

Master-Thesis

Identifikation und Vergleich von Autorenangaben zu Software
zwischen verschiedenen Datenquellen

Eingereicht am: 21. November 2024

von: Kevin Jahrens
geboren am 05.08.1999
in Bad Oldesloe

Matrikelnummer: 480592

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

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

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“ Softwarepaketen
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

Abstract

Maximal eine halbe Seite.

Inhaltsverzeichnis

| | | |
|----------|--|-----------|
| 1 | Einleitung | 6 |
| 1.1 | Motivation | 6 |
| 1.2 | Vorgehen | 6 |
| 1.3 | Gliederung | 6 |
| 2 | Grundlagen | 7 |
| 2.1 | Zitation von Software | 8 |
| 2.2 | Autorenrolle und Anerkennung in Open-Source-Software | 11 |
| 2.3 | Versionsverwaltung | 12 |
| 2.4 | Software-Verzeichnisse und Paketverwaltung | 15 |
| 2.4.1 | PyPI | 15 |
| 2.4.2 | CRAN | 19 |
| 2.5 | Zitierformate | 20 |
| 2.5.1 | Citation File Format | 20 |
| 2.5.2 | BiBTeX | 23 |
| 2.6 | Named Entity Recognition | 25 |
| 2.7 | Named Entity Disambiguation | 26 |
| 2.8 | Unschärfe Suche | 27 |
| 3 | Methodik | 29 |
| 3.1 | Datenbeschaffung | 30 |
| 3.1.1 | Git | 31 |
| 3.1.2 | PyPI | 33 |
| 3.1.3 | CRAN | 35 |
| 3.1.4 | Beschreibung | 39 |
| 3.1.5 | Citation File Format | 40 |
| 3.1.6 | BiBTeX | 43 |
| 3.2 | Abgleich | 44 |
| 3.3 | Auswertung | 47 |
| 4 | Ergebnisse | 49 |
| 4.1 | Ergebnisse der Gegenwart | 49 |
| 4.2 | Ergebnisse mit Zeitverlauf | 49 |
| 5 | Diskussion | 50 |
| 5.1 | Limitierungen | 50 |
| 6 | Fazit und Ausblick | 51 |
| 6.1 | Fazit | 51 |
| 6.2 | Ausblick | 51 |

| | |
|------------------------------------|-----------|
| Anhang A Beispielanlage | 52 |
| Literaturverzeichnis | 53 |
| Abbildungsverzeichnis | 57 |
| Tabellenverzeichnis | 58 |
| Quellcodeverzeichnis | 59 |
| Abkürzungsverzeichnis | 60 |
| Selbstständigkeitserklärung | 62 |

1 Einleitung

1.1 Motivation

1.2 Vorgehen

1.3 Gliederung

2 Grundlagen

In Abschnitt 2.1 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 Abschnitt 2.2 auf die Rolle von Autoren in Open-Source-Software eingegangen und ein wissenschaftliches Paper dargestellt, welches dies schon analysiert hat.

Autoren von Software werden in unterschiedlichen Quellen zitiert. Einige dieser Quellen sind stark mit der Softwareentwicklung verbunden. So gibt es verschiedene Systeme, die ein Entwickler verwenden kann, um seine Arbeit zu erleichtern bzw. überhaupt sinnvoll zu ermöglichen, in denen Sie anschließend als Autoren genannt werden. In den Abschnitten 2.3 und 2.4 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 Abschnitt 2.5 eingegangen. Außerdem können in Fließtexten, beispielsweise der Beschreibung einer Software, ebenfalls Autoren genannt werden. In Abschnitt 2.6 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 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 Abschnitt 2.7 beschrieben wird. Eine weitere einfache Möglichkeit des Abgleichs ist ein einfacher Abgleich von Zeichenfolgen. Dieser funktioniert jedoch nicht immer, da Autoren unterschiedliche Schreibweisen ihres Namens verwenden können. Aus diesem Grund wird in Abschnitt 2.8 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 Zitier Arbeitsgruppe“ Prinzipien der Software Zitation erstellt, welche eine breite Akzeptanz in der wissenschaftlichen Gemeinschaft finden sollen. Im Folgenden werden die Prinzipien vorgestellt und erläutert Smith, Katz und Niemeyer 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, zum Beispiel in der Referenzliste eines Artikels. Software sollte auf derselben Grundlage zitiert werden wie jedes andere Forschungsprodukt auch, wie zum Beispiel ein Aufsatz oder ein Buch. Das bedeutet, dass Autoren die entsprechend verwendete Software zitieren sollten, so wie sie die entsprechenden Publikationen zitieren würden.
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, die sie beschreiben.
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. **Spezifizität:** Softwarezitate sollten die Identifizierung 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 verstärkt auf das Prinzip der Wichtigkeit eingegangen, da besonders im Citation File Format (CFF) und BibTeX Format die Möglichkeit besteht nicht die Software, sondern beispielsweise ein Artikel anzugeben. Diese Zitierweise würde dann das Prinzip der Wichtigkeit verletzen, da die Software nicht die gleiche Bedeutung zugeschrieben bekommt wie andere Forschungsprodukte. Diese Diskrepanz wird in Kapitel 4 dargestellt.

Es gibt verschiedene Gründe, warum die Zitation von Software ebenfalls wichtig ist und auch, dass Standards der Zitation eingehalten werden, ähnlich wie es der Fall bei anderen wissenschaftlichen Arbeiten ist. Einige dieser Gründe werden im Folgenden genannt (Smith, Katz und Niemeyer 2016):

- Forschungsfelder verstehen: Software ist ein Produkt der Forschung und wenn sie nicht zitiert wird, werden Lücken in der Aufzeichnung der Forschung über den Fortschritt in diesem Forschungsfeld entstehen.
- Anerkennung: Akademische Forscher auf allen Ebenen, einschließlich Studenten, Postdocs, Dozenten und Mitarbeiter, sollten für die Softwareprodukte, die sie entwickeln und zu denen sie beitragen, anerkannt werden, insbesondere wenn diese Produkte die Forschung anderer ermöglichen oder fördern. Nicht-akademische Forscher sollten ebenfalls für ihre Softwarearbeit anerkannt werden, obwohl die spezifischen Formen der Anerkennung sich von denen für akademische Forscher unterscheiden.
- Software entdecken: Mithilfe von Zitaten kann die in einem Forschungsprodukt verwendete Software gefunden werden. Weitere Forscher können dann dieselbe Software für andere Zwecke verwenden, was zu einer Anerkennung der für die Software Verantwortlichen führt.
- Reproduzierbarkeit: Die Angabe der verwendeten Software ist für die Reproduzierbarkeit notwendig, aber nicht ausreichend. Zusätzliche Informationen wie Konfigurationen und Probleme auf der Plattform sind ebenfalls erforderlich.

Wie bereits erwähnt werden Autoren in unterschiedlichen Quellen angegeben. Einige dieser Quellen werden in dieser Masterarbeit untersucht, wie beispielsweise das CFF und BibTeX Format. Die Autoren in diesen Quellen werden von dem jeweiligen

Softwareprojekt angegeben. Falls an dem Projekt nicht nur Softwareentwickler beteiligt sind, sondern beispielsweise auch Grafikdesigner oder Übersetzer, so sollten diese ebenfalls in den Quellen angegeben 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 Code Beitrag einsetzen. Ein Beispiel für ein solches Projekt ist „All Contributors“ (All Contributors 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 dabei von einem Bot automatisch generiert und aktualisiert, sodass sie der Spezifikation von „All Contributors“ entspricht. Über einen Befehl in einem Issue oder einem Pull Request kann ein neuer Mitwirkender hinzugefügt werden. Autoren, welche keinen Code beigetragen haben, stellen im weiteren Verlauf der Masterarbeit ein Problem dar, da diese nicht mit den Autoren aus den Quellen abgeglichen werden können, da sie keine Commits haben. Dies wird in Abschnitt 3.2 genauer erläutert.

