

Master-Thesis

Identifikation und Vergleich von Autorenangaben zu Software
zwischen verschiedenen Datenquellen

Eingereicht am: 3. Januar 2025

von: Kevin Jahrens
geboren am 05.08.1999
in Bad Oldesloe

Matrikelnummer: 480592

Betreuer: Prof. Dr. -Ing. Frank Krüger^{ID}
Hochschule Wismar, Fakultät für Ingenieurwissenschaften
Bereich Elektrotechnik und Informatik, Wismar, Deutschland

Zweitbetreuer: M.A. Stephan Druskat^{ID}
Deutsches Zentrum für Luft- und Raumfahrt (DLR),
Institut für Softwaretechnologie, Berlin, Deutschland

Wismar, 1. Juli 2024

Master-Thesis

für: Herr Kevin Jahrens

Identifikation und Vergleich von Autorenangaben zu Software zwischen verschiedenen Datenquellen

Identifikation and comparison of authors of software across different data sources

Disposition

Software spielt eine zentrale Rolle in der Wissenschaft und sollte daher in wissenschaftlichen Arbeiten zitiert werden. Insbesondere für Autoren wissenschaftlicher Software ist die Zitation wesentlicher Bestandteil der wissenschaftlichen Anerkennung, sodass diese auch zunehmend in wissenschaftlichen Lebensläufen genannt werden und Beachtung finden. Anders als bei wissenschaftlichen Publikationen ist bei wissenschaftlicher Software aktuell noch unklar, welcher Anteil an der Entwicklung zu einer Nennung als Autor führt. Darüber hinaus existieren in verschiedenen Datenquellen widersprüchliche Angaben für Zitationsvorschläge bzgl. der Autoren einer Software.

Ziel dieser Masterarbeit ist es zu untersuchen inwieweit sich die Angaben von Autoren für Open Source Software unterscheiden. Dazu sollen öffentlich verfügbare Repositorien mit R und Python Paketen – als Stellvertreter für wissenschaftliche Software – hinsichtlich ihrer Autorenangaben untersucht werden. Insbesondere sollen die angegebenen Metadaten in den Repositorien (z.B. citation.cff) mit den Metadaten in Paketdatenbanken (<https://pypi.org/> und <https://cran.r-project.org/>) und den Entwicklungsanteilen automatisch verglichen werden.

1. Literaturrecherche Autorenrolle in Open Source Software und zur Disambiguierung von Autorennamen
2. Datensammlung: Identifikation und Download verfügbarer Metadaten zu „wichtigen“ Software-paketen
3. Automatische Auflösung und Abgleich der Autorennennungen aller Datenquellen
4. Analyse von Unterschieden in der Nennung von Autoren
5. Dokumentation der Ergebnisse in einer schriftlichen Master-Thesis

Startdatum: 16.09.2024

Abgabedatum: 17.03.2024


Prof. Dr. rer. nat. Litschke
Chairman of the Examination Committee



Prof. Dr.-Ing. Krüger
Supervisor

Kurzfassung

Die zunehmende Digitalisierung der Wissenschaft erhöht die Bedeutung von Open-Source-Software (OSS) und fordert neue Ansätze zur Anerkennung von Autoren. In dieser Arbeit werden Git-Repositories und andere Quellen wie das Citation File Format (CFF) analysiert, um die Qualität der Autorenangaben zu ermitteln. Eine Software extrahiert, aggregiert und visualisiert Autoreninformationen aus Comprehensive R Archive Network (CRAN)- und Python Package Index (PyPI)-Repositories. Die Ergebnisse zeigen Lücken bei der Nennung von Autoren in OSS. Abschließend wird anhand der Ergebnisse diskutiert, was Softwareentwickler leisten müssen, um als Autor genannt zu werden.

Abstract

The increasing digitalization of science increases the importance of Open-Source-Software (OSS) and calls for new approaches to author recognition. In this work, Git repositories and other sources such as the Citation File Format (CFF) are analyzed to determine the quality of author information. A developed software extracts, aggregates and visualizes author information from Comprehensive R Archive Network (CRAN) and Python Package Index (PyPI) repositories. The results show gaps in the naming of authors in OSS. Finally, the results are used to discuss what software developers need to achieve in order to be recognized as an author.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Wissenschaft wurde in den letzten Jahrzehnten zunehmend digitalisiert und ist mittlerweile deutlich stärker auf OSS angewiesen als zuvor. In wissenschaftlichen Arbeiten werden längst nicht mehr nur Paper und Bücher veröffentlicht, sondern auch Software, Daten und andere digitale Artefakte. Dies steigert die Reproduzierbarkeit und Nachvollziehbarkeit der Arbeiten. Der Wandel der Wissenschaft bringt allerdings die Herausforderung mit sich, Software angemessen zu würdigen. Neue Zitierformate und -weisen werden unerlässlich, um diesen Ansprüchen gerecht zu werden. Außerdem benötigt es eine gerechte Würdigung der einzelnen Autoren¹, die maßgeblich zur Weiterentwicklung von OSS beitragen.

Oftmals engagieren sich Softwareentwickler dafür in ihrer Freizeit, um die Qualität und Funktionalität der Software zu verbessern. Dabei kommt es immer wieder vor, dass ihre Arbeit nicht anerkannt wird, da sie Viele Menschen beteiligen sich regelmäßig und freiwillig an der Entwicklung von OSS. Sie entwickeln neue Funktionen, lösen Probleme oder verbessern die Codequalität. Selbst wenn sie einen signifikanten Beitrag zum aktuellen Stand des Projekts beigetragen haben, kann es vorkommen, dass sie gar nicht als Autor der Software oder sogar ihres Beitrags genannt aufgeführt werden.

Füllwort Dies ist beispielsweise dem Entwickler Ariel Miculas widerfahren, welcher einen Beitrag zum Linux-Kernel geleistet hat und nicht als Autor genannt wurde ([miculas_how_2023](#)). Im Gegensatz zu wissenschaftlichen Arbeiten ist die Autorschaft von Software nicht semantisch geklärt ([schmidt_software_nodate](#)).

Um zu prüfen, welchen Einfluss dies auf OSS hat, stellen sich folgende Forschungsfragen:

F1 Wie gut können Autoren untereinander abgeglichen werden?

F2 Was muss ein Softwareentwickler leisten, um als Autor genannt zu werden?

¹Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) verzichtet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

F3 Wie gut werden Autoren in den einzelnen Quellen gepflegt?

1.2 Vorgehen

In ~~der-dieser~~ Arbeit werden Autoren aus unterschiedlichen Quellen extrahiert. Eine wichtige Quelle ist Git, welche als Referenz genutzt wird. Andere Quellen, wie beispielsweise das CFF, werden mit den Autoren abgeglichen. Dadurch ist es möglich, Daten zu erheben, die zeigen, wie einzelne Git-Autoren in den Quellen repräsentiert werden. Die Daten werden anschließend graphisch aufbereitet und präsentiert.

Es muss berücksichtigt werden, dass in ~~der-dieser~~ Masterarbeit ausschließlich **Git-Autoren an Git-Repositorys beteiligte Autoren** betrachtet werden. Dadurch werden nur Personen betrachtet, die Code beigetragen haben. Allerdings wird somit nicht das gesamte Spektrum von Beiträgen zur Software abgedeckt, da beispielsweise Grafikdesigner nicht genannt werden. Diese sind allerdings für die Entwicklung von Software ebenfalls wichtig und sollten berücksichtigt werden. Dies soll jedoch nicht Inhalt dieser Arbeit sein.

1.3 Gliederung

In dieser Arbeit wird Software ~~erzeugt entwickelt~~, die es erlaubt, für eine beliebige Liste an CRAN- oder PyPI-Repositorys Autorenangaben aus verschiedenen Datenquellen zu extrahieren, diese im Anschluss zu aggregieren und graphisch aufzuarbeiten. Dabei werden in ?? die benötigten Grundlagen geschaffen, um beispielsweise die Themen Versionsverwaltung, Paketverwaltung, Software-Verzeichnisse und im Allgemeinen verschiedene Zitierformate für Software verstehen zu können. Sie werden benötigt, um in ?? ~~Software zu entwickeln, über die grundlegenden Bestandteile und Funktionen der entwickelten Software vorzustellen~~, die Autorenangaben aus verschiedenen Datenquellen extrahiert und ~~abgeglichen werden können abgleicht~~. Dabei werden die wesentlichen Designentscheidungen und ihre Umsetzung erläutert. Anschließend werden die Ergebnisse in ?? präsentiert. Dabei wird auf Ergebnisse der Gegenwart und jene mit Zeitverlauf ~~in der Vergangenheit~~ eingegangen. Im darauffolgenden ?? werden die Probleme und Herausforderungen ~~der dieser~~ Arbeit diskutiert. Außerdem werden die Forschungsfragen beantwortet. Im ?? wird die Arbeit zusammengefasst und ein Ausblick auf weitere Forschungsansätze gegeben, welche mit dieser Arbeit möglich geworden sind.

2 Grundlagen

In ?? wird auf die Prinzipien der Software-Zitation eingegangen. Es wird beschrieben, warum die Zitation von Software ebenfalls wichtig ist, ähnlich wie die Zitation von anderen wissenschaftlichen Arbeiten. Außerdem wird darauf eingegangen, dass ebenfalls Personen zitiert werden sollten, welche nicht aktiv an der Software programmieren. Zusätzlich dazu wird in ?? auf die Rolle von Autoren in OSS eingegangen und ein wissenschaftliches Paper ~~dargestellt~~vorgestellt, welches dies bereits analysiert hat.

Autoren von Software werden in unterschiedlichen Quellen zitiert. Einige dieser Quellen sind stark mit der Softwareentwicklung verbunden. Es existieren verschiedene Systeme, die Entwicklern zur Verfügung stehen, um ihre Arbeit effizienter zu gestalten oder überhaupt praktikabel zu machen. In diesen Systemen können Sie außerdem als Autoren genannt werden. In den Abschnitten ?? und ?? wird auf die Versions- und Paketverwaltung eingegangen, welche zwei dieser Systeme darstellen. Des Weiteren existieren spezielle Zitierformate, in welchen Autoren explizit angegeben werden können. Auf diese Formate wird in ?? eingegangen. Zudem können in Fließtexten, beispielsweise der Beschreibung einer Software, ebenfalls Autoren genannt werden. In ?? wird auf die *Named Entity Recognition* eingegangen, welche eine Methode darstellt, um Personen in Texten zu erkennen.

Alle Quellen, welche beschrieben werden, dienen im Verlauf ~~der~~dieser Masterarbeit als Grundlage für die Extraktion von Autoren und deren Metainformationen. Die extrahierten Autoren müssen anschließend zugeordnet werden. Der Prozess dafür heißt *Author Name Disambiguation*, welcher in ?? beschrieben wird. Eine weitere Möglichkeit ~~des Abgleichs der Zuordnung~~ ist ein Abgleich von Zeichenfolgen. Dieser funktioniert jedoch nicht immer, da Autoren unterschiedliche Schreibweisen ihres Namens verwenden können. Aus diesem Grund wird in ?? auf die unscharfe Suche eingegangen, welche eine Möglichkeit darstellt, um ähnliche Zeichenfolgen miteinander zu vergleichen, beispielsweise für den Abgleich von Namen mit oder ohne genannten Zwischennamen.

2.1 Zitation von Software

Software ist ein wesentlicher Bestandteil moderner Forschung. In der wissenschaftlichen Literatur ist es üblich, Quellen zu zitieren, um die Nachvollziehbarkeit und Reproduzierbarkeit von wissenschaftlichen Arbeiten zu gewährleisten. Im Gegensatz dazu ist dies bei wissenschaftlicher Software aktuell in diesem Umfang noch nicht gegeben. Hier gibt es aktuell kaum Anerkennung und Unterstützung für die Leistungen einzelner Autoren. Aus diesem Grund hat die „FORCE11 Software Citation Working Group“ Prinzipien der Software-Zitation erstellt, welche eine breite Akzeptanz in der wissenschaftlichen Gemeinschaft finden sollen. Im Folgenden werden die Prinzipien vorgestellt und erläutert (**smith_software_2016**):

1. **Wichtigkeit:** Software sollte ein seriöses und zitierbares Produkt wissenschaftlicher Arbeit sein. Software-Zitierungen sollten im wissenschaftlichen Kontext die gleiche Bedeutung zugeschrieben bekommen wie Zitierungen anderer Forschungsprodukte, ~~wie Publikationen~~. Sie sollten wie Publikationen auch in der Arbeit enthalten sein, z. B. in der Referenzliste eines Artikels.
2. **Anerkennung und Zuschreibung:** Softwarezitate sollten die wissenschaftliche Anerkennung und die normative, rechtliche Würdigung aller Mitwirkenden an der Software ermöglichen, wobei anerkannt wird, dass ein einziger Stil oder ein Mechanismus für die Namensnennung nicht auf jede Software anwendbar sein kann.
3. **Eindeutige Identifikation:** Ein Softwarezitat sollte eine Methode zur Identifikation enthalten, die maschinell verwertbar, weltweit eindeutig und interoperabel ist und zumindest von einer Gemeinschaft der entsprechenden Fachleute und vorzugsweise von allgemeinen Forschern anerkannt wird.
4. **Persistenz:** Eindeutige Identifikatoren und Metadaten, die die Software und ihre Verwendung beschreiben, sollten bestehen bleiben – auch über die Lebensdauer der Software hinaus.
5. **Zugänglichkeit:** Softwarezitate sollten den Zugang zur Software selbst und zu den zugehörigen Metadaten, Dokumentationen, Daten und anderen Materialien erleichtern, die sowohl für Menschen als auch für Maschinen notwendig sind, um die referenzierte Software sachkundig nutzen zu können.
6. **Spezifität:** Softwarezitate sollten die Identifikation und den Zugang zu der spezifischen Version der verwendeten Software erleichtern. Die Identifizierung der Software sollte so spezifisch wie nötig sein, z. B. durch Versionsnummern, Revisionsnummern oder Varianten wie Plattformen.

In dieser Arbeit wird verstrt auf das Prinzip der Wichtigkeit eingegangen, da ~~besonders~~ im CFF- und BIBTEX-Format die ~~Mglichkeit besteht, nicht die Software~~, ~~sondern~~ Option besteht, anstelle der Software beispielsweise einen Artikel anzugeben. Diese Zitierweise wrde dann das Prinzip der Wichtigkeit verletzen, da die Software nicht die gleiche Bedeutung zugeschrieben bekommt wie andere Forschungsprodukte. Diese Diskrepanz wird in ?? dargestellt.

~~Es gibt verschiedene Im Folgenden werden einige~~ Grnde genannt, warum die Zitation von Software ebenfalls wichtig ist und auch, dass Standards der Zitation eingehalten werden. ~~Einige dieser Grnde werden im Folgenden genannt~~ (smith_software_2016):

Wichtige?

Primre?

Hauptgrnde?

~~Forschungsfelder verstehen:~~ Software ist ein Produkt der Forschung und wenn sie nicht zitiert wird, werden Lcken in der Aufzeichnung der Forschung ber den Fortschritt in diesem Forschungsfeld entstehen.

- **Anerkennung:** Akademische Forscher auf allen Ebenen, einschlielich Studenten, Postdocs, Dozenten und Mitarbeiter, sollten fr die Softwareprodukte, die sie entwickeln und zu denen sie beitragen, anerkannt werden, insbesondere wenn diese Produkte die Forschung anderer ermglichen oder frdern. Nicht-akademische Forscher sollten ebenfalls fr ihre Softwarearbeit anerkannt werden, obwohl die spezifischen Formen der Anerkennung sich von denen fr akademische Forscher unterscheiden.
- **Software entdecken:** Mithilfe von Zitaten kann die in einem Forschungsprodukt verwendete Software gefunden werden. Weitere Forscher knnen dann dieselbe Software fr andere Zwecke verwenden, was zu einer Anerkennung der Softwareverantwortlichen fhrt.
- **Reproduzierbarkeit:** Die Angabe der verwendeten Software ist fr die Reproduzierbarkeit notwendig, aber nicht ausreichend. Zustzliche Informationen wie Konfigurationen und Probleme auf der Plattform sind ebenfalls erforderlich.

Wie bereits erwhnt, werden Autoren in unterschiedlichen Quellen genannt. Einige dieser Quellen werden in ~~der dieser~~ Masterarbeit untersucht, wie beispielsweise das CFF- und BIBTEX-Format. Die Autoren in diesen Quellen werden vom jeweiligen Softwareprojekt angegeben. Falls an dem Projekt nicht nur Softwareentwickler beteiligt sind, sondern beispielsweise auch Grafikdesigner oder bersetzer, so sollten diese ebenfalls in den Quellen zitiert werden. Dies ist wichtig, da diese Personen

ebenfalls einen Beitrag zum Projekt geleistet haben und somit auch Anerkennung verdienen.

Es gibt ebenfalls ~~bereits~~-Projekte, welche sich für die halb-automatische Nennung von Autoren ohne Codebeitrag einsetzen. Ein Beispiel für ein solches Projekt ist „All Contributors“ ([bolam_recognize_2024](#)). Bei der Verwendung von „All Contributors“ werden die Mitwirkenden standardmäßig in der README-Datei angegeben, welche im Projekt enthalten ist. Außerdem wird eine `.all-contributorsrc`-Datei erstellt, welche im JSON-Format die Mitwirkenden und deren Beiträge enthält. Die Liste wird automatisch generiert und aktualisiert, sodass sie den Spezifikationen von „All Contributors“ entspricht. Über einen Befehl in [einem Issue oder einem Pull Request](#) [GitHub](#) kann ein neuer Mitwirkender hinzugefügt werden. Autoren, welche keinen Code beigetragen haben, stellen im weiteren Verlauf [der dieser](#) Masterarbeit ein Problem dar, ~~da diese~~, [Sie können](#) nicht mit den Autoren aus den Quellen abgeglichen werden [können](#), da sie keine Commits haben. Dies wird in ?? genauer erläutert. [In wird auf die Begriffe Pull Request, Issue und Commit eingegangen.](#)

[In der In dieser](#) Arbeit werden Pakete untersucht, bei denen der Quellcode öffentlich auf GitHub zugänglich ist und von einer Gemeinschaft an Entwicklern weiterentwickelt werden kann. Die Entwickler arbeiten dabei in der Regel ehrenamtlich und ohne Bezahlung an der Software, wobei einige der untersuchten Pakete auch von [Organisationen](#) [Unternehmen](#) veröffentlicht werden, wie beispielsweise die „google-auth-library-python“ von Google. [Diese wird primär von Google Mitarbeiter entwickelt und gepflegt. Wie bereits beschrieben sind Prinzipien und Gründe definiert, warum Software ebenfalls zitiert werden sollte.](#) Welche Autoren in den Quellen angegeben werden sollten, ist jedoch nicht so genau definiert, wie es beispielsweise im Bereich von wissenschaftlich-medizinischen Artikeln der Fall ist. In diesem Bereich hat das „International Committee of Medical Journal Editors“ Richtlinien für die Rolle von Autoren und Beitragenden in wissenschaftlichen Artikeln definiert ([icmje_icmje_2024](#)). Aus diesem Grund ist [unter anderem](#) die Menge der Autoren, welche in den Quellen angegeben werden, unterschiedlich und hängt von dem jeweiligen Projekt ab. Außerdem ist es möglich, dass ausschließlich Autoren in den Quellen angegeben werden, welche aktuell nicht mehr an dem Projekt beteiligt sind.

2.2 Autorenrolle und Anerkennung in Open-Source-Software

In wissenschaftlichen Arbeiten existieren bereits Analysen zur Rolle von Autoren in OSS. Ein Paper analysiert die Zuschreibung von Entwicklern in OSS ([young_which_2021](#)). Sie beantworten in dem Paper folgende Fragen:

- Wie unterscheiden sich die Modelle für die Anerkennung von Beiträgen?
- Wie viele Informationen gehen verloren bei einer Festlegung auf Repository-Änderungen als Modell für den Beitrag?
- Wie entwickeln sich die Beiträge zu OSS über die Zeit?
- Können wir Projekte auf der Grundlage von Beitragsmustern klassifizieren?

In dem Paper werden vier Modelle für die Anerkennung unterschieden. Im ersten Modell werden die Änderungen an einem Repository als Beitrag betrachtet. Dabei wird auf die Top-100-Benutzer in den Projekten geschaut, wie sie durch die GitHub-API ausgegeben werden. Eine API ist eine Programmierschnittstelle, welche eine Kommunikation verschiedener Softwareanwendungen ermöglicht. Dabei werden vordefinierte Funktionen und Daten durch die API zur Verfügung gestellt, welche von anderen Programmen aufgerufen werden können ([github_about_2022](#)). Das zweite Modell betrachtet die Autoren, welche automatisch über ein Tool identifiziert werden. In dem Paper wird hierbei das Programm *octohatrack* verwendet ([young_which_2021; mclaughlin_octohatrack_2020](#)). Beim dritten Modell werden die Autoren anhand der `.all-contributorsrc`-Datei von „All Contributors“ analysiert ([young_which_2021; bolam_recognize_2024](#)). Das vierte Modell betrachtet die Autoren, welche mittels ad-hoc-Methoden identifiziert werden. Diese können beispielsweise durch die Analyse von nicht standardisierten Quellen stammen, wie zum Beispiel Webseiten oder unstrukturierten Textdateien. Das Modell wird in dem Paper aufgrund der Komplexität nicht fokussiert betrachtet.

Die Autoren betrachten die Fragen und Modelle aus einer gehobenen Perspektive. Dabei werden die einzelnen Fragen mithilfe von verschiedenen generellen Metriken wie der Anzahl der Autoren, welche Commits erstellt haben oder als Autoren in „All Contributors“ genannt werden, beantwortet. Sie betrachten keine einzelnen Autoren oder Projekte, sondern analysieren die gesamte OSS-Landschaft primär auf Basis der Anzahl der Autoren. Es werden keine genaueren Analysen durchgeführt, ~~wie beispielsweise in dieser Masterarbeit, in der unter anderem betrachtet wird, ob die genannten Autoren noch aktiv an dem Projekt beteiligt sind~~. Außerdem wird von den Autoren nicht auf Daten eingegangen, welche aus weiteren Quellen wie dem CFF- oder BIBTEX-Format stammen.

2.3 Versionsverwaltung

Die Versionsverwaltung ist ein System, um zumeist Quellcode und dessen Änderungen zu verwalten. Der Code und getätigte Änderungen werden in einem Repository gespeichert. Dadurch ist die Versionsverwaltung eine Art Logbuch, in dem alle Änderungen festgehalten werden. Dabei wird zusätzlich zu der Änderung der Autor und der Zeitpunkt der Änderung festgehalten (**ponuthorai_version_2022**). Dies ermöglicht es ~~in der Masterarbeit~~ empirisch die Menge an Arbeit der einzelnen Autoren zu ermitteln.

Zusätzlich zum Code können in einem Repository andere Dateien, wie beispielsweise eine README, eine Lizenz, und Zitationsinformationen, beispielsweise in Form einer CFF-Datei, gespeichert werden. Eine README-Datei ist eine Datei, welche in der Regel Informationen über das Projekt enthält, beispielsweise wie es installiert und verwendet wird. Sie wird standardmäßig im Stammverzeichnis des Repositorys gespeichert und wird an dieser Stelle auch von Diensten wie GitHub dargestellt.

Es gibt zwei verschiedene Arten von Versionsverwaltungssystemen. Zum einen gibt es die zentralen Systeme, bei denen alle Änderungen zentral verwaltet werden, beispielsweise SVN. Zum anderen gibt es die verteilten Systeme, bei denen jeder Entwickler eine Kopie des gesamten Repositorys und dessen Vergangenheit hat (**ponuthorai_version_2022**). Ein solches System ist Git, welches sich mit einem Marktanteil von ungefähr 75 % gegenüber anderen Systemen durchgesetzt hat (**lindner_version_2024**). ~~Aus diesem Grund und weil Git-Repositorys in der Arbeit untersucht werden, wird im folgenden Im Folgenden wird~~ auf Git eingegangen. ~~. Dabei werden und~~ Begriffe erklärt, mit denen es möglich ist, die geleistete Arbeit von einzelnen Autoren innerhalb eines Repositorys zu untersuchen. Außerdem wird auf grundlegende Funktionen von Git eingegangen, da diese für die Arbeit relevant sind. ~~Der Aufbau der einzelnen Git-Komponenten ist in dargestellt. Dabei ist zu erkennen, dass Git in einen Git-Server und Git-Anwendungen aufgeteilt wurde.~~

~~Übersicht über die Git-Komponenten Die Git-Komponenten bestehen aus einem Git-Server, welcher das Repository hostet, und Git-Anwendungen, welche auf das Repository zugreifen (indirekt aus).~~

Bei der Benutzung von Git ~~ist ein Server nicht zwingend erforderlich, jedoch steigert dies die Komplexität der Verwaltung und ist komplizierter in der Handhabung. Der Git-Server kann ein Server verwendet werden. Dieser ermöglicht die eine einfache kollaborative Entwicklung von Code, da dieser er ständig erreichbar ist und zentral verwaltet wird~~ (**ponuthorai_version_2022**). Standardmäßig wird auf dem

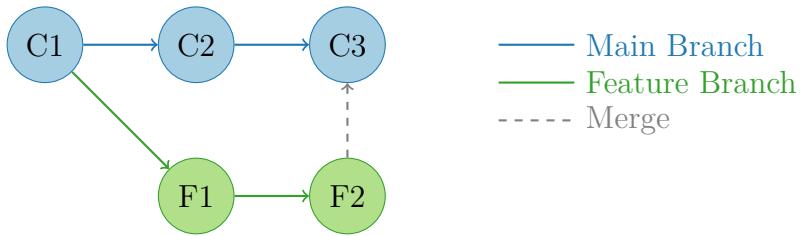
Git-Server die neueste Version des Repositorys gespeichert. Ein möglicher Anbieter, welcher Git-Server bereitstellt ist GitHub, auf welchen im späteren Verlauf weiter eingegangen wird. Git-Anwendungen sind Programme, welche mit dem lokal auf dem Computer befindlichen Repository interagieren (**ponuthorai_version_2022**). Diese Programme können ebenfalls auf entfernte Git-Repositorys zugreifen, wie beispielsweise Repositorys, welche auf GitHub gehostet werden und diese klonen. Anschließend arbeiten die Programme auf der lokalen Kopie und können die Änderungen, wenn nötig, auf das entfernte Repository übertragen.

In Repositorys werden verschiedene Arten von Statistiken gespeichert. Git verwaltet Revisionen als *Snapshot*. Anders als in anderen Systemen wird keine Serie von Änderungen gespeichert, sondern ein *Snapshot* der Änderungen gesamten Datei zu einem bestimmten Zeitpunkt erstellt (**ponuthorai_version_2022**). Dies wird ein Commit genannt. Mit einem Commit werden verschiedene Metainformationen gespeichert, beispielsweise eine Commit-Nachricht, der Autor und der Zeitpunkt der Änderungen.

Die Änderung wird dabei als zwei Zeitpunkte mit zwei Zeitpunkten angegeben. Zum einen wird der Zeitpunkt der Änderung des Autors angegeben, also jener Zeitpunkt, zu dem der Autor die Änderung vorgenommen und committet hat, dies wird in Git *author date* genannt. Zum anderen wird der Zeitpunkt des Einfügens des Commits in das Repository gespeichert, dies wird in Git *commit date* genannt. Der Commit kann von einer anderen Person, z. B. durch einen Projektverantwortlichen, mittels eines Pull Requests in das Repository übernommen worden sein. Durch dieses Verhältnis ist der *commit date* Zeitpunkt immer später oder gleich dem *author date* Zeitpunkt. Außerdem ist gewährleistet, dass beide verantwortlichen Verantwortlichen Anerkennung für die geleistete Arbeit erhalten (**chacon_pro_2024**).

Die Commit-Nachricht, sowie der Autor mit E-Mail-Adresse und Namen können in den Einstellungen von Git frei gewählt werden, müssen jedoch vorhanden sein, um einen Commit erstellen zu können. Mehrere Commits bilden die Commit-Historie bzw. die Vergangenheit eines Repositorys. Weitere Eigenschaften, welche sich aus dem Repository exportieren lassen, sind die Anzahl der eingefügten und gelöschten Zeilen. Zudem lässt sich die Anzahl der geänderten Dateien ermitteln. Diese Werte können für das gesamte Repository oder für einzelne Autoren ermittelt werden.

Ein Repository kann verschiedene Branches enthalten, muss jedoch mindestens einen Standardbranch enthalten. Ein Branch ist eine separate Entwicklungslinie, welche unabhängig von anderen Branches ist. Beim Erstellen von einem Branch wird der

**Abbildung 1: Git-Workflow**

In dieser Abbildung stellen Kreise Commits dar. Die unterschiedlichen Farben repräsentieren Commits einzelner Branches. Die gestrichelte Linie stellt eine Zusammenführung der Branches dar.

aktuelle Zustand des Branches, auf welchem der neue Branch erstellt wird, kopiert ([ponuthorai_version_2022](#)). Dadurch können Änderungen in dem neuen Branch durchgeführt werden, ohne dass diese Änderungen den ursprünglichen Branch beeinflussen. Diese Änderungen werden mittels Commits festgehalten. Unterschiedliche Branches können anschließend zusammengeführt werden, um die Änderungen aus einem Branch in einen anderen Branch zu übernehmen. [Ein Beispiel für einen solchen Workflow ist in ?? dargestellt.](#)

Die Statistiken der Repositorys können auf verschiedene Arten aufgearbeitet werden. Zum einen können einige direkt mittels Git-Befehlen ausgelesen werden ([chacon_git_2024](#)). Andere wiederum benötigen komplexere Abfragen, welche beispielsweise mittels Skripten oder speziellen Programmen ausgelesen werden können. Ein Beispiel für ein [Programm, welches Git-Statistiken aufarbeitet, solches Programm](#) ist [git-quick-stats](#) ([mestan_git-quick-stats_2024](#)). Außerdem bieten Onlinedienste zur Versionsverwaltung, wie GitHub, Statistiken über APIs an, welche jedoch im Umfang der Anfragen limitiert sind. [GitHub erlaubt beispielsweise für authentifizierte Nutzer 5.000 Anfragen pro Stunde](#) ([github_rate_2022](#)). Bei der Benutzung der API von GitHub zum Abfragen der Autoren eines Repositorys werden automatisch alle E-Mail-Adressen der Autoren in Git mit den E-Mail-Adressen, welche die Autoren in GitHub angegeben haben, abgeglichen ([github_rest-api-endpunkte_2022](#)). Dadurch werden die Autoren eindeutig zugeordnet und deren Commits addiert. Diese Werte werden ebenfalls in der Weboberfläche von GitHub angezeigt.

GitHub [ist eine Plattform, auf welcher Git-Repositorys gehostet werden können und dient somit als ein Git-Server. GitHub bietet neben der Bereitstellung eines Git-Servers](#) zusätzliche Funktionen an, welche über die Standardfunktionen von Git hinausgehen. Diese umfassen unter anderem die kollaborative Entwicklung von Code, [Automatisierung](#) mittels CI/CD, Sicherheitsaspekte, Pro-

jektmanagement, Team Administration und Client-Anwendungen zur Verwaltung von Repositorys (**ponuthorai_version_2022**). Aktuell benutzen GitHub über 100 Millionen Entwickler und mehr als 4 Millionen Organisationen. Insgesamt verwaltet die Plattform über 420 Millionen Repositorys **Außerdem ist GitHub in 90 % der Fortune 100 Unternehmen im Einsatz** (**github_about_2024**). Um die zusätzlichen Funktionen von GitHub bereitzustellen, werden sogenannte Issues, Pull Requests, geschützte Branches, Actions, Diskussionen und Wikis eingesetzt. GitHub-Issues sind eine Möglichkeit, um Probleme und Aufgaben zu verfolgen. Pull Requests dienen dazu, Änderungen in einem Branch eines Repositorys anzufragen und über diese zu informieren. In dem Pull Requests kann der Code überprüft und diskutiert werden.

