("win_error.md", id)
16      copy_file("promise_rejection.md", id)
17      copy_file("exception.md", id)
18
19      # Plugin aus Vault entfernen
20      shutil.rmtree(PLUGIN_PATH + id)
21
22      time.sleep(5)
```

Auflistung 16: Auschnitt des Smoketest Skripts

Diese Analyse kann, anders als andere nicht vollautomatisch auf einem Server ausgeführt werden. Sie muss auf einem System mit Bildschirm ausgeführt werden, welches mehrere Stunden in Anspruch nimmt.

Durch eine Anpassung des Quellcodes von Obsidian wäre eine vollständige Automatisierung möglich. Weitere Analysen der Funktionalität können nicht durchgeführt werden, weil diese eine Instrumentalisierung der einzelnen Plugins erfordern würde.

6. Auswertung

In diesem Kapitel soll untersucht werden in welchem Umfang sich einzelne Metriken dazu eignen inkompatible oder ungepflegte Plugins zu erkennen. In dem ersten Abschnitt werden verschiedene Metriken analysiert, die sich potenziell dazu eignen Aussagen über den Pflegezustand der einzelnen Plugins zu treffen, während im zweiten Abschnitt hauptsächlich Metriken ausgewertet werden die Aussagen über die Kompatibilität eines Plugins treffen könnten. Im letzten Abschnitt werden die Erweiterungsrichtlinien anderer Produkte untersucht, ob diese Richtlinien enthalten die ungepflegte oder inkompatible Erweiterungen unterbinden sollen.

Die einzelnen Metriken sollen auf ihre Eignung für einen produktiven Einsatz überprüft werden, dabei ist wichtig, dass die Metriken möglichst automatisch erfasst und ausgewertet werden können.

6.1. Erkennung von ungepflegten Erweiterungen

In diesem Abschnitt sollen verschiedene Metriken zur Erkennung von ungepflegten Erweiterungen ausprobiert werden, ob sich diese für einen produktiven Einsatz eignen.

In Abbildung 3 werden die Tage seit der Erstellung der einzelnen GitHub Repositories gezeigt. Dabei ist ein gleichbleibender Trend zu erkennen, die Anzahl neuer Erweiterungen pro Zeiteinheit ist vergleichsweise stabil. Aber auch einige Plugins sind dabei, die schon länger vor der Einreichung existiert haben. Dies ist zu erkennen daran, dass diese Einträge höhere Balken haben als ihre Nachbarn.

In einzelnen Fällen wurde bisher der Wechsel des genutzten GitHub Repositories zugelassen. Da bisher immer auf einen Fork gewechselt wurde, sind die entsprechenden Repositories teilweise deutlich jünger als das Original. Diese Einträge haben in der Abbildung einen deutlich niedrigeren Balken als ihre Nachbarn.

Vermutet wird das viele der älteren Erweiterungen nicht mehr gepflegt werden, oder die Pflege nur eingeschränkt durchgeführt wird, auch weil der durchschnittliche Bus-Faktor bei 0,9 liegt(vgl. Abschnitt 3.2), sprich durchschnittlich kennt nur ein Entwickler den Quelltext ausreichend um Änderungen schnell durchführen zu können. Das Durchschnittliche Plugin hat 3 Entwickler.

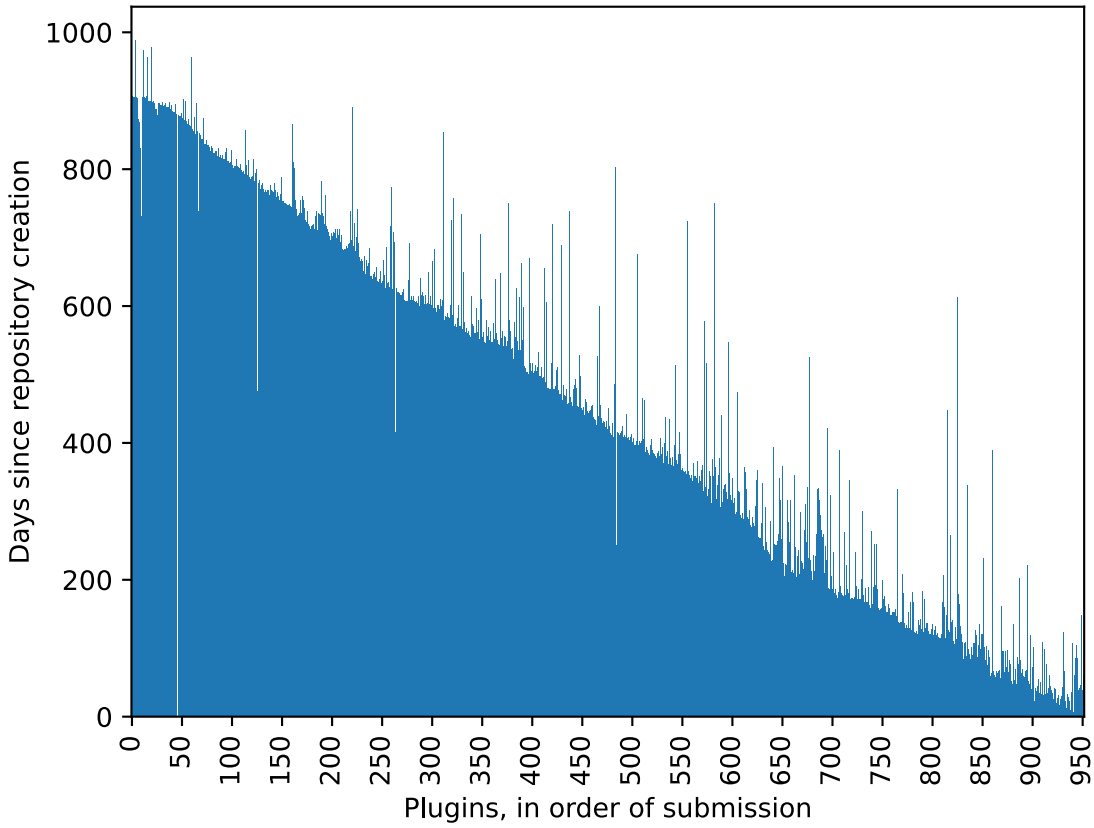


Abbildung 3: Tage seit Erstellung des Repositories

6.1.1. Popularität

Obsidian sammelt keine Daten von Nutzern, wie bereits in Abschnitt 4.1 erläutert. Auch Community Plugins dürfen keine Daten über Nutzer sammeln. Aus diesem Grund sind keine Daten über die Anzahl der aktiv genutzten Plugins vorhanden. Als Metrik für den Einfluss eines Plugins auf das Obsidian Community Plugin Ökosystems wird hier daher die Popularität betrachtet.

Anhand dieser Metrik soll entschieden werden wie mit einem als ungepflegt/inkompatibel erkannten Plugin vorgefahren werden soll. So soll bei populären Erweiterungen erst nach neuen Entwicklern gesucht werden, bevor diese entfernt werden, während dies bei kaum genutzten Erweiterungen nicht wichtig ist.

