




# Master-Thesis

Identifikation und Vergleich von Autorenangaben zu Software  
zwischen verschiedenen Datenquellen

Eingereicht am: 28. September 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 e.V. (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

<b>1</b>	<b>Einleitung</b>	<b>6</b>
1.1	Motivation . . . . .	6
1.2	Vorgehen . . . . .	6
1.3	Gliederung . . . . .	6
<b>2</b>	<b>Grundlagen</b>	<b>7</b>
2.1	Zitation von Software . . . . .	8
2.2	Versionsverwaltung . . . . .	11
2.3	Software-Verzeichnisse und Paketverwaltung . . . . .	13
2.3.1	PyPI . . . . .	14
2.3.2	CRAN . . . . .	18
2.4	Zitierformate . . . . .	19
2.4.1	Citation File Format . . . . .	19
2.4.2	BibTeX . . . . .	19
2.5	Named Entity Recognition . . . . .	19
2.6	Disambiguierung von Autorennamen . . . . .	19
2.7	Fuzzy Suche . . . . .	19
<b>3</b>	<b>Methodik</b>	<b>20</b>
3.1	Datenbeschaffung . . . . .	20
3.1.1	Git . . . . .	20
3.1.2	PyPI . . . . .	20
3.1.3	CRAN . . . . .	20
3.1.4	Beschreibung . . . . .	20
3.1.5	Citation File Format . . . . .	20
3.1.6	BibTeX . . . . .	20
3.2	Abgleich . . . . .	20
<b>4</b>	<b>Ergebnisse</b>	<b>21</b>
<b>5</b>	<b>Diskussion</b>	<b>22</b>
5.1	Limitierungen . . . . .	22
<b>6</b>	<b>Fazit und Ausblick</b>	<b>23</b>
6.1	Fazit . . . . .	23
6.2	Ausblick . . . . .	23
	<b>Anhang A Beispielanlage</b>	<b>24</b>
	<b>Literaturverzeichnis</b>	<b>25</b>

<b>Abbildungsverzeichnis</b>	<b>27</b>
<b>Tabellenverzeichnis</b>	<b>28</b>
<b>Algorithmenverzeichnis</b>	<b>29</b>
<b>Quellcodeverzeichnis</b>	<b>30</b>
<b>Abkürzungsverzeichnis</b>	<b>31</b>
<b>Selbstständigkeitserklärung</b>	<b>33</b>

## **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.

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 2.2 und 2.3 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 2.4 eingegangen. Außerdem können in Fließtexten, beispielsweise der Beschreibung einer Software, ebenfalls Autoren genannt werden. In Abschnitt 2.5 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 2.6 beschrieben wird. Eine weitere einfache Möglichkeit des Abgleichs ist ein einfacher String Abgleich. Dieser funktioniert jedoch nicht immer, da Autoren unterschiedliche Schreibweisen ihres Namens verwenden können. Aus diesem Grund wird in Abschnitt 2.7 auf die Fuzzy-Suche eingegangen, welche eine Möglichkeit darstellt, um ähnliche Strings 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. **Spezifität:** 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 jedoch nicht in einem speziellen Format angegeben, sondern standardmäßig in der README-Datei, welche im Projekt enthalten ist. 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



**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).

müssen wie beispielsweise bei Neuauflagen von Büchern.

## 2.2 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.

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-Repositorys 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 Repositorys zu untersuchen. Außerdem wird auf grundlegende Funktionen von Git eingegangen, da diese für die Arbeit relevant sind. Der Aufbau der einzelnen Git Komponenten ist in 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). 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 (ebd.). 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. Unter anderem wird eine Commit-Nachricht, der Autor und der Zeitpunkt der Änderungen gespeichert. 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 Hauptbranch *master* genannt. Seit 2020 wird dieser jedoch in *main* umbenannt, um rassistische Konnotationen zu vermeiden (GitHub 2024b). 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.

## 2.3 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 Aktua-

lisieren 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).

### 2.3.1 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. Owner können dabei alle Änderungen am PyPI Projekt vornehmen und Betreuer können neue Versionen des Paketes veröffentlichen (Ingram 2023). In Abbildung 2 ist dargestellt, wie verifizierte und unverifizierte Daten auf PyPI dargestellt 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

## 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).

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 neustens 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 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 und Owner 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 Owner und Betreuer, welche über die API ausgegeben werden, enthalten den Benutzernamen auf PyPI, ein Vollname wird hierbei nicht ausgegeben. 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.

### 2.3.2 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 einem String 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.

## 2.4 Zitierformate

### 2.4.1 Citation File Format

### 2.4.2 BibTeX

## 2.5 Named Entity Recognition

## 2.6 Disambiguierung von Autorennamen

## 2.7 Fuzzy Suche

## **3 Methodik**

### **3.1 Datenbeschaffung**

#### **3.1.1 Git**

#### **3.1.2 PyPI**

#### **3.1.3 CRAN**

#### **3.1.4 Beschreibung**

#### **3.1.5 Citation File Format**

#### **3.1.6 BibT<sub>E</sub>X**

### **3.2 Abgleich**

## 4 Ergebnisse

## **5 Diskussion**

### **5.1 Limitierungen**

## **6 Fazit und Ausblick**

### **6.1 Fazit**

### **6.2 Ausblick**

## A Beispielanlage

Beispieltext.



## Literaturverzeichnis

- 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).
- Chacon, Scott (2024). *Git - git-shortlog Documentation*. URL: <https://git-scm.com/docs/git-shortlog> (besucht am 21.05.2024).
- CRAN Team (Mai 2024). *The Comprehensive R Archive Network*. URL: <https://cran.r-project.org/> (besucht am 21.05.2024).
- Csárdi, Gábor und Maëlle Salmon (2023). *pkgsearch: Search and Query CRAN R Packages*. URL: <https://github.com/r-hub/pkgsearch>.
- 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).
- (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).
  - (20. Sep. 2024b). *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).
- 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).
- Lindner, Jannik (3. Mai 2024). *Version Control Systems Industry Statistics*. URL: <https://worldmetrics.org/version-control-systems-industry-statistics/> (besucht am 21.05.2024).
- Meššan, Lukáš (18. Mai 2024). *git-quick-stats*. Version 2.5.6. URL: <https://github.com/arzzzen/git-quick-stats> (besucht am 21.05.2024).
- 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).

- Python Software Foundation (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).
- 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).
- 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).
- 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).

## Abbildungsverzeichnis

1	Übersicht über die Git-Komponenten . . . . .	11
2	PyPI Verifizierte und unverifizierte Daten . . . . .	15

## Tabellenverzeichnis

## Algorithmenverzeichnis

## Quellcodeverzeichnis

## Abkürzungsverzeichnis

CFF	Citation File Format. 8
CRAN	Comprehensive R Archive Network. 13, 17
PEP	Python Enhancement Proposal. 13
PyPI	Python Package Index. 13–17

## 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 large initial 'K' and a long, sweeping underline.

Wismar, den 28. September 2024

Ort, Datum

Unterschrift