Die untersuchten Pakete in dieser Arbeit sind Open Source, welche auf GitHub veröffentlicht werden. Open-Source-Software ist Software, deren Quellcode öffentlich zugänglich ist und von einer Gemeinschaft von Entwicklern entwickelt wird. Die Entwickler arbeiten dabei in der Regel ehrenamtlich und ohne Bezahlung an der Software, wobei einige der untersuchten Pakete auch von Organisationen veröffentlicht werden, wie beispielsweise die „google-auth-library-python“ von Google. Diese wird primär von Google Mitarbeitern 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 wissenschaftlichen medizinischen Artikel 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 / Recommendations / Defining the Role of Authors and Contributors* 2024). Unter anderem aus diesem Grund ist 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, aber beispielsweise vor fünf Jahren aktiv das Projekt geführt haben. Diese und weitere Auffälligkeiten werden in der Masterarbeit untersucht. Auch dies ist in

anderen Bereichen anders definiert, sodass auch die neuen Autoren genannt werden müssen wie beispielsweise bei Neuauflagen von Büchern.

2.2 Autorenrolle und Anerkennung in Open-Source-Software

In der Wissenschaft existieren bereits Analysen zur Rolle von Autoren in Open-Source-Software (OSS). Ein Paper analysiert die Zuschreibung von Entwicklern in OSS (Young u. a. 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. Das zweite Modell betrachtet die Autoren, welche automatisch über ein Tool identifiziert werden. In dem Paper wird hierbei das Programm *octohatrack* verwendet (Young u. a. 2021; *LABHR/octohatrack* 2024). Als drittes Modell werden die Autoren mittels Taxonomien identifiziert. Hierbei wird die `.all-contributorsrc`-Datei von „All Contributors“ analysiert (Young u. a. 2021; All Contributors 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 beispielsweise Webseiten oder unstrukturierten Textdateien. Das Modell wird in dem Paper aufgrund der Komplexität nicht fokussiert betrachtet.

Die Autoren betrachten die Fragen und Modelle dabei von einer gehobenen Perspektive. Dabei werden die einzelnen Fragen mithilfe von verschiedenen generellen Metriken wie die 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 von 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 nicht auf Daten eingegangen, welche aus weiteren Quellen wie dem CFF oder BibTeX Format stammen.

2.3 Versionsverwaltung

Die Versionsverwaltung ist ein System, um verschiedene Versionen von Software zu verwalten. Es bietet Zugang zu Code und dessen Änderungen in der Vergangenheit. 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 und Loeliger 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 Informationen über das Projekt enthält, beispielsweise wie es installiert und verwendet wird. Sie wird Standardmäßig im Stammverzeichnis des Repositories 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 Repository und dessen Vergangenheit hat (ebd.). Ein solches System ist Git, welches sich mit einem Marktanteil von ungefähr 75 % gegenüber anderen Systemen durchgesetzt hat (Lindner 2024). Aus diesem Grund und weil Git-Repositories in der Arbeit untersucht werden, wird auf Git eingegangen. Dabei werden Begriffe erklärt, mit denen es möglich ist, die geleistete Arbeit von einzelnen Autoren innerhalb eines Repositories 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 Abbildung 1 dargestellt. Dabei ist zu erkennen, dass Git in einem Git-Server und Git-Anwendungen aufgeteilt wurde.

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 ermöglicht die einfache kollaborative Entwicklung von Code, da dieser ständig erreichbar ist und zentral verwaltet wird (Ponuthorai und Loeliger 2022).

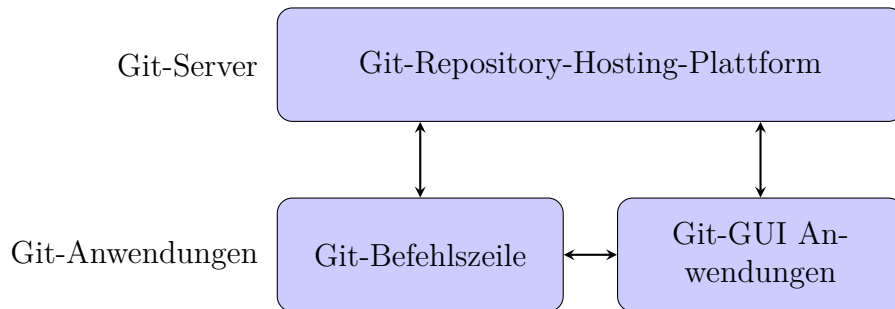


Abbildung 1: Übersicht über die Git-Komponenten

Die Git-Komponenten bestehen aus einem Git-Server, welcher das Repository hostet, und Git-Anwendungen, welche auf das Repository zugreifen (indirekt aus Ponuthorai und Loeliger 2022).

Standardmäßig wird auf dem Git-Server die neuste Version des Repositorys gespeichert. Ein möglicher Git-Server ist GitHub, auf welchen im späteren Verlauf weiter eingegangen wird. Git-Anwendungen sind Programme, welche mit dem lokalen Repository interagieren (Ponuthorai und Loeliger 2022). Diese können auf entfernte Git-Repositorys zugreifen wie beispielsweise Repositorys, welche auf GitHub gehostet werden. 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 zu einem bestimmten Zeitpunkt erstellt (ebd.). Dies wird ein Commit genannt. An einem Commit werden verschiedene Metainformationen gespeichert beispielsweise eine Commit-Nachricht, der Autor und der Zeitpunkt der Änderungen gespeichert.

Die Änderung wird dabei als zwei Zeitpunkte angegeben. Zum einen wird der Zeitpunkt der Änderung des Autors angegeben, also jener Zeitpunkt, zu dem der Autor die Änderung vorgenommen und committed 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 Anerkennung für die geleistete Arbeit erhalten (Chacon und Straub 2024).

Die Commit-Nachricht, sowie der Autor mit E-Mail-Adresse und Name 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. Außerdem 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 enthalten. In der Vergangenheit wurde der Standardbranch *master* genannt. Seit 2020 wird dieser jedoch in *main* umbenannt, um rassistische Konnotationen zu vermeiden (GitHub 2024c). Ein Branch ist eine separate Entwicklungslinie, welche unabhängig von anderen Branches ist. Beim Erstellen von einem Branch wird der aktuelle Zustand des Branches, auf welchem der neue Branch erstellt wird, kopiert (Ponuthorai und Loeliger 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 in einem Branch in einen anderen Branch zu übernehmen.

Die Statistiken der Repositorys können auf verschiedene Arten aufgearbeitet werden. Zum einen können einige direkt mittels Git-Befehlen ausgelesen werden (Chacon 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, ist *git-quick-stats* (Meštan 2024). Außerdem bieten Onlinedienste zur Versionsverwaltung, wie GitHub, Statistiken über APIs an, welche jedoch im Umfang der Anfragen limitiert sind (GitHub 2022a). 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 2022b). 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 zusätzliche Funktionen an, welche über die Standardfunktionen von Git hinausgehen. Diese umfassen unter anderem die kollaborative Entwicklung von Code, Automatisierung mittels CI/CD, Sicherheitsaspekte, Projekt Management, Team Administration und Client-Anwendungen zur Verwaltung von Repositorys (Ponuthorai und Loeliger 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 2024a). 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 kompilierter Code. Zusätzlich werden in einem Paket Metadaten gespeichert. Diese Metadaten können beispielsweise eine Beschreibung, Version, Abhängigkeiten und Autoren des Paketes 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 2012).

In dieser Arbeit wird auf zwei Software-Verzeichnisse eingegangen. Zum einen wird auf Python Package Index (PyPI) eingegangen, welches das Verzeichnis für Python ist und von der Paketverwaltung *pip* verwendet wird. Zum anderen wird auf Comprehensive R Archive Network (CRAN) eingegangen, welches das Verzeichnis für R ist. In PyPI sind aktuell mehr als 500.000 unterschiedliche Projekte mit über 5 Millionen Veröffentlichungen verfügbar (Python Software Foundation 2024c). Im Gegensatz dazu sind in CRAN aktuell mehr als 20.000 Pakete verfügbar (CRAN Team 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 2024b). 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 auf GitHub. Diese Links werden von PyPI als verifiziert angesehen, wenn der Upload des Paketes auf PyPI über eine GitHub Action erfolgt. Außerdem 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.