2.4 Software-Verzeichnisse und Paketverwaltung

Im Gegensatz zur Versionsverwaltung verwaltet die Paketverwaltung keinen Code und dessen Änderungen, sondern fertige Softwarepakete, welche von Entwicklern erstellt und in einem Software-Verzeichnis abgelegt werden. Inhalt eines Pakets können **beispielsweise** standardisierter Code von Software-Modulen **sein** oder komplizierter Code **sein**. Zusätzlich werden in einem Paket Metadaten gespeichert. Diese Metadaten können **beispielsweise** eine Beschreibung, Version, Abhängigkeiten und Autoren des Pakets enthalten. Sie lassen sich aus dem Paket mithilfe des Paketverwaltungssystems auslesen oder über APIs des Software-Verzeichnisses abrufen. Außerdem übernimmt das Paketverwaltungssystem das Installieren und meistens auch das Aktualisieren und Deinstallieren von Paketen. Zusätzlich wird das System verwendet, um fehlende Abhängigkeiten von Paketen automatisch zu installieren (**spinellis_package_2012**).

In dieser Arbeit wird auf die Software-Verzeichnisse PyPI und CRAN eingegangen. PyPI ist das Verzeichnis für Python-Pakete und CRAN ist das Verzeichnis für R-Pakete. In PyPI sind aktuell mehr als 500.000 unterschiedliche Projekte mit über 5 Millionen Veröffentlichungen verfügbar (**python_software_foundation_pypi_2024**). CRAN listet aktuell mehr als 20.000 Pakete (**cran_team_comprehensive_2024**).

2.4.1 PyPI

PyPI hat zu Beginn des Jahres 2024 ein Python Enhancement Proposal (PEP) veröffentlicht, welches die Verifizierung von Daten auf PyPI beschreibt (**python_software_foundation_2024**). In dem PEP werden Änderungen an der API beschrieben, welche zum Hochladen

von Paketen genutzt wird, um sogenannte „Attestation objects“ zu unterstützen, in denen digitale Signaturen enthalten sind. Das Ziel ist es verifizierte Daten auf PyPI zu ermöglichen, sodass Anwender direkt erkennen können, ob die Daten vertrauenswürdig sind. Aktuell wird dieses PEP umgesetzt, wodurch es zu vielen Änderungen in der API und auch in der Weboberfläche von PyPI kommt.

Außerdem sind noch nicht alle Daten über die APIs erreichbar und es ist auch aktuell nicht über die API erkennbar, ob es sich um verifizierte oder nicht verifizierte Daten handelt. Ein Beispiel für verifizierte Daten sind Links, beispielsweise zu GitHub. Diese Links werden von PyPI als verifiziert angesehen, wenn der Upload des Pakets auf PyPI über eine GitHub-Action erfolgt. Zudem werden die Personen als verifiziert dargestellt, welche in PyPI als Owner oder Betreuer des Pakets eingetragen sind, sie haben somit einen Account bei PyPI. Der dargestellte Owner kann nicht über eine API abgefragt werden.

PyPI bietet verschiedene APIs und Quellen an, um die Daten der Pakete abzufragen. Im Folgenden werden auf einige der Zugriffsmöglichkeiten eingegangen, welche mit Ausnahme der BigQuery alle in dieser Arbeit verwendet werden.

JSON-API

Die JSON-API ist die bevorzugt zu verwendende API von PyPI und bietet die Möglichkeit, die Metadaten eines Pakets abzufragen (**python_software_foundation_warehouse_2024**). Diese API ist nicht in der Anzahl der Anfragen beschränkt. Daten der neuesten Version des Pakets werden von der API zurückgegeben. Die Werte in den Metadaten stammen aus den Daten, welche beim Hochladen auf PyPI angegeben wurden. Die Daten des ersten Uploads eines Releases werden dabei als Metadaten der Version gespeichert und bei weiteren Uploads dieser Version nicht überschrieben.

Inhalt der Metadaten sind beispielsweise die Autoren und Maintainer zuzüglich deren E-Mail-Adresse, die Beschreibung, Lizenz und Links zu unterschiedlichen Quellen, beispielsweise einem GitHub-Repository (**python_software_foundation_warehouse_2024**). Die Beschreibung kann den Text aus der README-Datei des Pakets auf GitHub enthalten. Es ist jedoch auch möglich, eine eigene Beschreibung für PyPI anzugeben, sodass sich die README-Datei in GitHub und die Beschreibung auf PyPI unterscheiden können. Die Metadaten werden von den Entwicklern des Pakets eingetragen. In ?? sind einige der Metadaten des Pakets *matplotlib* unter dem Punkt „Meta“ dargestellt. Weitere Daten sind unter dem Punkt „Projekt-Links“ unter den verifizierten Details dargestellt.

Verified details ✓*These details have been verified by PyPI***Projekt-Links** Bug tracker Source code**Owner**

Matplotlib

GitHub Statistics Repository Sterne: 20027 Forks: 7577 Open issues: 1204 Open PRs: 392**Betreuer**

ivanov



matthew.brett



mdboom2

Unverified details*These details have **not** been verified by PyPI***Projekt-Links** Documentation Donate Download Forum Homepage**Meta**

- **Lizenz:** Python Software Foundation License (License agreement for matplotlib versions 1.3.0 and later)
- =====
- =====...)

- **Autor:** [John D. Hunter](#), [Michael Droettboom](#) 
- **Benötigt:** Python >=3.9
- **Provides-Extra:** `dev`

Abbildung 2: PyPI Verifizierte und unverifizierte Daten

Die Abbildung stellt die verifizierten und unverifizierten Daten des Pakets *matplotlib* auf PyPI dar ([python_software_foundation_pypi_2024](#)).

Die Autoren und Maintainer aus den Metadaten müssen nicht den verifizierten Betreuern und Owner des Pakets auf PyPI entsprechen, da dies unterschiedliche Systeme sind. Zum einen sind es die Benutzer, welche Rechte auf PyPI haben, um das Paket dort anzupassen und zum anderen sind es die Personen, welche durch die Entwickler des Pakets angegeben werden. Die Autoren und Maintainer können jedoch ebenfalls verifiziert sein, wobei dies über die API noch nicht abgefragt werden kann, jedoch in der Weboberfläche bereits für einige Pakete dargestellt wird. Es gibt aktuell ~~wenig wenige~~ Pakete, bei denen die Autoren verifiziert sind, ~~ein~~. Ein Beispiel ist das Paket *hololinked*, ~~welches zum Zeitpunkt der Erstellung dieser Arbeit darüber verfügt das Stand 28.12.2024 solche Angaben enthält (venkatasubramanian_vaidyanathan_hololinked_2024)~~. *Matplotlib* unterstützt dies aktuell noch nicht, wie in ?? dargestellt ist. Die verifizierten Betreuer und Owner können aktuell nicht über die JSON-API abgerufen werden, sondern müssen über die XML-RPC-API abgefragt werden.

~~PyPI XML-RPC~~XML-RPC-API

Die PyPI XML-RPC-API ist eine veraltete API, welche jedoch noch genutzt werden kann, um einige Informationen zu den Paketen abzufragen. Es wird empfohlen, diese API nicht mehr zu verwenden und den RSS-Feed oder die JSON-API als mögliche Alternativen zu verwenden ([python_software_foundation_warehouse_2024](#)). ~~Dadureh~~ Außerdem ist die API in der Anzahl der möglichen Anfragen stark limitiert und auch die Abstände zwischen den Anfragen müssen ~~relativ~~ groß sein. PyPI macht keine genauen Angaben darüber, wie viele Anfragen in welchem Zeitraum möglich sind. Diese API ist jedoch die einzige Quelle, um die Betreuer eines Pakets abzufragen, ohne einen Web-Scraper einsetzen zu müssen. Web-Scraping bezeichnet das Extrahieren von Daten aus Webseiten, indem der HTML-Code der Webseite analysiert wird ([richardson_beautifulsoup4_2024](#)).

Die Betreuer, welche über die API ausgegeben werden, enthalten den Benutzernamen auf PyPI, ein Vollname wird hierbei nicht ausgegeben. Außerdem enthalten sie eine Rollenbezeichnung, welche entweder *Maintainer* oder *Owner* sein kann, welcher in der Oberfläche nicht dargestellt wird. Owner können alle Änderungen am PyPI Projekt vornehmen und Betreuer können neue Versionen des Pakets veröffentlichen ([ingram_deprecate_2023](#)). Die Benutzernamen für die Betreuer des Pakets *matplotlib* sind in ?? unter dem Punkt „Betreuer“ dargestellt. Aktuell stellt PyPI keine API bereit, um den Namen eines Benutzers abzufragen, sodass nur der Benutzername über eine API abgefragt werden kann ([python_software_foundation_add_2024](#)).

BigQuery

Ebenfalls bietet PyPI über Google BigQuery einen Datensatz an, in dem alle Pakete mit ihren Versionen und Metadaten enthalten sind ([python_software_foundation_warehouse](#)). BigQuery ist ein Dienst von Google, welcher auf der Infrastruktur der Google Cloud-Plattform ausgeführt wird. Der Dienst ist eine vollständig verwaltete Datenanalyseplattform, welche es ermöglicht, große Datenmengen zu analysieren und mittels SQL abgefragt werden kann ([google_bigquery_2024](#)). Es ist möglich, den kompletten Datensatz auf BigQuery in mehreren einzelnen CSV-Dateien herunterzuladen. Dabei kann ausgewählt werden, welche Daten heruntergeladen werden sollen.

Nicht alle Metadaten, welche über die JSON-API abgefragt werden können, stehen in der BigQuery zur Verfügung. Ebenfalls stehen nicht alle Daten der BigQuery in der JSON-API zur Verfügung. Die wichtigsten Daten stehen in beiden Quellen zur Verfügung. Beispielsweise sind Autoren, Maintainer und deren E-Mail-Adressen in beiden enthalten. Ebenso umfassen beide Quellen die Beschreibung, Version und Abhängigkeiten. Die jeweiligen Daten, wie die Autoren eines Pakets, sind in beiden Quellen identisch.

2.4.2 CRAN

CRAN selbst bietet keine API an, um die Metadaten der Pakete abzufragen. Jedoch gibt es das METACRAN-Projekt, welches eine Kollektion von kleinen Diensten für das CRAN-Repository zur Verfügung stellt. Eines dieser Dienste ist eine API. Über diese API ist es unter anderem möglich, eine Liste der meist heruntergeladenen Pakete in einem bestimmten Zeitraum abzufragen ([csardi_cranlogsapp_2024](#)). Ein weiterer Dienst, welcher durch das METACRAN-Projekt verwaltet wird, ist eine CouchDB, welche die Metadaten aller Pakete von CRAN bereitstellt. Dieser Dienst wird ebenfalls von dem R-Paket *pkgsearch* genutzt, welches dazu dient, in R die Metadaten anderer Pakete abzufragen. Die beiden Projekte sind keine CRAN Projekte, was bedeutet, dass sie nicht von den Entwicklern von CRAN betrieben werden. Eine CouchDB ist eine Apache-Datenbank, welche nativ eine HTTP/JSON-API bereitstellt ([the_apache_software_foundation_apache_2024](#)). Die Datenbank ist eine Kopie des CRAN-Repository und wird regelmäßig aktualisiert ([csardi_pkgsearch_2023](#)).

Um in CRAN ein Paket hinzufügen zu können, muss ein Formular ausgefüllt werden. Dabei muss der eigene Name sowie eine E-Mail-Adresse und das Paket angegeben werden. Anschließend werden die Daten von einem teilweise automatisierten Prozess

überprüft und nach einer erfolgreichen Überprüfung wird das Paket in CRAN veröffentlicht ([altmann_comprehensive_2024](#)). Aus diesem Grund gibt es in CRAN keine Unterscheidung zwischen verifizierten und nicht verifizierten Daten, da sie im Gegensatz zu PyPI manuell geprüft werden.

Die Metadaten der einzelnen Pakete, welche über die METACRAN-API erreichbar sind, sind dabei ähnlich zu denen in PyPI ([csardi_pkgsearch_2023](#)). Es werden beispielsweise die Autoren, Maintainer, eine Beschreibung, die Version und Abhängigkeiten über die API ausgegeben. Die Autoren werden in zwei unterschiedlichen Formaten ausgegeben. Zum einen wird ein Feld [Author](#) ausgegeben, welches die Autoren in einer Zeichenfolge enthält. Dieses Feld enthält den Namen der Autoren, sowie ggf. deren Rolle und ORCID iD. Zum anderen wird ein Feld [Authors@R](#) ausgegeben, welches die Autoren in einem R-Format ausgibt. Dieses Feld enthält ebenfalls die Werte des Feldes [Author](#), sowie ggf. eine E-Mail-Adresse des jeweiligen Autors. Im Gegensatz zu PyPI gibt es bei CRAN keine Benutzer für das Software-Verzeichnis. Außerdem lassen sich alle Metadaten über die gleiche API abfragen. Es konnten keine Informationen über mögliche Limitierungen in der Anzahl der Anfragen gefunden werden.

2.5 Zitierformate

In ~~wissenschaftlichen Arbeiten gibt es viele verschiedene Zitierweisen, die sich je nach Fachgebiet unterscheiden. In dieser Arbeit soll jedoch nicht auf die verschiedenen Zitierweisen eingegangen werden, sondern auf Zitierformate, welche für die Zitation verwendet werden können und~~ [diesem Abschnitt wird auf unterschiedliche Zitierformate eingegangen, welche](#) die Datenstruktur hinter ~~der einer~~ Zitation beschreiben. [Hierbei gibt es ebenfalls unterschiedliche Formate, wobei sich in dieser Arbeit](#) [In dieser Arbeit wird sich](#) auf das CFF und das BIBTEX-Format beschränkt wird. Das CFF ist ein Format, welches speziell für die Zitation von Software entwickelt wurde, weshalb es in dieser Arbeit besonders interessant ist. Das BIBTEX-Format wird dazu verwendet, um zumeist in Verbindung mit LATEX, Bibliographien zu erstellen und ist daher ebenfalls von Interesse, da es auch für Software verwendet werden kann.

2.5.1 Citation File Format

Das CFF ist ein Format, welches in der `CITATION.cff`-Datei gespeichert wird und in YAML 1.2 geschrieben wird. Das Format beschreibt die Zitation von Software und kann von Menschen und Maschinen gelesen werden. Es enthält Metadaten,

welche für die Zitation von Software benötigt werden. Außerdem wird es öffentlich auf GitHub verwaltet. ~~Insgesamt haben Stand 07.11.2024 Auf GitHub enthalten~~ 2.512 Repositorys eine `CITATION.cff`-Datei ([Stand 07.11.2024](#)). Softwareentwickler können das CFF in ihre Repositorys einbinden, um anderen die Zitation ihrer Software zu erleichtern und vorzugeben, wie die Software richtig zu zitieren ist ([druskat_citation_2021](#)).

Da die Datei von Menschen gelesen werden kann, kann diese manuell erstellt werden und in das Repository eingebunden werden. Die Spezifikationen für das CFF werden auf GitHub verwaltet und sind öffentlich einsehbar ([druskat_citation_2021](#)). Ebenfalls existieren Programme, welche das CFF verarbeiten können. Beispielsweise kann das Programm `cffinit` genutzt werden, um eine `CITATION.cff`-Datei zu erstellen, sodass der Prozess der Erstellung vereinfacht wird ([spaaks_cffinit_2023](#)). Ein weiteres Beispiel ist das Programm `cffconvert`, welches das CFF in verschiedene Formate umwandeln kann, wie z. B. `BIBTEX` oder `RIS`. Außerdem kann das Programm genutzt werden, um CFF-Dateien zu validieren ([spaaks_cffconvert_2021](#)).

Zusätzlich wird das CFF von unterschiedlichen Plattformen unterstützt, wie z. B. von GitHub. Erkennt GitHub eine `CITATION.cff`-Datei im Repository auf dem Standardbranch, wird sie automatisch auf der Repository-Startseite verlinkt und kann direkt im `BIBTEX`-Format kopiert werden ([druskat_citation_2021; github_about_2024-1](#)). Ebenfalls ist es möglich, die in der Datei eingetragenen Autoren in der APA-Zitierweise zu kopieren. Die Funktionen sind in ?? dargestellt. ~~Eine weitere unterstützte Plattform ist `Zotero`, ein Programm zur Literaturverwaltung. Falls der Benutzer das Browser-Plugin von `Zotero` installiert hat, liest dieses die Daten aus der Datei, welche in dem GitHub-Repository gespeichert ist und importiert die Daten in `Zotero`.~~

In dem CFF existieren verschiedene Felder, welche für die Zitation von Software relevant sind. ~~In dieser Masterarbeit wird auf einige dieser Felder eingegangen, welche für die spätere Auswertung relevant sind.~~ Das wichtigste Feld ist das `authors`-Feld, welches die Autoren der Software enthält und zwingend erforderlich ist. In diesem Feld können die Autoren als Liste angegeben werden. Ein Autor ist dabei entweder eine Person oder eine Entität. Eine Entität kann beispielsweise eine Organisation ~~oder ein Unternehmen~~ sein. Die Entität kann mit einem Namen mittels `name` angegeben werden ([druskat_citation_2021](#)). Sie kann ebenfalls eine OORCID iD und eine E-Mail-Adresse enthalten. Besonders wichtig für diese Arbeit ist die Referenz auf eine Person, da dies die einzige Information ist, welche aus Git extrahiert werden kann. Eine Person enthält ebenfalls die genannten Werte einer Entität und wird jedoch über den Vor- und Nachname separiert mittels

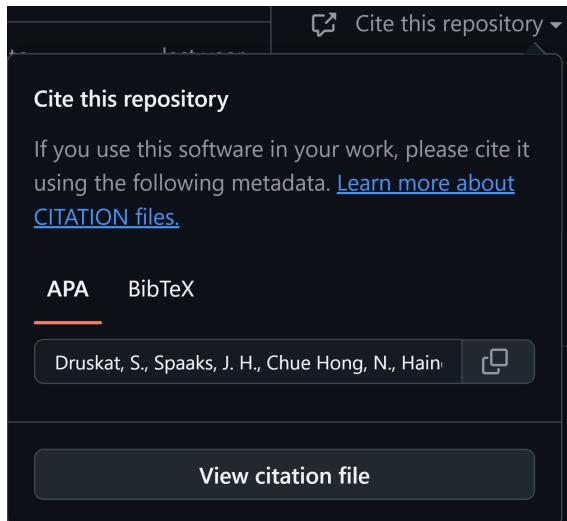


Abbildung 3: GitHub-Repository mit `CITATION.cff`-Datei

Die Abbildung stellt den Link auf die `CITATION.cff`-Datei dar, wie ihn GitHub aktuell darstellt. Außerdem ist die Möglichkeit sichtbar, die Datei im `BIBTeX`-Format und in der `APA`-Zitierweise zu kopieren (`druskat_citation_2021-1`).

given-names und family-names `given-names` und `family-names` angegeben. Dadurch ist es möglich, die Personen von den Entitäten zu unterscheiden.

Ein weiteres Feld „welches in der Masterarbeit relevant ist, ist das preferred-citation“ ist das `preferred-citation`-Feld. Mit diesem Feld ist es möglich, die Anerkennung für die Arbeit auf eine andere Arbeit zu übertragen (`druskat_citation_2021`). Ein Beispiel hierfür ist ein Paper über die Software, welches bevorzugt zitiert werden soll, anstelle der eigentlichen Software. Hierbei können ebenfalls Personen und Entitäten angegeben werden. Das Format entspricht dabei dem, welches bereits beschrieben worden ist. Durch die Angabe einer `preferred-citation` kann das Prinzip der Wichtigkeit vernachlässigt werden. Auf dieses Verhalten wird in der Diskussion konkreter eingegangen.

Weitere in `der dieser` Arbeit verwendete Felder sind `type, year, month, date-released, date-published, doi, collection-doi und identifiers`. Das Feld `type` ist zwingend erforderlich, hat jedoch als Standardwert „software“, sodass dies nicht angegeben werden muss. Es gibt an, ob es sich um eine Software oder einen Datensatz handelt. Dabei sind lediglich die Werte „software“ oder „dataset“ erlaubt. Das Feld `date-released` gibt an, wann die Software oder der Datensatz veröffentlicht wurde. Das Feld `doi` kann einen Digitalen Objektbezeichner (DOI) der Software oder des Datensatzes enthalten. Mittels `identifiers` können weitere Identifikatoren angeführt werden,

wie z. B. eine DOI oder eine URL, wobei die `identifiers` mit einer Beschreibung erweitert werden können. Die beschriebenen Felder können zusätzlich alle unter dem Feld `preferred-citation` angegeben werden, um eine andere Arbeit zu referenzieren.

Die Felder `year`, `month` und `date-published` können zusätzlich zu dem Feld `date-released` unter dem Feld `preferred-citation` angeführt werden, um das Jahr und das Datum der Veröffentlichung anzugeben. Außerdem können weitere Typen mittels `type` aufgeführt werden, wie beispielsweise „thesis“ oder „manual“, sodass diese Arbeiten ebenfalls referenziert werden können. Zusätzlich kann das Feld `collection-doi` verwendet werden, um auf eine Sammlung von Arbeiten zu verweisen, die die Arbeit enthält. Ein Beispiel einer `CITATION.cff`-Datei ist in ?? dargestellt. Dabei wurde sich auf die beschriebenen und notwendigen Felder beschränkt.

2.5.2 BIBTEX

BIBTEX ist eine Software, welche zur Erstellung von Literaturangaben und -verzeichnissen in LATEX-Dokumenten verwendet wird. Außerdem existiert mit BIBTEX ein gleichnamiges Format, welches in der `CITATION.bib`-Datei gespeichert wird und auf keinem anderen Format basiert. BIBTEX ist ein weit verbreiteter Standard und wird von vielen Autoren in der Wissenschaft verwendet. Auf GitHub [haben enthalten](#) 2.144 Repositorys eine `CITATION.bib`-Datei [enthalten](#) (Stand 07.11.2024). Das Format beschreibt die Zitation von Literatur und kann von Menschen und Maschinen gelesen werden. Es beschränkt sich dabei nicht auf eine spezielle Art von Literatur, sondern kann für viele unterschiedliche Arten von Literatur verwendet werden. Beispielsweise können Bücher und Masterarbeiten in BIBTEX zitiert werden ([patashnik_bibtexing_1988](#)). Ein offizieller Literaturtyp für Software existiert nicht. In der Datei können mehrere Einträge vorhanden sein, wobei jeder Eintrag eine Literaturangabe darstellt.

BIBTEX-Dateien können von Menschen manuell erstellt und in das Repository eingebunden werden. Außerdem existieren viele Literaturverwaltungsprogramme wie Zotero, welche BIBTEX-Dateien erstellen und verarbeiten können ([zotero_zotero_2024](#)). Ebenfalls ist die Integration in andere Plattformen möglich, wie z. B. in GitHub. Hier wird die `CITATION.bib`-Datei auf der Repository-Startseite verlinkt, sie lässt sich jedoch im Gegensatz zu dem CFF nicht direkt kopieren oder in andere Formate umwandeln ([github_about_2024-1](#)).

```
1 cff-version: 1.2.0
2 title: "CompAuthorsBetweenDS"
3 message: "If you use this software, please cite it using the metadata from
   ↵ this file."
4 type: software
5 authors:
6   - given-names: "Kevin"
7     family-names: "Jahrens"
8     email: "k.jahrens@stud.hs-wismar.de"
9   - name: "Hochschule Wismar"
10 date-released: "2025-01-05"
11 doi: "10.1000/182"
12 identifiers:
13   - description: "The DOI of the work."
14     type: "doi"
15     value: "10.1000/182"
16   - description: "The versioned DOI for version 1.0.0 of the work"
17     type: "doi"
18     value: "10.1000/182"
19 preferred-citation:
20   title: "Identifikation und Vergleich von Autorenangaben zu Software
   ↵ zwischen verschiedenen Datenquellen"
21   type: thesis
22   year: 2025
23   month: 01
24   date-published: "2025-01-05"
25   date-released: "2025-01-05"
26   doi: "10.1000/182"
27   collection-doi: "10.1000/182"
28   identifiers:
29     - description: "The DOI of the work."
30       type: "doi"
31       value: "10.1000/182"
32     - description: "The versioned DOI for version 1.0.0 of the work"
33       type: "doi"
34       value: "10.1000/182"
35   authors:
36     - given-names: "Kevin"
37       family-names: "Jahrens"
38       email: "k.jahrens@stud.hs-wismar.de"
39     - name: "Hochschule Wismar"
```

Listing 1: Beispiel einer CITATION.cff -Datei

In dem Listing ist die CFF-Datei dieser Arbeit dargestellt. Dabei wurden primär Felder angegeben, welche in der Masterarbeit verwendet werden.

```

1 @masterthesis{Jahrens:2025,
2   author = {Jahrens, Kevin},
3   title = {Identifikation und Vergleich von Autorenangaben zu Software
4   ↳ zwischen verschiedenen Datenquellen},
5   school = {Hochschule Wismar},
6   month = {1},
7   year = {2025}
7 }

```

Listing 2: Beispiel einer CITATION.bib -Datei

In dem Listing ist die BIBTEX-Datei dieser Arbeit dargestellt. Dabei wurden primär Felder angegeben, welche in der Masterarbeit verwendet werden.

In dem BIBTEX-Format existieren verschiedene Felder, welche für die Zitation von Literatur relevant sind. Einige Felder sind dabei zwingend erforderlich und andere nicht. Welche Felder zwingend erforderlich sind, hängt vom Literaturtyp ab jeweiligen

Ergibt sich aus der Sache
Klingt auch bisschen so als würde der Literaturtyp optional bleiben

Einige Felder sind dabei zwingend erforderlich und andere optional bleiben. Welche Felder zwingend erforderlich sind, hängt vom Literaturtyp ab, während andere optional bleiben. Zudem sind die verfügbaren Felder ebenfalls vom Literaturtyp abhängig (patashnik_bibtexing_1988). In dieser Arbeit wird auf einige dieser Felder eingegangen, welche für die Masterarbeit relevant sind. Das wichtigste Feld ist das author-Feld, welches die Autoren der Literatur enthält und zwingend erforderlich ist. Die Vor- und Nachnamen der Autoren werden mit einem Komma separiert und mehrere Autoren werden über ein „and“ getrennt. Weitere Felder, welche für die Masterarbeit verwendet werden, sind year und month, welche das Jahr und den Monat der Veröffentlichung angeben. Ein Beispiel einer CITATION.bib -Datei ist in ?? dargestellt. Es ist zu erkennen, dass in dem BIBTEX-Eintrag Informationen fehlen, welche in dem CFF-Eintrag vorhanden waren. Dies liegt daran, dass in dem BIBTEX-Eintrag nur eine Referenz auf die Masterarbeit möglich ist und nicht auf die entwickelte Software.

2.6 Named Entity Recognition

Die Named entity recognition (NER) beschreibt den Prozess der automatischen Erkennung und Klasseneinteilung von Substantiven, sogenannten Entitäten im Text (mohit_named_2014). Typische Entitäten sind Personen, Orte oder Organisationen. Im folgenden Beispiel sind die Entitäten markiert, dabei ist nach erfolgreicher NER die Klasse der Entität bekannt, welche hinter der Entität in Klammern gesetzt wurde.

PyTorch (Organisation) is currently maintained by **Soumith Chintala** (Person), **Gregory Chanan** (Person), **Dmytro Dzhulgakov** (Person), **Edward Yang** (Person), and **Nikita Shulga** (Person) with major contributions coming

*from **hundreds** (Digit) of talented individuals in various forms and means.*

NER wird in vielen Bereichen eingesetzt, wie z. B. Informationsextraktion, Frage-Antwort-Systeme und der maschinellen Übersetzung, um eine Wort-für-Wort-Übersetzung zu vermeiden. Die NER wurde in vielen Sprachen untersucht, darunter auch Arabisch und Hebräisch (**mohit_named_2014**). In dem Fall ~~der~~dieser Masterarbeit kann das System verwendet werden, um die Beschreibungen der Pakete zu verarbeiten und die Namen der Entwickler zu extrahieren.

In der NER gibt es verschiedene Herausforderungen. Diese sind zum einen das Erkennen des Entitäten-Anfangs und Endes und zum anderen die Erkennung der korrekten Entitätenklasse. Ebenfalls gibt es ähnlich wie in vielen anderen Bereichen der natürlichen Sprachverarbeitung das Problem der Polysemie, also dass ein Wort mehrere Bedeutungen haben kann (**mohit_named_2014**). Ein Beispiel hierfür ist das Wort „Bank“, welches sowohl eine Sitzgelegenheit als auch ein Finanzinstitut sein kann. ~~Dieses Problem ist besonders schwerwiegend in der , auf welche im nächsten Abschnitt eingegangen wird.~~

Für die NER gibt es verschiedene Programme und trainierte Modelle, ~~welche verwendet werden können~~. Ein Programm im Python-Umfeld für die NER ist *spaCy*, welches eine Open-Source-Bibliothek für die Verarbeitung natürlicher Sprache ist. Die Bibliothek ist nicht beschränkt auf die NER, sondern bietet auch Funktionalitäten wie Tagging, Parsing und Text Klassifikation. Die Bibliothek hat zum Anspruch, fertige Modelle für den industriellen Einsatz bereitzustellen, welche sowohl schnell sind, als auch eine hohe Genauigkeit haben (**honnibal_spacy_2024**). Für die unterschiedliche Genauigkeit sowie für unterschiedliche Sprachen existieren verschiedene Modelle, ~~welche verwendet werden können~~ die sich ebenfalls in der Geschwindigkeit |

Unnötig?

2.7 Named Entity Disambiguation

Die Named entity disambiguation (NED) ist ein automatischer Prozess, bei dem ein Name einer Entität einer gegebenen Datenmenge zugeordnet wird (**cucerzan_large-scale_2007**; **yamada_global_2022**). Die Entitäten können beispielsweise durch die NER extrahiert worden sein. Dabei kann es dazu kommen, dass eine Entität mehrfach extrahiert wird und mehrfach in der Datenmenge vorhanden ist, jedoch mit anderen Bedeutungen. In der natürlichen Sprachverarbeitung ist dies das Problem der Polysemie, ~~auf das bereits eingegangen wurde~~. Es ist aber auch möglich, dass Personen

mit dem gleichen Namen, sogenannte Namensvetter extrahiert werden, welche unterschieden werden müssen. Dies beschreibt die Author name disambiguation, welche konkret individuelle Personen disambiguierter und ein Teil der NED ist. Diese Probleme kann die NED mittels Modellen lösen, die die Entitäten anhand von Kontexten disambiguieren. Beispiele für erkannte Entitäten sind (**cucerzan_large-scale_2007**):

- George W. Bush (George W. Bush)
- George Bush (George W. Bush)
- Bush (George W. Bush)
- Reggie Bush (Reggie Bush)
- Bush (Reggie Bush)
- Bush (Rock band)

In dem Beispiel ist die Entität dargestellt, gefolgt von der Entität in der Datenmenge, welche in Klammern steht. Hierbei ist auffällig, dass Nennungen von „Bush“ mehrfach vorkommen und durch den Kontext, in dem sie stehen, durch die NED unterschieden werden müssen.