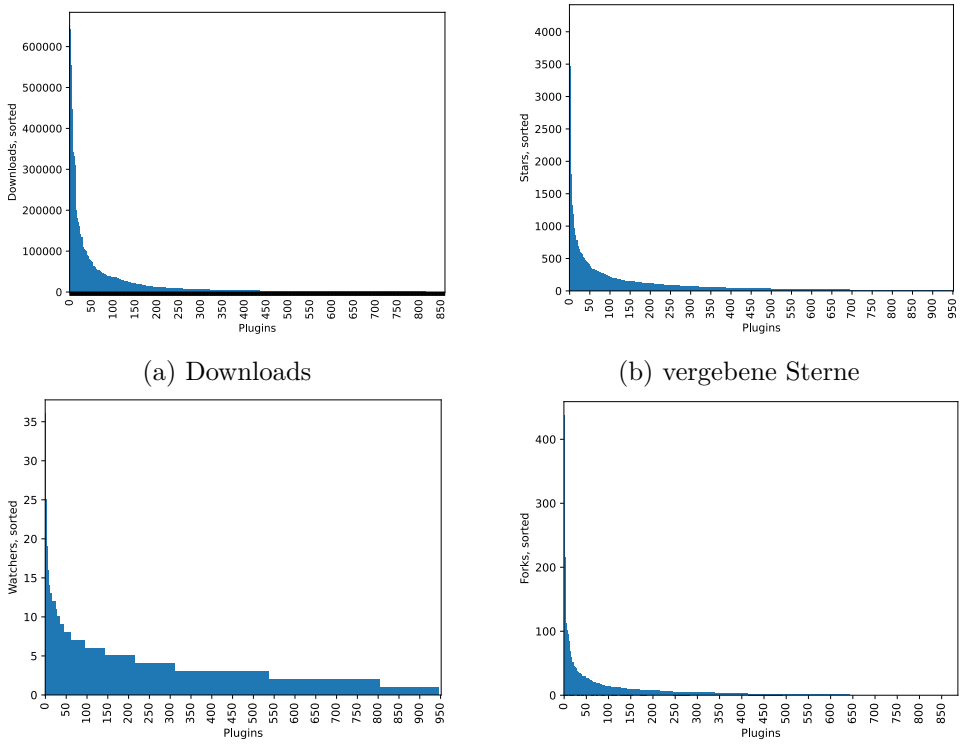
GitHub bietet öffentlich nur gebündelte Statistiken an. So decken die verfügbaren Daten den gesamten Zeitraum seit Erstellung des Repositories oder der Veröffentlichung des Releases ab. Nur Nutzer mit Push-Zugriff auf ein Repository haben Zugriff auf weitere Statistiken mit Zeitkomponente [50]. Entsprechend können keine Aussagen darüber getroffen werden, ob aktuell Interesse an einem Plugin existiert, oder beispielsweise im letzten Jahr niemand besagtes Plugin installiert hat.

Um trotzdem einen groben Überblick zu erhalten, werden Daten zur Downloadzahl,

[50] https://docs.github.com/en/rest/metrics/traffic?apiVersion=2022-11-28

vergebenen Sternen[51], Beobachter[52] und Forks erhoben.

Entsprechende Daten sind in Abbildung 4 visualisiert. Ein sehr kleiner Teil der vorhandenen Plugins sind sehr beliebt, wie an den vergleichsweise hohen Downloadzahlen (a) und vergebenen Sternen (b) zu erkennen ist. Die Verteilung der Forks (d) lässt auf ein geringes Interesse am Beitragen an der Entwicklung der meisten Plugins schließen. Bei der Anzahl der Beobachter (c) ist ein ähnlicher Trend zu beobachten, wenn auch in einem kleineren Rahmen.



(a) Downloads

(b) vergebene Sterne

(c) Beobachter

(d) Forks

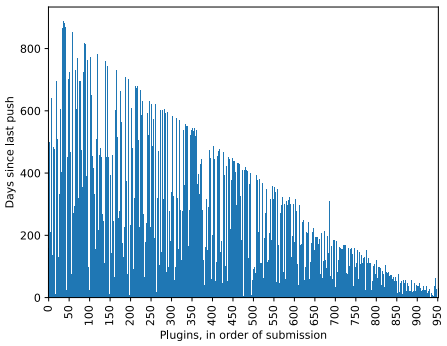
Abbildung 4: Verschiedene Popularitätsmetriken

6.1.2. Aktivität

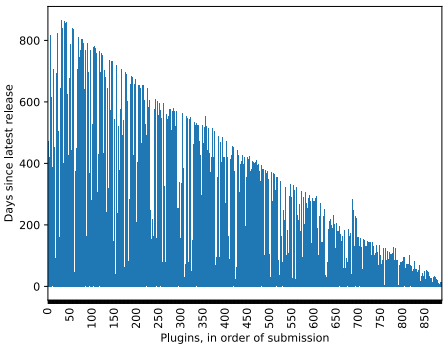
Bei Betrachtung der Aktivität auf den GitHub Repositories ist ein teils deutlicher Trend zu erkennen. Verglichen mit dem Alter der Repositories (Abbildung 3) ist zu erkennen, das die letzte Änderung des Quellcodes (Abbildung 5 a) und der letzte Release (Abbildung 5 b)) bei vielen Plugins nah am Zeitpunkt der Einreichung liegen. So kann angenommen werden das viele Plugins seit der Einreichung nicht mehr großartig geändert wurden.

[51] Auf GitHub kann dies einem "Gefällt mir" gleichgesetzt werden

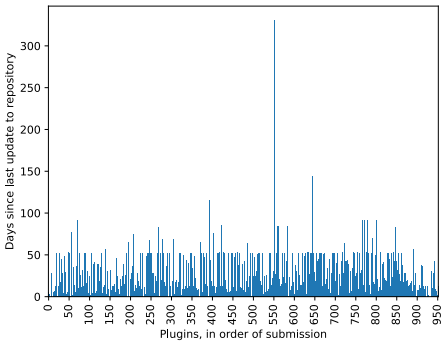
[52] diese Nutzer erhalten Benachrichtigungen über Aktivitäten, daher werden diese häufiger zu Mitentwicklern als andere Nutzer[28]



(a) Tage seit letzter Quellcodeänderung



(b) Tage seit letztem Release



(c) Tage seit letzter Aktivität auf dem GitHub Repository

Abbildung 5: Verschiedene Aktivitätsmetriken

Aus der Erfahrung des Autors dieser Arbeit erfüllt eine nicht gerade geringe Zahl von vorhandenen Plugins nur einen einzigen Zweck, häufig werden auch nur stabile APIs verwendet. Diese benötigen Erfahrungsgemäß wenig Pflege. Bei alleiniger Anwendung einer aktivitätsbasierten Metrik würden diese Plugins daher fehlerhafterweise aussortiert werden.

Würde zum Beispiel eine Metrik von einem Jahr ohne Aktivität verwendet werden, wie in der Arbeit von Khondhu, Capiluppi und Stol vorgeschlagen[19], würden 122 Plugins bei Betrachtung der Zeit seit der letzten Änderung des Quellcodes, oder 200 Plugins nach der Zeit seit dem letzten Release, betroffen sein.

Bei einem Zeitraum von 2 Jahren würden 7 Plugins bei Verwendung der letzten Quellcodeänderung, sowie 23 Plugins bei Verwendung des letzten Releases, betroffen sein. Hier wurden zuvor mehr betroffene Erweiterungen vermutet, auch aufgrund der Popularitätsmetriken aus dem vorherigen Abschnitt.

Anhand des Zeitraums seit der letzten Änderung des GitHub Repositories (Abbildung 5 c) kann abgeschätzt werden wie aktiv das Interesse von Nutzern an dem entsprechenden Plugin noch ist. Bei 18 Plugins ist die letzte Aktivität auf dem Repository über 150 Tage her. Als Aktivität zählen etwa Quellcodeänderungen, das Erstellen von Issues, Pull Requests sowie das Hinzufügen von Kommentaren. Auch das Forken, Starren und hinzufügen zu einer Watchlist(siehe Abschnitt 6.1.1) werden als Aktivität von der GitHub API gezählt[29].

Die Zeiträume sind hier willkürlich gewählt worden, eine korrekte Festsetzung eines solchen Zeitraums ist nicht trivial. Bei einer manuellen Untersuchung der 18 Plugins ohne Repository Aktivität innerhalb von 150 Tagen wurde bei 2 Repositories Hinweise auf fehlerhafte Funktionalität oder eine Vernachlässigung gefunden[53]. Eine Überprüfung durch Installation der Plugins wurde nicht durchgeführt. Aufgrund des benötigten manuellen Aufwands, und der großen Anzahl an falsch-positiven Einträgen eignet sich diese Metrik nicht für den produktiven Einsatz.

6.1.3. Archivierte Repositories

Entsprechend der GitHub Dokumentation ist der Archivierungsstatus dafür vorgesehen, deutlich zu machen das keine Pflege des Repositories mehr vorgenommen wird.

> Du kannst ein Repository archivieren, damit es allen Benutzer*innen nur mit Lesezugriff zur Verfügung steht und damit klar ist, dass es nicht mehr aktiv unterhalten wird.[30]

Bei einer Untersuchung der Repositories wurden 11 archivierte Repositories gefunden (siehe Tabelle 2). Dieser Status kann als eindeutiges Zeichen gesehen werden, das die entsprechenden Plugins nicht mehr gepflegt werden.

Entsprechend werden diese Plugins aus der offiziellen Liste entfernt werden. In dieser Arbeit werden diese Repositories weiterhin betrachtet, um untersuchen zu können, ob diese Plugins bei Auswertung weiterer Analysen ebenfalls auffällig sind.