Die dargestellten GitHub Statistiken werden dabei von PyPI ermittelt und nicht über APIs ausgegeben. Das Ziel von PyPI ist es, dass alle Daten verifiziert werden können und keine Daten mehr in der unverifizierten Form vorliegen.

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 2024d). Diese API ist nicht in der Anzahl der Anfragen beschränkt. Dabei werden die Daten der neusten Version des Pakets zurückgegeben. Die Werte der 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 (ebd.). 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 Abbildung 2 sind einige der

Verified details

These details have been [verified by PyPI](#)

Projekt-Links

 Bug tracker

 Source code

Owner



Matplotlib

GitHub Statistics

 Repository

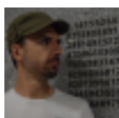
 Sterne: 20027

 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 2024c).

Metadaten des Pakets *matplotlib* unter dem Punkt „Meta“ dargestellt. Weitere Daten sind unter dem Punkt „Projekt-Links“ unter den verifizierten Details dargestellt. Aktuell gibt die JSON API noch keine Auskunft darüber, ob die Daten verifiziert sind oder nicht die Weboberfläche stellt dies bereits dar.

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 Pakete, bei denen die Autoren verifiziert sind, ein Beispiel ist das Paket *hololinked*, welches zum Zeitpunkt des Schreibens darüber verfügt. *Matplotlib* unterstützt dies aktuell noch nicht, wie in Abbildung 2 dargestellt ist. Die verifizierten Betreuer und Owner können aktuell nicht über die JSON API abgefragt werden, sondern müssen über die XML-RPC API abgefragt werden, da diese die einzige API ist, welche die Daten zur Verfügung stellt.

PyPI XML-RPC

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 diesen API nicht mehr zu verwenden und auf den RSS Feed oder die JSON API umzusteigen (Python Software Foundation 2024d). Dadurch ist die API stark limitiert in der Anzahl der möglichen Anfragen 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 dabei das Extrahieren von Daten aus Webseiten, indem der HTML-Code der Webseite analysiert wird (Richardson 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 dabei alle Änderungen am PyPI Projekt vornehmen und Betreuer können neue Versionen des Paketes veröffentlichen (Ingram 2023). Der Benutzername für die Betreuer des Pakets *matplotlib*

sind in Abbildung 2 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 2024a).

BigQuery

Ebenfalls bietet PyPI über Google BigQuery einen Datensatz an, in dem alle Pakete mit ihren Versionen und Metadaten enthalten sind (Python Software Foundation 2024d). BigQuery ist ein Dienst von Google, welcher auf der Infrastruktur der Google Cloud Plattform ausgeführt wird (Google 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 sind in beiden Quellen enthalten. So ist beispielsweise der Autor und Maintainer und deren E-Mail-Adressen in beiden Quellen enthalten. Auch die Beschreibung, Version und Abhängigkeiten sind in beiden Quellen enthalten. 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 bereitstellt. Eines dieser Dienste 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 2024). Die Datenbank ist eine Kopie des CRAN-Repository und wird regelmäßig aktualisiert (Csárdi und Salmon 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 u. a. 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 (Csárdi und Salmon 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 wurden keine Informationen über mögliche Limitierungen in der Anzahl der Anfragen gefunden.

2.5 Zitierformate

In der Wissenschaft 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 die Datenstruktur hinter der Zitation beschreiben. Hierbei gibt es ebenfalls unterschiedliche Formate, wobei sich in dieser Arbeit auf das CFF und das BibTeX-Format beschränkt werden. 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 L^AT_EX Bibliographien zu erstellen und ist daher ebenfalls von Interesse, da es auch für Software verwendet werden kann und von vielen eingesetzt wird.

2.5.1 Citation File Format

Das CFF ist ein Format, welches in der Datei `CITATION.cff` 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 und ist dort weit verbreitet. Insgesamt haben Stand 07.11.2024 2.512 Repositorys eine `CITATION.cff`-Datei. 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 u. a. 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 (ebd.). 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 u. a. 2023). Ein weiteres Beispiel ist das Programm *cffconvert*, welches das CFF in verschiedene Formate umwandeln kann, wie zum Beispiel BibTeX oder RIS. Außerdem kann das Programm genutzt werden um CFF-Dateien zu validieren (Spaaks 2021).

Zusätzlich wird das CFF von unterschiedlichen Plattformen unterstützt, wie zum Beispiel 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 u. a. 2021; GitHub 2024b). Ebenfalls ist es möglich die in der Datei eingetragenen Autoren in der APA-Zitierweise zu kopieren. Die Funktionen sind in Abbildung 3 dargestellt. Eine weitere Plattform, welche unterstützt wird, ist Zotero, welches ein Literaturverwaltungsprogramm ist (Druskat u. a. 2021; Zotero 2024). Falls der Benutzer das Browser-Plugin von Zotero installiert hat, liest dieses die Daten aus der `CITATION.cff`-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 u. a. 2021). Sie kann ebenfalls eine ORCID 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



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.

über den Vor- und Nachname separiert mittels „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“-Feld. Mit diesem Feld ist es möglich die Anerkennung für die Arbeit auf eine andere Arbeit zu übertragen (Druskat u. a. 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 in dem gleichen Format wie bereits beschrieben. Durch die Angabe einer „preferred-citation“ kann das Prinzip der Wichtigkeit vernachlässigt werden. Auf dieses Verhalten wird in den Ergebnissen konkreter eingegangen.

Weitere in der Arbeit verwendete Felder sind „type“, „year“, „month“, „date-released“, „date-published“, „doi“, „collection-doi“ und „identifiers“ (ebd.). 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. Das Feld „date-released“ gibt an, wann die Software oder der Datensatz veröffentlicht wurde. Das Feld „doi“ kann eine DOI der Software oder des Datensatzes enthalten. Mittels „identifiers“ können weitere Identifikatoren angegeben werden, wie zum Beispiel 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“ angegeben werden, um das Jahr und das Datum der Veröffentlichung anzugeben. Außerdem können weitere Typen mittels „type“ angegeben werden, wie beispielsweise „thesis“ oder „manual“, so dass diese Arbeiten ebenfalls referenziert werden können. Zusätzlich kann das Feld „collection-doi“ angegeben werden, um auf eine Sammlung von Arbeiten zu verweisen, die die Arbeit enthält. Ein Beispiel einer `CITATION.cff`-Datei ist in Listing 1 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 Datei `CITATION.bib` gespeichert wird und auf keinem anderen Format basiert. BibTeX ist ein weit verbreiteter Standard und wird von vielen verwendet, um Literatur zu zitieren. Insgesamt haben Stand 07.11.2024 2.144 Repositories auf GitHub eine `CITATION.bib`-Datei. 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 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 werden und in das Repository eingebunden werden. Außerdem existieren viele Literaturverwaltungsprogramme wie Zotero, welche BibTeX-Dateien erstellen und verarbeiten können (Zotero 2024). Ebenfalls ist die Integration in andere Plattformen möglich, wie zum Beispiel 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 2024b).