NED wird in vielen Bereichen eingesetzt, wie z. B. Textanalysen, semantische Suche und der Gruppierung von Software-Nennungen in wissenschaftlichen Arbeiten (**cucerzan_large-scale_2007; yamada_global_2022; schindler_somesci_2021**).

Zusammen oder getrennt? Finde im Internet mehr zusammen
In der dieser Masterarbeit kann die NED verwendet werden, um Autoren aus verschiedenen Quellen miteinander abzugleichen.

Ähnlich zu der NER gibt es für die NED verschiedene Modelle. Viele Modelle sind jedoch spezifisch auf einzelne Aufgabenbereiche trainiert. Allgemeine Modelle sind primär für die Disambiguierung von Entitäten in Texten trainiert, welche in der dieser Arbeit nicht immer vorhanden sind. Ein Beispiel ist ein Modell, welches auf BERT basiert und von Yamada u. a., welches Wörter, sowohl als auch Entitäten als Tokens erhält und diese mit Entitäten aus einem Text disambiguert (**yamada_global_2022**).

2.8 Unscharfe Suche

Die unscharfe Suche ist ein Verfahren, um ähnliche Zeichenfolgen zu finden, die sich in ihrer Schreibweise unterscheiden (**hall_approximate_1980**). Dieses Verfahren hat viele Anwendungsgebiete in der Informatik. Ein Beispiel ist das Finden eines Personennamens in einem Index. Falls der Name exakt in dem Index vorhanden ist,

ist die Suche trivial. Falls der Name unterschiedlich geschrieben ist, beispielsweise durch Abkürzungen oder Tippfehler, schlägt die triviale Suche fehl. Die unscharfe Suche kann in diesem Fall helfen, den Namen dennoch zu finden. Ein weiteres Anwendungsgebiet ist eine allgemeine Suche, beispielsweise von Produkten in einem Online-Shop. Hierbei müssen auch Tippfehler berücksichtigt werden. **Dadurch ist die Suche nach einer Zeichenfolge, welche nahezu korrekt ist, ein häufiges Problem in der Informatik.**

Die unscharfe Suche ~~basiert~~ kann auf der Levenshtein-Distanz basieren (`levenshtein_binary_1965`). Als Ergebnis der unscharfen Suche wird in vielen Implementierungen die Distanz zwischen zwei Zeichenfolgen in Prozent angegeben. Die Levenshtein-Distanz verwendet drei Arten von einzelzeichenbasierten Editieroperationen, um die Distanz zwischen zwei Zeichenfolgen zu berechnen.

1. Einfügen eines Zeichens zur Zeichenfolge (`Suhe` → `Suche`)
2. Löschen eines Zeichens aus der Zeichenfolge (`Suchee` → `Suche`)
3. Ersetzen eines Zeichens in der Zeichenfolge (`Siche` → `Suche`)

Zusätzlich zu der Levenshtein-Distanz existiert eine Erweiterung, die Damerau-Levenshtein-Distanz (**damerau_technique_1964**). Diese erweitert die Levenshtein-Distanz um eine vierte Editieroperation. Mittels dieser vier Operationen werden ungefähr 80 % der menschlichen Tippfehler abgedeckt (**damerau_technique_1964**).

4. Vertauschen von zwei benachbarten Zeichen in der Zeichenfolge (`Suhce` → `Suche`)

Eine Herausforderung bei der unscharfen Suche ist die Laufzeit. Sie ist ungefähr proportional zum Produkt der beiden Zeichenfolgenlängen, wodurch eine unscharfe Suche nach langen Zeichenfolgen unpraktisch wird. Eine weitere Herausforderung ist das Finden des richtigen Prozentsatzes, um die unscharfe Suche zu verwenden. Ein zu hoher Prozentsatz führt dazu, dass die Suche zu viele Ergebnisse zurückgibt, während ein zu niedriger Prozentsatz dazu führt, dass die Suche zu wenige Ergebnisse zurückgibt.

Für die unscharfe Suche gibt es viele Implementierungen, die auf verschiedenen Algorithmen basieren. Ein Programm, welches in Python implementiert ist, ist *TheFuzz* (**bachmann_thefuzz_2023**). Das Programm basiert auf *RapidFuzz*, welches die Levenshtein-Distanz in Python und C++ implementiert (**bachmann_rapidfuzz_2023**).

3 Methodik

In diesem Kapitel wird beschrieben, wie die Daten der einzelnen Quellen beschafft, abgeglichen und anschließend ausgewertet werden. Die Datenbeschaffung wird in ??, der Abgleich in ?? und die Auswertung in ?? beschrieben.

Die Datenbeschaffung wurde in die einzelnen Quellen untergliedert. Einige Methoden zur Datenbeschaffung sind dabei ähnlich, worauf im konkreten Fall eingegangen wird.

Der Abgleich findet jeweils zwischen Git und einer weiteren Quelle statt. Es existiert kein Abgleich zwischen einzelnen Quellen wie den Daten aus PyPI und der Beschreibung. Der Abgleich wird in jeder Datenbeschaffung außer der von Git automatisch durchgeführt. Die Ergebnisse des Abgleichs werden in einer CSV-Datei gespeichert. Die Datei wird nur erstellt, falls mindestens ein Eintrag enthalten ist. Allgemeine Daten, beispielsweise ob die Quelle valide ist, werden ebenfalls in einer CSV-Datei gespeichert. Falls in einer Quelle keine Daten vorhanden sind, wird keine CSV-Datei für diese erstellt.

Sämtliche Ergebnisse werden, falls verfügbar, zu verschiedenen Zeitpunkten in denen Änderungen an der Quelle vorgenommen wurden ermittelt und gespeichert. Falls aus der Quelle verschiedene Zeitpunkte der Änderungen vorliegen, wird der Abgleich mit Git jeweils mit der neuesten Version durchgeführt und mit der Version, welche zu dem Zeitpunkt der Änderung in der Quelle vorhanden war. Dadurch entstehen für ein Paket mehrere Dateien, welche unterschiedliche Werte enthalten. Es entsteht die in ?? dargestellte Ordnerstruktur, welche die Ergebnisse der Datenbeschaffung darstellt.

Der Prozess findet für jedes zu untersuchende Paket statt und ist in ?? visualisiert. Die Pakete stammen aus den Software-Verzeichnissen PyPI und CRAN. Die Ergebnisse des Prozesses werden in fünf Ordner gespeichert jeweils für PyPI, CRAN, CFF, PyPI CFF und CRAN CFF. Diese Ordner bilden die fünf Top-100-Listen, die in ?? näher beschrieben sind. In jedem Ordner sind jeweils Unterordner für die einzelnen Pakete. Diese Daten werden für die anschließende Aus-

```

/ ..... Wurzelverzeichnis
  |----- 20210819_161452-0400_bib_authors.csv BIBTEX Autoren abgeglichen mit
  |----- den Git-Werten zu diesem Zeitpunkt
  |----- 20210819_161452-0400_bib_authors_new.csv BIBTEX Autoren abgeglichen
  |----- mit den Git-Werten zum neuesten Zeitpunkt
  |----- 20210819_161452-0400_git_contributors.csv Git-Autoren bis zu diesem
  |----- Zeitpunkt
  |----- 20221010_124020-0400_readme_authors.csv Autoren in der Beschreibung
  |----- abgeglichen mit den Git-Werten zu diesem Zeitpunkt
  |----- 20221010_124020-0400_readme_authors_new.csv ..... Autoren in der
  |----- Beschreibung abgeglichen mit den Git-Werten zum neuesten Zeitpunkt
  |----- 20221010_124020-0400_git_contributors.csv Git-Autoren bis zu diesem
  |----- Zeitpunkt
  |----- bib.csv ..... Allgemeine Informationen zur BIBTEX-Datei
  |----- readme.csv ..... Allgemeine Informationen zur Beschreibung
  |----- git_contributors.csv ..... Git-Autoren zum neuesten Zeitpunkt
  |----- pypi_maintainers.csv ..... PyPI Maintainer
  |----- python_authors.csv ..... In Python angegebene Autoren

```

Abbildung 4: Ergebnisse der Datenbeschaffung

Die Abbildung stellt einen Ausschnitt der CSV-Dateien der Datenbeschaffung dar.

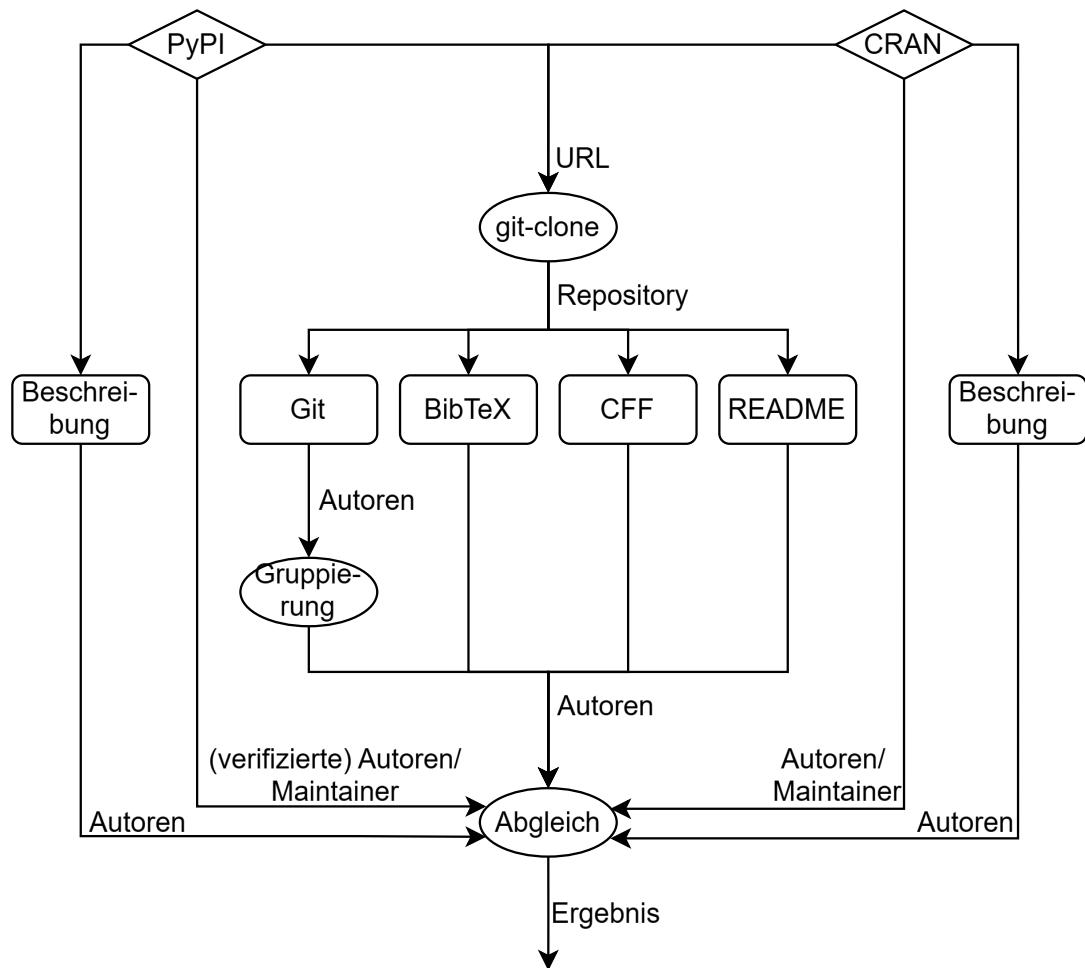
wertung in ?? verwendet. In diesem Abschnitt werden die Ergebnisse der Abgleiche analysiert und zusammengefasst, um Aussagen über alle Pakete hinweg zu treffen. Die Datenbeschaffung, der Abgleich und die Auswertung sind in Python programmiert und verwenden OSS auf welche in den jeweiligen Abschnitten eingegangen wird. Der entwickelte Quellcode ist in einem Git-Repository verfügbar ([jahreens_t20240710-softwareauthors-kj_2025](#)). Über alle Abschnitte hinweg wird Pandas in der Version 2.2.2 verwendet, um die Tabellen zu erstellen und zu verarbeiten ([the_pandas_development_team_pandas_2024](#)).

3.1 Datenbeschaffung

~~In diesem Abschnitt wird beschrieben, wie die Top-100-Listen der Pakete erstellt wurden, welche in der Masterarbeit untersucht werden. Außerdem wird erläutert, wie das Skript zur Datenbeschaffung aus den einzelnen Quellen aufgebaut ist. Hierbei~~

In diesem Abschnitt wird *tqdm* in der Version 4.66.5 verwendet, um den Fortschritt der Datenbeschaffung anzuzeigen ([costa-luis_tqdm_2024](#)). Der Abschnitt ist in die einzelnen zu untersuchenden Quellen untergliedert.

Insgesamt werden in ~~der dieser~~ Masterarbeit 5 Top-100-Listen untersucht. Die erste Liste enthält die Top-100 PyPI Pakete, welche am häufigsten im August 2024 herun-

**Abbildung 5:** Visualisierter Prozess

In der Abbildung ist der Prozess visualisiert. Die runden Knoten stellen Prozesse dar. Rautenförmige Knoten repräsentieren Software-Verzeichnisse und rechteckige Knoten repräsentieren die einzelnen zu untersuchenden Quellen, in welchen Autoren genannt werden können. Kanten stellen den Informationsfluss dar, wobei die Kantenbeschriftung die jeweiligen Informationen repräsentiert.

tergeladen wurden (**kemenade_top-pypi-packages_2024**). Die zweite Liste enthält die Top-100 meist heruntergeladenen CRAN Pakete im Zeitraum von 02.08.2024 bis 31.08.2024 (**csardi_cranlogsapp_2024**). [Beide Listen wurden direkt aus den genannten Quellen entnommen, wobei sie auf 100 Einträge limitiert worden sind.](#)

Zusätzlich zu den beiden Listen, welche ausschließlich PyPI und CRAN Pakete mit ihren Namen auf der jeweiligen Plattform enthalten, werden noch drei weitere Listen erstellt und anschließend untersucht. Hierbei wurde eine Liste vom Zweitgutachter [dieser Arbeit](#) untersucht, welche die Links zu allen Repositorys auf GitHub enthält, welche eine CFF-Datei enthalten. In der Liste sind 20.870 unterschiedliche Links enthalten, wobei einige der Links aufgrund von Umbenennungen auf das gleiche Repository zeigen. Die Liste wurde am 19.10.2024 um die Anzahl der Sterne auf GitHub erweitert, um über diese Metrik anschließend die Top-100 zu definieren. Dafür wurde ein Skript entwickelt, welches einmalig die gesamte Liste analysiert und für jedes Paket die GitHub-API verwendet, um die Anzahl der Sterne zu erhalten.

Anschließend wurden aus der Liste die Top-100-Pakete ausgewählt und mittels ecosyste.ms um die Paketverwaltung und den Namen des Pakets in dem jeweiligen Verzeichnis erweitert (**nesbitt_ecosystems_2024**). Ecosyste.ms ist eine Plattform, welche öffentlich zugängliche APIs bereitstellt, mit denen es möglich ist, Software-Metadaten abzufragen. Beispielsweise bietet eine API von ecosyste.ms die Möglichkeit, eine GitHub-URL zu übergeben und anschließend eine Liste von Metadaten zu erhalten, wie beispielsweise das Software-Verzeichnis, in welchem das Paket verwaltet wird. Die Liste besteht dabei teilweise aus vielen Einträgen, da beispielsweise ebenfalls Einträge für *nightly* Versionen enthalten sind. Um jeweils nur einen Eintrag pro Paket zu erhalten, wird nur das Paket betrachtet, welches die meisten Downloads hat. Es wurde davon ausgegangen, dass dies die Hauptversion des Pakets ist. Für jedes Paket wird zusätzlich eine *purl* ausgegeben, mit der es möglich ist, die doppelten Einträge in der Liste aller GitHub-Repositorys mit CFF zu identifizieren und anschließend nur jeweils einmal zu speichern (**ombredanne_purl-spec_2024; nesbitt_ecosystems_2024**).

Die Liste enthält anschließend 100 Pakete, welche nicht alle analysiert werden können, da sie beispielsweise nicht in CRAN oder PyPI verwaltet werden. In solchen Fällen kann es sich beispielsweise um Dokumentation handeln, welche in GitHub verwaltet wird. Diese Pakete werden in der Datenbeschaffung nicht betrachtet.

Konkret sind in der Liste null CRAN und 46 PyPI Pakete enthalten. Die restlichen 54 Pakete sind in keiner der beiden zu untersuchenden Paketverwaltungen enthalten

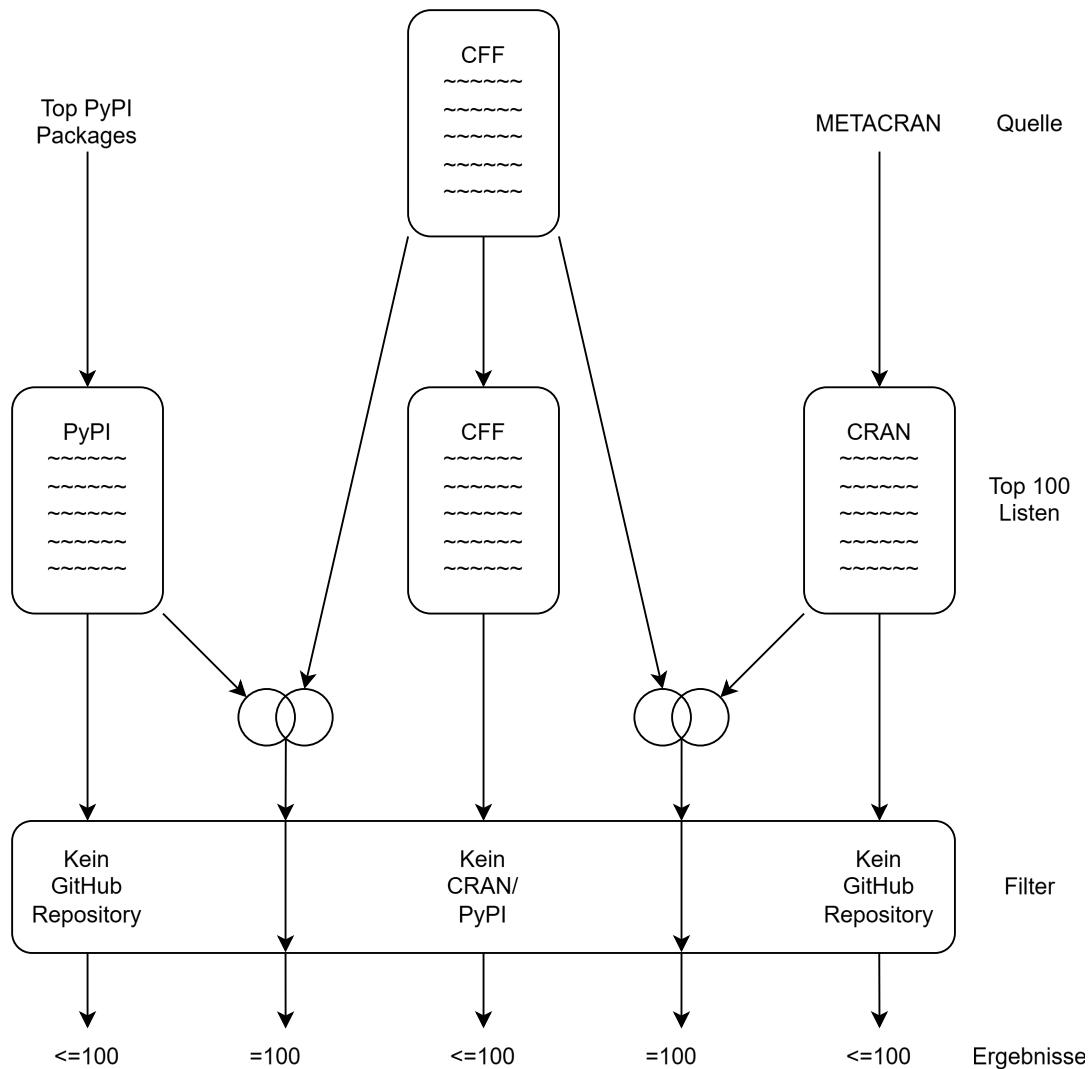
und können dadurch nicht untersucht werden. Außerdem können von den 46 PyPI Paketen zwei Pakete zusätzlich nicht untersucht werden, da diese nicht zurück auf das GitHub-Repository verweisen. Aus diesem Grund werden zwei weitere Listen erstellt. Die Listen enthalten die Top-100 PyPI und CRAN Pakete, welche eine CFF-Datei in GitHub haben. Um zu ermitteln, ob und in welchem Verzeichnis die Software liegt, wurde erneut ecosyste.ms verwendet. Anschließend wird für jedes Paket geprüft, ob ein Link aus dem PyPI oder CRAN Software-Verzeichnis zurück zu dem GitHub-Repository vorhanden ist. Zudem werden erneut die doppelten Einträge nicht betrachtet. Dadurch wird gewährleistet, dass in den Listen jeweils 100 Pakete enthalten sind, welche anschließend alle analysiert werden können. Der Prozess, wie die Listen erstellt wurden und wie viele Ergebnisse aus der Datenbeschaffung zu erwarten sind, ist in ?? dargestellt.

Aus den Quellen ??, ??, ?? und ?? können zeitliche Informationen extrahiert werden, da diese in Git verwaltet werden. Aus diesem Grund werden die Daten jeweils zu der Änderung der Quelle gespeichert. Dabei ist die maximale Anzahl der Änderungen in die Vergangenheit für die Beschreibung auf 50 beschränkt, um die Laufzeit des Skripts zu begrenzen. Die anderen Quellen haben keine Beschränkung, da diese nicht so häufig aktualisiert werden. Außerdem werden die Quellen jeweils für PyPI und CRAN betrachtet, da sie allgemein für alle Pakete unabhängig von der Paketverwaltung verfügbar sind.

In ?? ist es nicht möglich die Änderungen in der Zeit über einen Zeitraum zu betrachten. In der ?? Quelle ist es teilweise mit BigQuery möglich die Änderungszeitpunkte zu erhalten, jedoch ist dies mit finanziellen Kosten verbunden und erfordert eine andere Vorgehensweise als bei den anderen zeitlichen Daten, da diese nicht direkt aus Git extrahiert werden können. Die beiden Quellen werden aus diesem Grund nur in der neuesten Version betrachtet und enthalten keine Änderungshistorie.

3.1.1 Git

Die Git-Daten sind die grundlegenden Daten, welche für die weiteren Schritte benötigt werden. Sämtliche anderen Quellen werden mit den Git-Daten über den in ?? beschriebenen Prozess abgeglichen. Zu Beginn muss das Repository von GitHub geclont heruntergeladen werden, um die Daten lokal verarbeiten zu können. Dabei kommt die OSS *GitPython* in der Version 3.1.43 zum Einsatz. Die Bibliothek bietet eine Schnittstelle, um Git-Befehle in Python zu verwenden (**thiel_github_2024**). Beim Aufruf der Funktion wurde kein Branch spezifiziert, weshalb der Branch aus-

**Abbildung 6:** Erstellung der Listen

Die Abbildung zeigt den Prozess, wie die Top-100-Listen erstellt wurden. Dabei werden oben die Quellen für die Listen angegeben. Anschließend werden die Listen erstellt und **durch die Datenbeschaffung untersucht**, wobei die Pakete gefiltert werden. Am Schluss wird die mögliche Anzahl der Ergebnisse angegeben.

gewählt wird, auf welchen aktuell der *HEAD* vom remote Repository zeigt. Dieser Branch ist in der Regel der Standardbranch, auf welchem die Analyse durchgeführt werden soll. Für das Clonen muss außerdem der Link zum GitHub-Repository angegeben werden, welcher aus der PyPI oder CRAN Quelle stammt. Auf diese Quellen wird in den nächsten Abschnitten eingegangen.

Die Auswertung des Repositorys wird mit *git-quick-stats* in der Version 2.3.0 durchgeführt ([arzzen_git-quick-stats_2021](#)). *Git-quick-stats* bietet einfache und effiziente Möglichkeiten, um verschiedene Statistiken in einem Git-Repository zu ermitteln. Das Tool wird in dem Python-Skript mit dem Befehl *git-quick-stats -T* aufgerufen, um detaillierte Statistiken zu erhalten. Ausgegeben wird eine Liste aller Autoren, welche in dem Repository Änderungen vorgenommen haben. Diese Liste enthält unter anderem den Namen, die E-Mail, die Anzahl der Einfügungen, Löschungen, geänderten Zeilen, Dateien, Commits, sowie den ersten und letzten Commit, welche alle am Ende des Prozesses gespeichert werden. Die Anzahl der Commits beinhaltet keine Merge-Commits. Dieses Verhalten von *git-quick-stats* ist erwünscht, da diese nicht relevant für die Analyse sind. In dieser Masterarbeit werden ausschließlich die Autoren betrachtet, welche Änderungen an dem Code vorgenommen haben und nicht jene, die ausschließlich Änderungen von anderen in das Repository übernommen haben.

A

Die E-Mail-Adresse wird anschließend in Kleinbuchstaben umgewandelt, um die Daten zu vereinheitlichen. Zusätzlich wird eine Gruppierung auf der E-Mail durchgeführt und die anderen Werte summiert, mit Ausnahme des ersten und letzten Commits, bei denen der älteste und neueste Commit ausgewählt werden. Der Name ~~des Autors wird dabei ebenfalls summiert, was in dem Kontext bedeutet, dass Namen aneinander wird hintereinander gehängt werden, sodass beispielsweise bei einer Namensänderung von „Max M.“ zu „Max Mustermann“ der Name „Max M. Max Mustermann“ entsteht.~~ Das Gruppieren ist notwendig, da die Autoren den Namen und die E-Mail-Adressen in Git eigenständig festlegen können, wie in ?? beschrieben wurde. Durch dieses Vorgehen wird gewährleistet, dass zumindest keine E-Mail-Adressen doppelt vorhanden sind, falls ein Autor unterschiedliche Schreibweisen für seinen Namen verwendet. Die gruppierten Daten werden nach der Anzahl der Commits sortiert in der Datei `git_contributors.csv` gespeichert. In ?? sind die Felder der Datei aufgelistet.

Außerdem bietet das Tool die Möglichkeit, mit der gesetzten Umgebungsvariablen `_GIT_UNTIL=`, alle Änderungen nur bis zu einem bestimmten Zeitpunkt zu betrachten. Diese Funktion wird verwendet, um die Änderungen bis zur Aktualis-

Feld	Beschreibung
<i>name</i>	Name des Autors
<i>email</i>	E-Mail des Autors
<i>insertions</i>	Hinzugefügte Zeilen des Autors
<i>deletions</i>	Gelöschte Zeilen des Autors
<i>lines_changed</i>	Geänderte Zeilen des Autors
<i>files</i>	Geänderte Dateien des Autors
<i>commits</i>	Anzahl der Commits des Autors
<i>first_commit</i>	Zeitpunkt des ersten Commits des Autors
<i>last_commit</i>	Zeitpunkt des letzten Commits des Autors

Tabelle 1: Felder der `git_contributors.csv`-Datei

Die Tabelle zeigt die Felder der `git_contributors.csv` oder der `TIMESTAMP_git_contributors.csv`-Datei. Für jeden Autor der betrachteten Software werden die dargestellten Werte gespeichert.

sierung einer Quelle zu betrachten. Die Daten werden beispielsweise in der Datei `20210819_161452-0400_git_contributors.csv` gespeichert, wobei der erste Teil des Dateinamens den konkreten Tag und Uhrzeit mit zugehöriger Zeitzone angibt. Dabei hat die Datei die gleichen Felder wie die `git_contributors.csv`-Datei, jedoch nur für die Änderungen bis zu dem angegebenen Zeitpunkt. Für die Verarbeitung der Zeiten in unterschiedlichen Zeitzonen wird das Modul `pytz` in der Version 2024.2 verwendet (**bishop_pytz_2024**).

3.1.2 PyPI

Zu Beginn des Prozesses wird die Datei eingelesen, in denen die Top-100-Pakete von PyPI aufgelistet sind. In dieser Datei ist ausschließlich der Name des Pakets und die Anzahl der Downloads auf PyPI enthalten. Für die Beschaffung der Repository Daten von GitHub wird jedoch der Link zu der Versionsverwaltung benötigt. Dieser wird mithilfe von `aiohttp` in der Version 3.10.3 von der JSON-API von PyPI abgefragt (**svetlov_aiohttp_2024**). Dabei muss ein Paket nicht zwangsweise ein GitHub-Repository haben, weshalb die Daten nicht immer vorhanden sind. In diesem Fall wird das Paket übersprungen und nicht weiter betrachtet. In der PyPI Liste betrifft dies beispielsweise sieben der 100 Pakete.

Anschließend erfolgt die Verarbeitung der weiteren Daten, die über die JSON-API abgefragt wurden. Dabei lassen sich die nicht verifizierten Autoren und Maintainer mit Name und E-Mail des Pakets extrahieren. Welche Werte dort enthalten sind, können die Paketentwickler selbst entscheiden. Beispielsweise geben einige Paket-

entwickler mehrere Autoren mit Komma separiert an. Diese werden aufgeteilt und als einzelne Autoren gespeichert. Ebenfalls werden die E-Mail-Adressen anhand des Kommas separiert und jeweils mit dem Namen des Autors verbunden. Dies geschieht so, dass der 1.-erste Name mit der 1.-ersten E-Mail verbunden wird, der 2.-zweite Name mit der 2.-zweiten E-Mail und so weiter. Falls ein Autor keine E-Mail angegeben hat, wird der Name ohne E-Mail gespeichert und falls nur eine E-Mail angegeben ist, wird diese ohne einen Namen gespeichert.

Ebenfalls geben einige Paketentwickler keine E-Mail im E-Mail-Feld an, sondern nur den Namen und schreiben in das Namensfeld, zusätzlich eine E-Mail-Adresse, ~~auch der andere Weg ist möglich~~. Es gibt verschiedene weitere Sonderfälle, die nicht alle aufgezählt sind, da sie gleich behandelt werden. In diesen Sonderfällen wird keine weitere Betrachtung vorgenommen und die Daten so gespeichert, wie sie von der API zurückgegeben wurden. Anschließend erfolgt der Abgleich der Daten mit den zuvor beschafften Git-Daten anhand des in ?? beschriebenen Prozess, wobei die möglichen Sonderfälle berücksichtigt sind. Zudem werden die Daten in den Dateien `python_authors.csv` und `python_maintainers.csv` gespeichert. Der Inhalt umfasst den Namen und die E-Mail des Autors oder Maintainers, sowie die Daten aus dem Abgleich mit Git. Die Felder der Dateien sind in ?? aufgelistet.