Plugin-ID
slated-obsidian
code-block-copy
obsidian-query2table
obsidian-indent-lines
tq-obsidian
obsidian-code-copy
tag-page-preview
obsidian-dynamic-toc
obsidian-wordle
notion-like-tables
obsidian-embedded-note-paths

Tabelle 2: Betroffene Plugins

Diese Analyse kann vollautomatisch durchgeführt werden, dauert nur wenige Minuten und kann somit als sehr zuverlässig angesehen werden. Geplant ist es diese Analyse wöchentlich automatisch durchzuführen, und sobald ein neues Repository archiviert wird das Obsidian Team darüber in Kenntnisse zu setzen, damit entsprechende Schritte eingeleitet werden können.

[53] Stand: 18.01.2023

6.1.4. Releases

Wie bereits in Abschnitt 2.7.5 erläutert, werden Plugins direkt von den Releases auf dem GitHub Repository heruntergeladen. Entsprechend der dort definierten Regeln, wurde überprüft, ob die neuste markierte Version tatsächlich herunterladbar ist. Dabei wurden 4 Plugins gefunden bei denen dies nicht möglich ist (siehe Tabelle 3)[54].

Plugin-ID	**Version**	**letzter Push vor x Tagen**	**Grund**
obsidian-spaced-repetition	1.8.0	7	Version existiert nicht
import-foundry	0.8.1	47	Assets fehlen
link-info-server	1.0.1	309	Assets fehlen
obsidian-advanced-new-folder		271	Kein Release vorhanden

Tabelle 3: Plugins mit einer nicht herunterladbaren Version

Bei drei dieser Plugins (Zeile 2-4) kann auf dem entsprechenden GitHub Repository kein Issue gefunden werden, welches den Autor auf diesen Zustand hinweist. Das *Advanced New Folder* Plugin ist hier ein Sonderfall, da hier keine Issues aktiviert sind[55] und somit der Entwickler nicht auf diesem Wege informiert werden kann. Im Offiziellen Community *Discord* Server sind 4 Referenzen auf dieses Problem des Plugins zu finden. Nur beim *Spaced Repetition* wurde der Autor über diesen Zustand informiert, und ist nach Erhebung dieser Daten wieder installierbar gewesen.

Ein Plugin, das seit längerer Zeit nicht mehr installierbar ist, kann als ungepflegt angesehen werden. Erwartet wurde eine Anzahl von betroffenen Erweiterungen, die im selben Rahmen liegt wie die hier gefundenen 4 . Diese Metrik kann als sehr zuverlässig angesehen werden.

Zur Vermeidung dieses Problems in der Zukunft soll diese Analyse in Zukunft täglich durchgeführt werden, sollte ein Plugin nicht installierbar sein, soll automatisch ein Issue auf dem Repository geöffnet werden, um die Entwickler zu informieren. Sollte ein Plugin für mehrere Wochen nicht installierbar sein wird es aus der Liste entfernt werden.

6.1.5. Readme

Angelehnt an die Arbeit von Coelho und Valente (Abschnitt 3.2) wurde die README Datei[56] aller Repositories auf die folgenden Schlüsselwörter durchsucht:

- unmaintained

[54] Stand: 04.12.2022

[55] Und schon seit der Einreichung kein Release hat, da zu diesem Zeitpunkt nur eine manuelle Überprüfung des Codes erfolgte

[56] Der Inhalt einer solchen Datei wird als erstes gezeigt wen das GitHub Repository geöffnet, oder das Plugin im Obsidian Community Plugin Browser angesehen wird. Enthalten ist häufig eine Beschreibung des Plugins, mit Bedienungsanleitung.

- maintained
- maintainer
- deprecated

Zusätzlich wird auch auf *legacy editor* untersucht, da dies ebenfalls auf eine Vernachlässigung hindeuten kann(Siehe Abschnitt 2.7.6).

Insgesamt wurden 52 Instanzen dieser Schlüsselwörter gefunden. Nach einer manuellen Überprüfung wurden 16 relevante Einträge identifiziert. Die restlichen Einträge können als falsch positiv angesehen werden. Im folgenden zwei dieser falsch positiven Einträge.

> This official plugin maintained by the Readwise team enables you to ...[57]

Um Sätze wie ''this plugin is no longer maintained'' erkennen zu können wird auf das Wort ''maintained'' überprüft, in einem Fall wie diesem schlägt die Erkennung daher ebenfalls an.

> This functionality is deprecated and has been migrated to metadata-menu plugin[58]

In diesem Fall ist nur eine bestimmte Funktionalität als überholt dokumentiert, diese wurde zuvor in ein anderes Plugin migriert.

Acht der 16 relevanten Einträge deuten darauf hin das der Autor kein Interesse mehr an der Weiterentwicklung hat (Tabelle 4, Zeile 1 - 8). Bei drei Plugins wird ein Entwickler gesucht, der entweder die Pflege übernimmt, oder bei dieser mithilft(Zeile 9 - 11). Bei drei weiteren Einträge wird darauf hingewiesen das diese nur im *Legacy editor* (Abschnitt 2.7.6) funktionieren (Zeile 12 - 14). Die letzten zwei Einträge deuten darauf hin das dieses Plugin entwickelt wurde, weil ein anderes nach Ansicht des Entwicklers nicht mehr gepflegt wird (Zeile 15 - 16). Nur 2 Plugins empfehlen ein Plugin mit ähnlicher Funktionalität als Alternative.[59]

Entfernt werden sollen jene Plugins, bei denen der Entwickler kein Interesse mehr hat, und in den Issues kritische Fehler gemeldet wurden. Sobald der *legacy editor* aus der Anwendung entfernt wird, sollen auch Erweiterungen entfernt werden, die nur mit diesem funktionieren.

[57] `https://github.com/readwiseio/obsidian-readwise`
[58] `https://github.com/mdelobelle/obsidian_supercharged_links#link-context-menu-extra-options`
[59] Stand: 05.01.2023

	Plugin-ID	Text
1	markdown-prettifier	no longer maintained. for similar functionality i recommend
2	obsidian-autocomplete-plugin	This plugin is longer maintained as I'm not using Obsidian anymore.I might update it in the future
3	garble-text	This plugin is no longer maintained. Sorry for any inconvinience caused.
4	tag-page-preview	I'm unable to find the time to keep this repository well maintained and up-to-date with Obsidian. Life gets in the way.
5	obsidian-dynamic-toc	I'm unable to find the time to keep this repository well maintained and up-to-date with Obsidian. Life gets in the way.
6	obsidian-wordle	NOTE: This is no longer actively maintained.
7	sekund	This plugin won't receive any more updates. It is in other words in need of a maintainer.
8	notion-like-tables	NOTE: This plugin is no longer maintained. If you would like to continue development please fork the project and make your own version.
9	obsidian-checklist-plugin	LOOKING FOR ANOTHER MAINTAINER TO HELP OUT There's quite a bit of work to do on this plugin still and i've been neglecting it because work is too busy.
10	flashcards-obsidian	I am currently looking out for a co-maintainer.
11	obsidian-pocket	I am looking for someone to take over as the maintainer
12	obsidian-text-expander	CM6-based editor (default option in Obsidian >= v13.0.0) is not supported. Consider using Legacy editor.
13	obsidian-vale	Obsidian Vale doesn't yet support the new editor that was introduced in Obsidian 13. To continue using the plugin for version 13 you can enable the Legacy Editor.
14	obsidian-codemirror-options	That being said this plugin will continue to be maintained and enhanced in order to provide a form of "Live Preview" to anyone who wishes to stay on the legacy desktop editor version.
15	alx-folder-note	alx-folder-note is a maintained completely rewritten and improved folder note plugin that covers all core features of the original folder note plugin
16	editing-toolbar	Thanks to the cmenu plugin which gave me a lot of inspiration but this plugin has not been maintained for more than a year so I re-modified it and added a lot of interesting features

Tabelle 4: Identifizierte Plugins

Die Sammlung der benötigten Daten ist vollständig automatisierbar, die Auswertung der gesammelten Daten nimmt einige Minuten manueller Arbeit in Anspruch. Aufgrund des sehr eindeutigen Endergebnisses dieser Analyse kann diese Metrik als zuverlässig gesehen werden.