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. Außerdem sind die verfügbaren Felder vom Literaturtyp abhängig (Patashnik 1988).

```
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: "2025-01-01"
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: "2025-01-01"
25   date-released: "2025-01-01"
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


```

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}
8 }

```

Listing 2: Beispiel einer `CITATION.bib`-Datei

In dieser Arbeit wird auf einige dieser Felder eingegangen, welche für die spätere Auswertung 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 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 Listing 2 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 2014). Typische Entitäten sind Personen, Orte und Organisationen. Im folgenden Beispiel sind die Entitäten markiert, dabei ist nach erfolgreicher NER die Klasse der Entität bekannt, welche hinter die 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** (Ziffer) of talented individuals in various forms and means.*

NER wird in vielen Bereichen eingesetzt, wie zum Beispiel 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 (ebd.). In dem Fall der 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, welche schwierig zu lösen sind. Diese sind zum einen das Erkennen des Entitäten-Anfangs und Endes und zum anderen die Erkennung der korrekten Entitätenklasse, welche standardmäßig versucht werden parallel zu lösen. 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 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 Named entity disambiguation (NED), 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 u. a. 2020). Für die unterschiedliche Genauigkeit sowie für unterschiedliche Sprachen existieren verschiedene Modelle, welche verwendet werden können.

2.7 Named Entity Disambiguation

Die NED ist ein automatischer Prozess, bei dem ein Name einer Entität einer gegebenen Datenmenge zugeordnet wird (Cucerzan 2007; Yamada u. a. 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 welches 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 disambiguiert und ein Teil der NED ist. Diese Probleme kann die NER mittels Modellen lösen, die die Entitäten anhand von Kontexten disambiguieren. Beispiele für erkannte Entitäten sind (Cucerzan 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 zum Beispiel Text Analysen, semantische Suche und der Gruppierung von Software Nennungen in wissenschaftlichen Arbeiten (Cucerzan 2007; Yamada u. a. 2022; Schindler u. a. 2021). In der Masterarbeit kann die NED verwendet werden, um Autoren aus verschiedenen Quellen miteinander abzugleichen.

Ähnlich zu der NER gibt es für die NED verschiedene Modelle, welche verwendet werden können. Viele Modelle sind jedoch spezifisch auf einzelne Aufgabenbereiche trainiert. Allgemeine Modelle sind primär für die Disambiguierung von Entitäten in Texten trainiert, welche in der Arbeit nicht immer vorhanden sind. Ein Beispiel ist ein Modell, welches auf BERT basiert und Wörter, sowohl als auch Entitäten als Tokens erhält und diese mit Entitäten aus einem Text disambiguiert (Yamada u. a. 2022).

2.8 Unscharfe Suche

Die unscharfe Suche ist ein Verfahren, um ähnliche Zeichenfolgen zu finden, die sich in ihrer Schreibweise unterscheiden (Hall und Dowling 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, wird die triviale Suche fehlschlagen. Die unscharfe Suche kann in diesem Fall helfen, den Namen trotzdem zu finden. Ein weiteres Anwendungsgebiet ist eine allgemeine Suche beispielsweise von Produkten in einem Online-Shop. Hierbei müssen auch Tippfehler, welche häufig auftreten berücksichtigt werden. Dadurch ist die Suche einer Zeichenfolge, welcher nahezu korrekt ist ein häufiges Problem in der Informatik.

Die unscharfe Suche basiert auf der Levenshtein-Distanz (Levenshtein 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 einzelzeichen-basierten Editieroperationen, um die Distanz zwischen zwei Zeichenfolgen zu berechnen.

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

Zusätzlich zu der Levenshtein-Distanz existiert eine Erweiterung, die Damerau-Levenshtein-Distanz (Damerau 1964). Diese erweitert die Levenshtein-Distanz um eine vierte Editieroperation. Mittels dieser vier Operationen werden ungefähr 80 % der menschlichen Tippfehler abgedeckt (ebd.).

4. Vertauschen von zwei benachbarten Zeichen in der Zeichenfolge (Suhce \rightarrow Suche)

Eine Herausforderung bei der unscharfen Suche ist die Laufzeit. Sie ist ungefähr proportional zum Produkt der beiden Zeichenfolgenlängen, wodurch eine unscharfe Suche von 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* ([seatgeek/thefuzz](#) 2024). Das Programm basiert auf *RapidFuzz*, welches die Levenshtein-Distanz in Python und C++ implementiert ([rapidfuzz/RapidFuzz](#) 2024).

3 Methodik

In diesem Kapitel wird beschrieben wie die Daten der einzelnen Quellen beschafft, abgeglichen und anschließend ausgewertet werden. Die Datenbeschaffung wird in Abschnitt 3.1, der Abgleich in Abschnitt 3.2 und die Auswertung in Abschnitt 3.3 beschrieben.

Die Datenbeschaffung wurde in die einzelnen Quellen untergliedert. Einige Methoden zur Datenbeschaffung sind dabei ähnlich, worauf im konkreten Fall eingegangen wird. Diese werden jedoch nicht doppelt erläutert.

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 neusten 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 Abbildung 4 dargestellte Ordnerstruktur, welche die Ergebnisse der Datenbeschaffung darstellt.

Der Prozess findet für jedes zu untersuchende Paket statt. Die Pakete werden dabei aus dem PyPI und dem CRAN Software-Verzeichnis entnommen. Die Ergebnisse des Prozesses werden in fünf Ordnern gespeichert jeweils für PyPI, CRAN, CFF, PyPI CFF und CRAN CFF. Die Ordner stellen die 5 Top 100 Listen dar, welche in Abschnitt 3.1 beschrieben werden. In jedem Ordner sind jeweils Unterordner

geladen wurden (Kemenade u. a. 2024). Die zweite Liste enthält die Top 100 meist heruntergeladenen CRAN Pakete im Zeitraum von 02.08.2024 bis 31.08.2024 (Csárdi 2024). Außerdem werden die Top 100 Pakete, welche eine CFF-Datei in GitHub haben untersucht. Die Top 100 werden dabei durch die Anzahl der Sterne auf GitHub definiert. Die Liste wurde am 19.10.2024 erstellt. In der Liste sind Pakete enthalten, welche nicht im PyPI oder CRAN verfügbar sind. Diese Pakete werden nicht betrachtet.

Aus den Quellen Git, Beschreibung, Citation File Format und BibTeX 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 CRAN ist es nicht möglich die Änderungen in der Zeit zu betrachten. In der PyPI Quelle ist es teilweise mit BigQuery möglich die Änderungszeitpunkte zu erhalten, jedoch ist dies mit 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 neusten 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 Abschnitt 3.2 beschriebenen Prozess abgeglichen. Zu Beginn muss das Repository von GitHub geclont werden, um die Daten Lokal verarbeiten zu können. Dabei kommt die OSS *GitPython* in der Version 3.1.43 zum Einsatz, welches eine Bibliothek für die einfache Interaktion mittels Python und Git Befehlen darstellt (Thiel 2024). Beim Aufruf der Funktion wurde kein Branch spezifiziert, weshalb der Branch ausgewählt 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 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.