Zusätzlich zu den Informationen der Autoren und Maintainer wird die Beschreibung des Pakets von der API zurückgegeben. Einige Pakete haben in der Beschreibung ebenfalls Autoren angegeben, welche zusätzlich verarbeitet werden. Die Beschreibung wird als unstrukturierter Text zurückgegeben. Dieser Text wird mittels der NER Bibliothek *spaCy* in der Version 3.7.6 verarbeitet, um die Autoren zu extrahieren (**honnibal_spacy_2024**). Dabei wird das Programm so verwendet, dass nur die Entitäten *PERSON* extrahiert werden. Als Modell wurde das `en_core_web_trf` verwendet, welches auf Englisch trainiert ist und eine höhere Genauigkeit aufweist als das `en_core_web_sm` Modell. Das `en_core_web_sm` Modell hat einen F-F1 Score von 0.846, wohingegen das `en_core_web_trf` Modell einen F-F1 Score von 0.902 aufweist. Der F1-Score ist ein Maß für die Genauigkeit eines Modells, dabei liegt der Score zwischen 0 und 1, wobei 1 die höchste Genauigkeit darstellt (sasaki_truth_2007). Durch die Verwendung des genaueren Modells von spaCy ist die Laufzeit erhöht. In Experimenten hat sich jedoch gezeigt, dass das kleinere Modell keine guten Ergebnisse für die Beschreibungen liefert, da ~~sehr~~ viele Entitäten fälschlicherweise als Autoren erkannt werden. Die extrahierten Autoren werden ebenfalls mit den Git-Daten abgeglichen und in der Datei `description_authors.csv` gespeichert. Der Inhalt umfasst den Namen des Autors, sowie die Daten aus dem

Abgleich mit Git. Die Felder der Datei sind in ?? aufgelistet.

Außerdem erfolgt die Verarbeitung der verifizierten Maintainer, welche auf PyPI angegeben sind, da diese unter Umständen nicht den Autoren entsprechen, welche durch die API ausgegeben werden. Da PyPI diese nicht über die JSON-API bereitstellt, erfolgt der Abruf der Daten über die XML-RPC-API. Die API liefert den Benutzernamen sowie die Rolle des Autors. Die Rolle kann dabei *Maintainer* oder *Owner* sein. *Owner* ist hierbei nicht der *Owner*, welcher in ?? unter *Owner* aufgeführt ist, sondern in diesem konkreten Fall sind alle drei Betreuer des Pakets als *Owner* angegeben. Der Benutzer „Matplotlib„, welcher unter *Owner* aufgeführt ist, wird nicht über die API zurückgegeben. Dieser wird jedoch auch nicht benötigt, da unter *Owner* immer eine Organisation angeführt wird und in [der dieser](#) Masterarbeit nur Personen betrachtet werden. Im Allgemeinen wird aus der Dokumentation der XML-RPC-API nicht deutlich, welche Daten zurückgegeben werden.

Anschließend werden alle verifizierten Autoren, welche durch die API zurückgegeben wurden, analysiert, unabhängig von der angegebenen Rolle. Da die Autoren ausschließlich mit dem Benutzernamen zurückgegeben werden, wird ein Web-Scraper benötigt, um den vollständigen Namen des Autors zu erhalten. Falls ein Autor einen Namen angegeben hat, wird dieser auf der Profilseite des Benutzers dargestellt. Um diesen zu erhalten, wird eine Anfrage mittels *aiohttp* an die Profilseite des Benutzers gestellt. Anschließend wird mittels *BeautifulSoup* in der Version 4.12.3 der Name aus dem HTML extrahiert ([richardson_beautifulsoup4_2024](#)). Die Daten werden, nachdem sie mit den Git-Daten abgeglichen wurden, in der Datei *pypi_maintainers.csv* gespeichert. Inhalt ist dabei der Benutzername und der Name des Autors, sowohl als auch die Daten aus dem Abgleich mit Git. Die Felder der Datei sind in ?? aufgelistet.

3.1.3 CRAN

Ähnlich wie bei PyPI werden zu Beginn die Daten der Top-100-Pakete aus der zuvor beschafften Datei eingelesen. In dieser Datei sind ebenfalls ausschließlich der Name des Pakets und die Anzahl der Downloads auf CRAN enthalten. Aus diesem Grund wird zu Beginn eine Anfrage mittels *aiohttp* an die von METACRAN bereitgestellte API gestellt, um die Metadaten des Pakets zu erhalten. In den Metadaten kann der Link eines GitHub-Repositorys enthalten sein. Falls kein Link zu einem GitHub-Repository vorhanden ist, wird das Paket nicht weiter betrachtet. In der CRAN Liste betrifft dies beispielsweise sechs der 100 Pakete.

Feld	Beschreibung
<i>rank</i>	Rang des Autors sortiert nach der Anzahl der Commits
<i>insertions</i>	Hinzugefügte Zeilen des Autors
<i>deletions</i>	Gelöschte Zeilen des Autors
<i>lines_changed</i>	Geänderte Zeilen des Autors
<i>files</i>	Geänderte Dateien des Autors
<i>commits</i>	Anzahl der Commits des Autors
<i>first_commit</i>	Zeitpunkt des ersten Commits des Autors
<i>last_commit</i>	Zeitpunkt des letzten Commits des Autors
<i>score</i>	Zu wie viel Prozent der Abgleich mit Git übereinstimmt

Tabelle 2: Felder, welche durch den Abgleich mit Git entstehen

Die Tabelle zeigt die Felder, welche durch den Abgleich mit Git entstehen. Für jeden Autor oder Maintainer der betrachteten Software werden die dargestellten Werte gespeichert, falls der Autor abgeglichen werden konnte. Falls der Autor nicht abgeglichen werden konnte, sind die Felder leer und der Score ist 0.

Feld	Beschreibung
<i>name</i>	Name des Autors
<i>email</i>	E-Mail des Autors

Tabelle 3: Felder der `python_authors.csv`, `python_maintainers.csv` und `cran_maintainers.csv`-Datei

Die Tabelle zeigt die Felder der `python\authors.csv`, `python\maintainers.csv` und `cran\maintainers.csv`-Datei. Für jeden Autor oder Maintainer der betrachteten Software wird der Name und die E-Mail gespeichert, falls diese angegeben wurde. Außerdem werden weitere Felder durch den Abgleich mit Git in der Datei gespeichert, welche in ?? aufgelistet sind.

Feld	Beschreibung
<i>name</i>	Name des Autors

Tabelle 4: Felder der `description_authors.csv`, `TIMESTAMP_readme_authors(_new).csv` und `TIMESTAMP_bib_authors(_new).csv`-Datei

Die Tabelle zeigt die Felder der `description\authors.csv`, `TIMESTAMP\readme\authors(_new).csv` und `TIMESTAMP\bib\authors(_new).csv`-Datei. Für jeden Autor der betrachteten Software wird der Name gespeichert, welcher durch die NER ermittelt wurde. Außerdem werden weitere Felder durch den Abgleich mit Git in der Datei gespeichert, welche in ?? aufgelistet sind.

Feld	Beschreibung
<i>login</i>	Benutzername des Autors
<i>name</i>	Name des Autors

Tabelle 5: Felder der `pypi_maintainers.csv`-Datei

Die Tabelle zeigt die Felder der `pypi_maintainers.csv`-Datei. Für jeden Maintainer der betrachteten Software wird der Benutzername angegeben. Der Name kann leer sein, da er nicht angegeben werden muss. Außerdem werden weitere Felder durch den Abgleich mit Git in der Datei gespeichert, welche in ?? aufgelistet sind.

Anschließend werden die weiteren Daten verarbeitet, welche von der API bereitgestellt werden. Beispielsweise wird das Feld `Authors@R` bereitgestellt. Dieses Feld beinhaltet die Autoren mit dem Namen, der E-Mail und einer ORCID ID des Pakets in einer in R formatierten Zeichenfolge. Dabei müssen nicht zwingend alle Informationen vorhanden sein. Des Weiteren haben Autoren eine Rolle zugeordnet. Die Rolle ist nicht fest definiert und kann von den Autoren frei gewählt werden. Es existieren allerdings Standards, welche eingehalten werden sollten. Einem Autor können mehrere Rollen zugewiesen sein. Im R Journal wurden folgende Rollen definiert ([hornik_who_2011](#)):

aut(Autor): Autor (aut):

Vollständige Autoren, die wesentliche Beiträge zu dem Paket geleistet haben und in der Zitation des Pakets auftauchen sollten. com(Complier):

Compiler (com):

Personen, die Code (möglicherweise in anderen Sprachen) gesammelt, aber keine weiteren wesentlichen Beiträge zum Paket geleistet haben. ctb(Mitwirkender):

Mitwirkender (ctb):

Autoren, die kleinere Beiträge geleistet haben (z. B. Code-Patches usw.), die aber nicht in der Auflistung der Autoren auftauchen sollten. cph(Urheberrechtsinhaber):

Urheberrechtsinhaber (cph):

Personen, die das Urheberrecht an dem Paket besitzen. cre(Maintainer): Paket Maintainer ths(Betreuer der Thesis):

Maintainer (cre):

Paket Maintainer

Betreuer der Thesis (ths):

Betreuer der Thesis, wenn das Paket Teil einer Thesis ist. trl(Übersetzer):

Übersetzer (trl):

Übersetzer nach R, wenn der R-Code eine Übersetzung aus einer anderen Sprache (typischerweise S) ist.

Die Daten aus der Zeichenfolge werden mit der Software *rpy2* in der Version 3.5.16 verarbeitet (**gautier_rpy2_2024**). *Rpy2* ist eine Software, welche es ermöglicht R-Code in Python auszuführen. Die Software wird mit dem R-Befehl `eval(parse(text = '{cran_author}'))` ausgeführt, wobei `cran_author` die R-Zeichenfolge der Autoren ist. Anschließend werden die Autoren, welche die Rolle `aut` zugeordnet haben, nach dem Abgleich mit den Git-Daten in der Datei `cran_authors.csv` gespeichert. In der Datei wird, falls vorhanden, der Name, die E-Mail-Adresse und die ORCID iD der Autoren gespeichert. Die Felder der Datei sind in ?? aufgelistet.

Falls das Feld **Authors@R** keine Zeichenfolge enthält oder beim Verarbeiten der Zeichenfolge ein Fehler auftritt, wird das durch die API zurückgegebene Feld **Author** verarbeitet. In dem Feld stehen ebenfalls die Autoren des Pakets, jedoch ohne die zusätzliche Information der E-Mail. Außerdem ist das Feld nicht in R formatiert, sodass die Zeichenfolge mittels eines regulären Ausdrucks verarbeitet wird. Die Folge ist dabei unterschiedlich in verschiedenen Paketen, sodass keine allgemeine Regel definiert werden kann. Einige Paketautoren geben keine ORCID iD an, sodass die gesamte Zeichenfolge anders aufgebaut ist. Andere Autoren wiederum geben keine Rollenbezeichnungen an, sodass die Zeichenfolge beispielsweise nur den Namen enthält. Falls eine Rolle angegeben ist, werden nur jene Autoren verarbeitet, welche als Rolle `aut` zugeordnet haben. Andernfalls werden alle Autoren verarbeitet. Die Autoren werden in der Datei `cran_authors.csv` nachdem sie mit den Daten aus Git abgeglichen wurden gespeichert. Dabei wird nur der Name gespeichert ohne zusätzliche Informationen wie der ORCID iD, da dieses Vorgehen die Verarbeitung bei den unterschiedlichen aufgebauten Zeichenfolgen vereinfacht hat. Zudem wird diese Methode nur verwendet, falls die Verarbeitung der Zeichenfolge **Authors@R** fehlschlägt.

Ein weiteres Feld, welches von der API zurückgegeben und verarbeitet wird, ist das Feld **Maintainer**. Dieses Feld ist nicht in R formatiert, sondern eine einfache Zeichenfolge. Die Zeichenfolge enthält weniger Informationen als die **Author** Zeichenfolge. In ihr ist lediglich der Name und die E-Mail-Adresse des Maintainers enthalten. Außerdem ist immer nur ein Maintainer angegeben, welcher verarbeitet wird. Die Daten werden nach dem Abgleich mit den Daten aus Git in der Datei `cran_maintainers.csv` gespeichert. Ausgegeben wird in der Datei der

Feld	Beschreibung
<i>name</i>	Name des Autors
<i>email</i>	E-Mail des Autors
<i>orcid</i>	ORCID iD des Autors

Tabelle 6: Felder der `cran_authors.csv`, `TIMESTAMP_cff_authors(_new).csv` und `TIMESTAMP_cff_preferred_citation_authors(_new).csv`-Datei

Die Tabelle zeigt die Felder der `cran\authors.csv`, `TIMESTAMP\cff\authors(_new).csv` und `TIMESTAMP\cff\preferred\citation\authors(_new).csv`-Datei. Für jeden Autor der betrachteten Software wird der Name, die E-Mail und die ORCID iD angegeben, falls diese vorhanden sind. Außerdem werden weitere Felder durch den Abgleich mit Git in der Datei gespeichert, welche in ?? aufgelistet sind.

Name und die E-Mail-Adresse des Maintainers. Die Felder der Dateien sind in ?? aufgelistet.

Das letzte Feld, welches von der API verarbeitet wird, ist das Feld ~~Description~~Description. In diesem Feld ist die Beschreibung des Pakets enthalten. Dabei wird die Verarbeitung wie in ?? für die Beschreibung von PyPI durchgeführt.

3.1.4 Beschreibung

In diesem Abschnitt wird anders als zuvor keine API angefragt, sondern auf den bereits heruntergeladenen Git-Daten die Analyse durchgeführt. Untersucht wird die Beschreibung, welche in der `README.md`-Datei des Repositorys enthalten ist. Diese Datei wird in Git verwaltet, wodurch Änderungen ~~in der Zeit über einen Zeitraum~~ betrachtet werden können. Aus diesem Grund wird das Programm *Git-Python* verwendet, um die letzten 50 Änderungen der Datei auf dem Standard-branch zu erhalten. Anschließend wird für jede Änderung die Beschreibung wie zuvor mit *spaCy* analysiert. Für jeden Zeitpunkt wird eine CSV-Datei mit dem Namen `TIMESTAMP_readme_authors_new.csv` erstellt, wobei *TIMESTAMP* durch den konkreten Zeitpunkt der Änderung der Datei ersetzt wird. Hierbei wird der *commit date* Zeitpunkt aus Git verwendet. Die erstellte Datei enthält die extrahierten Namen aus der Beschreibung, welche mit den neuesten Git-Daten abgeglichen wurden. Die Felder der Datei sind in ?? aufgelistet. Der neueste Zeitpunkt entspricht hierbei dem Zeitpunkt, an dem die Repositorys heruntergeladen wurden, also dem Tag an dem der Prozess durchlaufen wurde.

Zusätzlich wird zu jedem Änderungszeitpunkt der Git-Prozess aus ?? bis zu diesem Zeitpunkt durchlaufen. Dies dient dazu, eine Liste aller Autoren zu erhalten, welche

Feld	Beschreibung
<i>committed_datetime</i>	Zeitpunkt des <i>commit date</i>
<i>authored_datetime</i>	Zeitpunkt des <i>author date</i>

Tabelle 7: Felder der `readme.csv`-Datei

Die Tabelle zeigt die Felder der `readme.csv`-Datei. Für jede Änderung der Beschreibung werden die dargestellten Werte gespeichert.

bis zu diesem Zeitpunkt Änderungen vorgenommen haben. Die Liste wird in der Datei `TIMESTAMP_git_contributors.csv` gespeichert, wobei *TIMESTAMP* durch den konkreten Zeitpunkt ersetzt wird. Der Inhalt ist der gleiche wie in ??, jedoch nur bis zu dem Zeitpunkt, an dem die `README.md`-Datei jeweils geändert wurde. Diese Daten werden ebenfalls mit den Autoren, welche aus der Beschreibung extrahiert wurden, abgeglichen und in der Datei `TIMESTAMP_readme_authors.csv` gespeichert. Die Felder der Datei sind ebenfalls in ?? aufgelistet. Der Inhalt ist hierbei der Name des Autors und weitere Informationen, welche durch den Abgleich ermittelt wurden.

Im Prozess werden ebenfalls allgemeine Informationen zur Beschreibung beschafft und anschließend in der Datei `readme.csv` gespeichert. Die Datei beinhaltet die Zeitpunkte *commit date* und *author date* für jede Änderung der Beschreibung. Die Felder der Datei sind in ?? dargestellt.

3.1.5 Citation File Format

In diesem Unterabschnitt wird ~~ähnlich wie im vorherigen Unterabschnitt~~ die CFF-Datei untersucht. Der Unterabschnitt ist dabei aufgeteilt in die allgemeinen Daten, welche für CFF und ~~preferred-citation~~preferred-citation gelten und die spezifischen Daten, welche ausschließlich für die ~~preferred-citation~~preferred-citation gelten. Die beiden Daten werden jeweils in einer `CITATION.cff`-Datei gespeichert, wie es in den Grundlagen erläutert wurde. Es wird ebenfalls ein zeitlicher Verlauf der Daten betrachtet, um die Änderungen ~~in der Zeit über einen Zeitraum~~ zu analysieren. Dabei wird der Verlauf nicht auf 50 Zeitpunkte beschränkt, sondern in der gesamten Historie betrachtet. Da die CFF-Datei in YAML geschrieben wird, wird die Bibliothek `pyyaml` in der Version 6.0.2 verwendet, um die Datei zu lesen ([simonov_pyyaml_2024](#)).

In der CFF können für die eigentliche Zitation und für die ~~preferred-citation~~preferred-citation Autoren angegeben werden. Diese werden jeweils getrennt extrahiert, anschließend

mit den neuesten Git-Daten abgeglichen und in separaten Dateien gespeichert. Dabei werden nur die Autoren betrachtet, welche Personen sind und keine Entitäten wie beispielsweise Organisationen. Die Autoren werden in der Datei `TIMESTAMP_cff_authors_new.csv` gespeichert, wobei `TIMESTAMP` durch den Zeitpunkt der Änderung ersetzt wird. Die Autoren aus der `preferred-citation`^{[preferred-citation](#)} werden in der Datei `TIMESTAMP_cff_preferred_citation_authors_new.csv` gespeichert. In beiden Dateien sind der Name, die E-Mail, die ORCID iD und die Git-Daten, welche durch den Abgleich ermittelt wurden, enthalten. Die Felder der Dateien sind in ?? aufgelistet.

Zusätzlich zu den neuesten Daten wird zu jedem Änderungszeitpunkt der Git-Prozess aus ?? bis zu diesem Zeitpunkt durchlaufen. Die Liste der Autoren wird in der Datei `TIMESTAMP_git_contributors.csv` gespeichert. Die Liste enthält die Autoren, welche bis zu diesem Zeitpunkt Änderungen vorgenommen haben. Außerdem wird die Liste der Autoren, welche aus der CFF extrahiert wurden, mit den Git-Autoren bis zu diesem Zeitpunkt abgeglichen. Dadurch entstehen die beiden Dateien `TIMESTAMP_cff_authors.csv` und `TIMESTAMP_cff_preferred_citation_authors.csv`. Enthalten sind die gleichen Informationen wie in den „new“ Dateien.

Zudem können in beiden Fällen ähnlich wie bei der Beschreibung allgemeine Daten extrahiert werden. Hierbei wird wie in der Beschreibung eine weitere CSV-Datei erstellt, welche die allgemeinen Daten enthält. In dem Fall der CFF werden zwei allgemeine Dateien erstellt, einmal für die CFF und einmal für die `preferred-citation`^{[preferred-citation](#)}. Die Dateien heißen `cff.csv` und `cff_preferred_citation.csv`. Die Felder der `cff.csv`-Datei sind in ?? aufgelistet und die der `cff_preferred_citation.csv`-Datei in ?. In den beiden Dateien werden ebenfalls die Zeitenpunkte *commit date* und *author date* gespeichert.

In den Dateien wird ebenfalls gespeichert, ob die CFF-Datei valide ist. Hierbei wird das Programm `cffconvert` in der Version 2.0.0 verwendet, um die Validität zu überprüfen (**spaaks_cffconvert_2021**). Außerdem benötigt `cffconvert` die Programme `jsonschema` und `pykwalify`, welche in ~~der~~^{[dieser](#)} Masterarbeit in der Version 4.23.0 und 1.8.0 verwendet werden (**berman_jsonschema_2024; grokzen_pykwalify_2020**). Diese werden von `cffconvert` benötigt, um die CFF-Datei zu validieren. Zusätzlich wird gespeichert, ob die CFF-Datei mit dem Tool `cffinit` erstellt wurde (**spaaks_cffinit_2023**). Dies ist möglich, da `cffinit` in der CFF-Datei mehrere Kommentare einfügt, welche auf das Tool hinweisen. Es wird davon ausgegangen, dass `cffinit` benutzt wurde, falls der Kommentar „This CITATION.cff file was generated with cffinit.“ in der CFF-

Datei vorhanden ist. Dies ist jedoch kein sicherer Indikator, da der Kommentar auch manuell hinzugefügt und vor allem entfernt werden kann. Diese Werte werden jeweils in den allgemeinen Dateien gespeichert. Dies führt zu einer doppelten Speicherung, allerdings ist dadurch in beiden Dateien separat zu erkennen, ob die CFF-Datei, aus denen die Daten extrahiert wurden, valide ist und ob *cffinit* benutzt wurde.

Alle weiteren Informationen werden direkt aus der CFF-Datei extrahiert und in die eigene CSV Datenstruktur übertragen. Falls ein Wert dabei nicht in der CFF-Datei vorhanden ist, wird der Eintrag in der CSV-Datei leer gelassen. Im Folgenden werden die Daten, welche direkt extrahiert werden, beschrieben. Eine extrahierte Information, welche in beiden Dateien, also der `cff.csv` und `cff_preferred_citation.csv` gespeichert wird, ist die DOI aus dem CFF Feld `identifiers`. In der `identifiers`. In dieser Masterarbeit wird ausschließlich die erste DOI in der Liste der `identifiers` betrachtet. Diese wird in der allgemeinen Datei gespeichert, um später zu überprüfen, ob eine DOI in diesem Feld vorhanden ist.

Zusätzlich wird in beiden Fällen das Feld `date-released` und `doi` extrahiert und gespeichert. Die `doi` wird in den allgemeinen Dateien gespeichert, um im weiteren Verlauf zu überprüfen, ob eine DOI in der CFF-Datei vorhanden ist. Als letzte gemeinsame Information wird die `type` extrahiert und in den allgemeinen Dateien gespeichert. Dabei können wie in den Grundlagen erwähnt für die allgemeinen Informationen in der CFF nur „software“ und „dataset“ vorkommen. Für die allgemeinen Daten der `preferred-citation` können weitere Typen vorkommen, welche in den Grundlagen erläutert wurden.

Alle weiteren Daten sind spezifisch für die `preferred-citation`. Hierbei wird das Feld `date-published` extrahiert und in der allgemeinen Datei `cff_preferred_citation.csv` gespeichert. Die Felder `year`, `month` und `collection-doi`, `year`, `month` und `collection-doi` werden ebenfalls extrahiert und in der allgemeinen Datei für die `preferred-citation` gespeichert. Die Felder der Datei sind in ?? aufgelistet.

3.1.6 BibTeX

In diesem Unterabschnitt wird wie bereits zuvor eine Datei untersucht, welche in Git verwaltet wird. Hierbei wird Dieser Unterabschnitt betrachtet die CITATION.bib-Datei zu jedem Änderungszeitpunkt untersucht, falls sie in dem zu untersuchenden Paket vorhanden ist. Es wird die Software `bibtexparser` in der Version 2.0.0b7 verwendet, um die BibTeX-Datei zu analysieren (`weiss_python-bibtexparser_2024`).

Feld	Beschreibung
<i>cff_valid</i>	Gibt an, ob die CFF-Datei valide ist
<i>cff_init</i>	Gibt an, ob <i>cff_init</i> benutzt wurde
<i>type</i>	Typ der CFF-Datei
<i>date-released</i>	Datum an dem das Werk zugänglich gemacht wurde
<i>doi</i>	Zu zitierende DOI
<i>identifier-doi</i>	Die erste DOI in den <i>identifiers</i>
<i>committed_datetime</i>	Zeitpunkt des <i>commit date</i>
<i>authored_datetime</i>	Zeitpunkt des <i>author date</i>

Tabelle 8: Felder der *cff.csv*-Datei

Die Tabelle zeigt die Felder der *cff.csv*-Datei. Für jede Änderung der CFF-Datei werden die dargestellten Werte gespeichert. Die Felder *date-released*, *doi* und *identifier-doi* sind optional.

Feld	Beschreibung
<i>cff_valid</i>	Gibt an, ob die CFF-Datei valide ist
<i>cff_init</i>	Gibt an, ob <i>cff_init</i> benutzt wurde
<i>type</i>	Typ der CFF-Datei
<i>date-released</i>	Datum an dem das Werk zugänglich gemacht wurde
<i>date-published</i>	Veröffentlichungsdatum
<i>year</i>	Veröffentlichungsjahr
<i>month</i>	Veröffentlichungsmonat
<i>doi</i>	Zu zitierende DOI
<i>collection-doi</i>	DOI der Sammlung
<i>identifier-doi</i>	Die erste DOI in den <i>identifiers</i>
<i>committed_datetime</i>	Zeitpunkt des <i>commit date</i>
<i>authored_datetime</i>	Zeitpunkt des <i>author date</i>

Tabelle 9: Felder der *cff_preferred_citation.csv*-Datei

Die Tabelle zeigt die Felder der *cff_preferred_citation.csv*-Datei. Für jede Änderung der CFF-Datei werden die dargestellten Werte gespeichert, falls das Feld *preferred-citation* angegeben ist. Die Felder *date-released*, *date-published*, *year*, *month*, *doi*, *collection-doi* und *identifier-doi* sind optional.

Die Software bietet Möglichkeiten, um in Python BBTEX-Dateien zu lesen und zu schreiben. Sie muss in einer Beta-Version verwendet werden, da Features, welche benötigt werden, noch nicht im offiziellen Release vorhanden sind.

Die Autoren werden aus der BBTEX-Datei extrahiert und mit den neuesten Git-Daten abgeglichen. Die erstellte Liste wird in der Datei `TIMESTAMP_bib_authors_new.csv` gespeichert, wobei `TIMESTAMP` durch den Zeitpunkt der Änderung ersetzt wird. Dabei wird in der Datei ausschließlich der Name des Autors zuzüglich der Daten aus dem Abgleich mit Git gespeichert. Die Felder der Datei sind in ?? aufgelistet.

Ebenfalls wird erneut der Git-Prozess aus ?? bis zu diesem Zeitpunkt durchlaufen. Die Liste der Autoren wird in der Datei `TIMESTAMP_git_contributors.csv` gespeichert. Der Inhalt ist identisch, wie bereits in den anderen Quellen beschrieben. Außerdem wird die Liste der Autoren, welche aus der BBTEX extrahiert wurden, mit den Git-Autoren bis zu diesem Zeitpunkt abgeglichen. Die Daten werden in der Datei `TIMESTAMP_bib_authors.csv` gespeichert und die Inhalte sind identisch mit der „new“ Datei. Hierbei wird ebenfalls ausschließlich der Name des Autors und die Daten aus dem Git-Abgleich gespeichert.

Zudem können erneut allgemeine Daten für die BBTEX-Datei extrahiert werden. Hierbei wird eine weitere CSV-Datei erstellt, welche die allgemeinen Daten enthält. Die Datei wird `bib.csv` genannt und enthält, wie bereits in den anderen Quellen, die Zeitpunkte *commit date* und *author date*. Außerdem wird der Literaturtyp aus der Datei extrahiert und in der Datei gespeichert. Die Literaturtypen können dabei unterschiedlich sein, wie in ?? erläutert wurde. Zusätzlich wird das Jahr und der Monat der Veröffentlichung extrahiert und gespeichert. Diese Werte müssen nicht für jeden Literaturtypen vorhanden sein und können aus diesem Grund leer sein. Ebenfalls müssen die DOI und die ISBN nicht für jeden Literaturtypen vorhanden sein, falls sie jedoch vorhanden sind, werden sie extrahiert und gespeichert. Die Felder der Datei sind in ?? aufgelistet.

3.2 Abgleich

Dieser Abschnitt beschreibt den Abgleich von Autoren aus verschiedenen Quellen mit den Git-Autoren eines Pakets. Der entwickelte Algorithmus nutzt die in den Quellen verfügbaren Daten (Name, E-Mail, Benutzername), um Übereinstimmungen zu ermitteln. Dabei wird keine NED verwendet, sondern ein an die Daten angepasster Ansatz.

Feld	Beschreibung
<i>type</i>	Literaturtyp der BIBTEX-Datei
<i>year</i>	Veröffentlichungsjahr
<i>month</i>	Veröffentlichungsmonat
<i>doi</i>	Zu zitierende DOI
<i>isbn</i>	Zu zitierende ISBN
<i>committed_datetime</i>	Zeitpunkt des <i>commit date</i>
<i>authored_datetime</i>	Zeitpunkt des <i>author date</i>

Tabelle 10: Felder der `bib.csv`-Datei

Die Tabelle zeigt die Felder der `bib.csv`-Datei. Für jede Änderung der BIBTEX-Datei werden die dargestellten Werte gespeichert. Die Felder *year*, *month*, *doi* und *isbn* sind abhängig vom Literaturtyp optional.

Die Git-Autorenliste wird nach der Anzahl der Commits sortiert, um Autoren mit mehr Commits zu bevorzugen. Anschließend werden Autoren aus den Quellen mit den Git-Autoren verglichen. Der Abgleich erfolgt anhand von:

- Namen:** Namen werden in Kleinbuchstaben umgewandelt, um Schreibweisen zu vereinheitlichen. Mithilfe des Python-Keywords *in* wird geprüft, ob ein Teil des Namens in einem anderen enthalten ist. Dieses Keyword erlaubt eine einfache Teilstring-Suche. Ergänzend wird *thefuzz* in der Version 0.22.1 für eine unscharfe Suche verwendet. Dabei wird eine Ähnlichkeit von mindestens 80 % als Übereinstimmung gewertet. Dies hilft, Namen mit leichten Abweichungen korrekt zuzuordnen.
- E-Mails:** Die E-Mails werden ebenfalls mit *in* abgeglichen, um zu prüfen, ob eine der Adressen (ganz oder teilweise) in der anderen enthalten ist. Zusätzlich wird *thefuzz* verwendet, um Ähnlichkeiten von mindestens 80 % zu erkennen.
- Benutzernamen:** Benutzernamen aus den Quellen werden mit Teilen der Git-E-Mails (lokaler oder Domain-Teil) verglichen, da Git keine Benutzernamen direkt bereitstellt. Auch hier kommt das Keyword *in* zum Einsatz, um einfache Übereinstimmungen zu finden, ergänzt durch die unscharfe Suche mit *thefuzz*.

Für jeden Vergleich wird ein Score berechnet, der die Übereinstimmungen relativ zur maximal möglichen Anzahl von Feldern darstellt. Der Git-Autor mit dem höchsten Score wird zugeordnet. Falls zwei Einträge in der Git-Liste existieren, welche einen gleichen Score erreichen, wird der Autor ausgewählt, welcher die meisten Commits hat. Falls keine Übereinstimmung gefunden wird, bleiben die Felder leer.

Das Ergebnis enthält die Autoren aus den Quellen, ergänzt um die Ergebnisse des Abgleichs, welche in ?? dargestellt sind.