6.1.6. Issues

Die von Nutzern erstellten Issues sollen in diesem Abschnitt darauf untersucht werden, wie viele Fehlermeldungen geöffnet sind, und wie schnell diese von den Entwicklern wieder behoben werden. Durch eine Auswertung der Titel von Issues soll hier zusätzlich identifiziert werden, welche Plugins von Nutzern als vernachlässigt angesehen werden.

Bei 16 Repositories sind keine Issues aktiviert gewesen, für diese Analysen werden diese Repositories daher nicht betrachtet. In den Community-Plugins dürfen diese Plugins weiterhin verbleiben, da dies kein eindeutiger auf eine Verwahrlosung hindeutender Faktor ist. Bei neuen Einreichung wird bereits durch eine automatische Warnung auf deaktivierte Issues hingewiesen.

In den meisten Fällen sind die Issues deaktiviert, weil diese Repositories entweder Forks eines anderen Plugins, oder Forks des offiziellen Beispiel-Plugins sind. Standardmäßig haben Forks auf GitHub keine Issues aktiviert, da Forks seitens GitHub's dazu gedacht sind die gemachten Änderungen dem originalen Projekt per Pull-Request wieder zuzuführen, und in einem solchen Fall keine Issues benötigt werden [31].

Für die Erkennung der von Nutzern als vernachlässigt angesehenen Erweiterungen werden hier die Titel der einzelnen Issues auf Schlüsselwörter durchsucht. Zusätzlich zu den in vorherigen Abschnitt verwendeten Worten, wurde zusätzlich auf die Schlüsselwörter

- abandoned
- enable plugin
- failed to load

durchsucht. Diese wurden gewählt durch eine manuelle Stichproben-artige Untersuchung einiger Issue-Titel.

Insgesamt wurden so 56 Issues gefunden, 14 davon können als falsch positive betrachtet werden, in Tabelle 5 alle 42 Issue Titel aufgezählt die hier relevant sind. Das erste hier aufgelistete Issue wurde von Entwickler selbst, die folgenden 11 von Nutzern erstellt, die fragen, ob das Plugin noch gepflegt wird. Die restlichen Issues melden Fehler beim Installieren oder starten des Plugins. Eine manuelle Auswertung aller hier gelisteten Issues ergab insgesamt 24 relevante Erweiterungen. Entfernt werden sollen jene Plugins, bei denen es keine Reaktion seitens der Entwickler auf diese von Nutzern erstellten Issues für einen noch nicht definierten Zeitraum gab, und wenn das gemeldete Problem reproduziert werden kann.

	Plugin-ID	**Titel**	**offen seit Tagen**
1	simple-embeds	Looking for maintainers	79
2	flashcards-obsidian	Looking for a co-maintainer	160
3	note-refactor-obsidian	Is this plugin still maintained?	282
4	obsidian-day-planner	Is this plugin still being maintained / developed?	417

5	periodic-notes	Is the repo still maintained? Do you need some help?	73
6	obsidian-gallery	Is this an abandoned plug-in?	123
7	text-snippets-obsidian	Maintainer Status and future ownership?	304
8	obsidian-pandoc	Maintainer status and future ownership?	452
9	workspaces-plus	Workspaces Plus Plugin Development Status	73
10	netwik	Is this dead?	351
11	obsidian-local-images	Is this repository still maintained?	132
12	remotely-save	Is this project abandoned?	48
15	table-editor-obsidian	Failed to enable plugin in 1.0	94
16	obsidian-day-planner	Failed to load plugin obsidian-day-planner	42
17	code-block-copy	Failed to load plugin manifest	578
18	workbench-obsidian	Failed to load plugin workbench obsidian	135
19	obsidian-mind-map	Failed to load mind-map	419
20	tag-wrangler	Failed to enable plugin in 1.0	94
21	obsidian-plugin-todo	failed to load this plugin	7
22	obsidian-plantuml	Failed to enable plugin in 1.0	94
23	obsidian-dice-roller	Failed to enable plugin in 1.0	94
24	obsidian-admonition	Failed to enable plugin in 1.0	94
25	obsidian-tabs	Failed to load plug-in obsidian-tabs	147
26	obsidian-excalidraw-plugin	BUG: failed to load plugin excalidraw	36
27	obsidian-kanban	[Bug]: Failed to enable plugin in 1.0	94
28	quickadd	[BUG] Failed to load plugin quickadd on iPad	98
29	mx-bili-plugin	Media Extended BiliBili Plugin:failed to load	364

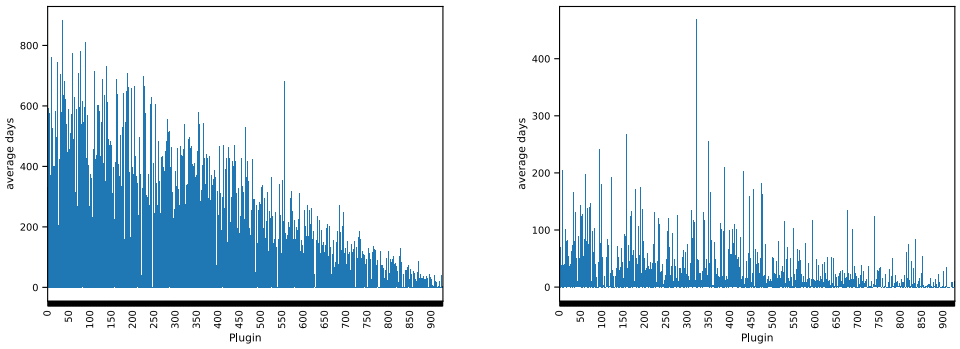
30	obsidian-markmind	Failed to load plugin obsidian-markmind	100
31	obsidian-prominent-starred-files	Failed to load	510
32	remotely-save	ios14.4 failed to load [Bug]:	28
33	remotely-save	[Bug]: Ipad & Android Failed to load plugin	72
34	remotely-save	[Bug]: Failed to enable plugin in 1.0	94
35	remotely-save	[Bug]: Android Pad encounter 'failed to load plugin remotely-save'	104
37	obsidian-memos	Failed to load memos with version 1.9.0	306
38	lapel	Failed to load plugin	227
39	obsidian-drag-n-drop-plugin	Failed to load ... out of range for changeset of length ...	153
40	writing	Failed to Load on Android Tablet	100
41	webpage-html-export	Can't enable plugin...	10
42	weekly-review	Failed to load plugin on windows 10	62

Tabelle 5: Issues, bei denen der Titel auf Verwarlosung des Repositories deuten könnte (Stand: 18.02.2023)

Die für diese Metrik benötigten Daten können vollautomatisch gesammelt werden, für das manuelle Aussortieren der nicht relevanten Issues werden nur wenige Minuten benötigt. Aus diesem Grund kann diese Metrik als zuverlässig angesehen werden, sowohl für die Erkennung von ungepflegten, als auch inkompatiblen Erweiterungen.

In Abbildung 6 a) wird gezeigt wie viele Tage die aktuell offenen Issues durchschnittlich auf einem Repository geöffnet ist. Auch hier ist wieder zu sehen das bei älteren Plugins Issues tendenziell schon länger geöffnet sind [60]. Da zu vermuten ist das einige der Issues Ideen für neue Features, und keine Fehlermeldungen sind, soll im nächsten Schritt untersucht werden wie die Verteilung der verschiedenen Kategorien ist.

[60] vgl. mit [32, S. 38] und mit Abbildung 6 b), welche das Durchschnittsalter der bereits geschlossenen Issues zeigt



(a) noch geöffnete Issues (b) bereits geschlossene Issues

Abbildung 6: Durchschnitt wie viele Tage Issues geöffnet sind

Dazu werden die einzelnen Issues anhand der von Kollaborateuren oder durch Vorlagen vergebenen Label kategorisiert. Unterteilt wird in 4 Kategorien:
Bug: Ein Nutzer meldet einen Fehler
Feature Request: Ein Nutzer hat eine Idee für ein neues Feature
Frage: Dem Nutzer ist nicht ganz verständlich wie ein Feature funktioniert
Unbekannt: Das Issue kann nicht klassifiziert werden, weil kein Label vergeben wurde
In Abbildung 7 ist die Verteilung dieser Kategorien dargestellt.