Die E-Mail-Adresse wird anschließend in Kleinbuchstaben umgewandelt, um die Daten zu vereinheitlichen. Anschließend 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 neuste Commit ausgewählt werden. Der Name des Autors wird dabei ebenfalls summiert, was in dem Kontext bedeutet, dass Namen aneinander gehängt werden. Das Gruppieren ist notwendig, da die Autoren den Namen und die E-Mail-Adressen in Git eigenständig festlegen können wie in Abschnitt 2.3 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 ermittelten und gruppierten Daten werden nach der Anzahl der Commits sortiert in der Datei `git_contributors.csv` gespeichert. In Tabelle 1 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 zu der Aktualisierung 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, welche in Tabelle 1 dargestellt ist, 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 2024).

| 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.

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 (*aio-libs/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.

Anschließend werden die weiteren Daten verarbeitet, welche von der JSON API abgefragt wurden. Dabei werden die nicht verifizierten Autoren und Maintainer jeweils mit Name und E-Mail des Pakets extrahiert. Welche Werte dort enthalten sind, können die Paketentwickler selbst entscheiden. Beispielsweise geben einige Paketentwickler 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. Name mit der 1. E-Mail verbunden wird, der 2. Name mit der 2. 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 noch verschiedene weitere Sonderfälle, welche jedoch nicht alle aufgezählt werden, 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 werden die Daten mit dem in Abschnitt 3.2 beschriebenen Prozess mit den zuvor beschafften Git Daten abgeglichen, dabei werden die verschiedenen Sonderfälle, welche auftreten können berücksichtigt. Anschließend werden die Daten in den Dateien `python_authors.csv` und `python_maintainers.csv` gespeichert. Inhalt ist dabei der Name und die E-Mail des Autors oder Maintainers, sowohl als auch die Daten aus dem Abgleich mit Git. Die Felder der Dateien sind in Tabelle 3 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 u. a. 2020). 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. Durch die Verwendung des genaueren Modells 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. Inhalt ist dabei der Name des Autors, sowohl als auch die Daten aus dem Abgleich mit Git. Die Felder der Datei sind in Tabelle 5 aufgelistet.

Außerdem werden die verifizierten Maintainer, welche auf PyPI angegeben sind verarbeitet, da diese unter Umständen nicht den Autoren entsprechen, welche durch die API ausgegeben werden. Diese werden durch PyPI nicht mittels der JSON API ausgegeben, weshalb die Daten mittels der XML-RPC API abgefragt werden. 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 Abbildung 2 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 Mas-

| 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.

terarbeit nur Personen betrachtet werden. Im Allgemeinen ist die Dokumentation der XML-RPC API nicht gut beschrieben und es wird 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 Webscraper benötigt, um den vollständigen Namen des Autor 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 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 Tabelle 4 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

| 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 Tabelle 2 aufgelistet sind.

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

Tabelle 4: 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 Tabelle 2 aufgelistet sind.

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

Tabelle 5: 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 Tabelle 2 aufgelistet sind.

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 ebenfalls nicht weiter betrachtet.

Anschließend werden die weiteren Daten verarbeitet, welche von der API bereitgestellt werden. Von der API 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, Murdoch und Zeileis 2011):

- „*aut*“ (Autor): Vollständige Autoren, die wesentliche Beiträge zu dem Paket geleistet haben und in der Zitation des Pakets auftauchen sollten.
- „*com*“ (Complier): Personen, die Code (möglicherweise in anderen Sprachen) gesammelt, aber keine weiteren wesentlichen Beiträge zum Paket geleistet haben.
- „*ctb*“ (Mitwirkender): Autoren, die kleinere Beiträge geleistet haben (z. B. Code-Patches usw.), die aber nicht in der Auflistung der Autoren auftauchen sollten.
- „*cph*“ (Urheberrechtsinhaber): Personen, die das Urheberrecht an dem Paket besitzen.
- „*cre*“ (Maintainer): Paket Maintainer
- „*ths*“ (Betreuer der Thesis): Betreuer der Thesis, wenn das Paket Teil einer Thesis ist.
- „*trl*“ (Übersetzer): 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 (*rpy2/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 Tabelle 6 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 zusätzlichen Informationen wie der ORCID-ID, da dieses Vorgehen die Verarbeitung bei den unterschiedlichen aufgebauten Zeichenfolgen vereinfacht hat. Außerdem wird diese Methode nur verwendet, falls die Verarbeitung der Zeichenfolge „Authors@R“ fehlschlägt.

Ein weiteres Feld, welches von der API zurückgegeben wird 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 Name und die E-Mail Adresse des Maintainers. Die Felder der Dateien sind in Tabelle 3 aufgelistet.

Das letzte Feld, welches von der API verarbeitet wird ist das Feld „Description“. In diesem Feld ist die Beschreibung des Pakets enthalten. Dabei wird die Verarbeitung wie in Unterabschnitt 3.1.2 für die Beschreibung von PyPI durchgeführt. Es wird ebenfalls die NER Bibliothek *spaCy* verwendet, um die Autoren zu extrahieren. Anschließend werden Namen der extrahierten Autoren in der Datei `description_authors.csv` nach einem Abgleich mit den Git Daten gespeichert. Die Felder der Datei sind in Tabelle 5 aufgelistet.

| 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 Tabelle 2 aufgelistet sind.

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. Dabei wird die Beschreibung wie zuvor mit *spaCy* untersucht. Anders ist jedoch, dass die `README.md` Datei in Git verwaltet wird und somit die Änderungen in der Zeit betrachtet werden können. Aus diesem Grund wird das Programm *Git-Python* verwendet, um die letzten 50 Änderungen der Datei auf dem Standardbranch 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 neusten Git Daten abgeglichen wurden. Die Felder der Datei sind in Tabelle 5 aufgelistet. Der neuste Zeitpunkt entspricht hierbei dem Zeitpunkt, an dem die Repositorys heruntergeladen wurden, also der Tag an dem der Prozess durchlaufen wurde.

Zusätzlich wird zu jedem Änderungszeitpunkt der Git Prozess aus Unterabschnitt 3.1.1 bis zu diesem Zeitpunkt durchlaufen. Dies dient dazu, eine Liste aller Autoren zu erhalten, welche 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 Unterabschnitt 3.1.1, 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

| 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.

Beschreibung extrahiert wurden, abgeglichen und in der Datei `TIMESTAMP_readme_authors.csv` gespeichert. Die Felder der Datei sind ebenfalls in Tabelle 5 aufgelistet. 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 Tabelle 7 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,, gelten und die spezifischen Daten, welche ausschließlich für die “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 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 (*yaml/pyyaml* 2024).

In der CFF können für die eigentliche Zitation und für die “preferred-citation,, Autoren angegeben werden. Diese werden jeweils getrennt extrahiert anschließend mit den neusten 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,, 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 Tabelle 6 aufgelistet.

Zusätzlich zu den neusten Daten wird zu jedem Änderungszeitpunkt der Git Prozess aus Unterabschnitt 3.1.1 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.

