Bachelorarbeit

ANALYSE DER VERWENDUNG UND VERBREITUNG VON GIT SUBMODULEN IN C/C++ PROJEKTEN

JONAS WAGNER

4. Januar 2024

Betreuer:
Sebastian Böhm Lehrstuhl für Software Engineering

Prüfer:
Prof. Dr. Sven Apel Lehrstuhl für Software Engineering
Prof. Dr. Jens Dittrich Big Data Analytics Group

Lehrstuhl für Software Engineering
Saarland Informatics Campus
Universität des Saarlandes

UNIVERSITÄT DES SAARLANDES

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 04.01.24
(Datum/Date)


(Unterschrift/Signature)

KURZFASSUNG

Die Nutzung von Drittanbieterbibliotheken ist ein gängiges Prinzip in der Softwareentwicklung, das Zeit spart und die Code-Wiederverwendbarkeit fördert. Eine Möglichkeit, um diese Bibliotheken zu verwalten und zu organisieren, ist die Verwendung von Git Submodulen. Submodule integrieren externe Git Repositorys in ein Hauptrepository, schaffen Referenzen zum externen Code und erleichtern dessen Aktualisierung sowie den Zugriff auf verschiedene Versionen, während das Hauptrepository separat verwaltet wird.

Das Hauptziel dieser Arbeit ist es, die Verbreitung und Verwendung von Submodulen in C/C++ Projekten umfassend zu analysieren. In diesen Projekten fehlt ein spezifischer Paketmanager, weshalb Git Submodule häufig als Ersatz für diesen dienen. Dabei zielt die Untersuchung darauf ab, quantitative Daten zur Häufigkeit der Submodulnutzung in C/C++ Projekten zu sammeln. Zusätzlich beabsichtigt sie, bestimmte Projekte im Detail zu untersuchen, um praxisnahe Erkenntnisse über die konkrete Anwendung von Submodulen zu gewinnen.

Ein weiterer Schwerpunkt liegt auf der Untersuchung der Häufigkeit der Git Submodulnutzung und deren Entwicklung im Zeitverlauf. Besonderes Augenmerk dabei ist die Häufigkeit der Aktualisierung der Submodule. Die Klassifikation der Rollen einzelner Submodule innerhalb der Projekte spielt dabei ebenfalls eine bedeutende Rolle.

Somit trägt diese Forschung dazu bei, das Verständnis über den Einsatz von Git Submodulen in C/C++ Projekten in der Softwareentwicklung zu erweitern.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

1 EINFÜHRUNG

Die Nutzung von Drittanbieterbibliotheken in der Softwareentwicklung ist heutzutage eine gängige Praxis, da sie eine effiziente Möglichkeit bietet Code wiederverwendbar zu machen und Zeit zu sparen, da nicht jedes Mal von Grund auf bestimmte Funktionen und Algorithmen neu entwickelt werden müssen. Stattdessen können Entwickler auf bereits vorhandene Lösungen zurückgreifen und sich auf die spezifischen Anforderungen des Projekts konzentrieren. Drittanbieterbibliotheken enthalten vorgefertigten Code, der bereits entwickelt, getestet und optimiert wurde. Dies ermöglicht eine effiziente Wiederverwendung von Code und reduziert den Entwicklungsaufwand. Beliebte Drittanbieterbibliotheken haben oft eine aktive Entwickler- und Nutzergemeinschaft, die Fragen beantwortet, Unterstützung bietet und regelmäßig Aktualisierungen veröffentlicht. Dies ermöglicht einen Austausch mit anderen Entwicklern und Nutzern der Bibliotheken, sowie den Zugang zu bewährten Praktiken und die Lösung von Problemen auf effiziente Weise.

Eine Möglichkeit zur Verwaltung von Drittanbieterbibliotheken bietet das Versionsverwaltungssystem Git, das es erlaubt ein Repository in einzelne Git Submodule zu unterteilen. Dies geschieht durch die Isolierung der gewünschten Teile in neue, leere Repositorys, die anschließend als Submodule in das Hauptrepository eingefügt werden können. Dadurch können Entwickler auf die Funktionen der Bibliothek zugreifen, ohne sie manuell herunterzuladen und einzubinden. Submodule finden gerade in C/C++ Projekten häufig Anwendung, da es dort keinen etablierten Paketmanager gibt, um Drittanbieterbibliotheken zu verwalten. Die Analyse der Verwendung und Verbreitung der Submodule wird durch die häufige Anwendung attraktiver, da es in C/C++ Projekten zu einer größeren Datenmenge über Submodule in diesen Sprachen im Vergleich zu anderen Programmiersprachen führt.

In unserer Arbeit untersuchen wir Submodule, da sie in der Softwareentwicklung mit C/C++ Projekten eine effiziente Verwaltung von Abhängigkeiten und externen Bibliotheken ermöglichen. Ihre Verwendung erlaubt die Strukturierung komplexer Projekte in handhabbare Module, was die Wartbarkeit, Skalierbarkeit und Zusammenarbeit innerhalb eines Entwicklerteams verbessert. Diese Untersuchung ermöglicht die Identifizierung genereller Rollen der Submodule und das Herausarbeiten potenzieller Herausforderungen bei der Integration und Aktualisierung dieser Module.

1.1 ZIEL DIESER ARBEIT

Das Hauptziel dieser Arbeit besteht darin, die Verbreitung und Verwendung von Submodulen in C/C++ Projekten zu erforschen und wertvolle Erkenntnisse über deren Einsatz zu gewinnen. Durch die Untersuchung quantitativer Daten, wie beispielsweise der Anzahl der verwendeten Submodule pro Repository, der Anzahl der Commits, welche die Aktualisierung der Submodule repräsentiert und deren Auswirkungen der Submodulverwendung auf die Gesamtanzahl der Commits, wollen wir ein umfassendes Verständnis für die Praxis der Submodulnutzung in C/C++ Projekten auf GitHub gewinnen. Zusätzlich untersuchen wir

ausgewählte Projekte detaillierter, um zu verstehen, wie Submodule in der Praxis eingesetzt werden. Dabei betrachten wir verschiedene Metriken wie die Anzahl der verwendeten Submodule pro Projekt und die Anzahl der Commits, die die Submodule betreffen.

1.2 ÜBERSICHT

Im Folgenden stellen wir die Struktur unserer Arbeit vor. Unser Arbeitsaufbau beginnt mit der Bereitstellung des grundlegenden Hintergrundwissens in Kapitel 2, das für das Verständnis unserer Arbeit von Bedeutung ist. Im Anschluss widmen wir uns den unterstützenden Programmen in Kapitel 3 und erklären, wie wir aus ihnen die benötigten Metadaten gewinnen. In Kapitel 4 folgt ein wesentlicher Abschnitt, in dem wir die Methodologie vorstelle. Hierbei fassen wir unsere Forschungsfragen zusammen und unterstreichen ihre Relevanz. Gleichzeitig skizzieren wir den Weg zur Umsetzung dieser Fragen und wie sie operationalisiert werden. Die Evaluation sowie die ausführliche Diskussion der Ergebnisse sind in Kapitel 5 zentrale Bestandteile. In diesem Zusammenhang werden wir auch auf die Validitätsaspekte eingehen, um die Zuverlässigkeit unserer Ergebnisse sicherzustellen. Es erfolgt in Kapitel 6 ein kritischer Vergleich unserer Arbeit mit verwandten Arbeiten, um herauszustellen, wie sie sich in den wissenschaftlichen Kontext einordnet. Kapitel 7 rundet unsere Arbeit durch Schlussbemerkungen ab, die eine kompakte Zusammenfassung bieten.

2 HINTERGRUND

In dieser Sektion erläutern wir wichtige Begriffe und Programme, die wir im weiteren Verlauf der Arbeit verwenden. Durch die Bereitstellung dieses Hintergrundwissens schaffen wir eine solide Grundlage, um den Kontext und die Bedeutung der in der Arbeit behandelten Konzepte zu verstehen.

GIT Git als verteiltes Versionsverwaltungssystem erlaubt jedem Benutzer, eine vollständige Kopie des *Repositorys* zu besitzen. Dabei können Änderungen unabhängig vom zentralen Server lokal vorgenommen werden. Wenn ein Benutzer bereit ist, seine Änderungen mit dem zentralen Server zu synchronisieren, erfolgt dies üblicherweise über einen Prozess, der als *Push und Pull* bekannt ist. `Push` bedeutet, dass der Benutzer seine lokalen Änderungen zum zentralen Repository hochlädt, während `Pull` verwendet wird, um die Aktualisierungen vom zentralen Server zu beziehen und in das lokale Repository zu integrieren. Diese bidirektionale Kommunikation ermöglicht es, Änderungen zwischen den lokalen Kopien und dem zentralen Repository effizient zu verwalten und zu synchronisieren.

REPOSITORYS Repositorys bestehen aus einer Sammlung von Dateien und Ordnern, die den Quellcode eines Projekts enthalten. Es dient als zentrale Ablage, in dem digitale Daten, Dokumente, Objekte und Programme mit ihren Metadaten verwaltet werden. Repositorys ermöglichen es Entwicklern, verschiedene Versionen des Codes zu verfolgen, zwischen ihnen zu wechseln, durch *Revisionen* Änderungen nachzuvollziehen und die Entwicklungshistorie zu verstehen. Softwareprojekte werden mithilfe von GIT Repositorys verwaltet, indem der gesamte Code, Ressourcen und Konfigurationen in einem zentralen Repository gespeichert werden. Entwickler können Kopien dieses Repositorys erstellen, um lokal zu arbeiten. Änderungen werden in lokalen Kopien vorgenommen, bevor sie in das zentrale Repository zurückgespielt werden.

REVISIONEN Revisionen beziehen sich auf einen spezifischen Zustand oder einen bestimmten Stand des Quellcodes in einem Repository zu einem bestimmten Zeitpunkt. Jeder *Commit* in GIT erzeugt eine neue Revision des Quellcodes, die den aktuellen Zustand des Projekts und die darin vorgenommenen Änderungen enthält. Repositorys nutzen Commits, um den Zustand des Projekts zu einem bestimmten Zeitpunkt festzuhalten.