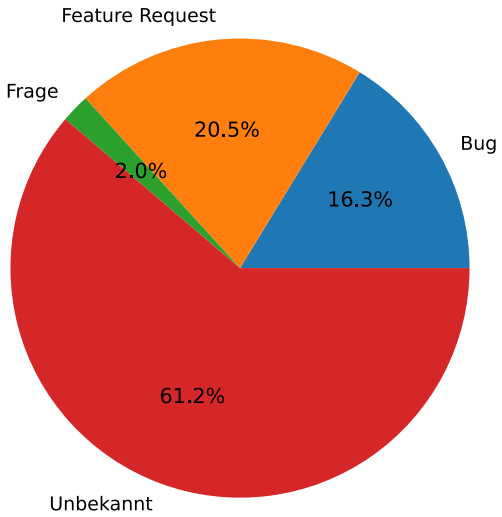


Abbildung 7: Verteilung der Issue-Kategorien

Untersucht werden sollte auch wie lange Fehlermeldungen ohne Reaktion der Entwickler bleiben. Aufgrund der geringen Anzahl von Issues die automatisch als Fehlermeldung klassifiziert werden können, hat diese Analyse eine geringe Aussagekraft für alle vorhandenen Erweiterungen. Wie in Abbildung 8 ersichtlich sind bei einigen wenigen Erweiterungen sowohl der letzte Release schon länger her, als auch die durchschnittliche Fehlermeldung schon länger geöffnet[61]. Insgesamt sind 194 Repositories auf dieser Grafik

[61] obere, rechte Ecke

abgebildet, bei den restlichen Repositories konnten keine als Bug klassifizierten Issues gefunden werden.

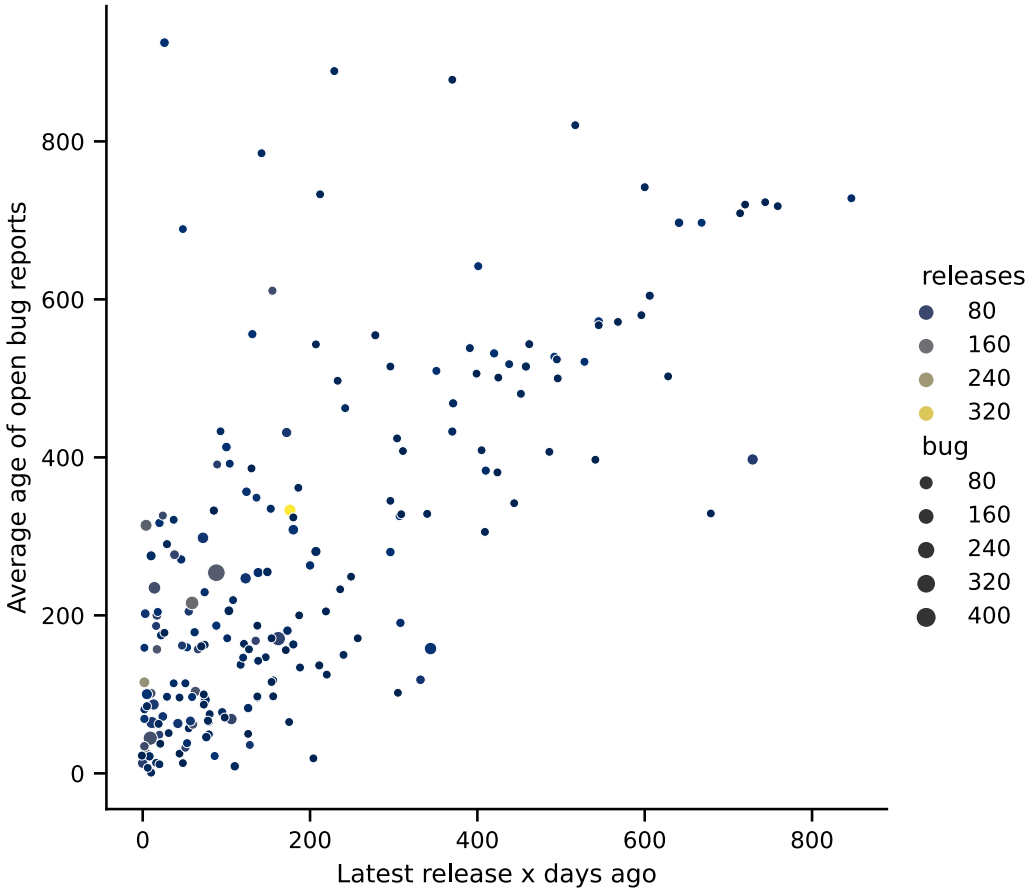


Abbildung 8: Durchschnittsalter von offenen Fehlermeldungen im Vergleich zum letzten Release

Werden alle offenen Issues betrachtet ist ein ähnlicher Trend zu beobachten(vgl. Abbildung 9). Bei vielen Erweiterungen ist das durchschnittliche Alter der geöffneten Issues proportional zu der Zeit, die seit dem letzten Release vergangen ist. Auf dieser Abbildung sind 705 Repositories dargestellt, bei den restlichen Repositories sind keine Issues vorhanden oder der letzte Release konnte nicht ermittelt werden.

Viele der Repositories mit lange geöffneten Issues und Releases, seit denen schon einige Zeit vergangen ist, sind auf den ersten Blick als ungepflegt anzusehen. Das Festlegen einer Grenze, bis zu der die einzelnen Repositories manuell überprüft werden sollten, ist nicht möglich, aus diesem Grund ist diese Metrik nicht vollständig für den Produktivbetrieb geeignet. Als Anhaltspunkt für Schwerpunktanalysen aber durchaus geeignet.

Wird etwa ein Zeitrahmen für beide Parameter von über 350 Tagen gesetzt, trifft dies auf 27 Plugins zu, durch eine manuelle Überprüfung konnten 9 dieser Einträge als ungepflegt bestätigt werden.

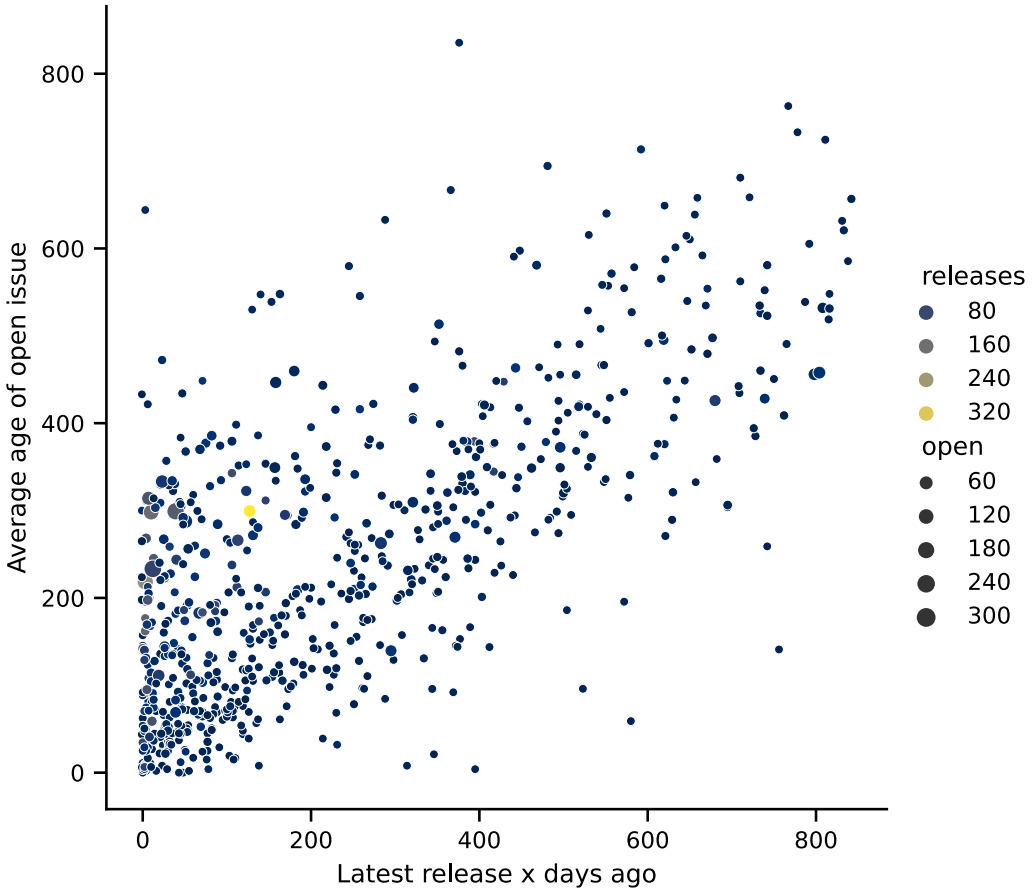


Abbildung 9: Durchschnittsalter von offenen Issues im Vergleich zum letzten Release

Die meisten geöffneten Issues haben keine Kommentare von Kollaborateuren erhalten, wie in Abbildung 10 ersichtlich. Dies deutet darauf hin das viele Entwickler diese nicht aktiv nutzen.