3.3 Auswertung

In diesem Abschnitt wird beschrieben, wie die Daten aus den Abschnitten ?? und ?? ausgewertet werden. Dabei wird der Abschnitt aufgeteilt in die Aggregierung der Daten, welche zuvor beschrieben wurden und anschließend wird kurz auf die Darstellung der aggregierten Daten eingegangen.

Die Auswertung der Daten erfolgt erneut mittels Python. Für die Auswertung werden alle Daten verarbeitet, welche durch die Datenbeschaffung gesammelt wurden. Einzige Ausnahme ist, dass immer die Dateien mit der *new* Endung betrachtet werden. Dies liegt daran, dass die Auswertung aktuell nur auf den neuesten Git-Daten durchgeführt wird und die Git-Daten zu den jeweiligen Zeitpunkten der Änderung nicht betrachtet werden. Außerdem ist darüber ein Abgleich zwischen den Git-Autoren und den Autoren aus den anderen Quellen möglich. Aber auch ein Abgleich zwischen den einzelnen Quellen ist möglich, da die Autoren in der neuesten Version immer die gleichen Git-Daten wie beispielsweise die gleiche Anzahl an Commits in allen Quellen haben. Der Abgleich der Autoren ist hierbei nur möglich, falls in der Datenbeschaffung die Autoren abgeglichen werden konnten.

Falls ein Abgleich zwingend erforderlich ist, beispielsweise bei der Untersuchung, wie lange ein Autor durchschnittlich in einer Quelle genannt ist, wird ein einfacher Vergleich des Namens über die Quelle durchgeführt. Dieser Vergleich ist jedoch nicht immer korrekt, da beispielsweise Namensänderungen in den Quellen so nicht berücksichtigt werden können, falls kein Abgleich in der Datenbeschaffung möglich war.

~~Das Skript ist so aufgebaut, dass nicht alle Daten auf einmal geladen werden, sondern jede Datei einzeln geladen und verarbeitet wird. Dies bietet den Vorteil, dass auch mit weniger Arbeitsspeicher gearbeitet werden kann und die Daten nicht alle auf einmal im Arbeitsspeicher gehalten werden müssen. Jedoch birgt dies auch Nachteile, beispielsweise dass zu keinem Zeitpunkt bekannt ist, wie viele Autoren maximal in einer Quelle vorkommen. Diese Information ist erst vorhanden, wenn alle Daten vollständig verarbeitet worden sind.~~

~~Zusätzlich ist die Die Aggregierung der Daten ist unterteilt in die Aggregierung der vollständigen Daten und die Aggregierung und die der neuesten Daten. In der vollständigen Datenverarbeitung werden Ergebnisse extrahiert, die die gesamte Historie der Daten betrachten. Hier werden alle Daten mit ihren Zeitstempeln verarbeitet. In der neuesten Datenverarbeitung werden nur die Daten betrachtet, die auf dem neuesten Stand der untersuchten Software sind. Dabei wird jeweils die neueste Datei~~

in den Quellen mithilfe des Zeitstempels ausgewählt und anschließend verarbeitet. In diese beiden Kategorien ist das ?? ebenfalls unterteilt.

4 Ergebnisse

In diesem Kapitel werden die Ergebnisse ~~der~~dieser Masterarbeit präsentiert. Das Kapitel ist in zwei Abschnitte unterteilt. Zum einen werden die Ergebnisse der Gegenwart in ?? präsentiert. Hierbei werden die jeweils neuesten Dateien aus der Datenbeschaffung untersucht und Statistiken aufgeführt, welche in Verbindung mit diesen Daten stehen. Zum anderen werden die Ergebnisse mit Zeitverlauf in ?? präsentiert. Hierbei werden die Dateien aus der Datenbeschaffung in allen Versionen analysiert. Durch die Betrachtung aller Versionen können Veränderungen über die Zeit aufgezeigt werden. In dem ?? wird zu Beginn auf die Statistiken des Abgleichs eingegangen und präsentiert, wie ~~genau~~der Abgleich durchgeführt wurde. Der Abgleich hat einen direkten Einfluss auf viele weitere Statistiken.

Die benötigten Daten für die Untersuchungen wurden in der Zeitspanne September 2024 bis November 2024 beschafft. Dies liegt daran, dass die Repositorys lokal gespeichert werden~~und bei der Entwicklung der Arbeit die Daten nicht erneut beschafft werden mussten, um diese bei der Erstellung dieser Arbeit nicht erneut beschaffen zu müssen~~. Die Daten einer Liste wurden jeweils an einem Tag beschafft, um eine Vergleichbarkeit zu gewährleisten. In dem gesamten Kapitel wird *matplotlib* in der Version 3.9.2 verwendet, um die Grafiken zu erstellen (**hunter_matplotlib_2007; hunter_matplotlib_2024**).

4.1 Ergebnisse der Gegenwart

Abgleich

Da die Ergebnisse ~~der~~dieser Masterarbeit direkt abhängig von der Qualität des Abgleichs sind, wurde diese untersucht. Dies wurde auf zwei Arten durchgeführt. Zum einen wurde automatisch die Anzahl der abgeglichenen und der nicht abgeglichenen Autoren untersucht. Zum anderen wurden manuell für eine ausgewählte Menge an Personen die Ergebnisse händisch überprüft. Es wurde in beiden Fällen die Überprüfung ausschließlich auf den neuesten Versionen der Daten durchgeführt.

Quelle	PyPI CFF	CRAN CFF
CRAN Autoren		204/238 (85,71 %)
CRAN Maintainer		99/100 (99,00 %)
Beschreibung	580/1070 (54,21 %)	21/80 (26,25 %)
README	868/1396 (62,18 %)	143/1008 (14,19 %)
CFF	477/598 (79,77 %)	184/233 (78,97 %)
CFF preferred citation	294/380 (77,37 %)	109/136 (80,15 %)
PyPI Maintainer	206/228 (90,35 %)	
Python Autoren	105/131 (80,15 %)	
Python Maintainer	15/25 (60,00 %)	
BIBTEX	0/1 (0,00 %)	
Summe	2545/3829 (66,47 %)	760/1795 (42,34 %)

Tabelle 11: Automatische Ergebnisse des Abgleichs

In der Tabelle sind die Ergebnisse des automatischen Abgleichs für die Listen PyPI CFF und CRAN CFF dargestellt. Die Werte geben an, wie viele der genannten Autoren in den Git-Repositorys gefunden wurden.

Die automatische Überprüfung erfolgt bei der Auswertung der Daten aus der Datenbeschaffung. Dabei wird für jede Liste und für jede Quelle z. B. CFF die Anzahl der abgeglichenen Autoren und der insgesamt vorhandenen Autoren ermittelt. Hierbei wurde die gesamte Menge an Autoren betrachtet. Die Ergebnisse für die Listen PyPI CFF und CRAN CFF sind in ?? dargestellt. Für die weiteren Listen sind die Ergebnisse in ?? zu finden.

In den Ergebnissen ist auffällig, dass die Autoren in der README-Datei und in der Beschreibung der Pakete am schlechtesten abgeglichen werden konnten. Dies liegt daran, dass hierbei die NER verwendet wurde und diese nicht immer korrekt arbeitet. Hierbei kommt es häufig vor, dass Entitäten erkannt werden, welche keine Person darstellen. Außerdem werden Personen mit weiteren Zusätzen wie z. B. dem Anfang einer Internetadresse zurückgegeben, was den Abgleich erschwert. Zusätzlich werden in der README und der Beschreibung häufig Personen genannt, welche z. B. Vorarbeit für die Software geleistet haben, aber in der Software selbst keinen Code beigetragen haben. Diese Personen können nicht abgeglichen werden. Die Python Maintainer konnten ebenfalls schlecht abgeglichen werden, da viele Pakete in diesem Feld keine Personen, sondern Organisationen nennen.

Allgemein bietet dieses Vorgehen keine Aussage darüber, ob die Personen nicht abgeglichen worden sind, weil sie keinen Code beigetragen haben oder weil sie nicht gefunden wurden. Zudem werden Personen, welche fälschlicherweise mit einer anderen Person abgeglichen wurden, in diesem Verfahren als gut bewertet. Um diese

Fälle ebenfalls betrachten zu können, wurde manuell eine Auswahl an Autoren überprüft. Es wurden nicht alle Autoren überprüft, da dies den zeitlichen Rahmen **der** **dieser** Arbeit überschritten hätte. Aus diesem Grund wurden die Autoren für jedes Paket und jede Quelle zufällig neu angeordnet und gespeichert. Anschließend wurden jeweils die ersten beiden Autoren anhand von vier Kriterien überprüft. Falls nur ein Autor in einer Quelle genannt wurde, wurde nur dieser überprüft. Jeder Autor wurde einer der vier Kategorien richtig positiv (TP), falsch negativ (FN), falsch positiv (FP) oder richtig negativ (TN) zugeordnet. Ein Autor ist als richtig positiv klassifiziert worden, falls er richtig zugeordnet wurde. Falls der Autor mehrfach in der Git-Autorenliste vorkommt, wurde der Autor als richtig positiv klassifiziert unabhängig davon, mit welchem von den doppelten Einträgen der Autor zugeordnet wurde. Aus diesen Daten wurde der F1-Score berechnet.

Außerdem wurde angegeben, ob es sich um keine Person handelt, sondern z. B. um eine Organisation. In diesen Fällen wurde der Eintrag dennoch in eine der vier Kategorien eingeteilt, wobei primär die Kategorie FP verwendet wurde, falls es eine Zuordnung gab. Falls die Zuordnung allerdings mit einem Git-Autor erfolgt ist, welcher beispielsweise ein Bot der Organisation ist, wurde der Eintrag als TP eingeteilt, da es keinen Mechanismus in der Datenbeschaffung gibt, welcher Personen von nicht Personen unterscheiden kann. Es wurde lediglich in der Datenbeschaffung darauf geachtet, möglichst keine Quellen zu berücksichtigen, welche keine Personen enthalten. Die Ergebnisse für alle Listen zusammen sind in ?? eingetragen. Die Ergebnisse für die einzelnen Listen sind in den Tabellen ??, ??, ??, ?? und ?? dargestellt.

In der ?? ist zu erkennen, dass über alle Listen hinweg ein F1-Score von 0,9411 erreicht wurde. Besonders auffällig sind die Werte für die Beschreibung und die README-Datei. Hier wurden F1-Scores von 0,8832 und 0,8351 erreicht. Außerdem sind in den beiden Quellen jeweils die meisten TN vorhanden. Zudem ist deutlich, dass in den Quellen von Python und PyPI viele Autoren keine Personen sind. In diesen Quellen sind zusammen 150 Autoren keine Personen. Dies entspricht 76,92 % aller nicht Personen in den manuellen Ergebnissen. Eine weitere Auffälligkeit ist die geringe Anzahl an BIBTEX Autoren. Hier werden insgesamt nur vier Autoren betrachtet, zusätzlich ist der F1-Score mit 0,8000 am schlechtesten für diese Quelle.

Commits gegenüber geänderten Zeilen

Aus den gesammelten Daten wurden weitere Statistiken berechnet. Dabei muss berücksichtigt werden, dass einige Statistiken auf dem Abgleich basieren und somit

Quelle	TP	FN	FP	TN	Keine Person	F1-Score
Beschreibung	87	5	18	125	15	0,8832
README	81	21	11	143	26	0,8351
CFF	168	13	3	24	3	0,9545
CFF preferred citation	78	8	2	13	0	0,9398
PyPI Maintainer	294	2	32	48	50	0,9453
Python Autoren	155	4	35	43	75	0,8883
Python Maintainer	30	0	5	21	25	0,9231
CRAN Autoren	269	4	3	19	1	0,9872
CRAN Maintainer	193	2	0	0	0	0,9948
BIBTEX	4	2	0	0	0	0,8000
Summe	1359	61	109	436	195	0,9411

Tabelle 12: Manuelle Ergebnisse des Abgleichs für alle Listen zusammen

In der Tabelle sind die Ergebnisse des manuellen Abgleichs für alle Listen eingetragen. Dabei wurde für jede Quelle die Anzahl der TP, FN, FP und TN Autoren ermittelt. Zusätzlich wurde angegeben, ob es sich um keine Person handelt und der F1-Score berechnet.

nur so gut sind, wie der Abgleich selbst. Außerdem basieren einige Statistiken auf den Metriken der Commits und geänderten Zeilen. Die **Abbildung ??** stellt das Verhältnis der Commits und der geänderten Zeilen für die Autoren in den Paketen der Listen PyPI CFF und CRAN CFF dar. Für die anderen Listen sind die Ergebnisse in ?? dargestellt. Ein Punkt stellt dabei einen Autor dar. Es ist zu erkennen, dass einige Autoren nur einen Commit getätigt haben und dabei viele Zeilen geändert haben. Beispielsweise hat der Autor François Lagunas in dem Paket *huggingface/datasets* lediglich einen Commit getätigt, aber dabei ungefähr 29.000 Zeilen hinzugefügt. Inhalt sind in diesem konkreten Fall automatisch generierte README-Dateien für Datensätze, welche zuvor gefehlt haben.

Top-Git-Autoren

Die ?? zeigt den Anteil der Top-Git-Autoren in den einzelnen Quellen, wie beispielsweise der CFF. Dabei werden mehrere Abbildungen für die unterschiedlichen untersuchten Listen dargestellt. In der konkreten Abbildung sind die Listen PyPI CFF und CRAN CFF dargestellt. Die Abbildungen für die weiteren Listen sind in ?? dargestellt. Außerdem werden für jede Liste zwei unterschiedliche Abbildungen dargestellt. Die eine Abbildung bemisst die Top-Git-Autoren an der Anzahl der Commits und die jeweils andere an der Anzahl der geänderten Zeilen. Falls beispielsweise der Autor mit den meisten Commits in einer Quelle z. B. CFF genannt wird und die Autoren an den Commits bemessen werden und zusätzlich nur der

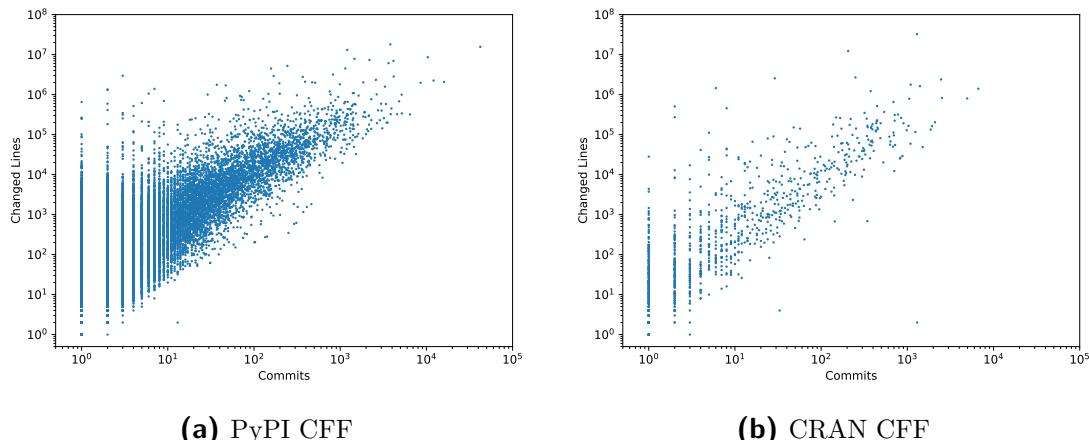


Abbildung 7: Commits und geänderte Zeilen gegenübergestellt

Die Abbildungen zeigen für die Autoren in den Paketen in den Listen PyPI CFF und CRAN CFF die Anzahl der Commits gegenüber der Anzahl der geänderten Zeilen. Die x-Achse stellt die Anzahl der Commits dar und die y-Achse die Anzahl der geänderten Zeilen. Es wird eine logarithmische Skalierung verwendet.

Top-Autor betrachtet wird, so wird in der Abbildung bei $x = 1$ dargestellt, dass 100 % der Git-Autoren in der CFF genannt werden.

Bei der Betrachtung von immer mehr Git-Autoren sinkt der Anteil der genannten Autoren in den Quellen. Beispielsweise werden in ?? von 100 betrachteten Top-Git-Autoren gemessen nach Commits nahezu 0 % als Python Maintainer genannt, jedoch werden knapp 10 % in der CFF preferred citation angegeben. Dabei muss berücksichtigt werden, dass unterschiedliche Quellen unterschiedlich häufig vorkommen. Beispielsweise haben weniger Pakete CFF preferred citation Autoren angegeben als CFF Autoren, da diese zwingend angegeben werden müssen. Dadurch kann es dazu kommen, dass nur ein Paket eine bestimmte Quelle angegeben hat und wenn dieses Paket in der Quelle viele Autoren nennt, ist der Anteil der genannten Autoren in der gesamten Quelle hoch.

Im Hintergrund der Abbildungen werden weitere Linien präsentiert, welche nicht in der Legende aufgeführt sind. Bei jeder Linie handelt es sich um eine einzige Quelle eines einzigen Pakets. Ein Paket wird somit mit mehreren Linien dargestellt. Auffällig ist dabei, dass einige Linien erst bei einem Wert von $x > 1$ beginnen. Dies ist der Fall, falls ein Paket in einer Quelle den Top-Autor nicht nennt. Beispielsweise gibt das Paket *faker* in der CFF als einzigen Autor „Daniele Faraglia“ an. Dieser Autor ist jedoch auf Rang 37 der Top-Git-Autoren, gemessen nach Commits.

Eine weitere Auffälligkeit, welche beispielsweise in der ?? vorkommt, ist, dass in der Legende BiBTeX aufgeführt ist, jedoch keine Linie in der Abbildung dargestellt

wird. Dies liegt daran, dass keiner der Top-100-Git-Autoren aller Pakete mit `BIBTEX` in einer `BIBTEX` als Autor genannt wird. Im konkreten Fall liegt das daran, dass nur eines der Pakete eine `BIBTEX`-Datei enthält und in dieser Datei nur ein Autor genannt wird, welcher nicht automatisch abgeglichen wurde. Dies ist ?? ebenfalls zu entnehmen. Dadurch liegt die Linie für alle x-Werte bei 0 % und ist nicht sichtbar.

Die letzte Auffälligkeit in den Graphen ist, dass einige Linien erst fallen und anschließend parallel zur x-Achse verlaufen. Andere wiederum fallen gar nicht und verlaufen direkt parallel zur x-Achse. Dies ist der Fall, falls ab einer bestimmten Anzahl an Git-Autoren alle weiteren Git-Autoren genannt werden. Dies tritt besonders dann auf, falls es wenig Git-Autoren gibt. Beispielsweise hat das Paket `netron`, welches in der PyPI CFF Liste vorkommt, nur einen Git-Autor. Dieser wird ebenfalls beispielsweise als PyPI Maintainer genannt. Dadurch sind alle Git-Autoren zu 100 % als PyPI Maintainer genannt und es kommt zu der Linie bei 100 % in ??.

Weitere Abbildungen, welche den Anteil zwischen genannten Autoren und den Top-Git-Autoren darstellen, sind in ?? abgebildet. Die Abbildung stellt die Graphen für die Listen PyPI CFF und CRAN CFF dar. Die Ergebnisse für die weiteren Listen sind in ?? dargestellt. In den Abbildungen wurden erneut zwei Graphen für jede Liste erstellt. Zum einen wurden die Autoren nach der Anzahl der Commits und zum anderen nach der Anzahl der geänderten Zeilen sortiert. In allen Abbildungen wird der Anteil der genannten Autoren unter den Top-Git-Autoren dargestellt. Durch dieses Verhältnis steigen die Linien, da mehr genannte Autoren unter den Top-Git-Autoren bei steigender Anzahl derer sind.

Die Abbildungen zeigen beispielsweise für $x = 0$ immer, dass 0 % der genannten Autoren unter den Top-0-Git-Autoren sind. Dies liegt daran, dass immer null Autoren unter den ersten null Git-Autoren sind. Der Anteil kann dabei 100 % erreichen, falls alle genannten Autoren in einer Quelle auch tatsächlich unter den Top-200-Git-Autoren sind und somit Quellcode entwickelt haben. Dabei muss berücksichtigt werden, dass nicht abgegliche Autoren ebenfalls berücksichtigt werden. Falls ein Autor nicht abgeglichen wurde, wird dieser als nicht unter den Top-Git-Autoren betrachtet und der Graph kann keine 100 % für die Quelle erreichen.

Um konkrete Werte zu nennen ist in ?? beispielweise dargestellt, dass ungefähr 80 85,53 % der als PyPI Maintainer genannten Autoren unter den Top-100-Git-Autoren nach der Anzahl der Commits sind und nur ungefähr 40 46,17 % der in der README Beschreibung genannten Autoren unter den Top-100-Git-Autoren nach Commits

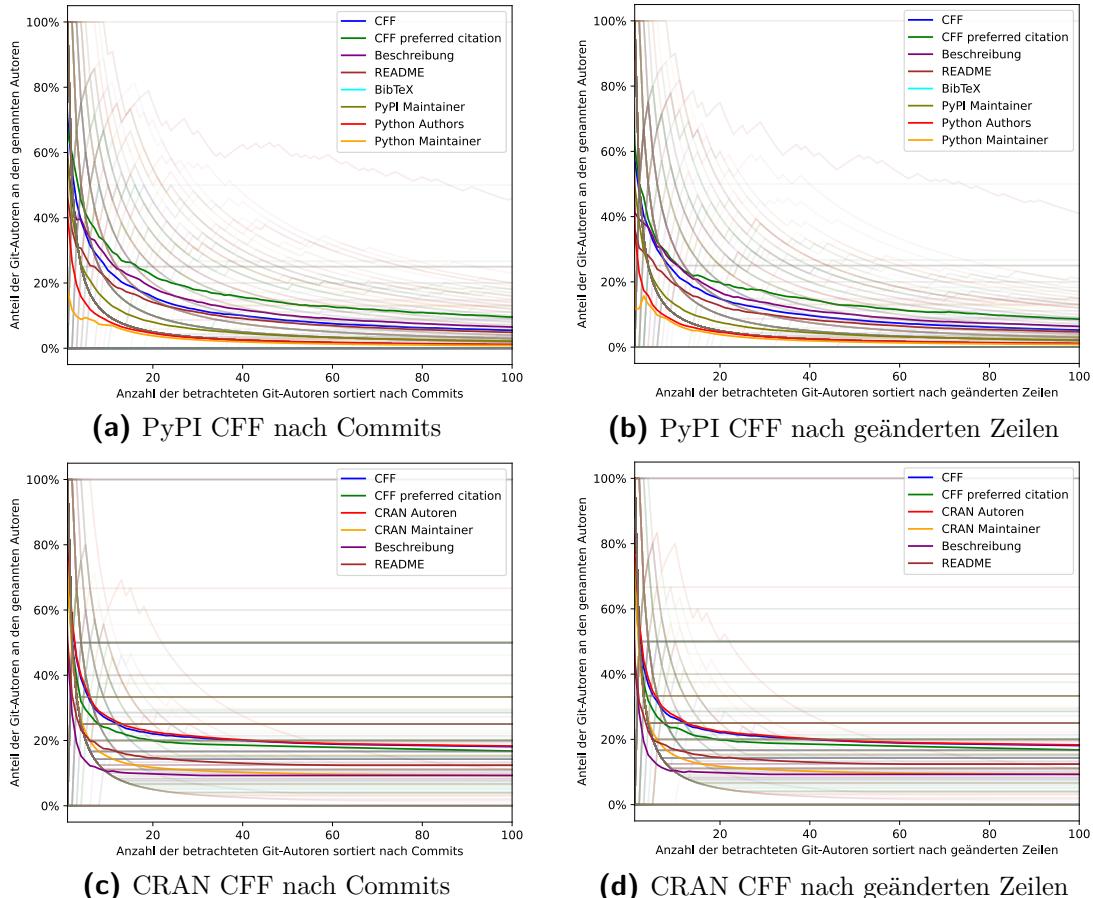


Abbildung 8: Anteil der Top-Git-Autoren an den genannten Autoren

Die Abbildungen zeigen für die Pakete in den Listen PyPI CFF und CRAN CFF den Anteil der Top-Git-Autoren in der Zitation. Die Linien stellen die unterschiedlichen Quellen dar. Auf der x-Achse wird die Anzahl der betrachteten Git-Autoren für ?? und ?? sortiert nach der Anzahl der Commits und für ?? und ?? sortiert nach der Anzahl der geänderten Zeilen angegeben. Die y-Achse stellt jeweils den Anteil der genannten Autoren an den Git-Autoren dar. Für die **blaue**-CFF-Linie in ?? gilt: Die Top x Commiter sind zu y % in der CFF gelistet.

sind. Die Linien im Hintergrund stellen erneut ein einzelnes Paket einer einzelnen Quelle dar, sodass ein Paket mit mehreren Linien für alle Quellen dargestellt wird.

In den Graphen sind erneut Auffälligkeiten zu erkennen. Beispielsweise existieren erneut Linien, welche für jeden x Wert 0 % oder 100 % darstellen. In ?? ist beispielsweise erneut die Linie für BIBTEX nicht zu erkennen, da diese erneut auf der x-Achse verläuft. Dies liegt daran, dass nur ein Autor in einer BIBTEX-Datei über alle Pakete genannt wird, welcher nicht abgeglichen werden konnte und somit nicht unter den Top-Git-Autoren ist. Für die Linien, welche bei 100 % verlaufen, handelt es sich um Pakete, bei denen beispielsweise nur ein Autor in der Quelle genannt wird und dieser Autor auf Rang eins der Top-Git-Autoren ist. Es kann aber auch komplexer sein, beispielsweise, dass drei Autoren in der Quelle genannt werden und alle drei unter den Top-drei-Git-Autoren sind. Die Reihenfolge, in der sie in der Quelle genannt werden, ist dabei irrelevant. Es muss dabei jedoch berücksichtigt werden, dass auch diese Linien nicht direkt bei 100 % starten, da bei der Betrachtung von null Git-Autoren keine genannten Autoren unter den Top-Git-Autoren sein können.

Ebenfalls ist auffällig, dass einige Linien erst bei einem Wert von $x > 2$ beginnen. Dies ist der Fall, falls kein Autor in einer Quelle unter den Top x Committern ist. Beispielsweise wird in dem PyPI Paket *faker* in der CFF nur der Autor „Daniele Faraglia“ genannt. Dieser ist jedoch erst auf Rang 37 der Top-Git-Autoren gemessen nach Commits gelistet.

Autoren ohne Commits

In ?? wird der Anteil der genannten Autoren in unterschiedlichen Quellen ohne Commits in den Paketen der Listen PyPI CFF und CRAN CFF dargestellt. Die Ergebnisse für die weiteren Listen sind in ?? dargestellt. In den Abbildungen werden die vergangenen zehn Jahre dargestellt. Dabei steigt mit steigender Jahreszahl der Anteil der Autoren, welche keinen Commit getätigten haben. Das Ende der x-Achse stellt den Tag der Datenbeschaffung dar. Aus diesem Grund ist der Anteil der Autoren ohne Commits am Ende der x-Achse bei 100 %, da es nicht möglich ist, in der Zukunft Commits zu tätigen.

In der ?? lässt sich erkennen, dass nahezu alle genannten Python Maintainer bis zum Jahr 2020 mindestens einen Commit getätigten haben. Außerdem ist erkennbar, dass ab diesem Jahr die in der README genannten Autoren im Vergleich zu den anderen Quellen anteilig die meisten Autoren ohne Commits genannt haben. Die

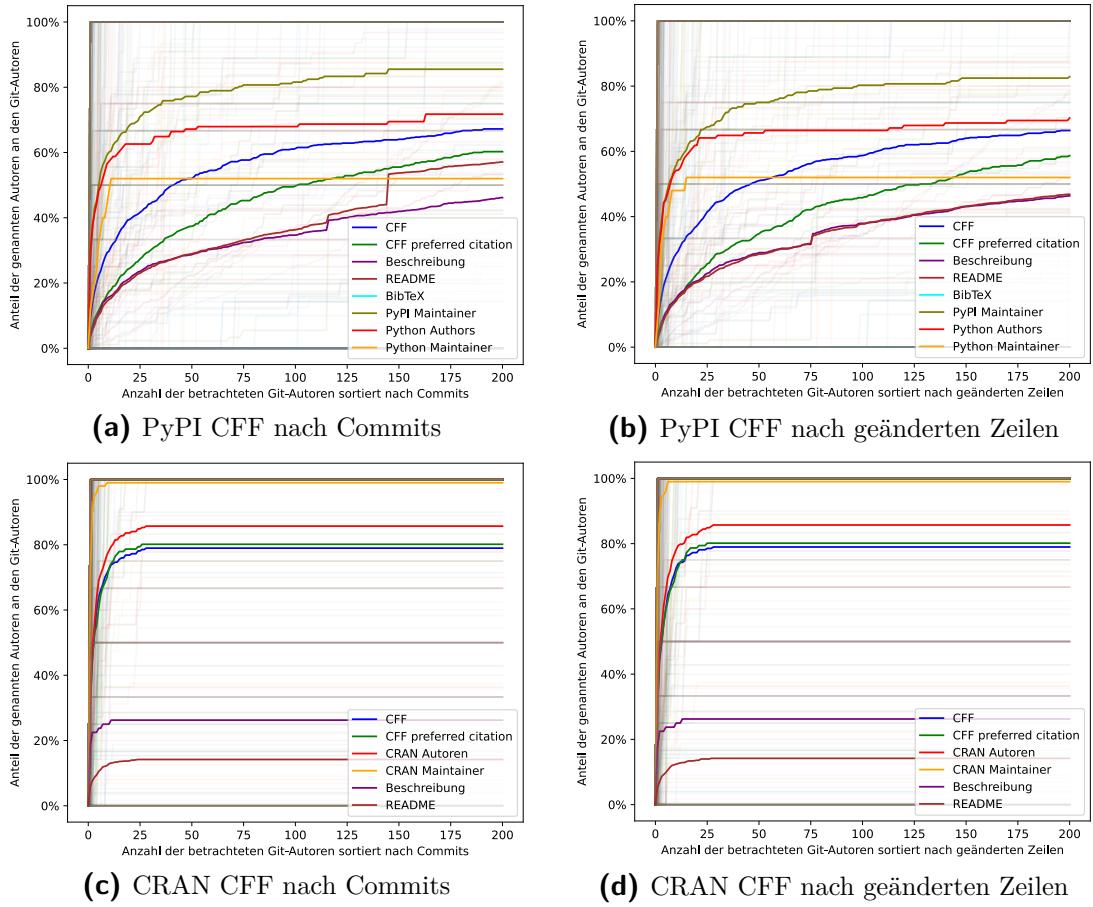
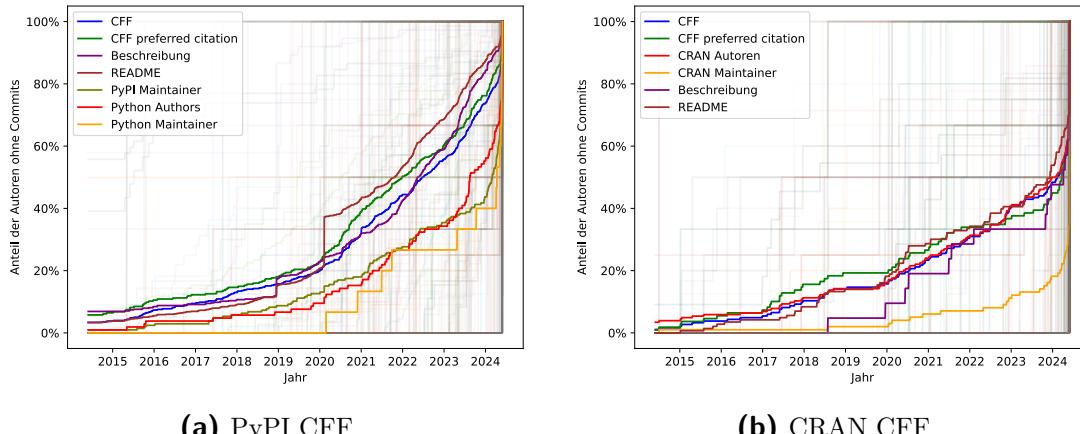


Abbildung 9: Anteil der genannten Autoren unter den Top-Git-Autoren

Die Abbildungen zeigen für die Pakete in den Listen PyPI CFF und CRAN CFF den Anteil der genannten Autoren unter den Top-Git-Autoren. Die Linien stellen die unterschiedlichen Quellen dar. Auf der x-Achse wird die Anzahl der betrachteten Git-Autoren für ?? und ?? sortiert nach der Anzahl der Commits und für ?? und ?? sortiert nach der Anzahl der geänderten Zeilen angegeben. Die y-Achse stellt jeweils den Anteil der genannten Autoren an den Git-Autoren dar. Für die **blaue**-CFF-Linie in ?? gilt: Autoren in der CFF sind zu y % unter den Top x Commitern.