Außerdem 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“. Die Dateien heißen `cff.csv` und `cff_preferred_citation.csv`. Die Felder der `cff.csv` Datei sind in Tabelle 8 aufgelistet und die der `cff_preferred_citation.csv` Datei in Tabelle 9. In den beiden Dateien werden ebenfalls die Zeitpunkte *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 2021). Außerdem benötigt *cffconvert* die Programme *jsonschema* und *pykwalify*, welche in der Masterarbeit in der Version 4.23.0 und 1.8.0 verwendet werden (*python-jsonschema/jsonschema* 2024; Grokzen 2024). 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 u. a. 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.

| 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 „identifiers“ |
| <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.

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 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 die „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 wir 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“ werden ebenfalls extrahiert und in der allgemeinen Datei für die „preferred-citation“ gespeichert. Die Felder der Datei sind in Tabelle 9 aufgelistet.

| 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 „identifiers“ |
| <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.

3.1.6 BiBTeX

In diesem Unterabschnitt wird wie bereits zuvor eine Datei untersucht, welche in Git verwaltet wird. Hierbei wird 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. Die Software bietet Möglichkeiten, um in Python BiBTeX 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 BiBTeX Datei extrahiert und mit den neusten 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 Tabelle 5 aufgelistet.

Ebenfalls wird erneut der Git Prozess aus Unterabschnitt 3.1.1 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 beschrie-

| Feld | Beschreibung |
|---------------------------|--|
| <i>type</i> | Literaturtyp der <code>BIBTEX</code> 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.

ben. Außerdem wird die Liste der Autoren, welche aus der `BIBTEX` 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 ist identisch mit der „new“ Datei. Hierbei wird ebenfalls ausschließlich der Name des Autors und die Daten aus dem Git Abgleich gespeichert.

Außerdem können erneut allgemeine Daten für die `BIBTEX` 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, verschiedene Typen sind in den Grundlagen erläutert. 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 Tabelle 10 aufgelistet.

3.2 Abgleich

In diesem Abschnitt wird beschrieben, wie die Autoren aus den unterschiedlichen Quellen, welche in der Datenbeschaffung erläutert wurden, abgeglichen werden. Dabei werden jeweils die Git Autoren mit den Autoren aus den anderen Quellen abgeglichen. Für diesen Prozess wird keine NED verwendet, sondern ein eigener Algorithmus, welcher auf die Daten angepasst ist. Dieser wurde entwickelt, indem die

Autoren, welche durch die Datenbeschaffung erhalten wurden, in den Paketen eingesehen wurden. So wurde durch viel Probieren eine möglichst gute Lösung gefunden, welche im Folgenden beschrieben wird. Dies führt allerdings dazu, dass der entwickelte Algorithmus nicht in der Lage ist zwei Autoren mit dem gleichen Namen zu unterscheiden, falls keine weiteren Daten wie eine E-Mail vorhanden sind.

Als Eingabe erhält der Algorithmus eine Liste von Autoren, welche in einer Quelle (z. B. CFF) gefunden wurden und jeweils die Liste der Git Autoren für das zu untersuchende Paket. In der Datenbeschaffung wurde gezeigt, dass die Autoren aus den Quellen einen Namen, eine E-Mail und einen Benutzernamen enthalten können, abhängig von der Quelle. In dem Algorithmus wird ermittelt, wie viele von den Daten vorhanden sind, um im Verlauf einen Score berechnen zu können wie viele der Daten übereinstimmen. Falls Beispielsweise die Quelle CFF ist, kann der Name und die E-Mail vorhanden sein. Dadurch ergibt sich, dass maximal zwei Daten mit denen von Git übereinstimmen können. Falls die Quelle die Beschreibung ist, ist maximal eine Übereinstimmung möglich. Diese Daten werden für den Abgleich verwendet. Die eingegebene Liste der Git Autoren muss für den Algorithmus nach der Anzahl der Commits sortiert sein, da der Algorithmus Personen mit den meisten Commits bevorzugt. Anschließend wird jeder Autor aus der Quelle mit jedem Git Autor abgeglichen. Dabei werden alle Werte in Kleinbuchstaben umgewandelt, um zu gewährleisten, dass auch bei unterschiedlicher Schreibweise eine Übereinstimmung gefunden werden kann.

Dabei wird so vorgegangen, dass die Daten, welche vorhanden sind, zum Vergleich herangezogen werden. Falls der Name vorhanden ist, wird dieser mit dem Namen des Git Autors über das Keyword *in* in Python verglichen. Dieses sorgt dafür, dass nur ein Teil der Zeichenkette in der anderen Zeichenkette vorhanden sein muss. Außerdem wird die andere Richtung ebenfalls probiert. Zusätzlich wird das Programm *thefuzz* in der Version 0.22.1 verwendet, um eine unscharfe Suche zu ermöglichen. Hierbei wird der Name des Autors aus der Quelle mit dem Namen des Git Autors verglichen und auf eine Übereinstimmung von 80 % oder mehr geprüft. Die Datenbeschaffung hat gezeigt, dass die Namen der Autoren manchmal Klammern enthalten, welche den beschriebenen Abgleich stören. Aus diesem Grund wird der zuvor beschriebene Abgleich erneut durchgeführt, wobei die Klammern in dem Namen entfernt worden sind.

Falls eine E-Mail vorhanden ist, wird diese mit der E-Mail des Git Autors über das Keyword *in* verglichen. Die andere Richtung wird ebenfalls probiert, also die E-Mail des Git Autors mit der E-Mail des Autors aus der Quelle über das Keyword

in zu vergleichen. Zusätzlich wird erneut die unscharfe Suche zwischen der E-Mail des Autors aus der Quelle und der E-Mail des Git Autors durchgeführt und auf eine Übereinstimmung von 80 % oder mehr geprüft. Falls kein Abgleich über den Namen stattgefunden hat, wird ein Vergleich zwischen der E-Mail des Autors aus der Quelle und dem Namen des Git Autors über das Keyword *in* durchgeführt. Außerdem wird der Abgleich erneut über die unscharfe Suche mit einer Übereinstimmung von 80 % oder mehr durchgeführt. Dies führt dazu, dass weitere Übereinstimmungen gefunden werden können, falls der Name zu keiner Übereinstimmung geführt hat. Dies liegt daran, dass viele Autoren in ihrer E-Mail-Adresse ihren Namen enthalten haben. Außerdem hat die Datenbasis gezeigt, dass es vorkommt, dass in dem E-Mail Feld keine E-Mail angegeben wurde, sondern ein Name und als Resultat das Namensfeld nicht ausgefüllt wurde.

Falls ein Benutzername vorhanden ist, wird dieser mit der E-Mail des Git Autors verglichen, da aus den Git Daten kein Benutzername extrahiert werden kann. Dies ist möglich, da viele Autoren für ihren Benutzernamen den lokal Teil der E-Mail verwenden. Für den Vergleich wird die E-Mail, falls sie ein @ Symbol enthält, an dieser Stelle getrennt. Anschließend wird der vordere lokale Teil für den Vergleich mit dem Benutzernamen aus der Quelle verwendet. Dabei wird der Vergleich erneut über das Keyword *in* in beiden Richtungen durchgeführt. Außerdem wird über die unscharfe Suche mit einer Übereinstimmung von 80 % oder mehr geprüft, ob eine Übereinstimmung gefunden wurde. Dieser Prozess wird ebenfalls erneut für den Domain-Teil der E-Mail durchgeführt, sodass dieser ebenfalls mit dem Benutzernamen verglichen wird.

Falls keiner dieser drei Vergleiche zu einem Erfolg geführt hat, wird der Name des Autors mit dem lokalen Teil der E-Mail des Git Autors über die unscharfe Suche verglichen. Dabei muss erneut eine Übereinstimmung von 80 % oder mehr vorhanden sein, um den Vergleich als gelungen zu bewerten.