Auch einem Kollaborateur zugewiesen sind die wenigsten geöffneten Issues. Diese beiden Metriken kombiniert sind als ein negatives Zeichen zu sehen, da davon auszugehen ist das die Entwickler potenziell nicht auf etwaige Fehlermeldungen reagieren. Möglich ist auch das ein Entwickler bereits die geöffneten Issues gesehen hat, aber für eine Implementation/Fehlerbehebung einen späteren Zeitpunkt vorgesehen wurde, dies aber nicht öffentlich kommuniziert wird.

Werden hier alle Repositories gefiltert bei denen die geöffneten Issues durchschnittlich älter als 600 Tage sind, sowie keine Kommentare von einem Kollaborateur auf all diesen Issues vorhanden sind, werden 19 Repositories erfasst. Eine manuelle Auswertung ergibt das nur 5 dieser Erweiterungen als wirklich ungepflegt und defekt angesehen werden können. Aufgrund des manuellen Aufwands der für diese Metrik aufgewandt werden muss und die vielen falsch-positiven Einträge eignet diese sich nicht für einen produktiven Einsatz.

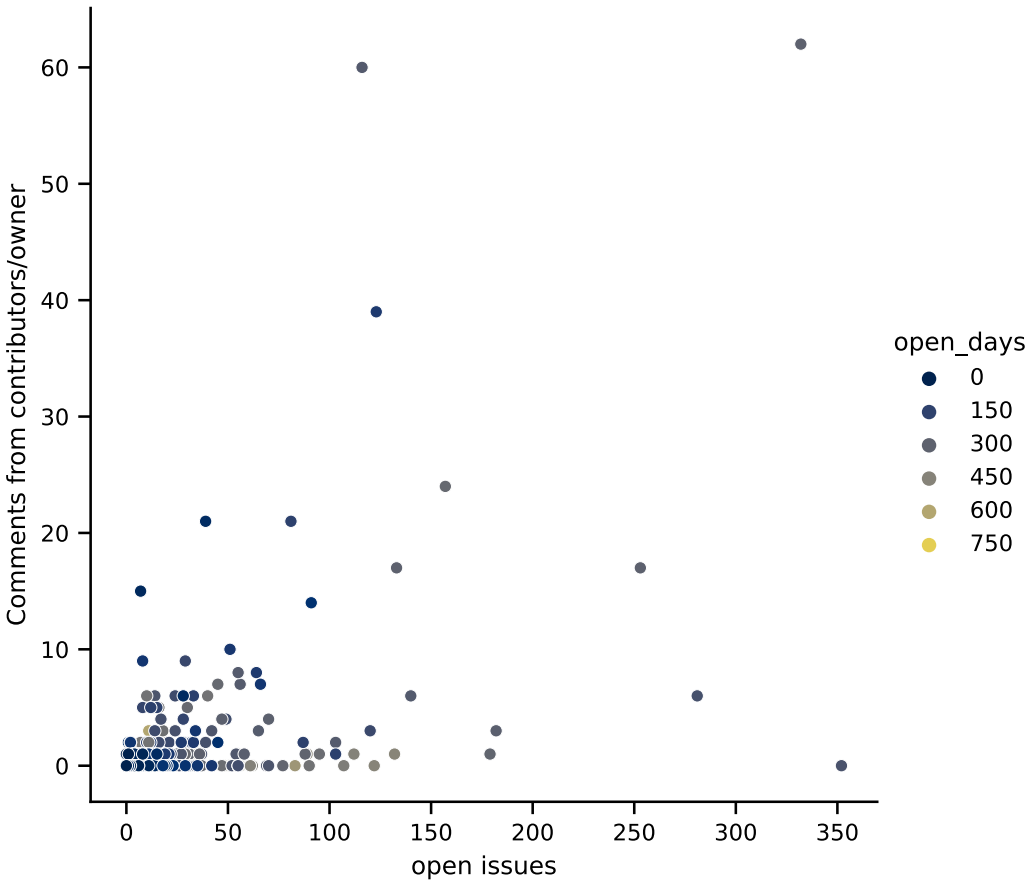


Abbildung 10: Verteilung der Kommentare von Kollaborateuren auf Issues

6.2. Erkennung von inkompatiblen Erweiterungen

In diesem Abschnitt sollen Daten zur Kompatibilität mit der aktuellen Obsidian Version ausgewertet werden. Entwickler können mit dem `minAppVersion` Parameter innerhalb der `manifest.json` Datei(vgl. Abschnitt 2.7.5) festlegen welche Obsidian Version mindestens für die Ausführung der aktuellen Plugin-Version benötigt wird, eine obere Grenze kann nicht gesetzt werden.

Hier sollen zuerst die von Erweiterungen genutzten API auf Aufrufe von nicht mehr vorhandenen, oder für die Entfernung vorgesehenen Methoden, untersucht werden. Des Weiteren werden alle Erweiterungen einen kleinen Smoketest durchlaufen, bei dem das Startverhalten der einzelnen Plugins betrachtet wird.

6.2.1. ES5 kompatible Erweiterungen

In der Zukunft soll die Unterstützung von für den ES5 Standard[62] kompilierten Erweiterungen abgeschaltet werden. Eine Analyse aller in TypeScript entwickelten Erweiterungen ergab das 84 Erweiterungen noch für ES5 kompilieren. Bevor die ES5 Unterstützung

[62] siehe 2.3

entfernt wird, sollen alle betroffenen Entwickler darüber benachrichtigt werden, sollten diese darauf nicht reagieren wird das betroffene Plugin entfernt. In den meisten Fällen ist das Aktualisieren des verwendeten Standards durch eine Änderung eines Parameters innerhalb der `tsconfig.json` Datei und das neu kompilieren, ausreichend.

6.2.2. Nutzung von APIs

In dieser Arbeit werden nur Erweiterungen betrachtet, die in TypeScript entwickelt wurden, weitere Programmiersprachen wurden aufgrund der großen Verbreitung von TypeScript nicht betrachtet (vgl. Abschnitt 4.3). Insgesamt wurde der Quellcode von 847 Erweiterungen für diesen Abschnitt automatisch ausgewertet.

Von den 7 offiziell als `@deprecated` markierten Funktionen/Variablen werden 5 im untersuchten Quellcode aufgerufen, in Tabelle 6 sind diese dargestellt. Sobald eine vollständige Entfernung dieser Funktionen/Variablen geplant sein, ist es geplant die Entwickler der betroffenen Plugins zu kontaktieren.

Methode/Variable	**Vorkommen**
activeLeaf: WorkspaceLeaf | null;	12
getUnpinnedLeaf(type?: string): WorkspaceLeaf;	9
splitActiveLeaf(direction?: SplitDirection): WorkspaceLeaf;	8
duplicateLeaf(leaf: WorkspaceLeaf direction?: SplitDirection): Promise<WorkspaceLeaf>;	2
cmEditor: CodeMirror.Editor;	1

Tabelle 6: Alle bisher als `@deprecated` markierten Funktionen

Die für den *Legacy Editor*(vgl. Abschnitt 2.7.6) reservierten Methoden wurden insgesamt 581-mal aufgerufen, aus 103 verschiedenen Plugins, vor der Entfernung des *Legacy Editor* soll manuell untersucht werden, inwiefern diese Plugins von diesem abhängen. Die Entwickler dieser Erweiterungen sollen dementsprechend informiert werden und Erweiterungen bei denen keine Änderungen durchgeführt werden entfernt werden. Zu diesen Methoden zählt die hierunter anderem auch die `cmEditor: CodeMirror.Editor` Variable aus der vorherigen Tabelle.

Unter den von Entwicklern manuell deklarierten Methoden/Variablen wurden keine gefunden, die intern bereits entfernt wurden. In Tabelle 7 sind die 20 am meisten deklarierten aufgelistet, diese sollen in Zukunft potenziell in die offizielle API Deklaration aufgenommen werden.