**Abbildung 10:** Autoren ohne Commits

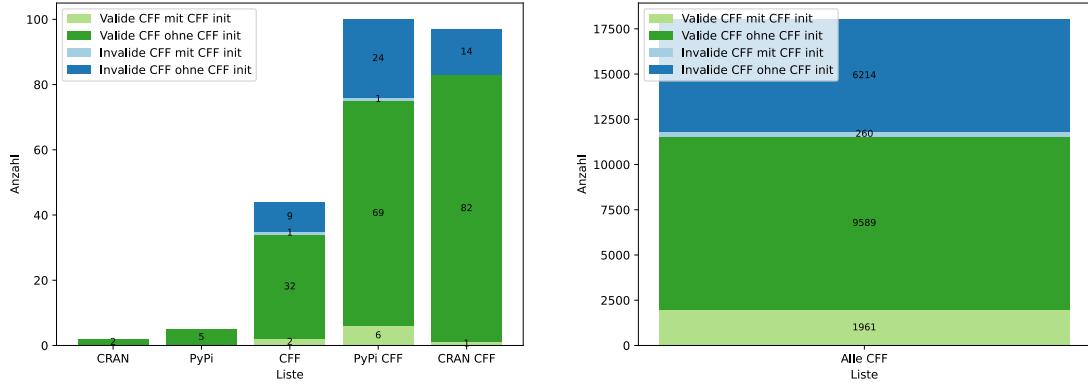
Die Abbildungen zeigen für die Pakete in den Listen PyPI CFF und CRAN CFF den Anteil der Autoren ohne Commits. Die Linien stellen die unterschiedlichen Quellen dar. Auf der x-Achse werden die Jahre dargestellt. Die y-Achse stellt den Anteil der Autoren ohne Commits dar. Für die **blaue**-CFF-**Linie** gilt in beiden Abbildungen: y % der Autoren in der CFF haben seit dem Jahr x keinen Commit getätigt.

Linien im Hintergrund stellen erneut ein einzelnes Paket einer einzelnen Quelle dar, sodass ein Paket mit mehreren Linien für alle Quellen dargestellt wird.

Validität der CFF-Dateien

Die ?? zeigt den Anteil der validen und invaliden CFF-Dateien in allen fünf Listen. Dabei wird außerdem unterschieden in Dateien, welche Anzeichen auf die Benutzung von *cffinit* aufweisen und Dateien ohne die Verwendung von dem Programm. Zudem lässt sich in der Abbildung erkennen, wie viele Pakete in welcher Liste das CFF verwendet haben. Dabei ist auffällig, dass in der CRAN CFF Liste keine 100 Pakete gelistet werden, obwohl die Liste 100 Pakete enthalten sollte, welche ein CFF verwenden. Dies liegt daran, dass in der Liste, welche vom Zweitgutachter [dieser Arbeit](#) bereitgestellt wurde, einige Pakete enthalten sind, welche keine CFF-Datei (mehr) enthalten. Konkret betrifft das in diesem Fall die drei Pakete *gtsummary*, *ggpp* und *ggnnards*. Aus diesem Grund sind in den weiteren Abbildungen ebenfalls keine 100 Pakete für die CRAN CFF Liste dargestellt.

Außerdem ist auffällig, dass in der CFF Liste und der PyPI CFF Liste jeweils ein Paket vorhanden ist, welches eine invalide CFF-Datei enthält, obwohl für die Erstellung *cffinit* verwendet wurde. Konkret handelt es sich in beiden Fällen um das PyPI Paket *modelyst-sqlmodel*. Das Paket ist in beiden Listen enthalten, da in der CFF Liste ausschließlich PyPI Pakete enthalten sind, da diese die meisten Sterne

**Abbildung 11:** Validität der CFF-Dateien

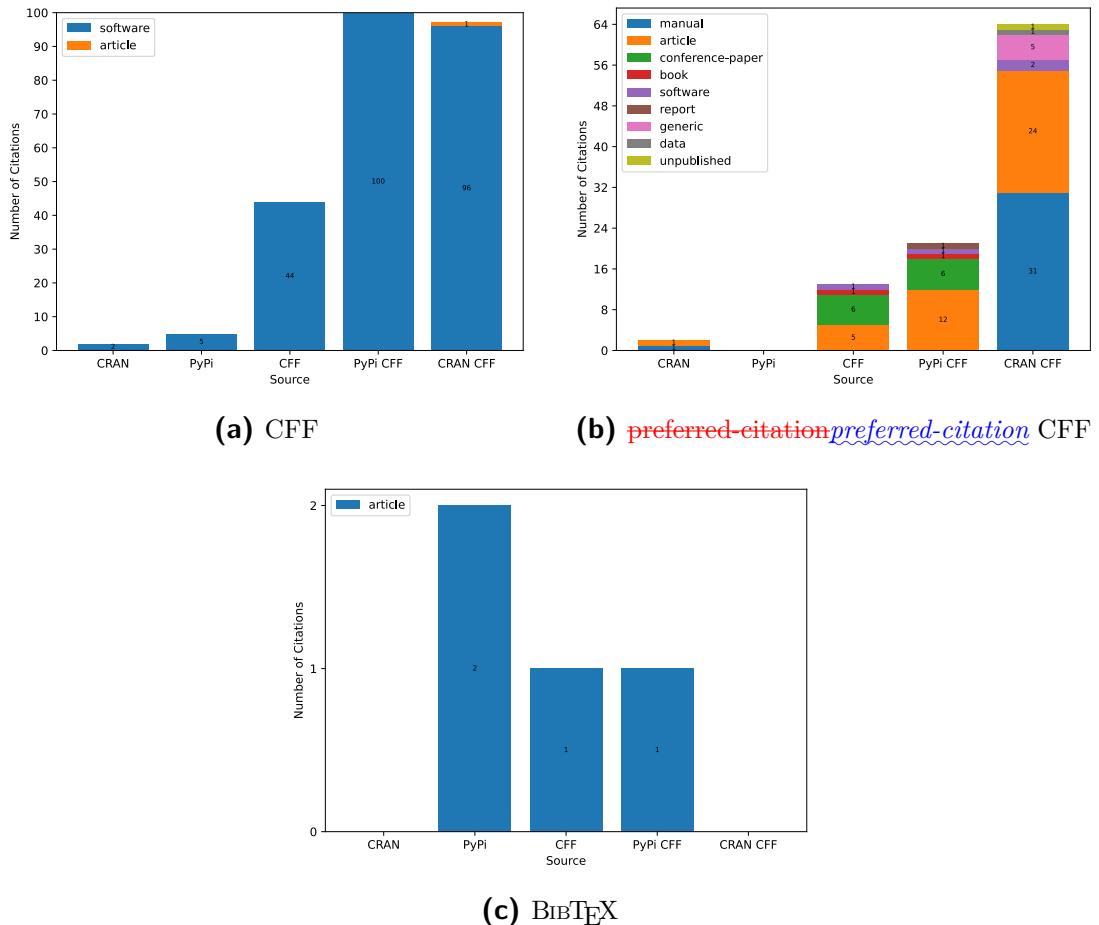
Die ?? zeigt für die verschiedenen analysierten Listen, wie viele der vorhandenen CFF-Dateien valide sind, und ob *cffinit* verwendet wurde oder nicht. Die ?? zeigt die gleichen Informationen für alle CFF-Dateien auf GitHub.

auf GitHub haben. In diesem Fall ist das Paket unter den Top-100 CFF Paketen auf GitHub und somit ebenfalls in der PyPI CFF Liste enthalten. Im Allgemeinen sind alle PyPI Pakete, welche in der CFF Liste enthalten sind, ebenfalls in der PyPI CFF Liste enthalten. In der CFF Liste ist kein einziges CRAN Paket enthalten, da die CRAN Pakete zu wenig Sterne auf GitHub haben, um unter den Top-100 CFF Paketen zu sein.

Die ?? zeigt die gleichen Informationen wie die zuvor beschriebene Abbildung. Diesmal wurde die Analyse allerdings nicht auf einer Liste, sondern für alle auf GitHub verfügbaren CFF-Dateien, welche analysiert werden konnten, durchgeführt. Dabei ist auffällig, dass der Anteil der validen CFF-Dateien zwar größer ist, jedoch der Anteil der invaliden CFF ebenfalls weiter gestiegen ist.

Typ der Zitation

?? zeigt für die Quellen CFF, [preferred-citation](#) CFF und BIBTEX den angegebenen Typ der Zitation. Anhand der Abbildungen ist außerdem zu erkennen, wie oft welche Quelle in den Listen vorkommt. Beispielsweise haben zwei der Top-100 CRAN Pakete eine CFF angegeben und fünf der Top-100 PyPI Pakete. Von diesen fünf Paketen hat kein Paket eine [preferred-citation](#) CFF angegeben. In den CRAN Listen haben beide Pakete hingegen ebenfalls eine [preferred-citation](#) CFF angegeben. Zudem ist auffällig, dass in allen Listen die Anzahl der Pakete, welche eine BIBTEX-Datei verwenden, gering ist. Bei dieser Betrachtung ist darauf zu achten, dass in dem CFF nicht zwingend eine

**Abbildung 12:** Typ der angegebenen Zitationen der einzelnen Quellen

Die Abbildungen zeigen für die drei unterschiedlichen Quellen, jeweils für alle fünf untersuchten Listen, welcher Typ von Zitation angegeben wurde. Es ist darauf zu achten, dass die y-Achsen unterschiedlich skaliert sind.

preferred-citation angegeben werden muss. Aus diesem Grund sind in ?? weniger Pakete dargestellt als in ??.

In ?? ist auffällig, dass ein CRAN Paket aus der CRAN CFF Liste als Typ der Zitation *article* angegeben hat. Dies ist in einer CFF-Datei nicht zulässig und führt zu einer ungültigen CFF-Datei. Dies betrifft das Paket *worcs*. Bei der Betrachtung aller CFF-Dateien auf GitHub ist zu erkennen, dass von 18.520 17.470 Pakete *software* angegeben haben und 412 *dataset*. Alle anderen 638 Pakete haben einen anderen Typ der Zitation angegeben und sind dadurch ungültig.

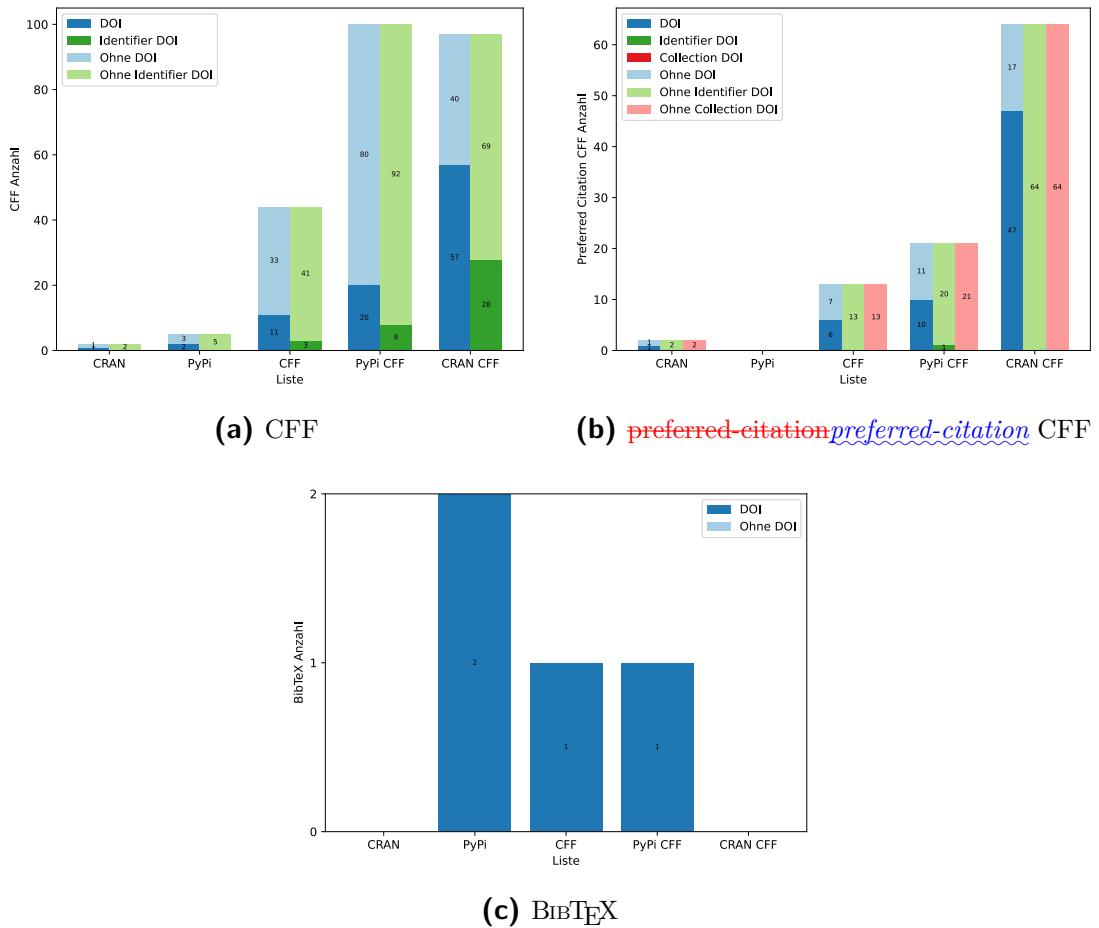
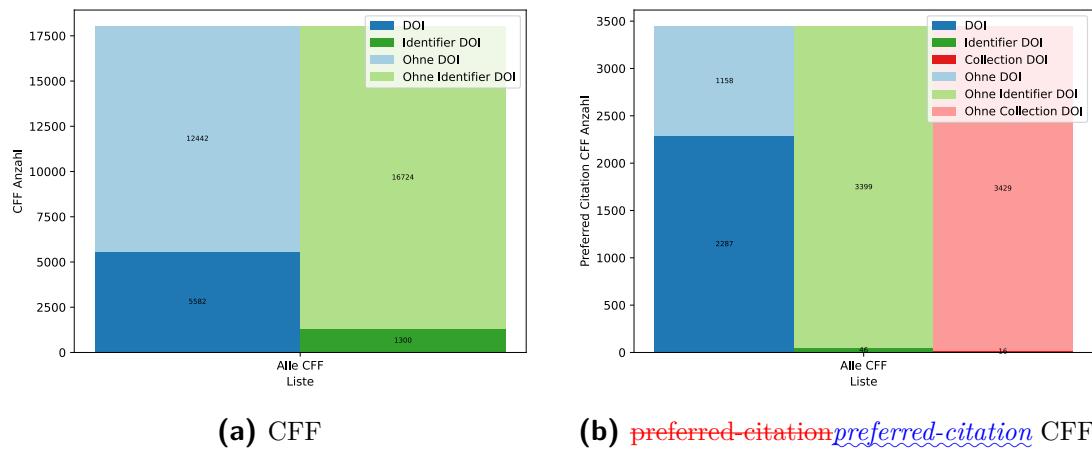


Abbildung 13: Verwendung von DOIs in den Zitationen der einzelnen Quellen
Die Abbildungen zeigen für die drei unterschiedlichen Quellen, jeweils für alle fünf untersuchten Listen, wie oft eine DOI auf verschiedenen Arten in den Zitationen angegeben wurde.

Verwendung von DOIs

In ?? ist dargestellt, wie viele der Quellen, welche potenziell eine DOI enthalten könnten, eine DOI in den Zitationen angegeben haben. Dabei wird in Quellen, in denen es möglich ist, zwischen den unterschiedlichen Angaben einer DOI unterschieden. In ?? ist beispielsweise zu erkennen, dass in der CRAN CFF Liste 57 Pakete eine DOI in dem CFF angegeben haben und 40 Pakete keine DOI angegeben haben. Auffällig ist außerdem, dass in der preferred-citation_{preferred-citation} in ?? kein einziges Paket eine Collection DOI angegeben hat und ebenfalls nur ein Paket in den identifiers_{identifiers} eine DOI angegeben hat. Zusätzlich wird in ?? dargestellt, wie viele aller CFF-Dateien auf GitHub eine DOI angegeben haben. Der Aufbau ist dabei wie zuvor.

**Abbildung 14:** Verwendung von DOIs in allen CFF auf GitHub

Die Abbildungen zeigen für die CFF und preferred-citation_{preferred-citation} Quellen, jeweils für alle CFF-Dateien auf GitHub, wie oft eine DOI auf verschiedenen Arten in den Zitationen angegeben wurde.

Quelle	CRAN	PyPI	CFF	PyPI CFF	CRAN CFF
README	257	184	73	83	133
CFF	484	326	555	488	240
BIBTEX		768	1159	1159	

Tabelle 13: Durchschnittliche Zeit seit der letzten Aktualisierung

Die Tabelle gibt an, wie viel Zeit im Durchschnitt über alle Pakete seit der letzten Aktualisierung der Quelle vergangen ist. Die Zeit ist in Tagen angegeben.

Weitere Ergebnisse

Zusätzlich zu den Graphen wurden weitere Ergebnisse aus den Daten berechnet. Einerseits-Unter anderem wurde berechnet, wie viel Zeit im Durchschnitt seit der letzten Aktualisierung vom Tag der Datenbeschaffung einer Quelle vergangen ist. In ?? wird zusätzlich darauf eingegangen, mit welcher Frequenz die Quellen aktualisiert werden. Die Berechnung ist dabei jeweils nur für das CFF-Format, das BIBTEX-Format und die README-Dateien möglich, da in den anderen Quellen kein Zeitstempel ermittelt werden kann. Die Ergebnisse sind in ?? dargestellt. Für CRAN ist jeweils keine Zeit für BIBTEX angegeben, da keines der Pakete in den Listen die Quelle verwendet.

In der Tabelle lässt sich beispielsweise ablesen, dass die README über alle Pakete in der PyPI CFF Liste im Durchschnitt 83 Tage nicht aktualisiert wurde. Es ist außerdem auffällig, dass die README-Dateien am aktuellsten sind und die BIBTEX-Dateien am längsten nicht aktualisiert wurden.

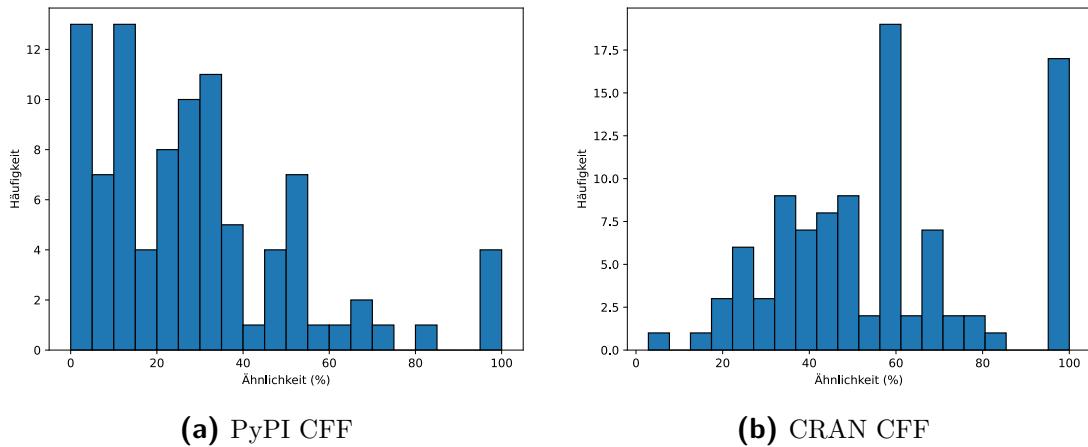


Abbildung 15: Übereinstimmung der Autoren in den Paketen

Das Histogramm zeigt für die PyPI CFF und CRAN CFF Listen, wie ähnlich die Autoren in den Paketen innerhalb der Quellen sind. Auf der x-Achse wird die Ähnlichkeit in Prozent angegeben. Die y-Achse stellt die absolute Anzahl der Pakete dar, welche die entsprechende Ähnlichkeit aufweisen.

In der ?? wird für die Listen PyPI CFF und CRAN CFF angegeben, wie ähnlich die Autoren in den Paketen innerhalb der Quellen sind. Für die weiteren Listen sind die Grafiken in ?? dargestellt. Dabei werden nur jene Autoren berücksichtigt, welche abgeglichen werden konnten. Es wird somit kein erneuter Abgleich zwischen den einzelnen Quellen anhand der Namen vorgenommen.

Die Histogramme sind jeweils in 20 Klassen unterteilt, sodass die Ähnlichkeit in 5 % Schritten angegeben wird. Für jedes einzelne Paket wurde die Ähnlichkeit der Autoren untereinander berechnet und in die entsprechenden Klassen eingesortiert. Die Häufigkeit ist dabei in absoluten Werten angegeben.

In der ?? ist beispielsweise zu erkennen, dass vier Pakete in der PyPI CFF Liste eine Ähnlichkeit von 100 % aufweisen. Das bedeutet, dass viermal folgender Fall aufgetreten ist: Alle Quellen eines Pakets haben die gleichen Autoren gelistet, wobei kein Autor in einer Quelle fehlt und auch kein Autor in einer Quelle zu viel gelistet ist. Dabei muss beachtet werden, dass dies davon abhängt, ob der Abgleich erfolgreich war oder nicht. Außerdem ist auffällig, dass 14 der Pakete in der PyPI CFF Liste eine Ähnlichkeit aller Autoren in den Quellen von weniger als 5 % aufweisen.

4.2 Ergebnisse mit Zeitverlauf

In diesem Abschnitt werden anders als zuvor die Daten mit Zeitbetrachtung analysiert und nicht lediglich die jeweils neueste Version untersucht. Dadurch ist es

möglich, Veränderungen in der Zeit über einen Zeitraum zu betrachten und zu analysieren. Jedoch ist dies nur für die Quellen CFF, preferred-citationpreferred-citation CFF und die README-Dateien möglich, da in den anderen Quellen keine Zeitinformation Zeitinformationen vorhanden sind.

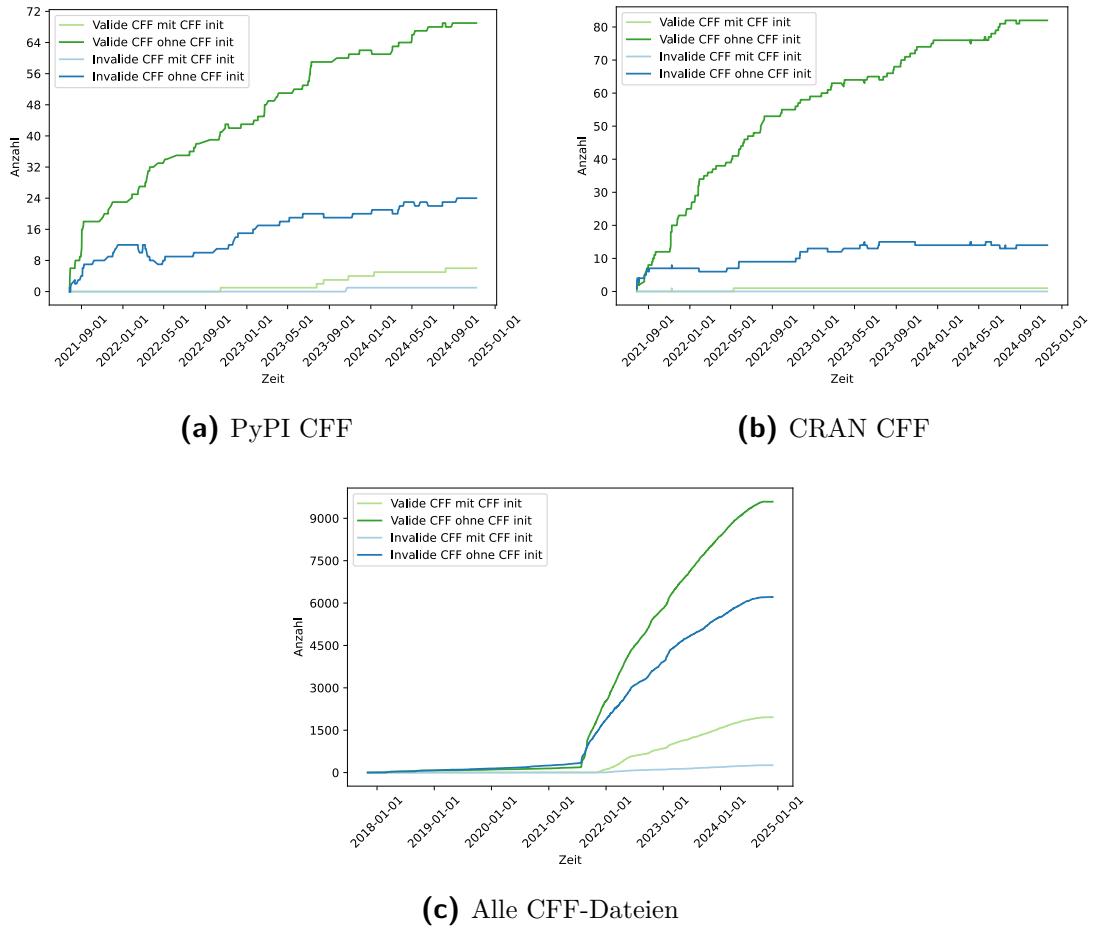
Validität der CFF-Dateien

Die ?? zeigt für die PyPI CFF und CRAN CFF Listen, sowie für alle CFF-Dateien auf GitHub, wie viele der CFF-Dateien valide sind. In ?? sind die Ergebnisse für die weiteren Listen dargestellt. In den Graphen ist jedes Repository nur einmal betrachtet worden. Falls ein Repository zu Beginn eine invalide CFF-Datei in dem Repository hatte und anschließend diese zu einer validen Datei geändert wurde, sinkt die Anzahl bei den invaliden und bei den validen steigt sie zu diesem Zeitpunkt. Dadurch ist gewährleistet, dass Änderungen korrekt dargestellt werden und keine Repositorys doppelt betrachtet werden.

In ?? ist beispielsweise zu erkennen, dass Anfang 2022 viele Repositorys von invaliden zu validen CFF-Dateien gewechselt sind. Außerdem lässt sich ablesen, dass die validen CFF-Dateien zu jedem Zeitpunkt mehr sind als die invaliden. In ?? ist hingegen zu erkennen, dass zu Beginn des Graphen die Anzahl der invaliden höher ist als die der validen. Zudem lässt sich anhand der Graphen ablesen, zu welchem Zeitpunkt wie viele Repositorys eine CFF-Datei enthalten haben. Besonders auffällig ist dabei in ??, dass die Anzahl der Repositorys mit einer CFF-Datei auf GitHub ab 2021 steil angestiegen ist. Dies stimmt überein mit der Beobachtung, dass ab 2021 die ersten Pakete in der PyPI CFF und CRAN CFF Liste eine CFF-Datei hinzugefügt haben. Außerdem lässt sich erkennen, dass ab 2018 die ersten GitHub-Repositorys eine CFF-Datei enthalten haben.

Hinzugefügte und entfernte Autoren

Die ?? zeigt für alle fünf Listen, wie viele Autoren in den einzelnen Quellen hinzugefügt und gelöscht wurden. Dabei werden ausschließlich genannten Autoren in dem CFF, BIBTEX und der README-Datei betrachtet, da nur bei diesen Quellen eine zeitliche Betrachtung möglich ist. Außerdem wird in der Abbildung gezeigt, wie viele Autoren nachträglich den Dateien hinzugefügt wurden, nachdem die Datei das erste Mal erstellt wurde. Die Informationen der Abbildungen werden ebenfalls erneut ohne die README Autoren dargestellt, da diese sehr-vielen Änderungen aufgrund der verwendeten NER unterliegen.

**Abbildung 16:** Validität der CFF-Dateien über die Zeit

Die Abbildung zeigt für die PyPI CFF und CRAN CFF Listen, sowie für alle CFF-Dateien auf GitHub, wie viele der CFF-Dateien valide sind. Die x-Achse stellt die Zeit dar und die y-Achse die Anzahl der validen und invaliden CFF-Dateien. Dabei wird außerdem in Repositorys, welche *cffinit* verwenden und Repositorys, welche dies nicht verwenden unterschieden.

Um zu betrachten, wie viele Autoren hinzugefügt und anschließend wieder entfernt worden sind und dabei möglichst Namensänderungen nicht als neue Autoren zu betrachten, wurde neben dem Abgleich aus der Datenbeschaffung in diesem speziellen Fall ein weiterer Abgleich durchgeführt. Dieser Abgleich wird ausschließlich dann durchgeführt, falls die Datenbeschaffung keine Übereinstimmung für den jeweiligen Autor ergeben hat. In diesem Fall wird ein erneuter Abgleich des Autors anhand des Namens durchgeführt. Dies führt dazu, dass, falls ein Abgleich in der Datenbeschaffung für einen Autor nicht erfolgreich war, dieser bei einer Namensänderung in der Grafik als gelöschter und hinzugefügter Autor betrachtet wird. Der erneute Abgleich über den Namen kann dies nicht verhindern, allerdings sorgt er dafür, dass auch nicht abgegliche Autoren in der Grafik als gelöschte und hinzugefügte Autoren betrachtet werden können. Zusätzlich ermöglicht dies im späteren Verlauf eine Analyse darüber, wie lange ein Autor durchschnittlich in einer Quelle genannt wird.

In ?? ist zu erkennen, dass in der PyPI CFF Liste die meisten Autoren in den betrachteten Quellen genannt werden. Außerdem ist auffällig, dass in der CRAN und der PyPI Liste in den Quellen nur Autoren hinzugefügt und keine gelöscht wurden. Im Allgemeinen ist deutlich, dass kaum Autoren, nachdem sie hinzugefügt wurden, wieder entfernt werden. In Kombination mit ?? ist zu erkennen, dass die meisten Autoren direkt hinzugefügt werden, sobald die Datei erstellt wird. Beispielsweise wurde in der CRAN und PyPI Liste in der CFF und der BIBTEX-Datei kein Autor hinzugefügt, nachdem die Datei erstellt wurde. In der PyPI CFF Liste wurden fünf Autoren nachträglich hinzugefügt, wobei hier berücksichtigt werden muss, dass dies auch aufgrund von Namensänderungen geschehen sein kann. Außerdem muss berücksichtigt werden, dass die CFF-Datei erst seit 2021 in den meisten Fällen genutzt wird und beim Hinzufügen der Datei alle Autoren zu dem Zeitpunkt eingetragen werden. Dadurch werden diese Autoren ausschließlich in der ?? dargestellt.