COMMITS In GIT repräsentieren Commits einen bestimmten Zustand des Projekts zu einem bestimmten Zeitpunkt. Jeder Commit erzeugt eine neue Revision mit einer eindeutigen Nummer im Verlauf des Projekts. Somit steht jede Revision in direktem Zusammenhang mit einem Commit. Ein Commit enthält typischerweise Informationen über den aktuellen Projektzustand, den Autor der letzten Änderungen, Datum und Uhrzeit des erstellten Commits und eine Commit-Nachricht, die die durchgeführte Änderung beschreibt. Während der Commit eine Momentaufnahme des Projektzustandes zu einem bestimmten Zeitpunkt und

mit den entsprechenden Informationen darstellt, ermöglicht die Revision die Identifizierung und Verfolgung dieses spezifischen Zustands innerhalb des Versionsverlaufs des Projekts.

GIT SUBMODULE Git Submodule bieten die Möglichkeit, während der Arbeit an einem Projekt das Repository eines anderen Projekts mit einer fest definierten Revision in das eigene Repository einzubinden. Submodule werden einem Repository über den Befehl `git submodule add` über die Befehlszeile eines Terminals mit der URL des externen Repositorys und einem Pfad im Hauptrepository hinzugefügt. Dabei kann es sich beispielsweise um eine Bibliothek handeln, die von Dritten entwickelt wurde oder die seperat entwickelt wird und in mehreren übergeordneten Projekten verwendet werden soll. Mit Submodulen lässt sich ein Repository als Unterverzeichnis eines anderen Repositorys führen. Damit bleiben beide Projekte getrennt, dennoch kann das eine vom anderen aus im selben Dateibaum genutzt werden. Die Rolle von Git Submodulen ist jedoch vielseitiger, denn sie können durch die klare Trennung der Repositorys auch als Auslagerung von Codeabschnitten genutzt werden, statt nur eine externe Bibliothek einzubinden.[1] Um Submodule zu aktualisieren, nutzt man `git submodule update -remote`, um alle Submodule auf die neueste in ihren externen Repositorys verfügbare Version zu aktualisieren. Wenn man nur ein bestimmtes Submodul aktualisieren möchte, navigiert man in dessen Verzeichnis und führt `git pull` mit dem entsprechenden Zweig aus, um das Submodul auf diese Version zu aktualisieren.

.GITMODULES Die Konfigurationsdatei `.gitmodules` in Git speichert Informationen über Submodule in einem Repository. In einem Beispiel aus dem Projekt `grpc` in Abbildung 2.1 werden Details wie der Pfad im übergeordneten Repository und die URL des entsprechenden Submoduls festgehalten. Diese Datei ermöglicht eine zentrale Verwaltung der Submodule in einem Projekt und aktualisiert sich entsprechend. Sie zeigt somit an, wie viele Submodule in einem Projekt verwendet werden. Wenn die `.gitmodules` Datei leer oder nicht vorhanden ist, bedeutet dies, dass das Projekt keine Submodule verwendet.

GITHUB GitHub[2] bietet Entwicklern die Möglichkeit, mit Hilfe von `Forks`, `Branches` und `Pull Requests` zusammenzuarbeiten. Ein Entwickler erstellt einen `Fork`, eine persönliche Kopie des Hauptrepositorys, um Änderungen vorzunehmen. Nachdem Änderungen in einem separaten `Branch` im eigenen `Fork` vorgenommen wurden, wird ein `Pull Request` erstellt, um diese Änderungen dem ursprünglichen Repository vorzuschlagen. Der Projektbetreuer kann dann den `Pull Request` überprüfen, diskutieren und die vorgeschlagenen Änderungen entweder akzeptieren, um sie in das Hauptrepository zu integrieren, oder ablehnen. Dadurch wird eine transparente und strukturierte Methode für die Zusammenarbeit und die Übernahme von Änderungen in einem Projekt geschaffen. Neben dem Hosting von Code bietet GitHub auch die Möglichkeit zur kollaborativen Code-Überprüfung und eine integrierte Problemverfolgung. Darüber hinaus verfügt GitHub über soziale Funktionen. Benutzer können Projekte *beobachten* und anderen Benutzern *folgen*, um Informationen über diese Projekte und Benutzer zu erhalten. Benutzer haben auch Profile, in denen sie persönliche Informationen hinterlegen können und die ihre kürzliche Aktivität auf der Website anzeigen [6]. Man kann auf GitHub Projekte besonders kennzeichnen, indem

1 https://git-scm.com/book/de/v2/Git-Tools-Submodule
2 https://github.com/

.gitmodules

```
[submodule "third_party/abseil-cpp"]
        path = third_party/abseil-cpp
        url = https://github.com/abseil/abseil-cpp.git
[submodule "third_party/benchmark"]
        path = third_party/benchmark
        url = https://github.com/google/benchmark
[submodule "third_party/bloaty"]
        path = third_party/bloaty
        url = https://github.com/google/bloaty.git
[submodule "third_party/boringssl-with-bazel"]
        path = third_party/boringssl-with-bazel
        url = https://github.com/google/boringssl.git
[submodule "third_party/cares/cares"]
        path = third_party/cares/cares
        url = https://github.com/c-ares/c-ares.git
[submodule "third_party/envoy-api"]
        path = third_party/envoy-api
        url = https://github.com/envoyproxy/data-plane-api.git
[submodule "third_party/googleapis"]
        path = third_party/googleapis
        url = https://github.com/googleapis/googleapis.git
[submodule "third_party/googletest"]
        path = third_party/googletest
        url = https://github.com/google/googletest.git
[submodule "third_party/opencensus-proto"]
        path = third_party/opencensus-proto
        url = https://github.com/census-instrumentation/opencensus-proto.git
[submodule "third_party/opentelemetry"]
        path = third_party/opentelemetry
        url = https://github.com/open-telemetry/opentelemetry-proto.git
[submodule "third_party/protobuf"]
        path = third_party/protobuf
        url = https://github.com/protocolbuffers/protobuf.git
```

Abbildung 2.1: Die Abbildung zeigt anhand des Projekts grpc einen Ausschnitt der zugehörigen .gitmodules Datei. In dieser sind die Submodule mit ihrem Pfad und der URL angegeben.

man ihnen einen Stern gibt. Die Anzahl der Sterne spielt eine entscheidende Rolle für die Popularität eines Projekts und wird oft, wie auch in unserer Arbeit, als Auswahlkriterium herangezogen.

GITHUB REST-API Um in GitHub auf Daten und Funktionen zuzugreifen, verwenden wir die GitHub REST-API. Sie bietet eine umfassende Programmierschnittstelle, die es Entwicklern ermöglicht, mit den Ressourcen und Daten auf der GitHub-Plattform zu interagieren. Durch die Nutzung dieser API können Entwickler verschiedene Aufgaben automatisieren und benutzerdefinierte Abläufe erstellen. Die API ermöglicht den Zugriff auf eine Vielzahl von Daten, darunter Repositorys, Abonnenten, Mitwirkende, Commits, Submodule und mehr. Dadurch können Entwickler ihre Arbeitsabläufe optimieren, indem sie Daten abrufen, Anpassungen vornehmen und Aktionen durchführen, ohne unbedingt die Benutzeroberfläche von GitHub nutzen zu müssen.[3] In unseren Untersuchungen setzen wir die API ein, um Projekte anhand ihrer Sterne zu ordnen und relevante Metadaten zu extrahieren. Dies umfasst insbesondere Informationen wie die Namen und Anzahl der Submodule sowie die Anzahl der Commits mit den zugehörigen Erstellungsdaten.

3 https://docs.github.com/en/rest?apiVersion=2022-11-28

3 UNTERSTÜTZENDE PROGRAMME UND DATENERHEBUNG

In diesem Kapitel beschreiben wir die Datensätze, die wir in unserer Studie verwenden, im Detail. Zunächst definieren wir die Kriterien, nach denen wir Softwareprojekte auswählen. Im Anschluss beschreiben wir, welche Datensätze wir aus diesen Projekten bilden und welche Eigenschaften diese Datensätze haben. Dabei gehen wir auch auf die technischen Details der Datenerhebung und deren Implementierung ein.

Tabelle 3.1: Verwendete Datensätze

Name des Datensatzes	Anzahl Projekte	Anzahl aller Submodule
D10k	10.000	10.109
DS1.9k	1.938	10.109
DS100	100	511

Unser Ziel besteht darin, eine repräsentative Auswahl von C/C++ Projekten zu treffen, um eine umfassende Analyse ihrer Submodule durchzuführen. Eine repräsentative Auswahl bedeutet für uns, dass sie sowohl Projekte mit umfangreichem als auch mit geringem Code umfasst. Dadurch sollen sowohl größere als auch kleinere Projekte berücksichtigt werden, um eine angemessene Vielfalt abzubilden. Wir halten eine Anzahl von 10.000 Projekten für geeignet, um dieses Ziel zu erreichen. Dabei ist es wichtig, nicht nur Projekte mit, sondern auch Projekte ohne Submodule einzuschließen. Außerdem entscheiden wir uns dafür, insbesondere die populärsten Projekte zu untersuchen.

Für die Selektion der Projekte haben wir uns für GitHub entschieden, da diese Plattform eine umfangreiche Sammlung von Projekten bietet, die nach Programmiersprachen sortiert werden können. Die GitHub REST-API bietet uns zudem einen einfachen Zugriff auf die Projektinformationen. Unsere Suche wird mit den Parametern `query: <language:c++ language:c>, sort: <stars>, order: <desc>` durchgeführt. Dabei beschränken wir uns auf die Programmiersprachen C/C++ und sortieren die Projekte absteigend nach der Anzahl der Sterne, die die Popularität eines Projekts repräsentieren. Über die GitHub REST-API suchen wir gezielt nach Projekten, die die Datei `.gitmodules` enthalten, um festzustellen, welche C/C++ Projekte tatsächlich Submodule verwenden und welche nicht.