Anschließend wird der Score berechnet. Dabei wird die Anzahl der Übereinstimmungen durch die Anzahl der maximal möglichen Übereinstimmungen geteilt. Falls beispielsweise der Name und die E-Mail vorhanden sind, sind maximal zwei Übereinstimmungen möglich. Bei einer Übereinstimmung des Namens und die E-Mail ergibt sich ein Score von 1. Falls nur der Name übereinstimmt, ergibt sich ein Score von 0,5. Dieser Score wird für jeden Autor aus der Quelle mit jedem Git Autor berechnet und in der Tabelle der Autoren aus der Quelle gespeichert. Anschließend wird der Autor aus der Quelle mit dem Git Autor mit dem besten Score ausgewählt. Falls es zwei Einträge in der Git Liste gibt, welche einen gleichen Score erreichen, wird

der Autor ausgewählt, welcher die meisten Commits hat. Außerdem wird der Rang, welcher der Autor in der Git Autoren Liste belegt zurückgegeben. Als Ergebnis des Abgleichs werden alle Autoren aus der Quelle, welche in den Algorithmus eingegeben wurde, zurückgegeben. Die Autoren werden dabei mit den Ergebnissen des Abgleichs verbunden, welche in Tabelle 2 dargestellt sind. Das Ergebnis des Abgleichs ist in Tabelle 2 dargestellt. Falls kein Abgleich für einen Autor möglich war, werden die Felder für diesen Autor leer gelassen. Anschließend wird das Ergebnis nach dem Rang sortiert, also nach der Anzahl der Commits, welche der Autor hat.

3.3 Auswertung

In diesem Abschnitt wird beschrieben, wie die Daten aus dem Abschnitt 3.1 ausgewertet werden. Dabei wird der Abschnitt aufgeteilt in die Aggregation 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 neusten 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 neusten 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. Aus diesem Grund werden für einige Ergebnisse der Auswertung die Ergebnisse einmal inklusive der Autoren betrachtet, welche nicht abgeglichen werden konnten und einmal exklusive dieser Autoren. Für andere Ergebnisse ist dies nicht möglich, da genannte Autoren in den Quellen, welche nicht in den Git Daten vorhanden sind, ebenfalls zu keinem Abgleich führen.

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 das 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 Aggregierung der Daten unterteilt in die Aggregierung der vollständigen Daten und die Aggregierung der neusten Daten. In der vollständigen Datenverarbeitung werden Ergebnisse extrahiert, welche die gesamte Historie der Daten betrachten. Hier werden alle Daten mit ihren Zeitstempeln verarbeitet. In der neusten Datenverarbeitung werden nur die Daten betrachtet, welche auf dem neusten Stand der untersuchten Software sind. Dabei wird jeweils die neuste Datei in den Quellen mithilfe des Zeitstempels ausgewählt und anschließend verarbeitet. In diese beiden Kategorien ist das Kapitel 4 ebenfalls unterteilt.

4 Ergebnisse

4.1 Ergebnisse der Gegenwart

4.2 Ergebnisse mit Zeitverlauf

5 Diskussion

5.1 Limitierungen

6 Fazit und Ausblick

6.1 Fazit

6.2 Ausblick

A Beispielanlage

Beispieltext.

Literaturverzeichnis

- aio-lib*s/*aiohttp* (10. Aug. 2024). original-date: 2013-10-01T23:04:01Z. URL: <https://github.com/aio-lib/aiohttp> (besucht am 12.11.2024).
- All Contributors (2024). *Recognize all contributors*. All Contributors. URL: <https://allcontributors.org> (besucht am 01.06.2024).
- Altmann, Benjamin u. a. (2024). *The Comprehensive R Archive Network*. URL: <https://cran.r-project.org/> (besucht am 28.09.2024).
- arzzen (15. Juni 2021). *git-quick-stats/git-quick-stats*. Version 2.3.0. URL: <https://github.com/git-quick-stats/git-quick-stats> (besucht am 09.11.2024).
- Bishop, Stuart (11. Sep. 2024). *stub42/pytz*. Version 2024.2. original-date: 2016-07-12T12:22:30Z. URL: <https://github.com/stub42/pytz> (besucht am 09.11.2024).
- Chacon, Scott (2024). *Git - git-shortlog Documentation*. URL: <https://git-scm.com/docs/git-shortlog> (besucht am 21.05.2024).
- Chacon, Scott und Ben Straub (6. Nov. 2024). *Pro Git*. 2. Aufl. Apress. 440 S. ISBN: 978-1-4842-0077-3. URL: <https://github.com/progit/progit2>.
- Costa-Luis, Casper da u. a. (4. Aug. 2024). *tqdm: A fast, Extensible Progress Bar for Python and CLI*. Version v4.66.5. DOI: 10.5281/zenodo.13207611. URL: <https://zenodo.org/records/13207611> (besucht am 08.11.2024).
- CRAN Team (Mai 2024). *The Comprehensive R Archive Network*. URL: <https://cran.r-project.org/> (besucht am 21.05.2024).
- Csárdi, Gábor (7. Nov. 2024). *r-hub/cranlogs.app*. original-date: 2014-10-28T19:53:47Z. URL: <https://github.com/r-hub/cranlogs.app> (besucht am 08.11.2024).
- Csárdi, Gábor und Maëlle Salmon (2023). *pkgsearch: Search and Query CRAN R Packages*. URL: <https://github.com/r-hub/pkgsearch>.
- Cucerzan, Silviu (Juni 2007). „Large-Scale Named Entity Disambiguation Based on Wikipedia Data“. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Hrsg. von Jason Eisner. Prague, Czech Republic: Association for Computational Linguistics, S. 708–716. URL: <https://aclanthology.org/D07-1074>.
- Damerau, Fred J. (März 1964). „A technique for computer detection and correction of spelling errors“. In: *Commun. ACM* 7.3. Place: New York, NY, USA Publisher: Association for Computing Machinery, S. 171–176. ISSN: 0001-0782. DOI: 10.1145/363958.363994. URL: <https://doi.org/10.1145/363958.363994>.
- Druskat, Stephan u. a. (Aug. 2021). *Citation File Format*. Version 1.2.0. DOI: 10.5281/zenodo.5171937. URL: <https://github.com/citation-file-format/citation-file-format> (besucht am 29.09.2024).
- GitHub (28. Nov. 2022a). *Rate limits for the REST API*. GitHub Docs. URL: <https://docs.github.com/en/rest/using-the-rest-api/rate-limits-for-the-rest-api?apiVersion=2022-11-28> (besucht am 21.05.2024).

- GitHub (28. Nov. 2022b). *REST-API-Endpunkte für Repositorys*. GitHub Docs. URL: <https://docs.github.com/de/rest/repos/repos?apiVersion=2022-11-28#list-repository-contributors> (besucht am 03.06.2024).
- (2024a). *About*. Let’s build from here. URL: <https://github.com/about> (besucht am 23.09.2024).
 - (2024b). *About CITATION files*. GitHub Docs. URL: <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-citation-files> (besucht am 01.10.2024).
 - (20. Sep. 2024c). *github/renaming*. original-date: 2020-06-12T21:51:22Z. URL: <https://github.com/github/renaming> (besucht am 24.09.2024).
- Google (2024). *BigQuery: Data Warehouse für Unternehmen*. Google Cloud. URL: <https://cloud.google.com/bigquery> (besucht am 27.09.2024).
- Grokzen (14. Nov. 2024). *Grokzen/pykwalify*. original-date: 2013-01-17T09:20:37Z. URL: <https://github.com/Grokzen/pykwalify> (besucht am 19.11.2024).
- Hall, Patrick A. V. und Geoff R. Dowling (Dez. 1980). „Approximate String Matching“. In: *ACM Comput. Surv.* 12.4. Place: New York, NY, USA Publisher: Association for Computing Machinery, S. 381–402. ISSN: 0360-0300. DOI: 10.1145/356827.356830. URL: <https://doi.org/10.1145/356827.356830>.
- Honnibal, Matthew u. a. (2020). „spaCy: Industrial-strength Natural Language Processing in Python“. In: DOI: 10.5281/zenodo.1212303.
- Hornik, Kurt, Duncan Murdoch und Achim Zeileis (Nov. 2011). „Who Did What? The Roles of R Package Authors and How to Refer to Them“. In: *The R Journal* 4. DOI: 10.32614/RJ-2012-009.
- ICMJE / Recommendations / Defining the Role of Authors and Contributors (2024). URL: <https://www.icmje.org/recommendations/browse/roles-and-responsibilities/defining-the-role-of-authors-and-contributors.html> (besucht am 21.09.2024).
- Ingram, Dustin (5. Apr. 2023). *Deprecate the "Maintainer"role · Issue #13366 · pypi/warehouse*. GitHub. URL: <https://github.com/pypi/warehouse/issues/13366> (besucht am 25.09.2024).
- Kemenade, Hugo van u. a. (1. Sep. 2024). *hugovk/top-pypi-packages: Release 2024.09*. Version 2024.09. DOI: 10.5281/zenodo.13624792. URL: <https://zenodo.org/records/13624792> (besucht am 08.11.2024).
- LABHR/octohatrack (22. Apr. 2024). original-date: 2015-07-20T09:56:20Z. URL: <https://github.com/LABHR/octohatrack> (besucht am 04.11.2024).