Durchschnittliche Verweildauer der Autoren

Die ?? zeigt die durchschnittliche Zeit in Tagen, wie lange ein Autor in einer bestimmten Quelle genannt wird. Dabei wird die gleiche Berechnung wie für die hinzugefügten und gelöschten Autoren durchgeführt. Aus diesem Grund gelten die gleichen Einschränkungen wie zuvor. Es lässt sich beispielsweise in der Tabelle erkennen, dass in der PyPI CFF Liste die Autoren im Durchschnitt 333 Tage in der CFF-Datei genannt werden, falls sie entfernt wurden. Es werden keine Autoren berücksichtigt,

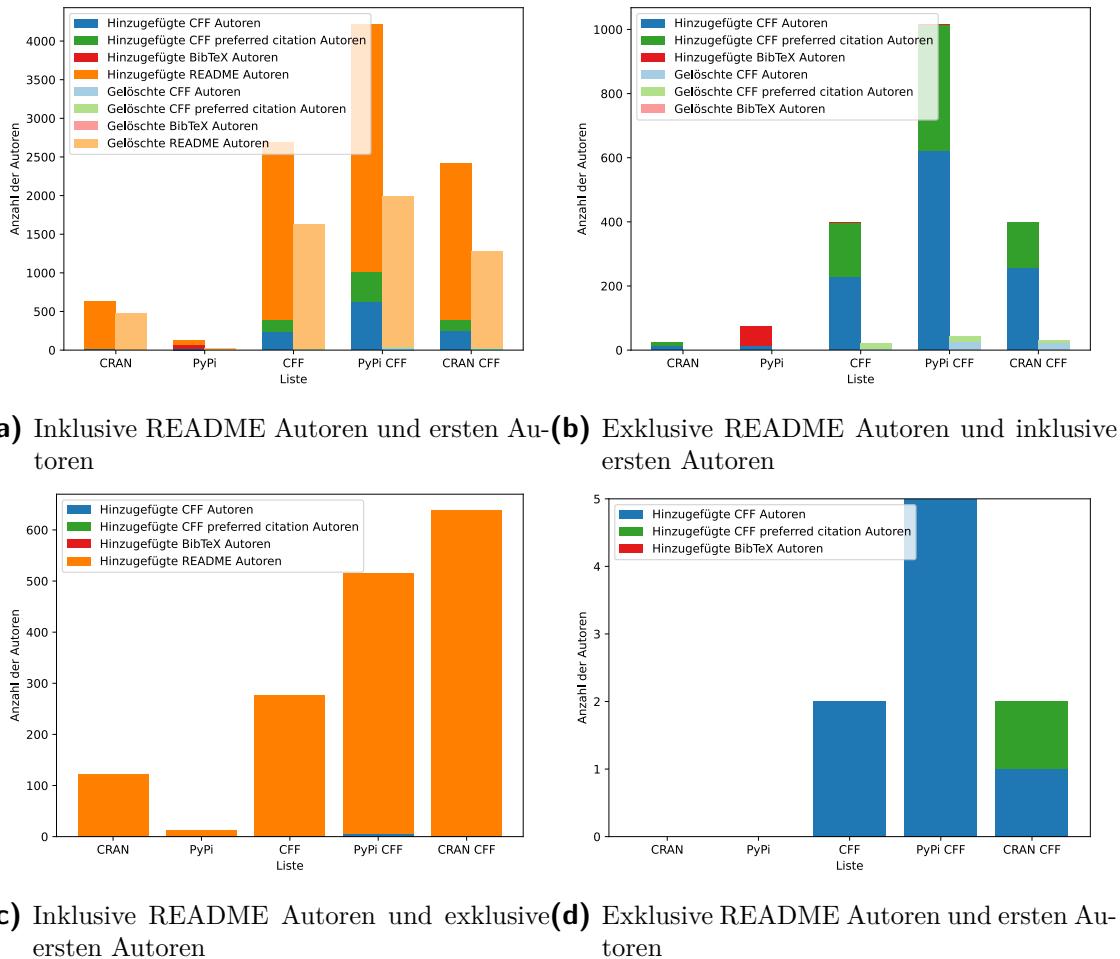


Abbildung 17: Hinzugefügtes und gelöschtes Autoren

Die beiden Abbildungen ?? und ?? zeigen wie viele Autoren über die Zeit in den Quellen hinzugefügt und gelöscht wurden. ?? inkludiert Autoren in der README-Datei, ?? exkludiert diese. Die Abbildungen ?? und ?? zeigen ausschließlich die hinzugefügten Autoren, wobei die Autoren, die zu Beginn direkt in die Quelle eingetragen wurden, nicht berücksichtigt werden.

Quelle	CRAN	PyPI	CFF	PyPI	CFF	CRAN	CFF
README	730	398	174		159		187
CFF	∞	∞	1		333		198
	∞		512		512		140
preferred-citation	<i>preferred-citation</i>						
CFF							
BIBTEX		∞		∞		∞	

Tabelle 14: Durchschnittliche Verweildauer der Autoren in den Quellen
Die Tabelle gibt an, wie lange im Durchschnitt über alle Pakete ein Autor in den Quellen genannt wird. Die Zeit ist in Tagen angegeben.

welche nie entfernt wurden. Im Gegensatz dazu werden in ?? ebenfalls Autoren berücksichtigt, welche nie entfernt wurden. Dabei wird der 02.12.2024 als Enddatum genommen.

In der ?? ist zu erkennen, dass einige Zellen nicht ausgefüllt sind und andere wiederum mit ∞ gekennzeichnet sind. Leere Zellen sind darin begründet, dass in der Liste kein Paket die Quelle verwendet und dadurch keine Zeit berechnet werden kann. Zellen, welche ∞ enthalten, sind dadurch begründet, dass in der Liste kein einziges Paket in der jeweiligen Quelle einen Autor, nachdem er hinzugefügt wurde, wieder entfernt hat. Beispielsweise hat kein Paket in der CRAN Liste einen Autor, nachdem er zur CFF hinzugefügt wurde, wieder entfernt. Außerdem wurden in der Liste keine Autoren in eine BIBTEX-Datei hinzugefügt, wodurch die Zelle leer ist.

Eine weitere Auffälligkeit ist in der CFF Liste zu erkennen. Hierbei ist eine durchschnittliche Verweildauer für die in der CFF eingetragenen Autoren von eins angegeben. Dies ist auf zwei unterschiedliche Pakete zurückzuführen. In dem Paket *pytorch-lightning* wurde ein Autor hinzugefügt, welcher zwei Tage später in der Schreibweise verändert wurde. Dabei konnte die erste Schreibweise des Autors nicht abgeglichen werden, wodurch es zu einer Löschung des Autors und einer Hinzufügung des Autors in der neuen Schreibweise kam. In dem zweiten Paket *torch-geometric* wurden wahrscheinlich fälschlicherweise in einer Version die beiden genannten Autoren in der CFF durch „Mona Lisa“ und „Hew Bot“ ersetzt. Noch am selben Tag wurden wieder die Originalautoren hinzugefügt. Dadurch ergibt sich allerdings für die zwei Autoren eine Verweildauer in der CFF von null Tagen. Insgesamt ergibt das $\frac{2+0+0}{3} \approx 1$, wodurch der Wert in der Zelle begründet ist. Nach manueller Prüfung wäre dies somit ein Fall für das ∞ Zeichen, da die eigentlichen Autoren bis zum Tag des Schreibens nicht entfernt worden sind.

Quelle	CRAN	PyPI	CFF	PyPI	CFF	CRAN	CFF
README	1082	1088	364		394		398
CFF	1021	257	850		651		745
	1021		676		559		723
<i>preferred-citation</i>	<i>preferred-citation</i>						
CFF							
BIBTEX		1197	1200		1200		

Tabelle 15: Durchschnittliche Verweildauer der Autoren in den Quellen inkl. Autoren, welche nie entfernt wurden

Die Tabelle gibt an, wie lange im Durchschnitt über alle Pakete ein Autor in den Quellen genannt wird. Falls ein Autor nicht entfernt wurde, wurde der 02.12.2024 als Enddatum festgelegt. Die Zeit ist in Tagen angegeben.

Quelle	CRAN	PyPI	CFF	PyPI	CFF	CRAN	CFF
README	125	149	30		28		45
CFF	24	132	95		98		91
BIBTEX		676					

Tabelle 16: Durchschnittliche Zeit zwischen der Aktualisierung der Quellen

Die Tabelle gibt an, wie oft im Durchschnitt über alle Pakete die Quellen aktualisiert werden. Die Zeit ist in Tagen angegeben.

Weitere Ergebnisse

In ?? wird dargestellt, mit welcher Frequenz die einzelnen Quellen in den unterschiedlichen Paketen der Listen aktualisiert werden. Dabei ist nur für die PyPI Liste ein Wert für BIBTEX angegeben, da in den anderen Listen entweder kein Paket eine BIBTEX-Datei verwendet hat oder diese nie aktualisiert wurde.

In der Tabelle lässt sich beispielsweise ablesen, dass die README-Datei am häufigsten aktualisiert wird. Beispielsweise wurde diese in der PyPI CFF Liste im Durchschnitt alle 28 Tage aktualisiert. Hingegen wird die BIBTEX-Datei in der PyPI Liste im Durchschnitt nur alle 676 Tage aktualisiert.

5 Diskussion

In diesem Kapitel wird die Arbeit diskutiert und die Ergebnisse interpretiert. Zuerst wird auf die grundlegenden Limitierungen ~~der~~ dieser Arbeit eingegangen und wie diese die Ergebnisse beeinflussen. Diese Limitierungen können nicht vollständig behoben werden. Anschließend erfolgt eine Diskussion über die Qualität des Abgleichs der Autoren. Daraufhin wird diskutiert, was Softwareentwickler leisten müssen, um zitiert zu werden. Natürlich lassen sich keine allgemeinen Aussagen für alle Pakete treffen, da dies individuell unterschiedlich ist. Zuletzt wird erörtert, wie sorgfältig die Autoren in den untersuchten Listen gepflegt sind.

5.1 Limitierungen

~~In diesem Kapitel wird strukturiert auf folgende Limitierungen eingegangen, welche bei der Erstellung der Arbeit aufgetreten sind:~~

- ~~Git Statistik: Doppelte Autoren innerhalb der Git-Autoren.~~
- ~~Autoren ohne Commits: Autoren, welche als Autor genannt werden, aber keinen Quellcode geschrieben haben.~~
- ~~Verlinkung auf andere Quellen: In manchen Fällen wird in Quellen auf eine andere Quelle verwiesen.~~
- ~~Ausführungszeit der Datenbeschaffung: Die Ausführungszeit ist sehr lang.~~

Git Statistik

In ?? wurde beschrieben, dass Autoren in Git ihren Namen und ihre E-Mail-Adresse ohne Einschränkungen eigenständig eintragen können. Dies bedeutet ebenfalls, dass ein Autor im Verlauf der Zeit seinen Namen und/ oder die E-Mail-Adresse ändern kann. Dadurch kann ein und dieselbe Person mit unterschiedlichen Git-Namen unterschiedlich viele Commits erstellt haben. Dies ist für die Masterarbeit nicht erwünscht, da möglichst ein Autor nur einmal in den Daten vorkommen soll und

sämtliche Commits diesem Autor angerechnet werden sollen. In der Datenbeschaffung wurde versucht, dieses Problem zu lösen, indem die E-Mail-Adresse als eindeutiger Identifikator genutzt wird und gleiche E-Mail-Adressen zu einem Autor zusammengefasst werden. Allerdings behebt dies nicht das Problem vollständig, da ein Autor seine E-Mail-Adresse ebenfalls ändern kann und diese nicht erneut abgeglichen wird.

Ein Abgleich, wie in ?? beschrieben, bei dem keine Unterscheidung von Namensvettern möglich ist, ist ebenfalls nicht möglich, da dies die Ergebnisse verfälschen könnte.

WW

Dies ist darin begründet, dass dadurch die Ergebnisse verfälscht werden könnten, da beispielsweise Autoren mit dem gleichen Namen zusammengefasst werden würden, obwohl es sich um unterschiedliche Autoren handelt. Die anderen Probleme, welche in beschrieben wurden, sind ebenfalls nicht zu vernachlässigen. Eine Möglichkeit, dieses Problem zu lösen, wäre die Verwendung der GitHub-API, welche automatisch die Git-Autoren anhand der in GitHub eingetragenen E-Mail-Adressen zusammenfasst. Ein weiterer Vorteil, der durch die Nutzung der GitHub-API bestehen würde, ist, dass dabei die Benutzernamen der GitHub-Benutzer abgefragt werden könnten. Diese werden häufig in der README und der Beschreibung mit einem @-Zeichen angegeben, um auf diese zu verweisen. Die verwendete NER erkennt die Benutzernamen, allerdings können sie häufig nicht mit den Git-Daten abgeglichen werden, da die Benutzernamen meistens nicht dem richtigen Namen oder der E-Mail-Adresse entsprechen. Die Verwendung der API ist jedoch nur mit viel Zeit möglich, da sie für jedes Paket einzeln abgefragt werden müsste und dadurch schnell das Ratenlimit von GitHub erreicht werden würde. Eine weitere Möglichkeit, welche bereits durch *git-quick-stats* verwendet wird, ist das Zusammenfassen der Autoren mittels einer *.mailmap*-Datei (**chacon_git_2024-1**). In dieser Datei kann eingetragen werden, dass zwei E-Mail-Adressen zusammengefasst werden sollen.

Durch die beschriebene Limitierung kommt es beispielsweise vor, dass in *torch* sechsmal der Autor „Edward Yang“ mit unterschiedlichen E-Mail-Adressen vorkommt, obwohl es sich um die gleiche Person handelt. Der erste Eintrag des Autors hat 1.925 Commits, alle weiteren Einträge zusammen haben nochmal 1.282 Commits getätigt. Diese Limitierung in der Datenbeschaffung verfälscht die Gesamtergebnisse. Aber auch bei einem erneuten Abgleich würde dies die Ergebnisse durch die Ungenauigkeit des Abgleichs verfälschen.

Autoren ohne Commits

Eine weitere Limitierung, welche bereits im Verlauf ~~der dieser~~ Masterarbeit häufiger thematisiert wurde, ist, dass Autoren als Autor genannt werden können und auch sollten, obwohl sie keinen Quellcode geschrieben haben. Eine reine Betrachtung der geleisteten Arbeit anhand der Änderungen am Quellcode, wie sie in ~~der dieser~~ Masterarbeit durchgeführt wurde, ist daher nicht ausreichend, um das gesamte Spektrum der Arbeit von Autoren zu erfassen. Aus diesem Grund kommt es in ~~der dieser~~ Arbeit dazu, dass Autoren im Abgleich nicht erkannt werden, da sie keinen Quellcode geschrieben haben und somit nicht als Git-Autoren gelistet sind. Dadurch können Ergebnisse wie beispielsweise in ?? schlechter ausfallen, da nur auf Git-Autoren geprüft wird. Es werden keine Autoren berücksichtigt, welche beispielsweise an der Dokumentation gearbeitet haben oder an der Organisation des Projekts beteiligt waren.

Verlinkung auf andere Quellen

In einigen Quellen werden Autoren durch die Entwickler nicht direkt genannt, sondern es wird auf eine andere Quelle verwiesen, in welcher die Autoren genannt werden. Beispielsweise wird in *pandas* in der CFF-Datei als Name „The pandas development team“ und anschließend die Webseite <https://pandas.pydata.org/about/team.html> angegeben. Auf dieser Seite sind anschließend alle aktiven Autoren gelistet. In ~~der dieser~~ Masterarbeit stehen lediglich die Autoren in der CFF-Datei mit Vor- und Nachnamen im Fokus. Es erfolgt keine zusätzliche Analyse anderer Quellen, die möglicherweise erwähnt sind. Daher kann es wie im Fall von *pandas* passieren, dass Autoren, die nicht direkt genannt sind, jedoch auf einer anderen Seite gelistet werden, unberücksichtigt bleiben. Dies schränkt die Ergebnisse ~~der dieser~~ Masterarbeit ein, da nicht alle Autoren einbezogen sind. Eine Betrachtung der verlinkten Seiten würde jedoch einen erheblichen Mehraufwand bedeuten, da für jedes Paket die verlinkten Seiten analysiert werden müssten. Außerdem sind die verlinkten Seiten unterschiedlich aufgebaut, sodass die Komplexität dadurch ebenfalls gesteigert wäre. Aus diesem Grund wurde in ~~der dieser~~ Masterarbeit darauf verzichtet, die verlinkten Seiten zu analysieren und mit weniger genannten Autoren gearbeitet.

Ausführungszeit der Datenbeschaffung

Eine weitere Limitierung ist die Laufzeit der Datenbeschaffung. Diese wird besonders durch den Aufruf von *git-quick-stats* beeinflusst, da das Programm für große

Repositorys eine lange Laufzeit hat. Beispielsweise benötigt die Ausführung des Programms für das Paket *pytorch* 1 Minute und 22 Sekunden. Außerdem hat die NER für die README und die Beschreibung eine hohe Laufzeit, weswegen für die README nur die letzten 50 Commits betrachtet werden. Die Laufzeit der NER beträgt für die README von *pytorch* 3,3588 Sekunden. Bei einer Ausführung der NER für 50 Versionen der README entspricht dies bereits einer Laufzeit von 2 Minuten und 80 Sekunden.

Um die Laufzeit weiter zu reduzieren, beispielsweise um sämtliche CFF Pakete auf GitHub analysieren zu können, wurde auf Faktoren, welche die Laufzeit erheblich erhöhen, verzichtet. Aus diesem Grund wurden bei der Analyse ausschließlich die CFF-Dateien betrachtet. Zudem wurden für den Abgleich die Git-Autoren benötigt, welche ausschließlich in der neuesten Version beschafft wurden, sodass *git-quick-stats* nur einmal aufgerufen werden musste. Dadurch konnte die Laufzeit der Datenbeschaffung für die gesamte CFF Liste mit 20.870 Einträgen auf dem internen HPC Server der Hochschule Wismar auf 55 Stunden reduziert werden. Der Server hat zwei Intel Xeon Gold 6346 Prozessoren mit jeweils 3,1 GHz je 16 Kerne verbaut. Bei Betrachtung aller Quellen hätte dieser Prozess erheblich mehr Zeit in Anspruch genommen. Allerdings muss berücksichtigt werden, dass das Programm nicht täglich, sondern einmalig ausgeführt wird, um die Daten zu einem bestimmten Zeitpunkt zu beschaffen. Falls ausgelassene Daten für künftige Arbeiten benötigt werden, könnten diese dem Prozess erneut ergänzt werden.

5.2 Wie gut können Autoren untereinander abgeglichen werden?

In den Tabellen ??, ??, ??, ??, ??, ??, ?? und ?? wurden die Ergebnisse des Abgleichs der Autoren dargestellt. In ?? ist aufgefallen, dass viele Autoren in den Python Quellen keine Personen sind. Dies ist darauf zurückzuführen, dass in den Quellen häufig Organisationen als Autoren genannt werden und keine individuellen Personen aufgeführt werden. Dies ist besonders der Fall, da die Top-100-Listen betrachtet wurden, welche häufig von Organisationen verwaltet werden und nicht von einzelnen Entwicklern gepflegt werden. Beispielsweise sind in der PyPI Liste vier Pakete enthalten, welche bereits den Namen Google enthalten. In allen vier Paketen sind in den Quellen PyPI Maintainer und Python Autoren keine Personen, sondern ausschließlich „gcloudpypi“, „google_opensource“ und „Google LLC“ genannt. Hier lässt sich diskutieren, ob eine Nennung von individuellen Personen dennoch erfolgen sollte, auch wenn sie beispielsweise bei einer Organisation einem Unternehmen wie

Google angestellt sind und für diese arbeiten. Dies soll allerdings kein Thema für diese Arbeit sein.

In ?? ist aufgefallen fällt auf, dass die README und die Beschreibung schlechte F1-Scores haben. Dies liegt daran, dass die NER viele Ergebnisse liefert, unter anderem FP, welche anschließend primär falsch zugeordnet werden. Ebenfalls sind viele FN Ergebnisse enthalten, da die NER ebenfalls Benutzernamen erkennt, wie bereits erläutert wurde. Außerdem ist aufgefallen fällt auf, dass viele TN in den Ergebnissen enthalten sind. Dies ist darauf zurückzuführen, dass die verwendete NER viele Entitäten erkennt, welche nicht relevant sind, da es sich beispielsweise nicht um Personen handelt. Dies ist besonders verwunderlich, dass da in der Methodik beschrieben wurde, dass nur Personen durch die NER erkannt werden sollen nur Personen erkennen soll.

Zudem ist in ?? aufgefallen, dass die BIBTEX-Quelle den schlechtesten F1-Score hat. Dies ist darin begründet, dass nur die ersten beiden Autoren in jeder BIBTEX-Datei betrachtet wurden. Da in allen Listen insgesamt nur vier BIBTEX-Dateien enthalten sind, ist die Anzahl der betrachteten Autoren auf maximal acht Autoren begrenzt. Wie in den Tabellen ?? und ?? erkennbar ist, sind insgesamt 63 in den BIBTEX-Dateien enthalten. In zwei der 4 Dateien konnten alle Autoren, was in diesen Fällen jeweils ein Autor entspricht, nicht abgeglichen werden. In den anderen beiden Dateien sind mehr Autoren enthalten, wobei viele der Autoren richtig abgeglichen werden konnten. Bei einer Betrachtung aller Autoren wäre der F1-Score für die BIBTEX-Quelle besser ausgefallen. In diesem Fall verschlechtern die beiden nicht abgeglichenen Autoren aufgrund der insgesamt geringen Anzahl an betrachteten Autoren den F1-Score erheblich.

Außerdem ist aufgefallen fällt auf, dass insgesamt viele FP in den Ergebnissen enthalten sind. Diese sind dadurch begründet, dass das Keyword *in* in ?? verwendet wird. In Bei der manuellen Überprüfung der Ergebnisse ist aufgefallen fällt auf, dass in einigen Git-Autorenlisten Autoren enthalten sind, welche einen Namen mit nur einem oder zwei Buchstaben haben. In diesen Fällen ist es möglich, fast jeden Autor aus der Quelle mit diesem speziellen Git Autor abzugleichen, da der Autor der Quelle nur diesen Buchstaben in seinem Namen enthalten haben muss. Falls über keine weiteren Attribute der Abgleich erfolgen kann, bedeutet dies immer, dass ein falscher Abgleich stattfindet. Dieses Problem entsteht grundlegend dadurch, dass es kaum Einschränkungen bei der Wahl des Namens und der E-Mail-Adresse in Git gibt. Außerdem führt das Problem im Allgemeinen zu schlechteren Ergebnissen im Abgleich, da die zu untersuchenden Daten nicht standardisiert sind.

*Sehr viel „fällt auf“ auf dieser Seite
ersetzen?*

Im Allgemeinen konnte gezeigt werden, dass der Abgleich der Autoren gut funktioniert hat, da ein F1-Score von über 0,9 „~~welcher erreicht wurde~~“ als gut zu bewerten ist. Außerdem wurde in diesem Abschnitt diskutiert, warum einige Quellen schlechtere Ergebnisse haben. Hierbei wurde deutlich, dass die schlechteren Ergebnisse primär nicht durch den Abgleich verursacht wurden, sondern durch andere Faktoren, wie beispielsweise die NER oder die Anzahl der manuell betrachteten Autoren in den BIBTEX-Dateien. Bei einer Betrachtung des F1-Scores ohne diese Gegebenheiten würde dieser nochmals verbessert werden.

5.3 Was muss ein Softwareentwickler leisten, um als Autor genannt zu werden?

Bei der Beantwortung der Frage muss beachtet werden, dass die Aussagen nur allgemein getroffen werden können und nicht für alle Pakete gelten. Einzelne Pakete können natürlich unterschiedlich sein und andere Anforderungen an die Autoren stellen. Nur weil ein Paket alle Autoren nennt, welche mindestens einen Commit getätigt haben, bedeutet das nicht, dass ein anderes Paket ebenfalls diesen Ansatz verfolgt. Die Ergebnisse in ?? zeigen im Allgemeinen aggregierte Werte für alle Pakete einer Liste und aus diesem Grund wird die Frage ebenfalls allgemeingültig diskutiert.

Die ?? zeigt, dass Autoren mit vielen Commits über alle Pakete hinweg häufiger als Autoren genannt sind. Sie zeigt, dass eine erhöhte Chance besteht, falls eine Person unter den Top-10-Autoren ist, dass diese Person auch als Autor genannt wird. Dies ist jedoch nicht garantiert, da die Abbildung gleichzeitig zeigt, dass nur etwa 50 % der Autoren mit den meisten Commits tatsächlich als Autor aufgeführt sind. Außerdem zeigt sie, dass in einigen Paketen die Autoren mit den meisten Commits oder geänderten Zeilen auch gar nicht als Autor genannt werden können. Aus diesen Gründen lässt sich annehmen, dass viel Arbeit in einem Projekt ein guter Ansatz ist, um als Autor genannt zu sein. Dies jedoch keine Garantie dafür genannt zu werden und in den meisten Fällen werden weitere Schritte benötigt, um tatsächlich aufgeführt zu werden.

Ein möglicher Schritt, welcher jedoch nicht umsetzbar ist, ist bei der Gründung des Paketes beteiligt zu sein. Dies geht aus der ?? hervor. Sie zeigt das Problem, dass die meisten Autoren direkt zu Beginn genannt werden und anschließend kaum weitere Autoren hinzugefügt werden und somit die Autorenliste nicht aktiv gepflegt wird. Insgesamt wurden in allen Paketen der untersuchten Listen nur neun Autoren, der CFF oder BIBTEX-Datei nachträglich hinzugefügt. ?? zeigt jedoch, dass in den Da-

teien viel mehr Autoren insgesamt enthalten sind. Hierbei muss allerdings beachtet werden, dass die CFF-Datei erst seit 2021 vermehrt verwendet wird, wie aus ?? hervorgeht. Dadurch sind erst drei Jahre vergangen, in welchen Autoren hinzugefügt werden konnten. Und diese Autoren müssten in den drei Jahren auch aktiv am Projekt beteiligt gewesen sein, um als Autor genannt zu werden. Ebenfalls spiegelt die ?? dieses Verhalten wider. Hier wird deutlich, dass viele genannte Autoren keinen Commit in den letzten Jahren getätigt haben. Jedoch muss berücksichtigt werden, dass inaktive Repositorys mit in diese Statistik einfließen, welche ebenfalls Autoren enthalten, welche nicht mehr aktiv am Projekt beteiligt sind, da das Projekt eingestellt wurde. Auf die Inaktivität von Autoren und deren **pflege**-**Pflege** wird im nächsten Abschnitt genauer eingegangen.

5.4 Wie gut werden Autoren in den einzelnen Quellen gepflegt?

In ?? wurde gezeigt, dass viele Autoren gar nicht unter den Top-100-Git-Autoren sind. Dies bedeutet, dass viele der genannten Autoren gar nicht mehr aktiv am Projekt beteiligt sind. Dies kann verschiedene Gründe haben, wie beispielsweise, dass die Autoren das Projekt verlassen haben. Außerdem muss beachtet werden, dass Autoren, welche nicht in der Datenbeschaffung abgeglichen werden konnten, hier ebenfalls enthalten sind. Die Abbildung deutet allerdings bereits darauf hin, dass Autoren, sobald sie einmal eingetragen wurden, nicht mehr entfernt werden, obwohl sie nicht mehr aktiv am Projekt beteiligt sind. Zudem zeigt es, in Verbindung mit ??, dass Autoren mit vielen Commits ebenfalls kaum genannt werden, was die Vermutung bestätigt, dass die Autorenlisten in den meisten Fällen nicht aktiv gepflegt werden.

Ein weiterer Indikator dafür ist die ?. Hier wird deutlich, dass viele genannte Autoren in den letzten Jahren keinen Commit getätigt haben. Dies lässt sich allerdings dadurch relativieren, dass die Statistik ebenfalls Pakete enthält, welche nicht mehr aktiv entwickelt werden, was allerdings bei den Top-100-Listen unwahrscheinlich ist. Des Weiteren ist die hohe Anzahl der invaliden CFF-Dateien, welche in ?? und ?? deutlich werden, ein Indikator dafür, dass die Pflege der Autoren den Entwicklern der Pakete nicht besonders wichtig zu scheinen sei.

Auch zeigt die Häufigkeit, mit der die Quellen aktualisiert werden, dass scheinbar kein großes Interesse darin besteht, die Autorenlisten zu pflegen. Aus ?? wird deutlich, dass zwei der drei untersuchten Quellen in fast jeder Liste durchschnittlich das letzte Jahr nicht aktualisiert wurden. Die README wird dabei öfter aktualisiert.

siert, wobei berücksichtigt werden muss, dass in der README nicht nur Autoren, sondern auch in vielen Fällen beispielsweise die Dokumentation vorhanden ist. Außerdem muss berücksichtigt werden, dass ein Jahr in der Softwareentwicklung keine lange Zeit ist und neue Autoren innerhalb dieser Zeit kaum hinzugefügt werden können, da ein Einarbeiten und etablieren in ein großes Softwareprojekt innerhalb eines Jahres schwer möglich ist. Dahingegen ist dies bei der BibTeX Quelle anders, da hier die letzte Aktualisierung zwei bis drei Jahre zurückliegt in der die Autorenliste ggf. um weitere Autoren ergänzt hätte werden können.

?? zeigt, dass die Übereinstimmung der Autoren über die Quellen hinweg gering ist. Besonders in der PyPI CFF Liste wird dies deutlich. Dabei werden allerdings auch Quellen wie die README betrachtet, in welcher bei vielen Paketen keine oder kaum Autoren genannt werden. Auch ist die Abbildung erneut stark abhängig von dem Abgleich in der Datenbeschaffung. Allerdings lässt sich hier erneut erkennen, dass die Autoren in den Quellen nicht automatisch gepflegt werden. Des Weiteren zeigen die Tabellen ?? und ??, sowie die ??, dass einmal hinzugefügte Autoren in den meisten Fällen nicht mehr entfernt werden. Diese Tatsache ist dabei nichts Negatives, da die Autoren Arbeit in den Paketen geleistet haben, allerdings sollten Autoren, welche aktuell das Paket aktiv pflegen ebenfalls genannt werden.

Im Allgemeinen lässt sich sagen, dass die Autoren in den betrachteten Listen nicht aktiv gepflegt werden und besonders in den meisten Fällen keine automatischen Prozesse vorhanden sind, welche die Autorenlisten aktualisieren. Dies könnte unter anderem daran liegen, dass viele der Pakete nicht in der Wissenschaft entstanden sind, sondern beispielsweise in Unternehmen, bei denen die Pflege der Autorenlisten nicht so wichtig ist, sondern die Nennung des Unternehmens im Fokus steht.