Unsere verwendeten Datensätze sind in Tabelle 3.1 aufgeführt. Wir können den prozentualen Anteil der Projekte mit Submodulen (**DS1.9k**) im Gesamtdatensatz (**D10k**) ermitteln. Der Datensatz **DS1.9k** entspricht dem Gesamtdatensatz **D10k** und umfasst ausschließlich die 1.938 Projekte, die Submodule enthalten. Insgesamt beinhaltet der Gesamtdatensatz 10.000 Projekte, in denen wiederum 10.109 Submodule enthalten sind.

Das Hinzufügen eines Submoduls erfolgt mit dem Befehl `git submodule add <URL des Submoduls>`. Dadurch wird automatisch eine `.gitmodules`-Datei im GitHub-Repository angelegt, in der alle Submodule gespeichert sind. Durch die Nutzung eines regulären

Ausdrucks[1] können wir dann für jedes Projekt die Anzahl der Submodule aus dieser Datei entnehmen, indem wir die Zeichenfolge `submodule` zählen. Um dies zu veranschaulichen, verweisen wir beispielhaft auf die `.gitmodules`-Datei des Projekts `grpc/grpc` in Abbildung 2.1.

Um eine präzisere Analyse der Submodule durchzuführen ist es notwendig die Projekte, die tatsächlich Submodule enthalten, auf lokaler Ebene zu klonen. Da das Klonen aller 1.938 Projekte zu viel Speicherplatz erfordert, entscheiden wir uns dafür gemäß einer Gleichverteilung eine zufällige Auswahl von 100 Projekten (**DS100**) aus dem **DS1.9k** zu treffen. Mit diesem kleineren Set können wir nun genauer die Aktualität der Git-Submodule untersuchen. Hierfür verwenden wir die `git log`-Funktion, die uns Zugriff auf die Commits gewährt. Diese Commits dokumentieren die Änderungen an den Submodulen und deren Auswirkungen auf das Hauptrepository. Durch Überprüfen des letzten Commits können wir feststellen, ob das Submodul auf dem neuesten Stand ist. Dies ermöglicht eine Überwachung der Submodul-Aktualität und der durchgeführten Änderungen.

Die Kategorisierung der Rolle der einzelnen Submodule innerhalb eines Hauptrepositorys erfolgt detaillierter. Die Repository-Übersicht auf GitHub bietet eine Fülle von Informationen, die Entwicklern und Benutzern dabei helfen, das Projekt besser zu verstehen und damit zu interagieren. Eine zentrale Rolle spielt dabei die README-Datei, die eine umfassende Einführung und allgemeine Informationen über das Projekt bereitstellt, einschließlich seiner Zwecke, Funktionen und Verwendungsmöglichkeiten. Diese Informationen sind entscheidend, um die Bedeutung und Funktion der einzelnen Submodule im Kontext des Hauptrepositorys zu erfassen und zu klassifizieren.

1 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet

4 METHODOLOGIE

In dieser Sektion stellen wir unsere Forschungsfragen vor und erörtern die angewandte Methodik, um die gestellten Forschungsfragen zu beantworten. Dabei erläutern wir die Operationalisierung unserer Experimente und den geplanten Ablauf. Durch eine methodische Vorgehensweise streben wir an, fundierte Antworten auf die Forschungsfragen zu erhalten und wertvolle Erkenntnisse zu gewinnen.

4.1 FORSCHUNGSFRAGEN

In diesem Abschnitt stellen wir die Forschungsfragen vor, deren Beantwortung ein tieferes Verständnis für die Verwendung von Git Submodulen in C/C++ Projekten auf GitHub zum Ziel hat. Jede Forschungsfrage zielt darauf ab, die Verwendung und Verbreitung von Submodulen zu beleuchten und Erkenntnisse zu liefern, die zur Lösung des Forschungsproblems beitragen. Die Forschungsfragen sind in zwei Teile gegliedert. Der erste Teil befasst sich mit der Verbreitung, der zweite Teil mit der Verwendung von Git Submodulen.

FF1 *Wie weit verbreitet sind Git Submodule in C/C++ Projekten auf GitHub?*

Die erste Forschungsfrage dieser Arbeit zielt darauf ab, die generelle Verbreitung von Git Submodulen in C/C++ Projekten auf GitHub zu untersuchen. Hierzu wird die Forschungsfrage in zwei Teilfragen unterteilt. Zunächst wird die allgemeine Häufigkeit der Submodule betrachtet, um festzustellen wie weit verbreitet sie in den untersuchten C/C++ Projekten sind. Anschließend wird die Frage nach der zeitlichen Veränderung der Submodul-Nutzung behandelt, um zu ermitteln, ob und wie sich die Verwendung von Submodulen im Laufe der Zeit entwickelt hat.

FF1.1 *Wie häufig verwenden Projekte Git Submodule?*

Um verlässliche Aussagen über die Verbreitung von Git Submodulen treffen zu können, ist es von entscheidender Bedeutung, herauszufinden, wie viele C/C++ Projekte tatsächlich auf Submodule zurückgreifen. Die Beantwortung dieser Forschungsfrage verschafft uns Überblick darüber, wie weit verbreitet die Verwendung von Submodulen in dieser Domäne in der Praxis ist. Zusätzlich können wir die gewonnenen Daten verwenden, um eine Auswahl an Projekten zur genaueren Untersuchung für die zweite Forschungsfrage zu treffen.

FF1.2 *Wie unterscheidet sich die Häufigkeit der Verwendung von Git Submodulen in älteren und neueren Projekten?*

Bei der Analyse der C/C++ Projekte ist es von Bedeutung zu untersuchen, ob ein deutlicher Unterschied zwischen älteren und neueren Projekten besteht, da es Git und damit auch die Funktion der Submodule erst seit 2007[1] gibt. Die gewonnenen Erkenntnisse ermöglichen

1 https://lore.kernel.org/git/7vodglr32i.fsf@gitster.siamese.dyndns.org/

eine Einsicht in die zeitliche Entwicklung der Submodul-Nutzung. Durch die Analyse verschiedener Zeiträume wird deutlich, wie sich die Verwendung von Submodulen in C/C++ Projekten im Laufe der Zeit verändert hat.

FF2 *Wie verwenden C/C++ Projekte Git Submodule?*

Die zweite Forschungsfrage dieser Arbeit untersucht, wie Git Submodule in C/C++ Projekten auf GitHub verwendet werden und welche spezifischen Funktionen und Rollen sie innerhalb dieser Projekte einnehmen. Sie zielt darauf ab, die Art und Weise zu beleuchten, wie Entwickler Submodule einsetzen, um damit verbundene spezifische Aufgaben oder Funktionen zu erfüllen. Dieses Verständnis ist von Bedeutung, um gängige Muster in der Verwendung von Submodulen zu identifizieren. Zudem bieten sie Einblicke in die Struktur und Organisation solcher Projekte.

FF2.1 *Wie häufig werden Submodule aktualisiert?*

Die regelmäßige Aktualisierung von Submodulen bietet folgende Vorteile: Erstens ermöglicht sie den Zugriff auf neue Funktionen und Verbesserungen externer Bibliotheken. Zweitens werden Fehlerbehebungen und Sicherheitsaktualisierungen durchgeführt, um die Stabilität und Sicherheit des Projekts zu gewährleisten. Drittens kann die Aktualisierung der Submodule bereits Aufschluss über die Rolle des Submoduls geben, da beispielsweise eine ausgelagerte Funktion, die konstant bleibt, nicht häufig aktualisiert werden muss. Durch die Durchführung dieser Untersuchung streben wir an, ein umfassenderes Verständnis für die Bedeutung und den Nutzen der Aktualisierung von Submodulen zu erlangen.

FF2.2 *Welche Rolle spielen Git Submodule innerhalb eines Projekts?*

Die Hauptaufgabe von Git Submodulen ist von besonderem Interesse. Werden Teile des Hauptprojekts ausgelagert oder dienen sie hauptsächlich dazu, zusätzliche Funktionalität einzubinden? Die Ergebnisse dieser Untersuchung ermöglichen Rückschlüsse darauf, ob die Verwendung von Submodulen überwiegend in eine bestimmte Richtung geht oder ob eine breitere Vielfalt an Einsatzmöglichkeiten besteht.

4.2 OPERATIONALISIERUNG

Im kommenden Abschnitt beschreiben wir die Experimente, mit denen wir die einzelnen Forschungsfragen beantworten. Die Experimente folgen der Aufteilung der Forschungsfragen: Zunächst untersuchen wir die allgemeine Verbreitung, anschließend die konkrete Verwendung von Submodulen. Durch diese strukturierte Vorgehensweise erlangen wir ein umfassendes Verständnis über die Rolle und den Einsatz von Submodulen in C/C++ Projekten.

FF1.1: Wir nutzen für diese Forschungsfrage den gesamten Datensatz **D10k**, da in diesem sowohl Projekte mit als auch Projekte ohne Submodule enthalten sind. Aus diesem filtern wir die Projekte heraus, die Submodule enthalten, was uns zu dem Datensatz **DS1.9k** führt. Durch die Betrachtung des Anteils der Projekte mit Submodulen zu der Gesamtanzahl an Projekten können wir dadurch durchschnittlich ermitteln, wie viele C/C++ Projekte Submodule nutzen. Wir erfassen auch die Gesamtanzahl der Submodule pro Projekt aus dem

Datensatz **DS1.9k**. Dabei identifizieren wir Ausreißerprojekte anhand einer signifikanten Abweichung der Submodulanzahl im Vergleich zu den vorhergehenden Werten.

FF1.2: Durch die GitHub REST-API können wir gezielt die Metrik `created at` aus allen Submodulen entnehmen. Diese Entstehungsdaten extrahieren wir aus dem Datensatz **DS1.9k** und ordnen sie chronologisch. Anschließend erstellen wir eine zeitliche Achse, die mit dem frühesten Entstehungsjahr beginnt. Danach setzen wir die Anzahl der Projekte, die in einem bestimmten Jahr mindestens ein Submodul integriert haben, ins Verhältnis zur Gesamtanzahl der Projekte in diesem Jahr. Durch die Berechnung von Durchschnittswerten über verschiedene Zeiträume hinweg identifizieren wir mögliche Trends oder erkennen, ob keine besondere Entwicklung besteht. Diese Analyse der zeitlichen Durchschnittswerte erlaubt eine Einschätzung der Entwicklung in Bezug auf die Anzahl der Submodule.

FF2.1: Für die Analyse der Submodul-Aktualisierungen verwenden wir unseren kleineren Datensatz **DS100** mit 100 Projekten, das insgesamt 511 Submodule umfasst, da das Klonen aller 1.938 Projekte zu viel Speicherplatz erfordert. Dabei zählen wir für jedes Projekt, wie oft jedes Submodul pro Jahr aktualisiert wurde. Über die `git log` Funktion können wir die Commits des Submoduls im Hauptrepository verfolgen und die Häufigkeit dieser Aktualisierungen ermitteln. Durch die chronologische Anordnung der Commits können wir Rückschlüsse auf die zeitliche Entwicklung der Aktualisierungen ziehen. Abbildung 4.1 zeigt ein Beispiel der `git log` Funktion. Jeder Commit besteht aus einem eindeutigen *Hashwert*, dem Autor des Commits, dem Datum einschließlich der genauen Uhrzeit und einer Beschreibung. Mit einem regulären Ausdruck, der das Datumsformat analysiert, können wir zählen, wie oft das Submodul in jedem Jahr aktualisiert wurde.

```
commit c81bdf7a835b758a7b9f28278916091e9e019421
Author: AJ Heller <hork@google.com>
Date:   Fri Jul 14 12:04:40 2023 -0700

    [Protobuf] Upgrade third_party/protobuf to 23.4 (#33695)

    This was done manually due to a problem with
    `tools/distrib/python/make_grpcio_tools.py`. ~I fixed it in this PR
    (depends on cl/547979185), so there is a fair chance this upgrade will
    work normally for the next release.~ The fix may be problematic for
    upgrading protobuf on older release branches, so the improvement will be
    worked on separately. CC @jtattermusch

    This also updates the UPB dep to the latest commit on the 23.x branch.

commit a3dae00f897a3a06ee17e042f43aa82befc525d0
Author: Esun Kim <veblush@google.com>
Date:   Thu May 18 14:46:59 2023 -0700

    [Deps] Upgrade Protobuf to v23.1 (#33164)

    Along with the required Abseil & upb upgrade.

commit dc9513314071eb87fcf168a1015c14bc250dcaa0
Author: Esun Kim <veblush@google.com>
Date:   Tue May 9 12:24:21 2023 -0700

    [Deps] Upgrade Protobuf v23 (#32914)

    Upgrading Protobuf and Upb to v23.0
```

Abbildung 4.1: Die Abbildung zeigt anhand des Projekts grpc mit dem Submodul protobuf einen Ausschnitt der git log Funktion. Die Commits enthalten einen eindeutigen Hashwert, einen Autor, das Datum des Commits und eine Beschreibung.

FF2.2: Um die genaue Rolle eines Submoduls zu identifizieren reicht es nicht aus die Daten der GitHub REST-API in beliebiger Form auszuwerten. Für eine detaillierte Untersuchung nutzen wir wieder den Datensatz **DS100**. Um die Rolle eines Submoduls zu bestimmen, untersuchen wir die Repository-Übersicht und die README-Datei. Diese methodische Vorgehensweise ermöglicht eine präzisere Beurteilung der Bedeutung und Funktionsweise der Submodule innerhalb der ausgewählten Projekte und trägt zu einem tieferen Verständnis ihrer Verwendung bei. Durch das *Card-Sorting* ordnen wir Submodule mit ähnlichen Funktionalitäten zusammen, um sie später in Kategorien zu gruppieren. Anhand des Testprojekts `grpc` konnten wir folgende fünf Kategorien vordefinieren:

1. **Bereitstellung von Funktionen**: Diese Kategorie umfasst Submodule, die hauptsächlich dazu dienen, zusätzliche Funktionen, Bibliotheken oder Erweiterungen bereitzustellen, die von anderen Teilen des Projekts verwendet werden können. Sie sind in der Regel dafür verantwortlich allgemeine Funktionalitäten bereitzustellen, die in verschiedenen Teilen des Projekts wiederverwendet werden und sehr vielseitig Anwendung finden.

2. **Analysewerkzeuge**: Hierbei handelt es sich um Submodule, die speziell für die Analyse des Codes oder der Daten im Projekt entwickelt wurden. Sie können Werkzeuge, Bibliotheken oder Skripte sein, die dazu beitragen die Qualität des Codes zu überwachen, Fehler zu finden oder Statistiken zu generieren.

3. **Telemetrie (Logging und Tracing)**: Submodule in dieser Kategorie sind darauf ausgerichtet Informationen über das Projekt oder das System zu sammeln, normalerweise in Form von `Logs` oder `Traces`. Sie helfen Probleme zu diagnostizieren, den Ablauf des Systems zu verfolgen und statistische Informationen zu sammeln.

4. **Domänenspezifische Funktion**: Diese Submodule sind speziell auf die Anforderungen und Bedürfnisse eines bestimmten Anwendungsbereichs oder einer speziellen Domäne zugeschnitten. Sie bieten Funktionen und Werkzeuge, die in diesem speziellen Kontext nützlich sind und werden demnach in der Regel von nur einem Projekt oder sehr wenigen Projekten verwendet.

5. **Testumgebung**: Submodule in dieser Kategorie enthalten Bibliotheken oder Werkzeuge, die für das Testen des Projekts verwendet werden. Sie sind darauf ausgelegt Testfälle zu erstellen, Tests durchzuführen und sicherzustellen, dass das Projekt ordnungsgemäß funktioniert.

5 EVALUATION

In diesem Abschnitt präsentieren wir zuerst die Ergebnisse unserer Experimente. Im Anschluss bewerten wir die Ergebnisse und Schlussfolgerungen. Hierbei analysieren wir die Ergebnisse im Zusammenhang mit unseren Forschungsfragen. Zusätzlich beleuchten wir kritisch die methodische Validität unserer Arbeit und diskutieren die Implikationen dieser Bewertung. Dies dient dazu die Gültigkeit unserer Schlussfolgerungen und die Bedeutung unserer Forschung zu verdeutlichen.

5.1 ERGEBNISSE

In den anfänglichen Experimenten zur Untersuchung von **FF1.1** stellt sich heraus, dass von den insgesamt 10.000 untersuchten C/C++ Projekten 1.938 Submodule enthalten. Dies entspricht einem Anteil von etwa 19,4%, wie in Abbildung 5.1 veranschaulicht wird. Diese Feststellung ermöglicht die Hochrechnung, dass ungefähr ein Fünftel aller C/C++ Projekte Submodule verwenden.

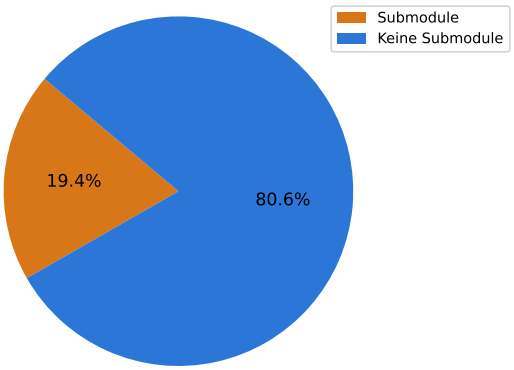

Abbildung 5.1: Prozentuale Anteile aller Projekte mit mindestens einem Submodul (orange) und aller Projekte ohne Submodule (blau).

In Abbildung 5.2 sind alle Projekte dargestellt, die Submodule verwenden. Die rote Linie kennzeichnet eine approximierte Verteilungskurve der Projekte in Abhängigkeit von der Anzahl der genutzten Submodule. Die Balken zeigen verschiedene Bereiche, wobei beispielsweise etwa 1.400 Projekte 1 bis 5 Submodule aufweisen. Die meisten Projekte mit Submodulen verwenden relativ wenige (etwa 72%), danach fällt die Verteilungskurve sehr schnell ab und nur etwa 28% der Projekte verwenden 5 oder mehr Submodule. Es ist jedoch wichtig zu beachten, dass es Ausreißerprojekte gibt. Als Ausreißer gelten alle Projekte ab 74 Submodulen, da hier eine signifikante Zunahme von 59 auf 74 Submodule erfolgt. Eine detaillierte Auflistung dieser Ausreißerprojekte ist in Tabelle 5.1 zu finden.

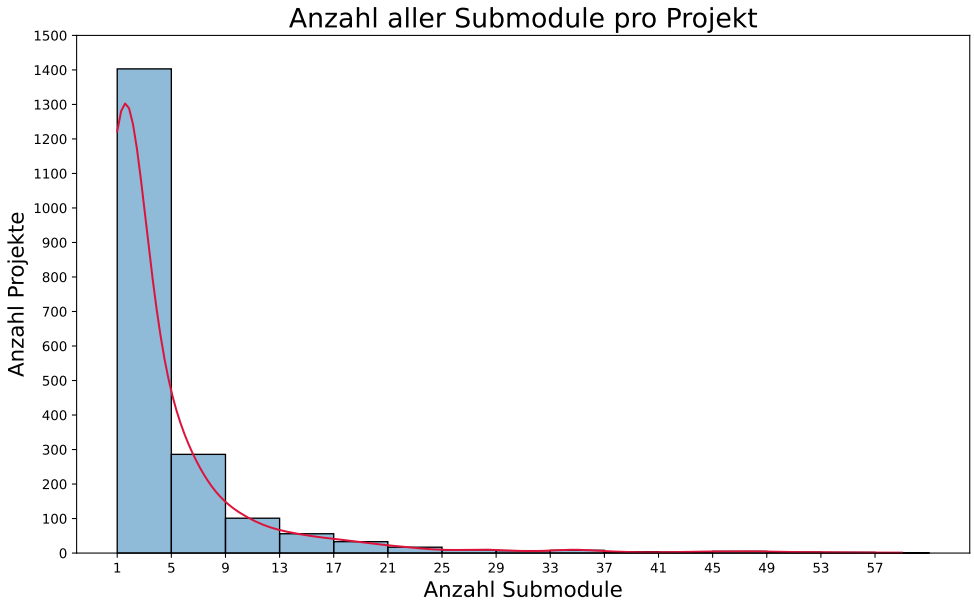

Abbildung 5.2: Approximierte Verteilungskurve und Balkendiagramm für die Anzahl der Submodule pro Projekt, das mindestens ein Submodul enthält.

Im Kontext von **FF1.2** erfolgt eine Analyse zur zeitlichen Entwicklung der Submodulnutzung, die in Abbildung 5.3 visualisiert ist. Dabei werden die Projekte nach ihrem Erstellungsjahr aufgeführt und die Anzahl der Submodule pro Jahr wird ermittelt. Die x-Achse stellt die Jahre dar, in denen die Projekte erstellt wurden, während die y-Achse die Anzahl der erstellten Projekte für jedes Jahr angibt. Projekte, die mindestens ein Submodul enthalten, sind im orangefarbenen Balken dargestellt, während Projekte ohne Submodule im blauen Balken zu finden sind. Zum Beispiel wurden im Jahr 2012 insgesamt 641 Projekte erstellt, von denen 103 mindestens ein Submodul enthalten. Die vorliegende Verteilung zeigt einen erwarteten Anstieg bis zum Jahr 2015, gefolgt von einem Rückgang. Durch das Sortieren der Sterne in absteigender Reihenfolge gehen wir davon aus, dass neuere Projekte ab 2020 unterrepräsentiert sind. Indem wir die Sterne in absteigender Reihenfolge sortieren, nehmen wir an, dass neuere Projekte ab 2020 möglicherweise unterrepräsentiert sind. Dies könnte daran liegen, dass ihre Bewertungen niedriger ausfallen, da die Sternenbewertungen sich kontinuierlich über einen längeren Zeitraum aufbauen.

Die Darstellung in Abbildung 5.4 veranschaulicht das Verhältnis der Submodulnutzung über die Jahre gemäß der Abbildung 5.3. Hierbei bleibt die x-Achse unverändert, während auf der y-Achse die Anzahl der Projekte mit Submodulen durch die Gesamtanzahl der Projekte dividiert und grafisch dargestellt wird. Ein Wert von 0 zeigt an, dass Projekte mit diesem Erstellungsjahr keine Submodule verwenden, während ein Wert von 1 bedeutet, dass alle Projekte mindestens ein Submodul enthalten. Unsere Ergebnisse pendeln sich in etwa zwischen $0,14$ und $0,27$ ein und wir können eine steigende Tendenz über die Jahre beobachten.

Im nachfolgenden Abschnitt präsentieren wir die Ergebnisse, die aus der Untersuchung der Forschungsfragen 2 hervorgegangen sind. Dabei liegt der Fokus auf den Aktualisierungen der Submodule sowie deren spezifischen Rollenzuweisungen innerhalb der untersuchten

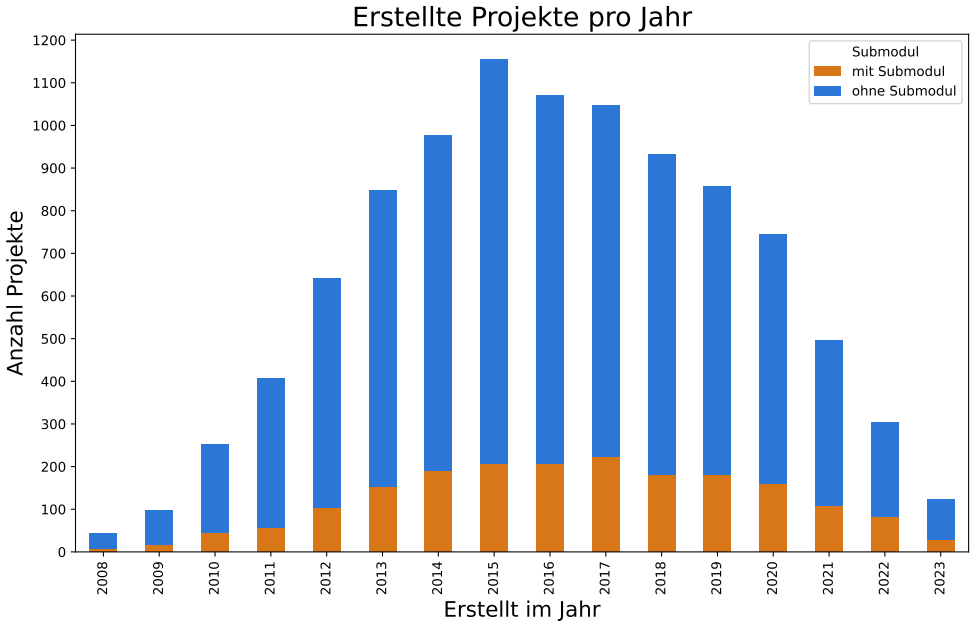

Abbildung 5.3: Die Balken repräsentieren die Gesamtanzahl der Projekte, die jeweils im entsprechenden Jahr erstellt wurden. Ihre Farbunterschiede zeigen an, ob ein Projekt mindestens ein Submodul enthält oder nicht.

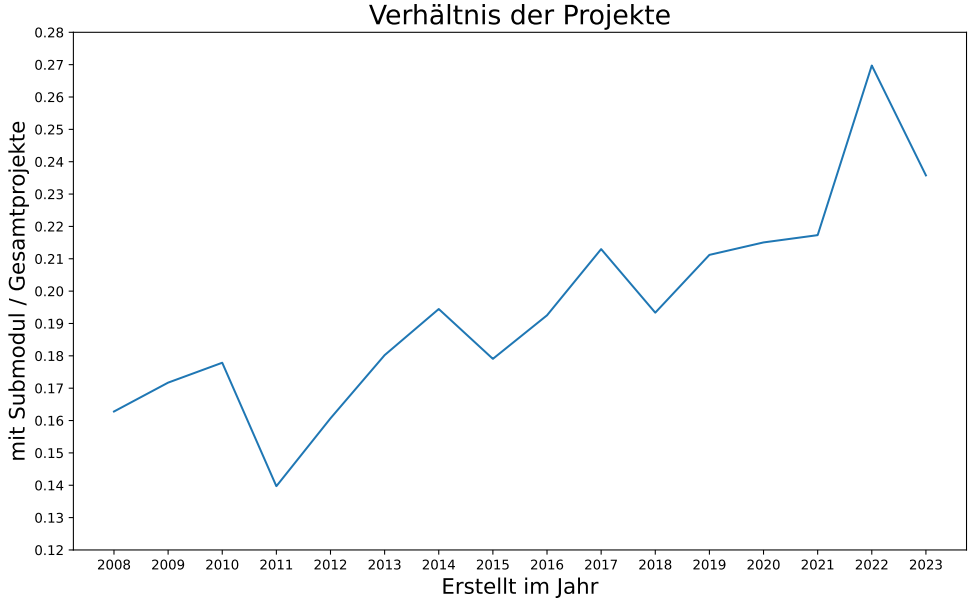

Abbildung 5.4: Der Graph stellt das Verhältnis der Projekte mit Submodulen zu allen erstellten Projekten pro Jahr dar.

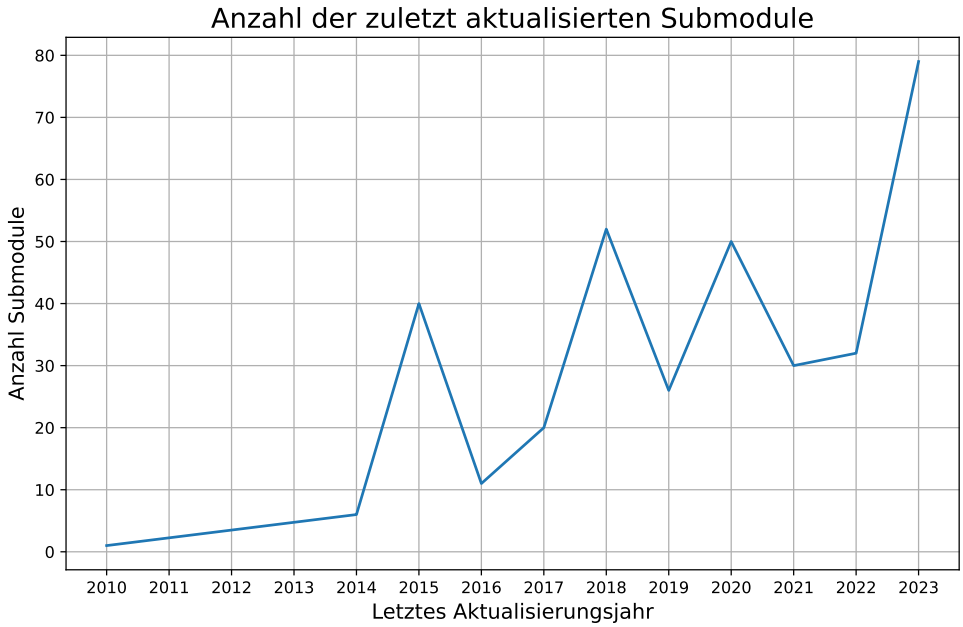

Abbildung 5.5: Die Abbildung veranschaulicht die Anzahl der Submodule, die im jeweiligen Jahr ihre letzte Aktualisierung erfahren haben. Submodule, die nach dem Jahr 2020 eingebunden wurden, sind hierbei nicht inbegriffen.

Projekte. Diese Analyse bietet Einblicke in die Häufigkeit der Aktualisierungen und die unterschiedlichen Funktionen, die diesen Modulen zugewiesen werden.

In Abbildung 5.5 wird zur Untersuchtung der **FF2.1** die Gesamtanzahl der zuletzt aktualisierten Submodule in ihren jeweiligen Hauptrepositorys dargestellt. Beachtenswert ist, dass diese Statistik ausschließlich die Submodule berücksichtigt, die bis zum Jahr 2020 in ein Projekt integriert wurden. Dieser Ansatz zielt darauf ab, die Verzerrung durch einen einzigen Commit zu vermeiden, der auf das Erstellungsdatum datiert ist und keine weiteren Aktualisierungen aufweist. Auf der x-Achse sind die Jahre der letzten Aktualisierung aufgeführt, während die y-Achse die Anzahl der Submodule in diesem spezifischen Jahr zeigt.

In unserer Analyse der Submodul-Aktualisierung zeigt Abbildung 5.6 die Verteilung dieser Aktualisierungen, wobei Ausreißer aus der Visualisierung ausgeschlossen sind. In dem Boxplot haben wir bei den 511 Submodulen 80 Ausreißer identifiziert, nämlich alle Datenpunkte außerhalb des Bereichs der Quartile. Hierbei stechen zwei größere Projekte hervor, nämlich `facebook/CacheLib` und `horsicq/XELFViewer`. In diesen Projekten erfolgen bis zu 1396 Aktualisierungen pro Jahr für Submodule. Bei genauerer Betrachtung haben wir festgestellt, dass die Aktualisierungs-Commits automatisiert generiert werden, was die ungewöhnlich hohe Anzahl erklärt. In der Box repräsentiert die dargestellte Linie den Median von 2 Aktualisierungen pro Jahr, der die Daten in zwei Hälften teilt. Die Box selbst zeigt den Interquartilsabstand, den Bereich zwischen dem 25. und 75. Perzentil, in dem sich die Werte zwischen 1 und 5 Aktualisierungen pro Jahr befinden. Die Linien außerhalb der Box zeigen die Variation der Daten, die nicht als Ausreißer betrachtet werden. Diese reichen von 0 bis 11 Aktualisierungen pro Jahr.

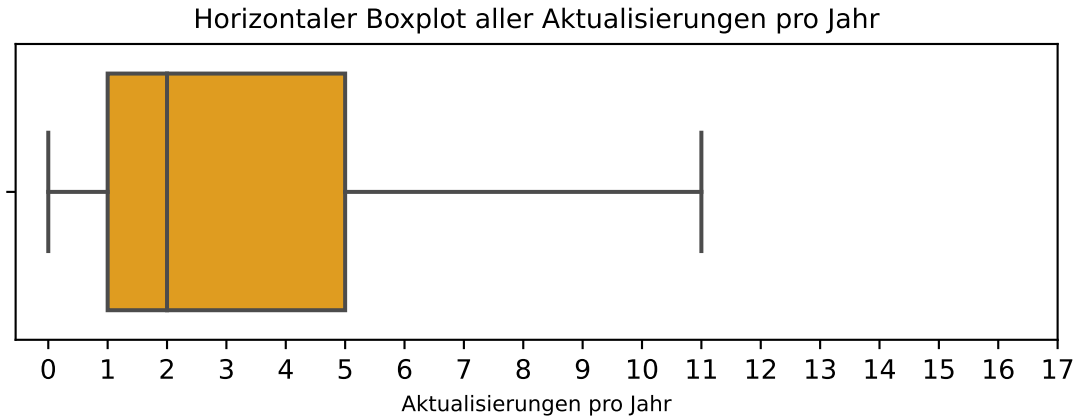

Abbildung 5.6: Der Boxplot zeigt die Anzahl an Submodul-Aktualisierungen pro Jahr ohne Ausreißer.

Im Rahmen von **FF2.1** wird die Beziehung zwischen dem Erstellungsjahr und dem Zeitpunkt der letzten Aktualisierung genauer betrachtet, um die Verteilung dieser zeitlichen Daten zu analysieren. Abbildung 5.7 veranschaulicht die zeitliche Beziehung zwischen dem Erstellungsjahr und dem Zeitpunkt der letzten Aktualisierung für verschiedene Submodule. Jeder Punkt auf der Abbildung repräsentiert ein Submodul, wobei die Position des Punktes auf der horizontalen Achse das Erstellungsjahr und auf der vertikalen Achse das Jahr der letzten Aktualisierung darstellt. Die Punkte sind so skaliert, dass ihre Farbintensität die Häufigkeit der Überlappung mehrerer Submodule an einem bestimmten Zeitpunkt anzeigen. Intensiver gefärbte Punkte deuten auf Bereiche hin, in denen mehrere Submodule mit ähnlichen Erstellungs- und Aktualisierungszeiten liegen, während weniger intensiv gefärbte Punkte auf weniger Überlappungen hinweisen. Die Häufung dunkler Punkte

Tabelle 5.1: Die Tabelle zeigt Projekte, bei denen ab 74 Submodulen Ausreißer festgestellt wurden, da der Anstieg von 59 auf 74 Submodule signifikant ist.

Projekt	**Submodule**
connectedhomeip	74
Cardinal	78
STMems_Standard_C_drivers	81
Sming	82
ByConity	84
fivem	95
RHVoice	107
chdb	111
ClickHouse	116
wifi-arsenal	618

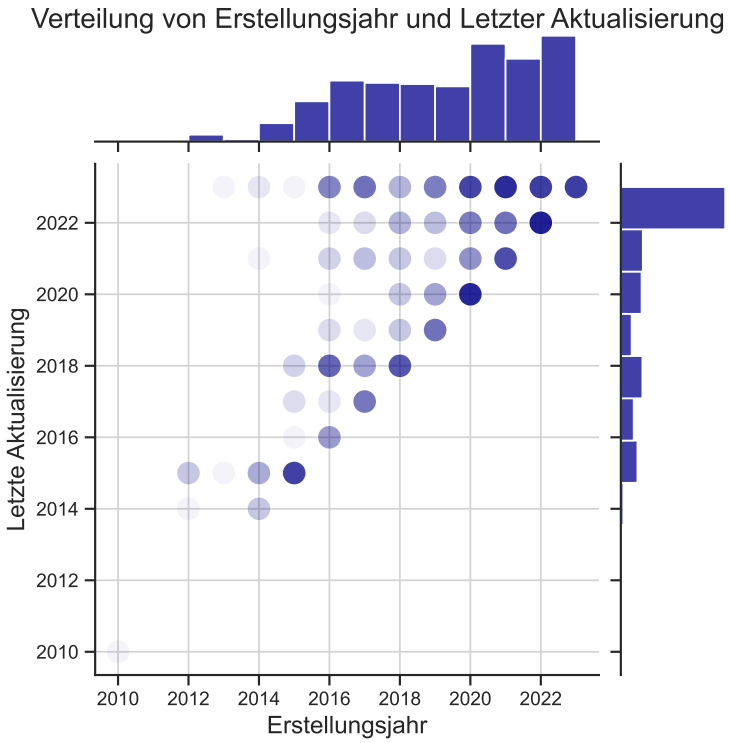

Abbildung 5.7: Die Abbildung illustriert die zeitliche Beziehung zwischen dem Erstellungsjahr und der letzten Aktualisierung für verschiedene Submodule.

im oberen rechten Bereich lässt vermuten, dass viele Submodule aus jüngeren Projekten kürzlich aktualisiert wurden.

Die Zuordnung der Submodule zu den fünf Rollen, die im Rahmen des Card-Sorting-Verfahrens festgelegt wurden, ist in Tabelle 5.2 visualisiert. Von den insgesamt 511 analysierten Submodulen gehören nur 6 zur Kategorie Analysewerkzeuge und lediglich 8 zur Kategorie Telemetrie. Die größte Gruppe von Submodulen, nämlich 327, fällt in die Kategorie Bereitstellung von Funktionen. Hierunter fallen hauptsächlich Submodule, die generelle Funktionalitäten oder Bibliotheken bereitstellen und in verschiedenen Teilen des Projekts wiederverwendet werden. Zusätzlich gibt es mehrfach auftretende Submodule innerhalb der

Tabelle 5.2: Die Tabelle listet die Anzahl aller Submodule nach ihren spezifischen Rollen auf, einschließlich derer, die mehrfach auftreten, sowie der reduzierten Anzahl ohne wiederholte Einträge.

	Funktionen	**Analyse-werkzeuge**	**Telemetrie**	**Domänen-spezifisch**	**Test-umgebung**
insgesamt	327	6	8	140	30
mehrfache abgezogen	301	5	8	125	14

Repositorys. Die zweite Zeile der Tabelle zeigt die Anzahl der Submodule ohne die mehrfach auftretenden Einträge, die insgesamt 453 Submodule zurücklässt. Absolut betrachtet zeigen sich mehrfach auftretende Submodule hauptsächlich in der Kategorie Bereitstellung von Funktionen. Prozentual betrachtet führt jedoch die Kategorie Testumgebung mit dem Submodul `googletest`, das sich insgesamt 14 mal auf verschiedene Repositorys verteilt. Für eine detailliertere Betrachtung der mehrfach auftretenden Submodule verweisen wir auf den Anhang in Tabelle A.1 und Tabelle A.2.

5.2 DISKUSSION

Aus unseren Untersuchungen lässt sich schließen, dass die Mehrheit der untersuchten Projekte entweder überhaupt keine Submodule einsetzt oder in einigen Fällen nur wenige. Wenn Submodule eingebunden werden, dann beschränkt sich die Anzahl der Submodule in den meisten Fällen auf 1 bis 5 Submodule, wobei genau 1 Submodul den größten Anteil mit insgesamt 720 Projekten ausmacht, gefolgt von genau 2 Submodulen mit insgesamt 366 Projekten. Dies liegt vor allem daran, dass viele Projekte Submodule nutzen, die dazu dienen, allgemeine Funktionen bereitzustellen, welche in verschiedenen Teilen des Projekts wiederverwendet werden können. Nur eine geringe Anzahl, etwa 50 Projekte von insgesamt 1.938, die Submodule verwenden, haben mehr als 17 Submodule. Es ist von besonderem Interesse, das umfangreichste Projekt namens `wifi-arsenal` genauer zu untersuchen, da es 618 Submodule aufweist. Die Erklärung für die große Anzahl von Submodulen in diesem Repository liegt darin, dass es primär als eine Sammlung von Drittanbieterbibliotheken dient, die sich auf WLAN-Funktionalitäten beziehen. Diese Bibliotheken können in anderen Projekten verwendet werden, um WiFi-bezogene Funktionalitäten zu integrieren. Die Projekte `ClickHouse` und `chdb` fallen ebenfalls auf, da sie mit 116 bzw. 111 Submodulen die zweit- und drittgrößten Submodul-Anzahlen aufweisen. Dies ist darauf zurückzuführen, dass es sich bei diesen Projekten um umfangreiche Datenbankmanagementsysteme handelt, die viele domänenspezifische Funktionalitäten in ihrem Bereich erfordern.

Wenn wir die Verteilung der Submodule über die Zeit betrachten, fällt auf, dass der Trend allgemein dahin geht, dass immer mehr Submodule eingebunden werden. Dies legt nahe, dass die Verwendung von Submodulen insgesamt an Popularität gewinnt. Der scheinbare Rückgang der Gesamtanzahl neu erstellter Projekte in Abbildung 5.3 könnte daran liegen, dass unser Datenset nicht immer die neuesten Projekte enthält. Neue Projekte scheinen generell weniger Sterne zu erhalten, da die auf GitHub von Nutzern vergebenen Sterne Zeit benötigen, um sich anzusammeln und zu akkumulieren. Diese Bewertungen spiegeln sich nicht sofort über kurze Zeiträume wider, was erklären könnte, warum neuere Projekte im Durchschnitt weniger Sterne aufweisen. Daher sollte dieser Rückgang nicht als Indikator für eine generelle Abnahme der Projekte auf GitHub interpretiert werden.

Unser Fazit zu **FF1.1** und **FF1.2** lautet, dass in unserer Untersuchung etwa jedes fünfte C/C++ Projekt Submodule einbindet. Der allgemeine Trend zeigt eine zunehmende Verwendung. Hierbei ist bemerkenswert, dass neuere Projekte im Vergleich zu älteren mehr Submodule integrieren.

In unserer zweiten Forschungsfrage widmen wir uns einer detaillierteren Analyse der Submodule. Die Grafik in Abbildung 5.5 zeigt, dass im Jahr 2023 die meisten Submodule aktualisiert wurden. Dies deutet darauf hin, dass die Mehrheit der Submodule regelmäßig

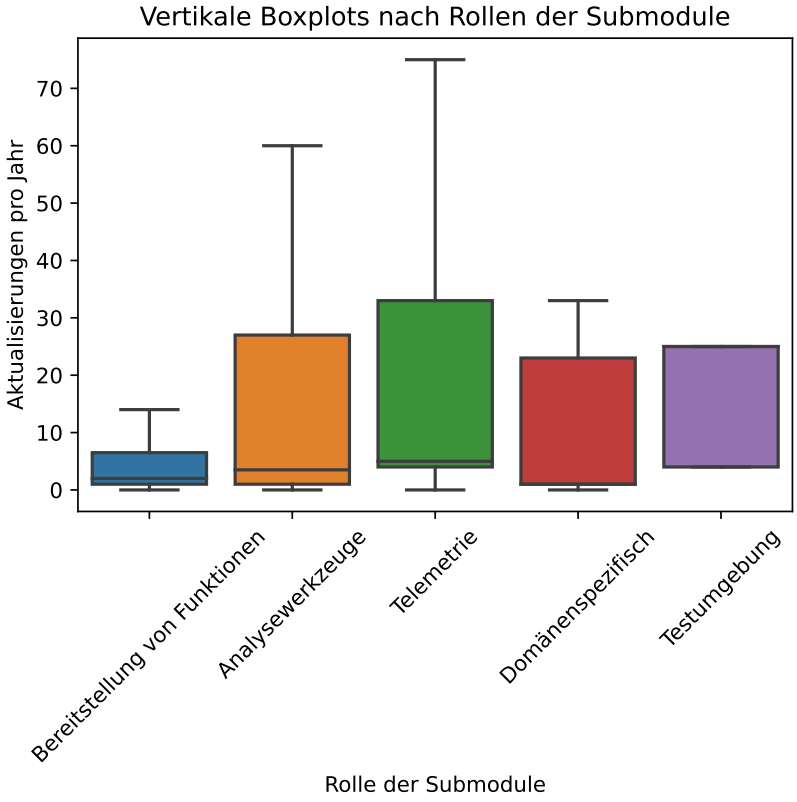

Abbildung 5.8: Die Boxplots zeigen für die verschiedenen Rollen die Aktualisierungen pro Jahr.

auf dem neuesten Stand gehalten wird. Dennoch gibt es bemerkenswerte Spitzen in den Jahren 2015, 2018 und 2020, für die wir keine zufriedenstellende Erklärung gefunden haben.

Der Boxplot in Abbildung 5.6 verdeutlicht, dass Aktualisierungen für Submodule häufig jährlich, halbjährlich oder vierteljährlich durchgeführt werden. Es stellt sich die Frage, ob die Rolle des Submoduls Einfluss auf die Häufigkeit der Aktualisierungen hat. Unsere Vermutung ist, dass Submodule, die allgemeine Funktionen bereitstellen, weniger häufig aktualisiert werden müssen, da ihre Funktionen in der Regel stabil und universell gültig sind.

Um diese Vermutung zu untersuchen haben wir die Boxplots aller Rollen miteinander verglichen. In Abbildung 5.8 sind die jährlichen Aktualisierungen dargestellt, wobei Ausreißer in dieser Analyse nicht berücksichtigt wurden. Auffällig ist, dass der Boxplot der Rolle *Bereitstellung von Funktionen* einen deutlich kleineren Bereich aufweist als die Boxplots der anderen Rollen. Zusätzlich liegt der Median dieser Rolle ebenfalls niedriger. Dies unterstützt unsere Annahme, dass die Submodule, die dieser Rolle zugeordnet sind, seltener aktualisiert werden müssen im Vergleich zu denen der anderen Rollen.

Die Verteilung der Aktualisierungen in Abbildung 5.7 zeigt, dass, obwohl viele Submodule in den Jahren 2015 bis 2023 eingebunden wurden, die Mehrheit der Aktualisierungen im Jahr 2022 und 2023 erfolgte. Dies deckt sich mit unserer Annahme, dass Submodule regelmäßig und bis in die aktuelle Zeit aktualisiert werden.

Unser Fazit zu **FF2.1** und **FF2.2** besagt, dass Git-Submodule bis zum Jahr 2023 regelmäßige Aktualisierungen erfahren haben. Die Rolle des Submoduls ist allerdings ein

ausschlaggebender Faktor. Bei der Rolle *Bereitstellung von Funktionen*, die etwa 64% des Datensatzes **DS100** ausmacht, ist die Häufigkeit der Aktualisierungen pro Jahr deutlich geringer im Vergleich zu den anderen Rollen. Dies lässt sich damit begründen, dass allgemeine Funktionen weniger häufige Aktualisierungen erfordern, da sie stabil und universell gültig sind.

5.3 VALIDITÄTSBEACHTUNGEN

Bei der Durchführung der Forschung müssen bestimmte Validitätsbeachtungen berücksichtigt werden, insbesondere im Hinblick auf die Wahl der Sprachen C/C++. Die Validitätsbeachtungen sind wie folgt:

Interne Validität: Es ist möglich, dass die GitHub-REST-API Projekte, die eigentlich den Anforderungen entsprechen auslässt, da sie die Daten nicht richtig verarbeiten kann, beziehungsweise eine Ergebnisbegrenzung von 1000 Projekten vorhanden ist. Um diese zu umgehen, haben wir diesen Filter auf kleine Bereiche eingegrenzt (10 Sterne pro Schritt), sodass jede einzelne Anfrage ein Ergebnis unterhalb der Grenze von 1000 Projekten liefert.

Es ist wichtig anzumerken, dass wir eine zufällige Auswahl von 100 Projekten als unseren Datensatz **DS100** definiert haben, basierend auf einer gleichmäßigen Verteilung. Es besteht die Möglichkeit, dass diese Projekte die Ergebnisse beeinflussen könnten, besonders wenn sie übermäßig viele oder wenige Submodule besitzen und somit eine Verzerrung verursachen könnten.

Externe Validität: Die Verallgemeinerung unserer Ergebnisse ist mit einigen Einschränkungen verbunden. Unsere Untersuchung konzentriert sich ausschließlich auf C/C++ Projekte, die auf der Plattform GitHub sind. Da es unmöglich ist, alle C/C++ Projekte auf GitHub zu analysieren, beschränken wir uns auf eine Stichprobe von 10.000 Projekten, die nach der Anzahl der Sterne des Repositorys sortiert sind. Diese Sterne gelten laut Borges u. a. [3] als Maß für die Popularität eines Projekts. Wir haben uns dafür entschieden, uns hauptsächlich auf die wohl bekannteren Projekte zu konzentrieren. Das hat einen Effekt auf die Allgemeingültigkeit, da neuere Projekte mit niedrigeren Sternebewertungen oft nicht in unserem Datensatz enthalten sind.

Ein weiteres Problem, das sich bei C/C++ Projekten ergibt, ist das Fehlen eines dedizierten Paketmanagers. Daher ist die Analyse von Submodulen besonders interessant, da es laut Tang u. a. [9] nur wenige sinnvolle Techniken gibt, um Drittanbieterbibliotheken in das C/C++ Ökosystem einzubinden. Die Ergebnisse können somit nicht auf andere Programmiersprachen angewendet werden.

6 VERWANDTE ARBEITEN

In diesem Abschnitt stellen wir relevante Arbeiten vor, die in Bezug auf Drittanbieterbibliotheken, C/C++ Projekte und Git Submodulen Parallelen zu unserer eigenen Forschung aufweisen. Wir analysieren, inwiefern diese Quellen zur Entwicklung unserer Arbeit beitragen und nehmen eine vergleichende Bewertung der erzielten Ergebnisse vor.

Die Arbeit von Borges u. a. [3] hat gezeigt, dass die Anzahl der Sterne eines GitHub Repositorys eine maßgebliche Metrik für dessen Popularität darstellt. Dies betonen auch Jiang u. a. [5], die für ihr Datenset vor allem die C/C++ Projekte verwenden, die über 1000 Sterne besitzen und analysieren die in der README-Datei enthaltenen Informationen. In unserer Arbeit bewerten wir C/C++ Projekte anhand ihrer Sterne und sortieren sie danach, um somit eine repräsentative Stichprobe populärer Projekte für unsere Analyse auszuwählen. Die Informationen aus den README-Dateien dienen uns zur Kategorisierung der Submodule.

Gousios [4] nutzt in seinem Beitrag die GitHub REST-API, um abfragbare Metadaten zu extrahieren, zu archivieren und weiterzugeben. Er präsentiert die Herausforderungen und Einschränkungen bei der Arbeit mit dem durch die GitHub REST-API zur Verfügung gestellten Datensatz. Dadurch können wir beispielsweise das Verständnis für die Anfragebegrenzung von 5000 Anfragen pro Stunde für authentifizierte Anfragen der GitHub REST-API gewinnen und diese in unseren Code integrieren. AlMarzouq u. a. [1] haben uns zusätzlich die Verwendung und Einschränkungen von Filter- und Sortierfunktionen aufgezeigt und somit die Nutzung erheblich erleichtert.

Die Verwendung und Verbreitung von Drittanbieterbibliotheken spielt in vielen Arbeiten eine große Rolle. In unserer Arbeit sind sie essenziell, da sie großer Bestandteil der einzelnen Submodule sind, die dazu dienen, diese in ein Projekt einzubinden. In der Arbeit von Wang und Guo [10] werden Drittanbieterbibliotheken in Verbindung mit mobilen Anwendungen auf deren Verteilung analysiert. Die Untersuchung zeigt auf, dass Drittanbieterbibliotheken in mobilen Anwendungen einen Anteil von mehr als 60% des Gesamtcodes ausmachen. Larios Vargas u. a. [7] beziehen sich vor allem auf die Rolle der einzelnen Drittanbieterbibliotheken und analysieren dies mithilfe dutzender Faktoren, wie zum Beispiel Qualität, Freigabefaktoren und entstehende Betriebskosten. Die Entscheidung, welche Drittanbieterbibliothek ausgewählt werden soll ist in jedem Projekt ein praktisches Problem und hängt mitunter von der Softwarearchitektur ab. Auch der Beitrag von Bauer u. a. [2] beschäftigt sich mit der Auswahl bestimmter Bibliotheken und verwendet dabei einen strukturierten Ansatz zur Bewertung von Bibliotheken in Softwareprojekten. Dieser Ansatz basiert auf der Idee, dass die Gesamtzahl der aufgerufenen Methoden einer Bibliothek es ermöglicht, alle externen Bibliotheken nach der Stärke ihrer direkten Beziehungen zum System zu ordnen. Indem die Anzahl der Methodenaufrufe als Maßstab verwendet wird, kann die Bedeutung und der Einfluss einer Bibliothek auf das gesamte System besser verstanden und bewertet werden. Die Arbeit von Nguyen u. a. [8] hilft bei der Bewertung mit einem Empfehlungssystem, das gezielt auf das Projekt zugeschnittene Drittanbieterbibliotheken

empfiehlt. Somit helfen uns die Ergebnisse der Arbeiten dabei, die Notwendigkeit und Rolle der Submodule einordnen zu können. Tang u. a. [9] haben große Parallelen zu unserer Arbeit, da sie auch die Einbindung von Drittanbieterbibliotheken in C/C++ Projekten mit Submodulen untersuchen. Sie betonen auch, dass ohne Paketmanager die bisherige Forschung nur wenige Erkenntnisse über die Abhängigkeiten der Drittanbieterbibliotheken im C/C++ Ökosystem hat. Darum gibt es nur wenige sinnvolle Techniken diese einzubinden und eine davon sind Git Submodule.

7 SCHLUSSBEMERKUNGEN

Diese Arbeit analysiert die Verwendung und Verbreitung von Git Submodulen in C/C++ Projekten. Vor dem Hintergrund des Mangels an einem dedizierten Paketmanager und der Notwendigkeit, Drittanbieterbibliotheken in diesen Sprachen einzubinden, ist es motivierend, die Rolle und Verbreitung von Git Submodulen zu untersuchen.

Die Untersuchung der Verbreitung auf GitHub offenbart, dass etwa ein Fünftel der untersuchten C/C++ Projekte Submodule einsetzt, wobei eine Tendenz zur vermehrten Nutzung in neueren Projekten festzustellen ist.

Die Analyse zeigt, dass Git Submodule regelmäßig aktualisiert werden, wobei die Rolle *Bereitstellung von Funktionen* eine zentrale Bedeutung einnimmt. Submodule, die allgemeine Funktionen bereitstellen, werden im Durchschnitt etwa zweimal pro Jahr aktualisiert, während andere Rollen häufiger Aktualisierungen erfahren.

Diese Ergebnisse betonen die Wichtigkeit von Git Submodulen in C/C++ Projekten und weisen darauf hin, dass ihre Verwendung sich vor allem in neueren Projekten weiter ausbreitet. Die Erkenntnisse über die Rollen der Submodule bieten Einblicke in ihre unterschiedlichen Verwendungsweisen und zeigen die Dynamik ihrer Aktualisierungen auf.

Es ist anzumerken, dass diese Studie einen Schwerpunkt auf die Analyse von C/C++ Projekten auf GitHub legt und weitere Untersuchungen könnten in anderen Repositorys oder auf Basis anderer Metriken durchgeführt werden. Zukünftige Forschung könnte sich auch darauf konzentrieren, wie Entwickler die Nutzung von Submodulen optimieren können.

Diese Arbeit liefert somit eine Grundlage für ein tieferes Verständnis der Verwendung und Verbreitung von Git Submodulen in C/C++ Projekten und bietet Platz für weitere Forschung in diesem Bereich.

A

ANHANG

Tabelle A.1: Die Tabelle zeigt alle mehrfach auftretenden Submodule und deren Anzahl in der spezifischen Rolle. (1/2)

Submodul	**Funktionen**	**Analyse-werkzeuge**	**Telemetrie**	**Domänen-spezifisch**	**Test-umgebung**
googletest					14
src				8	
stb	4				
sdl	4				
eigen	4				
rapidjson	4				
common	4				
esp-idf				4	
libbpf-bootstrap				2	
protobuf				2	
imgui	3				
harfbuzz	3				
zlib	3				
catch					3
cxxopts	2				
nanoflann	2				
JUCE	2				
mbedtls				2	
glm	2				
kenlm	2				
tinyxml2	2				
opus	2				
Pangolin	2				
Sophus	2				
motive	2				
fplbase	2				

Tabelle A.2: Die Tabelle zeigt alle mehrfach auftretenden Submodule und deren Anzahl in der spezifischen Rolle. (2/2)

Submodul	**Funktionen**	**Analyse-werkzeuge**	**Telemetrie**	**Domänen-spezifisch**	**Test-umgebung**
freetype	2				
cardboard-java	2				
flatbuffers	2				
fplutil	2				
mathfu	2				
abseil	2				
libunibreak	2				
vectorial	2				
webp	2				
args	2				
optional	2				
fontobene	2				
minhook	2				
cub	2				
fmt	2				
pybind11	2				
boost		2			
uhej				2	
onnx				2	
Catch2					2
Gesamt	61	2	0	22	19

LITERATUR

[1] Mohammad AlMarzouq, Abdullatif AlZaidan und Jehad AlDallal. „Mining GitHub for research and education: challenges and opportunities“. In: *International Journal of Web Information Systems*. 2020, S. 451–473.

[2] Veronika Bauer, Lars Heinemann und Florian Deissenboeck. „A structured approach to assess third-party library usage“. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. 2012, S. 483–492.

[3] Hudson Borges, Andre Hora und Marco Tulio Valente. „Understanding the Factors That Impact the Popularity of GitHub Repositories“. In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2016, S. 334–344.

[4] Georgios Gousios. „The GHTorent dataset and tool suite“. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. 2013, S. 233–236.

[5] Ling Jiang, Hengchen Yuan, Qiyi Tang, Sen Nie, Shi Wu und Yuqun Zhang. „Third-Party Library Dependency for Large-Scale SCA in the C/C++ Ecosystem: How Far Are We?“ In: *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2023. 2023, 1383–1395.

[6] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German und Daniela Damian. „The Promises and Perils of Mining GitHub“. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. 2014, 92–101.

[7] Enrique Larios Vargas, Maurício Aniche, Christoph Treude, Magiel Bruntink und Georgios Gousios. „Selecting Third-Party Libraries: The Practitioners' Perspective“. In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, 245–256.

[8] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio und Massimiliano Di Penta. „CrossRec: Supporting software developers by recommending third-party libraries“. In: *Journal of Systems and Software* 161 (2020), S. 110460.

[9] Wei Tang, Zhengzi Xu, Chengwei Liu, Jiahui Wu, Shouguo Yang, Yi Li, Ping Luo und Yang Liu. „Towards Understanding Third-Party Library Dependency in C/C++ Ecosystem“. In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2023.

[10] Haoyu Wang und Yao Guo. „Understanding Third-Party Libraries in Mobile App Analysis“. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 2017, S. 515–516.