- Levenshtein, Vladimir I. (1965). „Binary codes capable of correcting deletions, insertions, and reversals“. In: *Soviet physics. Doklady* 10, S. 707–710. URL: <https://api.semanticscholar.org/CorpusID:60827152>.
- Lindner, Jannik (3. Mai 2024). *Version Control Systems Industry Statistics*. URL: <https://worldmetrics.org/version-control-systems-industry-statistics/> (besucht am 21.05.2024).
- Mešťan, Lukáš (18. Mai 2024). *git-quick-stats*. Version 2.5.6. URL: <https://github.com/arzzzen/git-quick-stats> (besucht am 21.05.2024).
- Mohit, Behrang (2014). „Named Entity Recognition“. In: *Natural Language Processing of Semitic Languages*. Hrsg. von Imed Zitouni. Berlin, Heidelberg: Springer

- Berlin Heidelberg, S. 221–245. ISBN: 978-3-642-45358-8. URL: https://doi.org/10.1007/978-3-642-45358-8_7.
- Patashnik, Oren (8. Feb. 1988). *BibTEXing*.
- Ponuthorai, Prem Kumar und Jon Loeliger (Nov. 2022). *Version Control with Git*. 3. Aufl. Sebastopol: O'Reilly Media. ISBN: 978-1-4920-9119-6.
- Python Software Foundation (12. Apr. 2024a). *Add PyPI User API · Issue #15769 · pypi/warehouse*. GitHub. URL: <https://github.com/pypi/warehouse/issues/15769> (besucht am 27. 09. 2024).
- (8. Jan. 2024b). *PEP 740 – Index support for digital attestations* / *peps.python.org*. Python Enhancement Proposals (PEPs). URL: <https://peps.python.org/pep-0740/> (besucht am 25. 09. 2024).
- (Mai 2024c). *PyPI · Der Python Package Index*. PyPI. URL: <https://pypi.org/> (besucht am 21. 05. 2024).
- (Mai 2024d). *Warehouse documentation*. URL: <https://warehouse.pypa.io/index.html> (besucht am 21. 05. 2024).
- python-jsonschema/jsonschema* (18. Nov. 2024). original-date: 2011-12-30T03:37:43Z. URL: <https://github.com/python-jsonschema/jsonschema> (besucht am 19. 11. 2024).
- rapidfuzz/RapidFuzz* (5. Nov. 2024). original-date: 2020-02-29T14:41:44Z. URL: <https://github.com/rapidfuzz/RapidFuzz> (besucht am 06. 11. 2024).
- Richardson, Leonard (17. Jan. 2024). *beautifulsoup4: Screen-scraping library*. Version 4.12.3. URL: <https://www.crummy.com/software/BeautifulSoup/bs4/> (besucht am 27. 09. 2024).
- rpy2/rpy2* (31. März 2024). Version 3.5.16. original-date: 2018-12-14T18:51:22Z. URL: <https://github.com/rpy2/rpy2> (besucht am 12. 11. 2024).
- Schindler, David u. a. (2021). „SoMeSci- A 5 Star Open Data Gold Standard Knowledge Graph of Software Mentions in Scientific Articles“. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. event-place: Virtual Event, Queensland, Australia. New York, NY, USA: Association for Computing Machinery, S. 4574–4583. ISBN: 978-1-4503-8446-9. DOI: 10.1145/3459637.3482017. URL: <https://doi.org/10.1145/3459637.3482017>.
- seatgeek/thefuzz* (5. Nov. 2024). original-date: 2021-03-05T19:07:19Z. URL: <https://github.com/seatgeek/thefuzz> (besucht am 06. 11. 2024).
- Smith, Arfon M., Daniel S. Katz und Kyle E. Niemeyer (19. Sep. 2016). „Software citation principles“. In: *PeerJ Computer Science* 2. Publisher: PeerJ Inc., e86. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.86. URL: <https://peerj.com/articles/cs-86> (besucht am 10. 09. 2024).
- Spaaks, Jurriaan H. (Sep. 2021). *cffconvert*. Version 3.0.0a0. original-date: 2018-01-09T14:16:28Z. URL: <https://github.com/citation-file-format/cffconvert> (besucht am 30. 09. 2024).
- Spaaks, Jurriaan H. u. a. (Aug. 2023). *cffinit*. Version 2.3.1. original-date: 2021-07-14T09:53:13Z. URL: <https://github.com/citation-file-format/cff-initializer-javascript> (besucht am 30. 09. 2024).
- Spinellis, Diomidis (2012). „Package Management Systems“. In: *IEEE Software* 29.2, S. 84–86. DOI: 10.1109/MS.2012.38.

- The Apache Software Foundation (2024). *Apache CouchDB*. URL: <https://couchdb.apache.org/> (besucht am 22.05.2024).
- The pandas development team (10. Apr. 2024). *pandas-dev/pandas: Pandas*. Version 2.2.2. DOI: 10.5281/zenodo.10957263. URL: <https://github.com/pandas-dev/pandas>.
- Thiel, Sebastian (31. März 2024). *gitpython-developers/GitPython*. Version 3.1.43. original-date: 2010-11-30T17:34:03Z. URL: <https://github.com/gitpython-developers/GitPython> (besucht am 09.11.2024).
- Yamada, Ikuya u. a. (Juli 2022). „Global Entity Disambiguation with BERT“. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. NAACL-HLT 2022. Hrsg. von Marine Carpuat, Marie-Catherine de Marneffe und Ivan Vladimir Meza Ruiz. Seattle, United States: Association for Computational Linguistics, S. 3264–3271. DOI: 10.18653/v1/2022.naacl-main.238. URL: <https://aclanthology.org/2022.naacl-main.238> (besucht am 03.09.2024).
- yaml/pyyaml* (19. Nov. 2024). original-date: 2011-11-03T05:09:49Z. URL: <https://github.com/yaml/pyyaml> (besucht am 19.11.2024).
- Young, Jean-Gabriel u. a. (2021). „Which contributions count? Analysis of attribution in open source“. In: *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, S. 242–253. DOI: 10.1109/MSR52588.2021.00036.
- Zotero (2024). *Zotero / Your personal research assistant*. URL: <https://www.zotero.org/> (besucht am 01.10.2024).

Abbildungsverzeichnis

| | | |
|---|--|----|
| 1 | Übersicht über die Git-Komponenten | 13 |
| 2 | PyPI Verifizierte und unverifizierte Daten | 17 |
| 3 | GitHub-Repository mit <code>CITATION.cff</code> -Datei | 22 |
| 4 | Ergebnisse der Datenbeschaffung | 30 |

Tabellenverzeichnis

| | | |
|----|--|----|
| 1 | Felder der <code>git_contributors.csv</code> Datei | 33 |
| 2 | Felder, welche durch den Abgleich mit Git entstehen | 35 |
| 3 | Felder der <code>python_authors.csv</code> , <code>python_maintainers.csv</code> und <code>cran_maintainers.csv</code> Datei | 36 |
| 4 | Felder der <code>pypi_maintainers.csv</code> Datei | 36 |
| 5 | Felder der <code>description_authors.csv</code> , <code>TIMESTAMP_readme_authors(_new).csv</code> und <code>TIMESTAMP_bib_authors(_new).csv</code> Datei | 36 |
| 6 | Felder der <code>cran_authors.csv</code> , <code>TIMESTAMP_cff_authors(_new).csv</code> und <code>TIMESTAMP_cff_preferred_citation_authors(_new).csv</code> Datei | 39 |
| 7 | Felder der <code>readme.csv</code> Datei | 40 |
| 8 | Felder der <code>cff.csv</code> Datei | 42 |
| 9 | Felder der <code>cff_preferred_citation.csv</code> Datei | 43 |
| 10 | Felder der <code>bib.csv</code> Datei | 44 |

Quellcodeverzeichnis

| | | |
|---|---|----|
| 1 | Beispiel einer <code>CITATION.cff</code> -Datei | 24 |
| 2 | Beispiel einer <code>CITATION.bib</code> -Datei | 25 |

Abkürzungsverzeichnis

| | |
|------|--|
| CFF | Citation File Format. 8, 11, 19, 20, 22, 24, 28, 30, 39–42, 44 |
| CRAN | Comprehensive R Archive Network. 14, 18, 19, 28, 30, 34 |
| NED | Named entity disambiguation. 25, 26, 43 |
| NER | Named entity recognition. 24–26, 33, 35, 37 |
| OSS | Open-Source-Software. 10, 29, 30 |
| PEP | Python Enhancement Proposal. 14, 15 |
| PyPI | Python Package Index. 14–19, 28–30, 32–34, 37 |

Datenträger

```
/.....Wurzelverzeichnis
├── OrdnerA.....Ein Ordner auf dem Datenträger
│   ├── OrdnerB.....Ein Unterordner auf dem Datenträger
│   │   └── datei.xyz.....Eine Datei
│   └── thesis.pdf.....PDF-Datei dieser Bachelor-Thesis
```

Im Unterverzeichnis `tools` des Projekts findet sich das Perl-Skript `dirtree.pl`, mit welchem Inhalte für das `dirtree`-Environment (siehe oberhalb) semiautomatisch erstellt werden können.

Die Nutzung aus der Kommandozeile ist wie folgt:

```
perl dirtree.pl /path/to/top/of/dirtree
```

Quelle des Skripts:

<https://texblog.org/2012/08/07/semi-automatic-directory-tree-in-latex/>

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig 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-Plagiatssoftware zu ermöglichen.

A handwritten signature in black ink, reading "Kevin Zehner". The signature is written in a cursive style with a vertical line at the beginning.

Wismar, den 21. November 2024

Ort, Datum

Unterschrift