Eine zusätzliche Auffälligkeit, welche die Pflege und Nennung der Autoren indirekt betrifft, zeigt ?. Hier wird deutlich, dass in vielen Fällen, in denen eine ~~preferred citation~~preferred-citation angegeben ist, diese auf ein Paper verweist und nicht auf die Software. Falls Autoren von wissenschaftlichen Arbeiten ausschließlich diese Referenz zitieren, stellt dies ~~ist einen~~ Verstoß gegen das Prinzip der Wichtigkeit dar, welches in ?? beschrieben wurde. Dies könnte durch die Autoren von Software verhindert werden, indem sie keine ~~preferred citation~~preferred-citation angeben, sondern ausschließlich die Software und weitere Referenzen in der CFF als ~~references~~references angeben. Diese Referenzen wurden in ~~der dieser~~ Masterarbeit allerdings nicht untersucht.

bringt zu
 lockt
 "keine
 Priorität
 hat" z.B.
 auf jeden
 Fall "so"
 streichen

6 Fazit und Ausblick

In diesem Kapitel wird ein Rückblick auf die Masterarbeit gegeben. Es werden wichtige Themen aufgegriffen und ein Ausblick auf Arbeiten gegeben, die folgen könnten.

6.1 Fazit

In ~~der~~ dieser Masterarbeit wurde Software entwickelt, die es erlaubt, für eine beliebige Liste an CRAN- oder PyPI-Repositorys Autorenangaben aus verschiedenen Datenquellen zu extrahieren, diese im Anschluss zu aggregieren und graphisch aufzuarbeiten. Um ein Verständnis für die Thematik zu erlangen, wurde zu Beginn eine Literaturrecherche zur Autorenrolle in OSS durchgeführt. Außerdem wurden Prinzipien aufgezeigt, die die Bedeutung der Angabe von Autoren in OSS verdeutlichen. Anschließend wurden Grundlagen für die spätere Datenbeschaffung erarbeitet. Hierbei wurde unter anderem auf Software-Verzeichnisse und die Paketverwaltung eingegangen, welche als Datenquellen für die Autorenangaben dienen. Zudem wurden unterschiedliche Zitierformate für Software und wie diese ausgewertet werden können, betrachtet. Im weiteren Verlauf wurde auf die Disambiguierung von Autoren eingegangen.

Daraufhin entstand eine Software, die es erlaubt, die Autorenangaben aus den Datenquellen zu extrahieren und diese untereinander abzugleichen. Die generierten Daten wurden analysiert und graphisch aufbereitet. Dabei sind Diskrepanzen beispielsweise bei der Nennung der Top-Git-Autoren in den Datenquellen aufgefallen. Diese Diskrepanzen wurden diskutiert und die Fragen Frage beantwortet, wie gut der entwickelte Abgleich der Autoren funktioniert (**F1**)und. Dabei wurde deutlich, dass es möglich ist, die Autorenangaben aus verschiedenen Datenquellen zu extrahieren und untereinander abzugleichen. Außerdem wurde die Frage beantwortet, was ein Autor machen muss, damit er in den Autorenangaben gelistet wird (**F2**). Außerdem wurde die Frage beantwortet Hierbei wurde gezeigt, dass diese Frage nicht allgemeingültig beantwortet werden kann, allerdings erhöhen viele getätigte Commits die Chancen genannt zu werden, wobei auch Autoren mit den meisten Commits in einem Repository

*muss das? so hin und her.
von wegen u mach viel commits, dann hast du
bessere Chancen..... aber muss auch nicht*

~~nicht genannt werden.~~ Abschließend wurde die Frage diskutiert, wie gut Autoren gepflegt werden (F3). *Vor mir lies manes Abs. für F3, wenn die anderen Fragen keinen neuen Abs haben* Im Allgemeinen hat die Arbeit gezeigt, dass es möglich ist, Autorenangaben aus verschiedenen Datenquellen zu extrahieren und diese untereinander abzugleichen. Dadurch ist aufgefallen Es wurde deutlich, dass die Qualität der Autorenangaben in den Datenquellen unterschiedlich ist. Es wurde zudem gezeigt, dass und in vielen Paketen die Autoren nicht gut gepflegt werden. Hier besteht Verbesserungsbedarf, um die Qualität der Autorenangaben zu steigern und die Arbeit jedes einzelnen Entwicklers angemessen zu würdigen.

6.2 Ausblick

Bei der Entwicklung der Datenbeschaffung wurde bewusst auf die Analyse einiger Quellen verzichtet. So wurde beispielsweise nur für die Quellen, welche in Git verwaltet werden, ein zeitlicher Verlauf der Autorenangaben betrachtet. In weiteren Arbeiten könnte dies um Quellen ergänzt werden, welche ebenfalls eine zeitliche Betrachtung ermöglichen, wie beispielsweise PyPI mithilfe der Google BigQuery Daten. Außerdem wurden in ~~der~~dieser Masterarbeit ausschließlich GitHub-Repositorys betrachtet, obwohl es weitere Plattformen wie GitLab gibt, welche auf die gleiche Weise analysiert werden könnten.

Ebenfalls könnten die betrachteten Quellen um weitere Datenpunkte ergänzt werden. Beispielsweise sind im CFF weitere Referenzen zu anderen Arbeiten unter dem Feld ~~referenes~~references aufgelistet. In BIBTeX-Dateien werden ebenfalls weitere Daten gespeichert, welche aktuell noch nicht betrachtet worden sind, allerdings für eine noch detailliertere Analyse genutzt werden könnten.

Ein weiteres Problem, welches in ~~der~~dieser Masterarbeit auftrat, war die Disambiguierung von Autoren. Es wurde gezeigt, dass der Abgleich der Autoren nicht immer korrekt funktioniert und es wurden Gründe wie beispielsweise der Vergleich von nur einem Buchstaben als eines der Probleme erkannt. In weiteren Arbeiten könnte die Disambiguierung von Autoren weiter verbessert werden und so eine direkte Verbesserung der Ergebnisse erzielt werden.

Zusätzlich wurden in der Datenbeschaffung Daten ~~erzeugt~~erstellt, welche in dieser Masterarbeit noch nicht ausgewertet wurden. Dies betrifft die Git-Daten zu den Zeitpunkten einer Änderung an den Autorenangaben. Aus diesen Daten könnte bei einer Analyse beispielsweise geschlussfolgert werden, wie sich die Git-Autoren über

die Zeit verändern und bei wie vielen Commits ein Autor einer Quelle hinzugefügt wurde. Dies ist allerdings nur umsetzbar, falls die Entwickler häufiger ihre Autorenangaben aktualisieren, besonders jene mit zeitlichem Verlauf wie das CFF.

A Zusätzliche Abbildungen

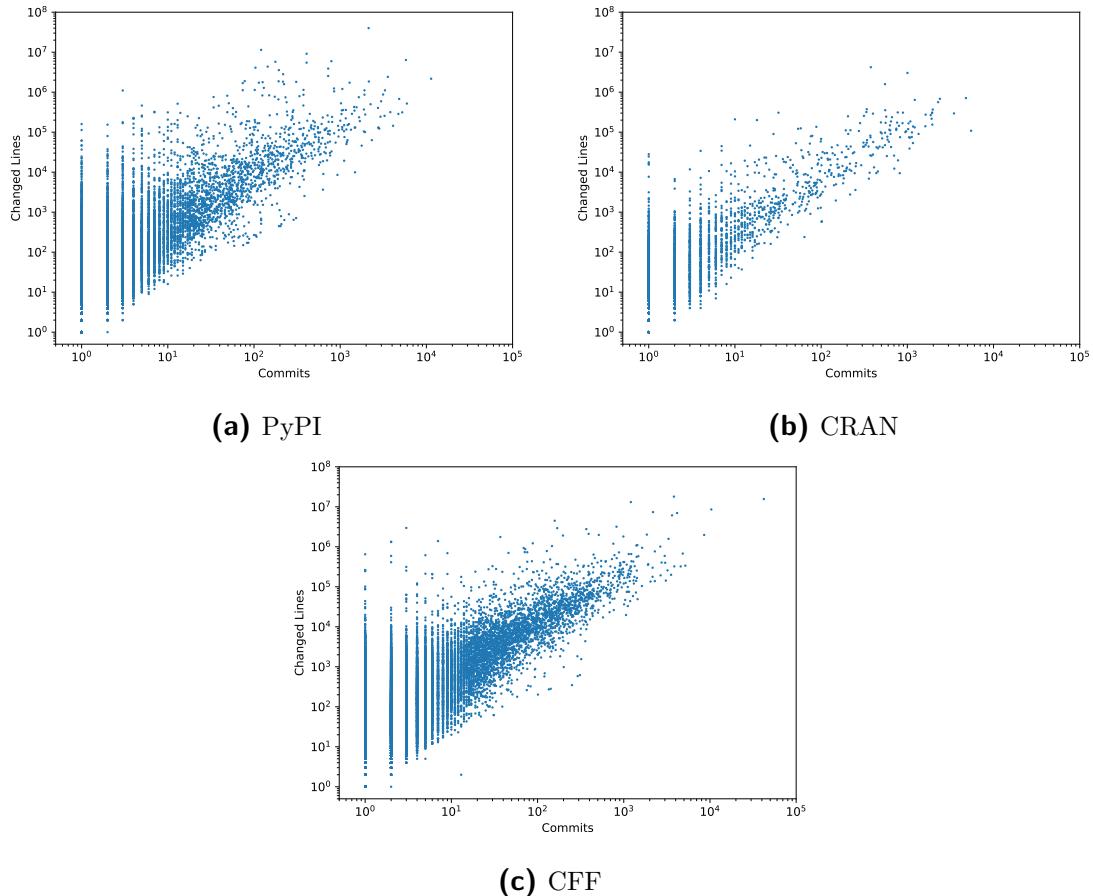


Abbildung 18: Commits und geänderte Zeilen gegenübergestellt

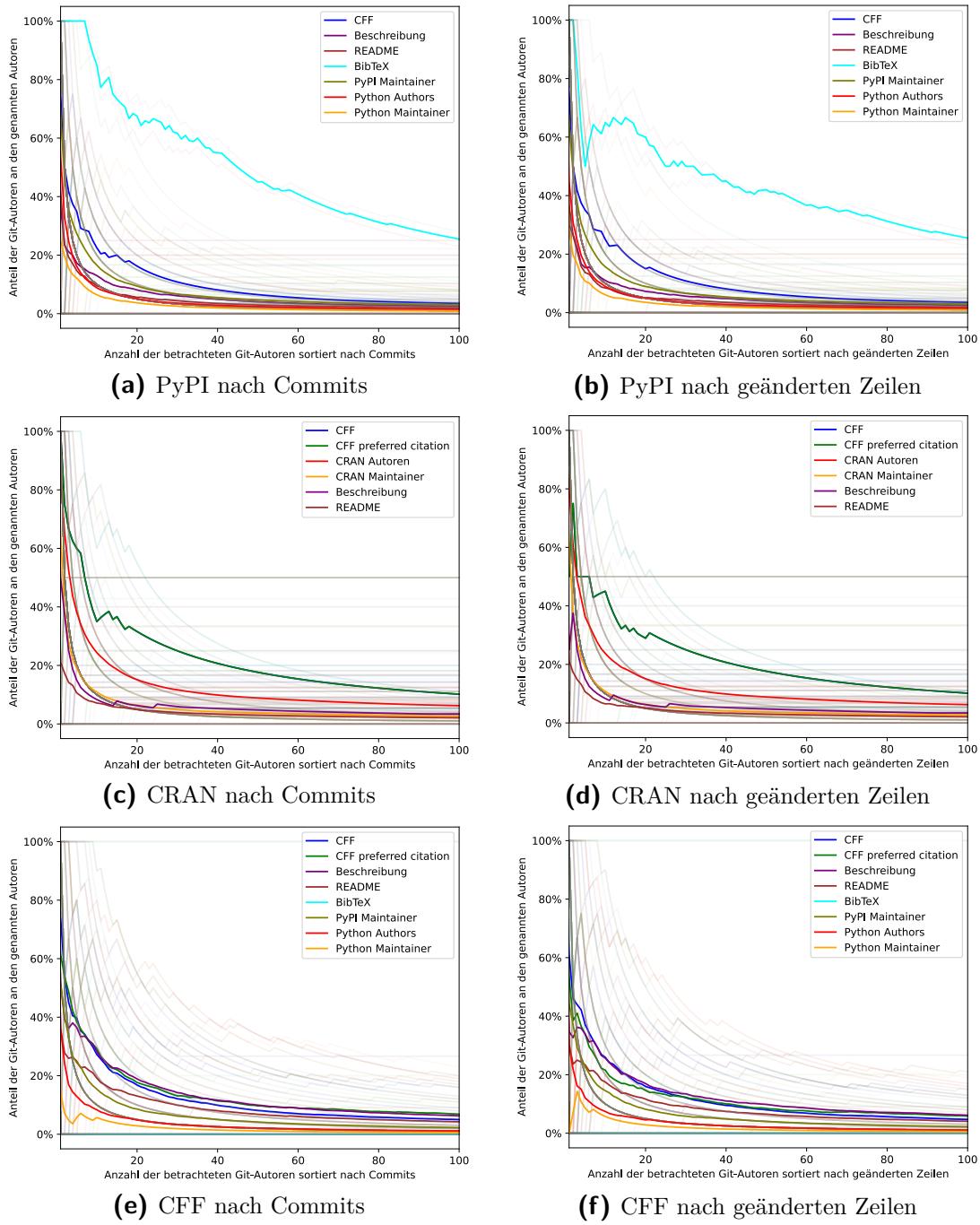


Abbildung 19: Anteil der Top-Git-Autoren an den genannten Autoren

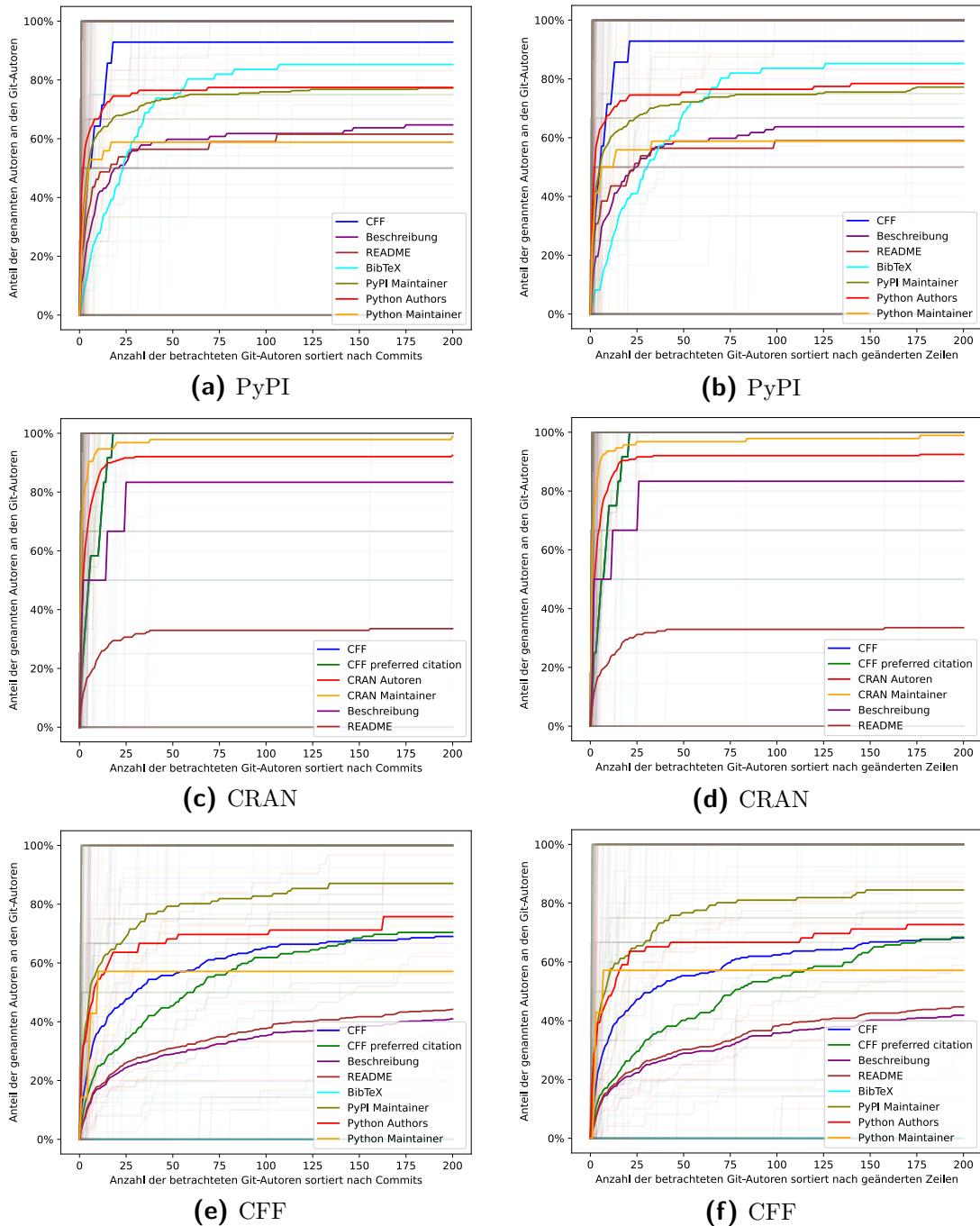


Abbildung 20: Anteil der genannten Autoren unter den Top-Git-Autoren

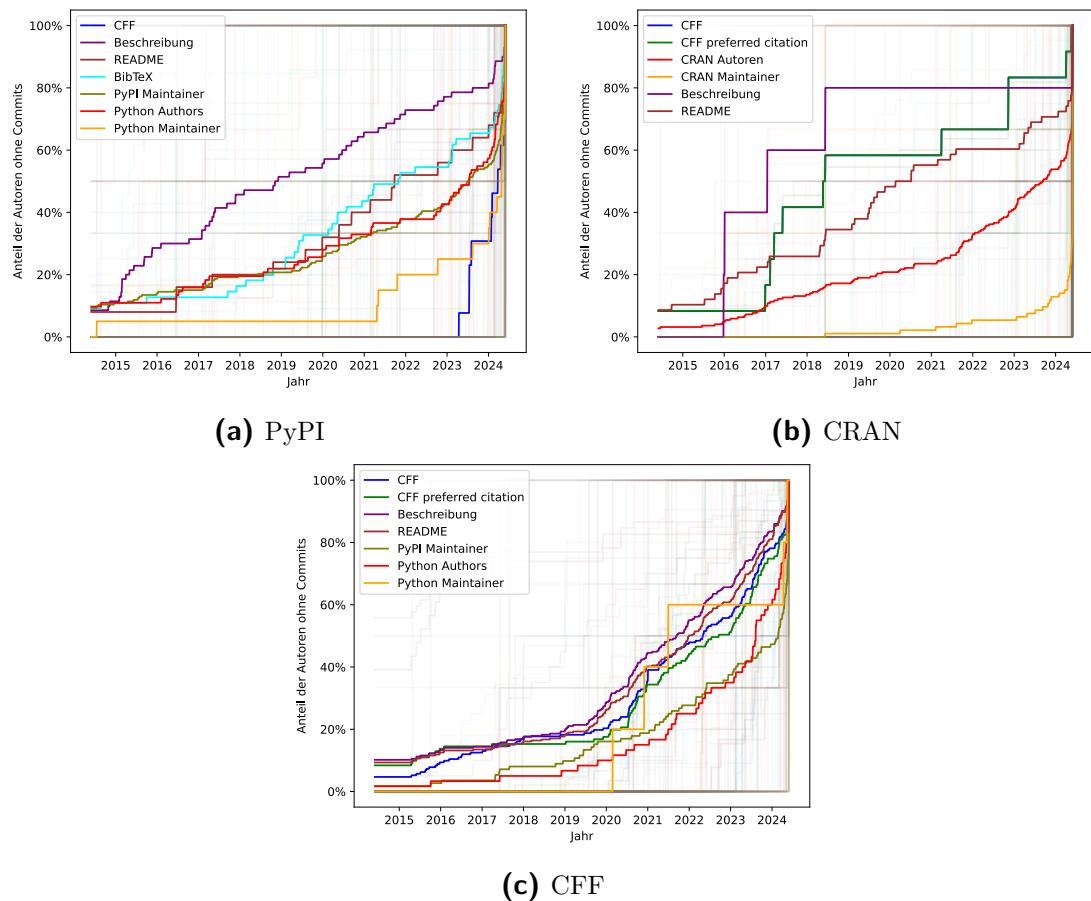


Abbildung 21: Autoren ohne Commits

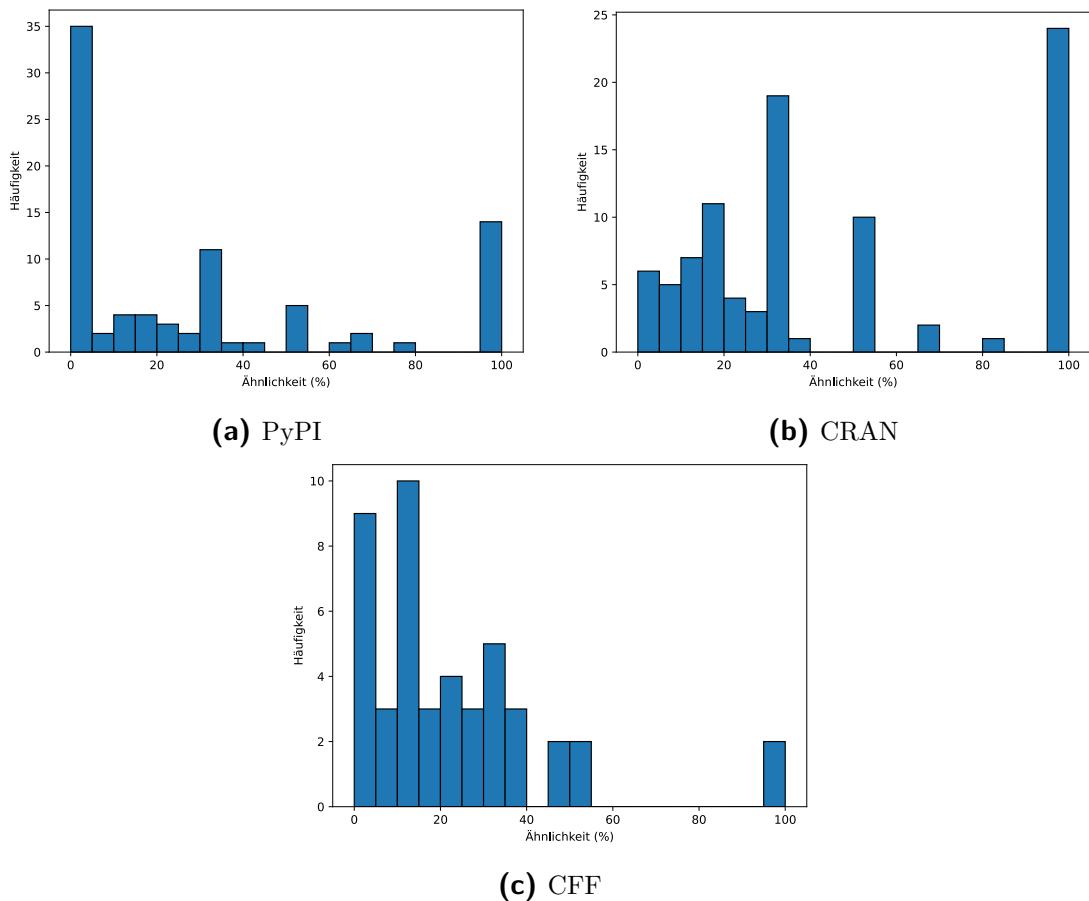


Abbildung 22: Übereinstimmung der Autoren in den Paketen

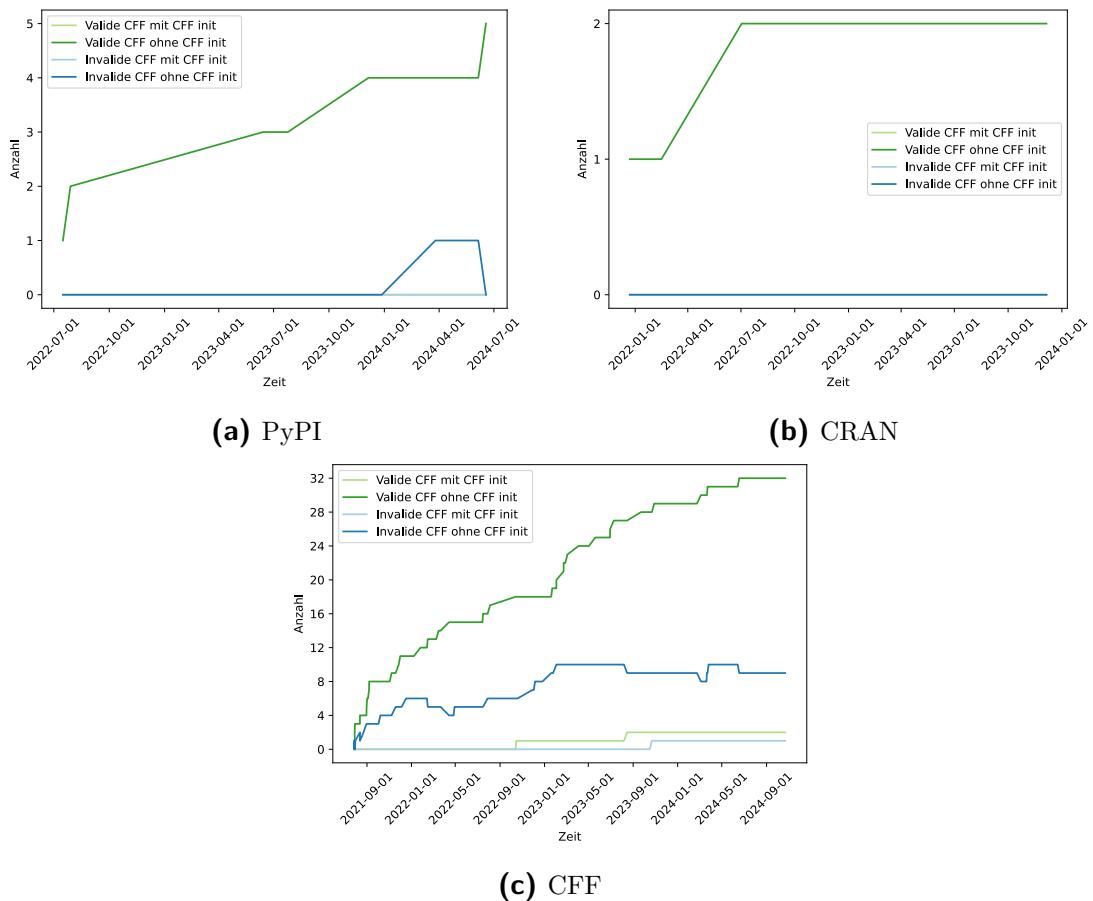


Abbildung 23: Validität der CFF-Dateien über die Zeit

B Zusätzliche Tabellen

Quelle	CRAN	PyPI	CFF
CRAN Autoren	221/239 (92,47 %)		
CRAN Maintainer	93/94 (98,94 %)		
Beschreibung	5/6 (83,33 %)	70/102 (68,63 %)	285/542 (52,58 %)
README	58/173 (33,53 %)	25/39 (64,10 %)	311/568 (54,75 %)
CFF	12/12 (100,00 %)	13/14 (92,86 %)	192/226 (84,96 %)
CFF preferred citation	12/12 (100,00 %)		131/152 (86,18 %)
PyPI Maintainer		193/237 (81,43 %)	112/116 (96,55 %)
Python Autoren		82/102 (80,39 %)	60/66 (90,91 %)
Python Maintainer		20/34 (58,82 %)	5/7 (71,43 %)
BIBTeX		55/61 (90,16 %)	0/1 (0,00 %)
Summe	401/536 (74,81 %)	458/589 (77,76 %)	1096/1678 (65,32 %)

Tabelle 17: Automatische Ergebnisse des Abgleichs

Quelle	TP	FN	FP	TN	Keine Person	F1-Score
Beschreibung	3	0	1	0	0	0,8571
README	15	4	0	33	2	0,8824
CFF	2	0	0	0	0	1,0000
CFF preferred citation	2	0	0	0	0	1,0000
CRAN Autoren	137	1	1	9	1	0,9928
CRAN Maintainer	94	1	0	0	0	0,9947
Summe	253	6	2	42	3	0,9844

Tabelle 18: Manuelle Ergebnisse des Abgleichs für die CRAN Liste

Quelle	TP	FN	FP	TN	Keine Person	F1-Score
Beschreibung	28	1	3	16	5	0,9333
README	8	0	2	9	5	0,8888
CFF	3	0	0	1	0	1,0000
PyPI Maintainer	113	1	9	25	20	0,9576
Python Autoren	63	2	4	15	21	0,9545
Python Maintainer	19	0	0	11	12	1,0000
BIBTEX	4	0	0	0	0	1,0000
Summe	238	4	18	77	63	0,9558

Tabelle 19: Manuelle Ergebnisse des Abgleichs für die PyPI Liste

Quelle	TP	FN	FP	TN	Keine Person	F1-Score
Beschreibung	12	1	6	26	2	0,7742
README	13	1	4	17	5	0,8387
CFF	27	2	0	4	0	0,9643
CFF preferred citation	7	1	1	4	0	0,8750
PyPI Maintainer	61	0	9	5	7	0,9313
Python Autoren	29	0	13	6	17	0,8169
Python Maintainer	3	0	2	2	3	0,7500
BIBTEX	0	1	0	0	0	0,0000
Summe	152	6	35	64	34	0,8812

Tabelle 20: Manuelle Ergebnisse des Abgleichs für die CFF Liste

Quelle	TP	FN	FP	TN	Keine Person	F1-Score
Beschreibung	26	2	8	52	7	0,8387
README	23	4	5	35	11	0,8364
CFF	56	3	3	13	3	0,9492
CFF preferred citation	15	1	1	5	0	0,9375
PyPI Maintainer	120	1	14	18	23	0,9412
Python Autoren	63	2	18	22	37	0,8630
Python Maintainer	8	0	3	8	10	0,8421
BIBTEX	0	1	0	0	0	0,0000
Summe	311	14	52	153	91	0,9041

Tabelle 21: Manuelle Ergebnisse des Abgleichs für die PyPI CFF Liste

Quelle	TP	FN	FP	TN	Keine Person	F1-Score
Beschreibung	18	1	0	31	1	0,9730
README	22	12	0	49	3	0,7857
CFF	80	8	0	6	0	0,9524
CFF preferred citation	54	6	0	4	0	0,9474
CRAN Autoren	132	3	2	10	0	0,9814
CRAN Maintainer	99	1	0	0	0	0,9950
Summe	405	31	2	100	4	0,9609

Tabelle 22: Manuelle Ergebnisse des Abgleichs für die CRAN CFF Liste

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Ich erkläre ferner, dass ich die vorliegende Arbeit in keinem anderen Prüfungsverfahren als Prüfungsarbeit eingereicht habe oder einreichen werde.

Die eingereichte schriftliche Arbeit entspricht der elektronischen Fassung. Ich stimme zu, dass eine elektronische Kopie gefertigt und gespeichert werden darf, um eine Überprüfung mittels Anti-Plagiatsssoftware zu ermöglichen.



Wismar, den 3. Januar 2025

Ort, Datum

Unterschrift