Methode/Variable	**Deklarationen**
enabledPlugins:Set<string>;	16
containerEl:HTMLElement;	14
file:TFile;	9
id:string;	9
enabled:boolean;	8
dom:HTMLElement;	8
viewRegistry:ViewRegistry;	8
containerEl:HTMLDivElement;	8
executeCommandById(id:string):void;	6
to:number;	5
from:number;	5
findCommand(id:string):Command;	5
folds:FoldPosition[];	4
pushScope(scope:Scope):void;	4
_loaded:boolean;	4
getPlugin<Textendskeyof-Plugins>(plugin:T):Plugins[T];	4
open():void;	4
onMarkdownFold():void;	4

Tabelle 7: Die 20 am meisten von Entwicklern selbst deklarierten, Obsidian internen Methoden/Variablen

In Tabelle 8 sind die 20 am meisten von Entwicklern per Kommentar von der *TypeScript* Compiler Überprüfung ausgeschlossen Quellcodezeilen aufgelistet, in der vollständigen Liste wurden ebenfalls keine Aufrufe identifiziert die nicht mehr innerhalb von Obsidian's Quellcode existieren. Eine Überprüfung welche der aufgerufenen Funktionen/Variablen in die offizielle API Deklaration aufgenommen werden steht noch aus.

Methode/Variable	Vorkommen
cb.containerEl.addClass("templater_search");	50
this.app.vault.adapter.basePath +	28
app.commands.executeCommandById(editor:focus");	25
autoCleanSetting.components[0].toggleEl.classList.remove(is-enabled");	24
app.setting.open();	23
const webContents = remote.webContents.fromId(this.webviewEl.getWebContentsId());	22
editor = leaf.view.sourceMode.cmEditor;	21
this.plugin.app.setting.close();	21
app.commands.executeCommandById(item.id);	20
this.app.commands.executeCommandById("app:reload")	18
this.plugin.settings.scriptEngineSettings[scriptName][	17
translator[setting] = service_settings[setting as keyof APIServiceSettings];	16
autoCleanSetting.components[0].toggleEl.classList.add("is-enabled");	16
if (!this.targetView || !this.targetView?._loaded) {	16
notice.noticeEl.innerHTML = `<b>Templater update</b>: $msg`;	15
this.chooser.useSelectedItem(evt);	15
this.picker.controlKeyPressed = (e.ctrlKey || e.metaKey);	15

Tabelle 8: Die 20 von Entwicklern am meisten per Kommentar von der Compiler Überprüfung ausgeschlossenen Codezeilen

Auch innerhalb der Aufrufe von internen Funktionen über das casten `as any` konnten keine entfernten Methoden gefunden werden. Eine Auflistung der 20 meistverwendeten Codezeilen ist in Tabelle 9.

Methode/Variable	Vorkommen
	131
	27
	26
	22
	22
	21
	18
	13
	12
	12
	11
	11
	9
	9
	8
	8
	8

Tabelle 9: Die 20 am meisten `as any` ge-casteten Funktionsaufrufe

Für die spätere regelmäßige Auswertung dieser Werte für die Erkennung von inkompatiblen Erweiterungen ist der manuelle Aufwand nicht mit dem Nutzen des Endergebnisses in Verhältnis zu setzen. Für die Verbesserung der offiziellen API sind diese Daten aber durchaus sehr gut verwendbar.

Innerhalb der per Monkey-Patching überschriebenen Methoden wurden keine identifiziert die in der nächsten Zeit entfernt werden sollen. Insgesamt nutzen 103 Erweiterungen Monkey-Patching, die einzelnen überschriebenen internen Methoden werden hier nicht aufgelistet.

6.2.3. Smoketest

Insgesamt haben 34 Plugins beim Initialisieren Fehlermeldungen oder Warnungen generiert[63].

Die meisten Fehlermeldungen sind als Programmierfehler oder fehlerhaftes Fehlermanagement zu deuten. 14 Warnungen wurden durch den Aufruf einer nicht definierten Methode oder Variable erzeugt. In einem Fall wurde versucht eine Variable aufzurufen, als wenn sie eine Funktion wäre. In 8 Fällen wurde versucht auf eine nicht existierende Datei zuzugreifen, 3 Plugins konnten nicht auf einen externen Server zugreifen, und 3 weitere Plugins meldeten Fehler, weil diese von anderen Plugins abhängig sind. Ein weiterer Fehler wurde beim Verarbeiten eines JSON Dokuments ausgelöst.

In 3 Plugins hätte ein Aufruf einer bestimmten veraltete API zu einem Absturz geführt, wurde aber von Obsidian abgefangen. Das hier bisher fehlende Plugin ist aufgefallen

[63] Stand: 24.01.2023

aufgrund einer Fehlermeldung der verwendeten Produktanalyse Bibliothek, diese Erweiterung wird hier daher nicht betrachtet. Aufgrund des Verbots von Telemetrie innerhalb von Plugins (siehe Abschnitt 4.1) wurden entsprechende Maßnahmen ergriffen.

Von den hier relevanten 30 Fehlermeldungen sind nur 4 als kritisch einzustufen, aus diesem Grund, und weil diese Analyse nicht vollständig automatisierbar ist, sowie einige Stunden zur manuellen Überprüfung in Anspruch nimmt, eignet sich diese Analyse nicht für einen produktiven Einsatz.

6.3. Erweiterungsrichtlinien anderer Produkte

Hier soll untersucht werden welches Vorgehen andere funktional oder technisch vergleichbare Anwendungen verwenden, um veraltete Erweiterungen auszusortieren.

Für Erweiterungen der Konkurrenzprodukte *Roam Research*[64] und *RemNote*[65] existieren zum Zeitpunkt dieser Arbeit keine Richtlinien. Der Hersteller von Entwicklungsumgebungen und anderen Entwicklungswerkzeugen *JetBrains*[33], der Code Editor *Visual Studio Code*, die IDE *Visual Studio*[34, S. 7], sowie der Browserhersteller *Mozilla*[35] reservieren sich das Recht Erweiterungen wieder zu entfernen, sollten diese die Richtlinien der entsprechenden Plattform verletzen, auf denen diese Erweiterungen für Nutzer angeboten werden.

Nur im *Chrome Web Store* von Google sind defekte Erweiterungen explizit unerwünscht.

> Extensions with broken functionality—such as dead sites or non-functioning features—are not allowed.[36]

Ob und mit welcher Frequenz Erweiterungen reevaluiert werden ist nicht definiert.

Im Hinblick auf veraltete/ungepflegte Erweiterungen konnten keine spezifischen Richtlinien, in keinem der genannten Produkte identifiziert werden.

[64] https://roamresearch.com
[65] https://remnote.com

7. Zusammenfassung und Ausblick

In dieser Arbeit wurden 11 verschiedene Metriken zur Erkennung von ungepflegten oder inkompatiblen Obsidian Plugins implementiert und analysiert. Für den produktiven Einsatz eignen sich 5 dieser Metriken. Besonders gut geeignet für einen produktiven Einsatz sind die folgenden Metriken:

- Repository archiviert (Abschnitt 6.1.3)
- fehlerhafte Releases (Abschnitt 6.1.4)
- README Inhalt (Abschnitt 6.1.5)
- Issue Titel (Abschnitt 6.1.6)

Mit diesen Metriken konnten insgesamt 55 Erweiterungen als eindeutig nicht gepflegt identifiziert werden und werden in Zukunft aus der offiziellen Liste entfernt werden. Weitere Erweiterungen konnten identifiziert werden, die als potenziell nicht gepflegt eingestuft werden können. Weil bei diesen keine kritischen Fehler beobachtet wurden, werden diese nicht entfernt werden. Die restlichen in dieser Arbeit betrachteten Metriken eignen sich nicht aufgrund eines zu großen manuellen Aufwands und einer hohen Anzahl von falsch-positiven Ergebnissen.

Die in dieser Arbeit ebenfalls gesammelten Daten zu von Erweiterungen verwendeten internen Funktionen/Variablen von Obsidian konnten ihren eigentlichen Zweck der Erkennung von inkompatiblen Erweiterungen nicht erfüllen, gesammelt werden diese weiterhin, zur Verbesserung der offiziellen API.

Der Autor dieser Arbeit vermutet das noch viele nicht identifizierte ungepflegte Erweiterungen innerhalb der Obsidian Community Plugins vorhanden sind. Weil diese nicht in den Auswertungen dieser Arbeit auffällig geworden sind, ist davon auszugehen das diese zwar ungepflegt sind, aber noch funktionstüchtig sind. Vollständig zufriedenstellend sind die Ergebnisse dieser Arbeit daher nicht, weil nicht alle ungepflegten Erweiterungen identifiziert werden konnten.

In Zukunft ist es geplant die hier, als produktiv einsetzbar identifizierten Metriken in regelmäßigen Abständen auszuwerten und wo möglich vollständig zu automatisieren. Die hier als nicht produktiv einsetzbar klassifizierten Metriken sollen potenziell weiterentwickelt werden, um eine frühere Erkennung von ungepflegten Erweiterungen zu ermöglichen. Die Entfernung aus der Community-Plugin Liste wird weiterhin manuell durch ein Mitglied des Obsidian Teams geschehen.

A. Literatur

[1] S. Chacon und B. Straub, „Pro Git,“ in *Apress*, 2009.

[2] J. Jiang, D. Lo, J.-H. He, X. Xia, P. S. Kochhar und L. Zhang, „Why and how developers fork what from whom in GitHub,“ *Empir. Softw. Eng.*, Jg. 22, Nr. 1, S. 547–578, 2017. DOI: 10.1007/s10664-016-9436-6.

[3] MDN Contributors. (31. Jan. 2023). Introduction, JavaScript, Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction (besucht am 13. 02. 2023).

[4] MDN Contributors. (31. Jan. 2023). JavaScript data types and data structures, JavaScript, Adresse: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures (besucht am 13. 02. 2023).

[5] MDN Contributors. (13. Jan. 2023). DOM (Document Object Model), Adresse: https://developer.mozilla.org/en-US/docs/Glossary/DOM (besucht am 13. 02. 2023).

[6] Typescript Documentation Contributors. (7. Feb. 2023). TypeScript for the New Programmer, Adresse: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (besucht am 14. 02. 2023).

[7] D. Kundel. (11. Juni 2020). ASTs, What are they and how to use them, Adresse: https://twilio.com/blog/abstract-syntax-trees (besucht am 14. 02. 2023).

[8] N. Raote. (22. Mai 2020). Monkey patching: What is it and should you be using it? Adresse: https://dev.to/napoleon039/monkey-patching-what-is-it-and-should-you-be-using-it-50db (besucht am 15. 02. 2023).

[9] T. Forte, *Building a Second Brain, A Proven Method to Organize Your Digital Life and Unlock Your Creative Potential*. Profile Books Limited, 2022, ISBN: 9781800812215.

[10] A.-L. L. Cunff. (o. D.). Exploring the power of note-making with the co-founder of Obsidian, Adresse: https://nesslabs.com/obsidian-featured-tool (besucht am 28. 01. 2023).

[11] Obsidian Team and contributors. (23. Jan. 2023). Credits, Obsidian Help, Adresse: https://help.obsidian.md/Obsidian/Credits (besucht am 29. 01. 2023).

[12] Obsidian Team and contributors. (26. Okt. 2022). GitHub - obsidianmd/obsidian-releases: Community plugins list, theme list, and releases of Obsidian., Adresse: https://github.com/obsidianmd/obsidian-releases/#how-community-plugins-are-pulled (besucht am 08. 12. 2022).

[13] M. Lehman, „Programs, life cycles, and laws of software evolution,“ *Proceedings of the IEEE*, Jg. 68, Nr. 9, S. 1060–1076, 1980. DOI: 10.1109/PROC.1980.11805.

[14] J. Coelho, M. T. Valente, L. Milen und L. L. Silva, „Is this GitHub Project Maintained? Measuring the Level of Maintenance Activity of Open-Source Projects,“ *CoRR*, Jg. abs/2003.04755, 2020.

[15] J. Coelho und M. T. Valente, „Why modern open source projects fail,“ *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017.

[16] V. Cosentino, J. L. C. Izquierdo und J. Cabot, „Assessing the bus factor of Git repositories,“ *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, S. 499–503, 2015.

[17] I. Samoladas, L. Angelis und I. Stamelos, „Survival analysis on the duration of open source projects,“ *Inf. Softw. Technol.*, Jg. 52, S. 902–922, 2010.

[18] C. Miller, D. G. Widder, C. Kästner und B. Vasilescu, *Why Do People Give Up FLOSSing? A Study of Contributor Disengagement in Open Source*, 2019. DOI: 10.1007/978-3-030-20883-7_11.

[19] J. Khondhu, A. Capiluppi und K.-J. Stol, „Is It All Lost? A Study of Inactive Open Source Projects," in *International Conference on Open Source Systems*, 2013.

[20] J. Coelho, M. T. Valente, L. L. Silva und E. Shihab, „Identifying unmaintained projects in github," *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018.

[21] A. A. Sawant, G. Huang, G. Vilén, S. Stojkovski und A. Bacchelli, „Why are Features Deprecated? An Investigation Into the Motivation Behind Deprecation," in *IEEE International Conference on Software Maintenance and Evolution*, 2018.

[22] J. Businge, A. Serebrenik und M. van den Brand, „Survival of Eclipse third-party plug-ins," *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, S. 368–377, 2012.

[23] G. Rauch. (8. Juni 2021). Was ist Telemetrie? Adresse: https://www.dev-insider.de/was-ist-telemetrie-a-1026971/ (besucht am 28. 01. 2023).

[24] H. Erz. (2021). Telemetry, Data Privacy, and Zettlr, Adresse: https://www.hendrik-erz.de/post/telemetry-data-privacy-and-zettlr (besucht am 18. 02. 2023).

[25] Y. Murakami, Y. Takatsuka und M. Tsunoda, „Analyzing Users' Attitude toward Software Data Collection," *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, S. 507–512, 2019.

[26] V. Lenarduzzi, A. C. Stan, D. Taibi, G. Venters und M. Windegger, „Prioritizing Corrective Maintenance Activities for Android Applications: An Industrial Case Study on Android Crash Reports," in *International Conference on Software Quality. Process Automation in Software Development*, 2018.

[27] S. Jansen, „Measuring the health of open source software ecosystems: Beyond the scope of project health," *Inf. Softw. Technol.*, Jg. 56, S. 1508–1519, 2014.

[28] J. Sheoran, K. Blincoe, E. Kalliamvakou, D. Damian und J. Ell, „Understanding "Watchers" on GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, Ser. MSR 2014, Hyderabad, India: Association for Computing Machinery, 2014, S. 336–339, ISBN: 9781450328630. DOI: 10.1145/2597073.2597114.

[29] GitHub Team and contributors. (o. D.). GitHub event types, GitHub Docs, Adresse: https://docs.github.com/en/developers/webhooks-and-events/events/github-event-types (besucht am 18. 01. 2023).

[30] GitHub Team. (o. D.). Repositorys archivieren, GitHub Dokumentation, Adresse: https://docs.github.com/de/repositories/archiving-a-github-repository/archiving-repositories (besucht am 11. 01. 2023).

[31] GitHub Team and contributors. (o. D.). Working with Forks, GitHub Docs, Adresse: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/about-forks (besucht am 24. 01. 2023).

[32] J. T. McKellar, „Monitoring the health of an open source project : a case study," 2010.

[33] JetBrains s.r.o. (28. Nov. 2022). JetBrains Plugin Marketplace Approval Guidelines, Adresse: https://www.jetbrains.com/legal/docs/plugins_site/approval-guidelines/ (besucht am 08. 02. 2023).

[34] Microsoft Corporation. (2019). Microsoft Visual Studio Marketplace Publisher Agreement, Adresse: https://cdn.vsassets.io/v/M146_20190123.39/_content/Visual-Studio-Marketplace-Publisher-Agreement.pdf (besucht am 08. 02. 2022).

[35] mozilla.org Contributors. (2022). Add-on Policies, Adresse: https://extensionworkshop.com/documentation/publish/add-on-policies/ (besucht am 08. 02. 2023).

[36] Google LLC. (15. Dez. 2022). Minimum Functionalility, Chrome Developers, Adresse: https://developer.chrome.com/docs/webstore/program-policies/minimum-functionality/ (besucht am 08. 02. 2023).

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis