



Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,
Arbeitsgruppe Software Engineering

Messung der Informationstypen-Häufigkeiten in der Python-Dokumentation

Sven Wildermann
Matrikelnummer: 4567553
bachelorarbeit@wildermann.berlin

Betreuer und Gutachter: Prof. Dr. Lutz Prechelt
Zweitgutachterin: Prof. Dr. Fehr

Berlin, 12. August 2014

Zusammenfassung

Walid Maalej und Martin P. Robillard veröffentlichten im September 2013 einen Artikel [MR13], in dem sie die Dokumentationen der Programmiersprachen Java und .NET auf ihren Informationsgehalt hin untersucht und verglichen haben. Diese Untersuchung wird im Rahmen dieser Bachelorarbeit auf die Dokumentation von Python mit einigen Abweichungen übertragen. Die von Maalej und Robillard eingeführte Taxonomie der in Dokumentationen anzutreffenden Wissenstypen wurde hierfür auf die Eigenheiten von Python angepasst. Die Einordnung von Teilen der Dokumentationen zu den verschiedenen Wissenstypen wird von Gutachtern im Rahmen eines Forschungspraktikums geleistet und erfolgt mit Hilfe eines eigens hierfür geschriebenen Werkzeuges. Dieses wurde mit Hilfe des auf Python basierenden Webframeworks Django umgesetzt. Für die Aufteilung der HTML-Gesamtdokumentation in kleinere Teile und den Import in die Datenbank wurde ein Python-Skript angefertigt, welches für die Syntaxanalyse das Paket BeautifulSoup4 verwendet. Die statistische Auswertung erfolgte ebenfalls mit Python. Die Ergebnisse werden wo möglich in Bezug auf die vorhergehende Studie verglichen und interpretiert.

Danksagungen

Zu erst möchte ich Prof. Dr. Lutz Prechelt für die intensive Betreuung und Begutachtung dieser Arbeit danken. Weiterhin möchte ich den Studenten der Freien Universität Berlin Jakob Warkotsch, Josephine Mertens, Jakob Lennart Dührsen, Leon Martin George, Malte Detlefsen, Michael Christian Koeck und Robert Kappler für die Datenerhebung ebenso danken wie Herrn Schmeisky und Herrn Zieris von der AG Software Engineering. Danke auch an Christian Salzmann vom technischen Support des Instituts für Informatik an der Freien Universität Berlin. Mein besonderer Dank geht an Phil Stelzer für die Unterstützung in der Webentwicklung. Außerdem möchte ich meiner Ehefrau, Anne Stephanie Wildermann, für die geistige Unterstützung während der Erstellung dieser Arbeit und für die Studienjahre davor bedanken. Meinen Eltern und Schwiegereltern danke ich für die Unterstützung während meiner gesamten Studienzeit.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

12. August 2014

Sven Wildermann

Inhaltsverzeichnis

1	Einleitung	1
1.1	Aufbau der Arbeit	1
2	Grundlagen	2
2.1	Artikel von Maalej und Robillard	2
2.2	Kodier-Handbuch	3
2.2.1	Markierungen	3
2.2.2	Kleinere Änderungen	5
3	Konzeption	6
3.1	Dokumentationseinheiten	6
3.2	Stichprobe	8
3.3	Goldstichprobe	9
3.4	Zeitaufwand	10
3.5	CADo-Tool vs. Eigenentwicklung	11
4	Entwicklung und Durchführung	12
4.1	Extrahierer	12
4.1.1	BeautifulSoup4	13
4.1.2	Implementierung	14
4.2	Typisierungswebsite	16
4.2.1	Anforderungen	16
4.2.2	Umsetzung	16
4.2.2.1	Navigationsleiste	17
4.2.2.2	Login	17
4.2.2.3	Einheit typisieren	17
4.2.2.4	Noch nicht typisierte Einheiten	18
4.2.2.5	Typisierte Einheiten	19
4.2.2.6	Persönliche Statistik	20
4.2.2.7	Kodierhandbuch	21
4.2.2.8	Bugtracker	21
4.2.2.9	Allgemeine Statistik	22
4.2.2.10	Zufällige Einheit	22
4.2.2.11	Logout	23
4.2.2.12	Datenbankmodell	23
4.2.2.13	Hosting	23
4.3	Gamification	23
5	Auswertungen	24
5.1	Konfusionen	24
5.1.1	Diskussion: Konfusion Nr. 7	26
5.1.2	Hilfsmittel	26

5.2	Zusammenführen der Goldstichproben	27
5.2.1	Zusammenführen der Stichproben	27
5.2.1.1	Markierungsergebnis bestimmen	28
5.3	Auswertung der Typisierungen	28
5.3.1	Übereinstimmung pro Wissenstyp nach Gutachtern . .	28
5.3.2	Übereinstimmung nach Wissenstyp	29
5.3.3	Übereinstimmung nach Einheiten	29
5.3.4	Rechenbeispiel	30
5.3.5	Unstimmigkeiten zur Goldstichprobe	31
5.3.6	Verträglichkeit von Einheiten	31
6	Fazit	34
7	Anhang	35
7.1	Technologien	35
7.1.1	Django	35
7.1.2	Python	35
7.1.3	BeautifulSoup4	35
7.1.4	Coffeescript	35
7.1.5	Postgres	35
7.1.6	Ajax	35
7.1.7	R	35
7.2	Kodierhandbuch - vollständig	35
7.3	Glossar	35

Abbildungsverzeichnis

1	Wissenstypen in [MR13]	2
2	Durchschnittliche Textlängen je Kategorie	7
3	Gesamthäufigkeiten der Einheiten	7
4	Gesamthäufigkeiten der Kategorien	8
5	Stichprobengrößen	9
6	Goldstichprobengrößen	9
7	Einheiten pro Student inkl. Zeitaufwand	10
8	Zeitaufwand für die Goldstichprobe	11
9	Beispielhafte Struktur der HTML-Elemente	13
10	Login-Bereich	17
11	Einheit typisieren	18
12	Einheit typisieren (2)	18
13	Noch nicht typisierte Einheiten	19
14	Abgespeicherte Einheiten	20
15	Persönliche Statistik	21
16	Bugtracker	22
17	Konfusionshäufigkeiten	24

18	Abschätzung der besseren Typen bei Konfusionen	25
19	Funktion für die Konfusions-Stichprobe	26
20	Übereinstimmung nach Gutachtern	29
21	Übereinstimmung nach Einheit je Gutachter	30
22	Übereinstimmung nach Einheit je Kategorie	30
23	Verträglichkeit nach Gutachtern	32
24	Verträglichkeit nach Kategorien	33

Offtopic: Verbesserungen für die Bachelorarbeit

1. Ein DOM-Knotenbaum einer typischen HTML-Datei einbauen
2. UML-Diagramm des Tools
3. Datenbank-Entitäten aufzeigen
4. Möglicher Workflow (als Nicht-deterministischer Automat oder so)
5. Screenshot z.B. von Dokumentationseinheit1898 hinzufügen, um das Interface zu zeigen
6. Alle Links als Fussnoten anzeigen - sieht besser aus und ist besser für den offline-Gebrauch!
7. Nummerierung bei den Code-Schnipseln hinzufügen
8. Bei Bildern immer Bildunterschriften einfügen, die etwas zu dem gezeigten aussagen
9. Alle Kommentare im Code (und den code selbst) auf Rechtschreibfehler überprüfen
10. Related Work
11. Future Work
12. Wort "Klassifizierung" statt "Typisierung"
13. Links im Literaturverzeichnis anzeigen lassen
14. HTML-Fehler in der Original-Dokumentation aufzeigen (mindestens 2)
15. HASH-Wert des Quellcodes abdrucken
16. Alle geschriebenen Kommandos erwähnen
17. Stichprobe von Robert erwähnen
18. Passiv vermeiden!
19. Zusammenfassung: Auf Resultate eingehen (dafür aber nicht auf bs4 etc.)
20. Mit der Goldstrichprobe vergewissern: Wie oft waren die Studenten sich fälschlicherweise einig? Damit findet man einen stochastischen Fehler.

1 Einleitung

Zu jeder Programmiersprache gehört eine Dokumentation über die bereitgestellten Funktionalitäten, auch Referenzhandbuch genannt. Während die Vor- und Nachteile der Programmiersprachen häufig diskutiert werden, findet man nur wenig Analyse zu den Dokumentationen. Dabei trägt eine Dokumentation nicht unwesentlich zum Erfolg oder Misserfolg einer Programmiersprache bei. Entscheidend ist, wie gut die Entwickler mit den gebotenen Informationen zu recht kommen und wie schnell Antworten auf Benutzungsfragen gefunden werden können. Die Anzahl und Qualität der Programmbeispiele ist dabei mindestens genauso wichtig wie die Erläuterung von Funktionalitäten, Konzepten und Abhängigkeiten.

Um etwas über die Qualität von Dokumentationen aussagen zu können, muss erst einmal verstanden werden, welche Informationen zu welchen Teilen in dem jeweiligen Handbuch vorhanden sind. Der erste Teil dieser Frage wurde bereits von den Herren Maalej und Robillard [MR13] beantwortet. Sie fanden bei der Analyse der JAVA und .NET Dokumentationen insgesamt 12 gut unterscheidbare Wissenstypen, im Original heißen diese knowledge types.

Die Analyse über die Häufigkeit dieser Typen in der Dokumentation wird von Studenten innerhalb eines Forschungspraktikums am Institut für Informatik durchgeführt. Sie erhalten Ausschnitte aus der Dokumentation über eine hierfür entwickelte Website und geben dann an, welche der 12 Typen auf diese Einheit passen. Die genauen Regeln für die Bewertung dieser Einheiten gibt das Kodier-Handbuch an, welches im Wesentlichen aus der Originalstudie übernommen und auf Python angepasst wurde.

1.1 Aufbau der Arbeit

Am Beginn stelle ich die vorausgegangene Arbeit von Walid Maalej und Martin P. Robillard [MR13] vor, da diese die Grundlage für diese Bachelorarbeit bildet. Hierbei werde ich insbesondere auf die verschiedenen Informationstypen (auch Wissenstypen genannt) eingehen. Im Anschluss werde ich erklären, welche Änderungen an diesen Wissenstypen notwendig waren, um auf die Analyse mit Python zu passen. Die Programmierung des Werkzeugs für die Typisierung der Dokumentationseinheiten wird dann ebenso erläutert wie die Begründung für eine eigene Entwicklung zu diesem Zweck. Die Durchführung und Organisation inklusive der aufgetretenen Schwierigkeiten des Forschungspraktikums, innerhalb dessen Studenten eine bestimmte Stichprobe an Dokumentationseinheiten erhalten haben, um diese zu typisieren, wird ebenso thematisiert.

Zuletzt werden dann die ausgewerteten Ergebnisse vorgestellt.

2 Grundlagen

2.1 Artikel von Maalej und Robillard

Walid Maalej and Martin P. Robillard veröffentlichten im September 2013 den Artikel „Patterns of Knowledge in API Reference Documentation“ [MR13] und besprechen darin zum Einen eine Taxonomie von in Dokumentationen vorkommenden Wissenstypen und zum Anderen die durchgeführte Analyse der Programmiersprachendokumentationen von Java SDK 6 und .NET 4.0. Zum Finden dieser Taxonomie stellten Sie zu jedem neu gefundenen Wissenstyp eine neue Frage auf und erhielten so über 100 verschiedene Fragestellungen. Daraufhin wurden ähnliche Fragen zusammengefasst und schließlich die 12 in Abbildung 1 gezeigten verschiedenen Wissenstypen ausfindig gemacht. Auf der Website zu dieser Studie wurde dann zu dem der „Coding Guide“ [MRa] veröffentlicht, welcher für jeden Typ einen Fragenkatalog, Anmerkungen und Beispiele angibt. Die Vorgehensweise sah vor, dass die Gutachter für

Knowledge Type	Description (Excerpt)
Functionality and Behavior	Describes what the API does (or does not do) in terms of functionality or features. Describes what happens when the API is used (a field value is set, or a method is called).
Concepts	Explains the meaning of terms used to name or describe an API element, or describes design or domain concepts used or implemented by the API.
Directives	Specifies what users are allowed / not allowed to do with the API element. Directives are clear contracts.
Purpose and Rationale	Explains the purpose of providing an element or the rationale of a certain design decision. Typically, this is information that answers a “why” question: Why is this element provided by the API? Why is this designed this way? Why would we want to use this?
Quality Attributes and Internal Aspects	Describes quality attributes of the API, also known as non-functional requirements, for example, the performance implications. Also applies to information about the API’s internal implementation that is only indirectly related to its observable behavior.
Control-Flow	Describes how the API (or the framework) manages the flow of control, for example by stating what events cause a certain callback to be triggered, or by listing the order in which API methods will be automatically called by the framework itself.
Structure	Describes the internal organization of a compound element (e.g. important classes, fields, or methods), information about type hierarchies, or how elements are related to each other.
Patterns	Describes how to accomplish specific outcomes with the API, for example, how to implement a certain scenario, how the behavior of an element can be customized, etc.
Code Examples	Provides code examples of how to use and combine elements to implement certain functionality or design outcomes.
Environment	Describes aspects related to the environment in which the API is used, but not the API directly, e.g., compatibility issues, differences between versions, or licensing information.
References	Includes any pointer to external documents, either in the form of hyperlinks, tagged “see also” reference, or mentions of other documents (such as standards or manuals).
Non-information	A section of documentation containing any complete sentence or self-contained fragment of text that provides only uninformative boilerplate text.

Abbildung 1: Wissenstypen in [MR13]

jeden Wissenstyp bestimmen, ob dieser in der vorgelegten Einheit vorkommt oder nicht. Hierzu konnten für jede Einheit die verschiedenen Wissenstypen mit Hilfe von „CheckBoxes“ angekreuzt werden. Jede Einheit wurde von 2 Gutachtern unabhängig bewertet. Die Einheiten wurden in drei verschiedene Kategorien aufgeteilt:

1. Module [engl. modules] (entsprechen „packages“ in Java und „assemblies“ in .NET)
2. Typen [engl. types] (vor allem Klassen und Schnittstellen)

3. Mitglieder [engl. members] (Felder und Methoden)

Die Einheiten der Kategorie „Module“ wurden auf Grund der geringen Anzahl, der Unterschiedlichkeit (vor allem bzgl. der Länge) in Java und .NET und des geringen Informationsgehalts bei .NET dann aber nicht analysiert. Damit beschränkt sich die Original-Studie also auf Typen und Mitglieder. Die Stichprobe über die verbleibenden vier Kategorien (zwei je Programmiersprache) wurden mit 95% Konfidenzintervall und 2,5% Fehlerspanne gezogen, so dass insgesamt 5.575 Einheiten (431.136 Wörter) zufällig ausgewählt und auf 17 Gutachter verteilt wurden.

So sind insgesamt $5.574 * 2 = 11.148$ Bewertungen vorgenommen worden. Diese wurden dann im Hinblick auf die Übereinstimmung unter den Gutachtern und bezüglich der Wissenstypen analysiert. Ebenso wurden Auswertungen über die Fälle getroffen, in denen sich zwei Gutachter uneinig waren. Zudem konnten so Aussagen darüber getroffen werden, welche Wissenstypen bei welcher Einheitskategorie wie häufig vorkommen und ob bestimmte Wissenstypen mit einander korrelieren, also ob z.B. der Typ „Structure“ häufig zusammen mit „Functionality and Behaviour“ auftritt. Ebenso wurde analysiert, ob und wie ein Zusammenhang zwischen der Anzahl der gefundenen Wissenstypen und der Länge der Einheiten besteht.

Besonders die vorgestellten Wissenstypen und das Kodier-Handbuch [Coding Guide] waren für diese Folgestudie sehr nützlich und wurden deswegen übernommen und angepasst.

2.2 Kodier-Handbuch

Das in der Original-Studie [MR13] verwendete Kodier-Handbuch findet auch in dieser Studie Verwendung, um von allen Gutachtern die selben bzw. sehr ähnliche Ergebnisse erwarten zu können. Es dient als Anleitung bei der Bewertung der Einheiten. Während der Großteil des Handbuchs übernommen wurde und unverändert bleibt, gab es jedoch ein paar wesentliche Anpassungen.

2.2.1 Markierungen

Der wichtigste Unterschied zu dem originalen Kodier-Handbuch [MRa] ist, dass die Gutachtern nicht nur pro Einheit bewerten sollen, ob bestimmte Wissenstypen vorhanden sind, sondern Markierungen in einer Einheit vornehmen und pro Markierung einen Typ festlegen. Wie mit einem Textmarker soll jede Einheit vollständig bearbeitet werden und abschließend jedes Zeichen (mit wenigen Ausnahmen) markiert sein. Dies führt zu ergänzten Regeln über :

- Die Art Markierungen vorzunehmen
- Der Länge von Markierungen

- Die Notwendigkeit von Doppelmarkierungen in einem Segment

Diese Änderungen wurden im ersten Absatz des Kodier-Handbuchs [PW] wie folgt formuliert:

You will be presented with documentation blocks extracted from API reference documentation (Javadocs and the like). For each block, you will be also presented with the name of its corresponding package/namespace, class, method, or field. Your task is to read each block carefully and evaluate where the block contains knowledge of the different types described below. Apply the following rules when doing so:

- Consider the documentation initially one paragraph at a time. If the paragraph contains only information of one knowledge type, mark the whole paragraph with that type in one stretch. Never mark more than one paragraph at once.
- If multiple knowledge types mix within the paragraph, mark a contiguous stretch of one or more sentences with one type and the next stretch with another.
- If necessary, treat subsentences connected with conjunctions such as „and“, „or“, „but“, or with colon or semicolon like complete sentences.
- A sentence (or such subsentence) as a whole is never marked with more than one type, but sometimes phrases within the sentence will require a separate marking with a different type. Double-marking the same text with two types is allowed (and required) in this case. To create such annotations uniformly, we work in two passes:
 - Pass 1: Prefer longer segments of a complete sentence or several. Annotate subsentences only rarely. If a sentence contains knowledge of more than one type (which happens quite often), look if one of them is clearly dominant for the overall role of the sentence in the documentation block. If so, annotate only that dominant type to the whole sentence and do not annotate any of the other types yet.
 - Pass 2: After pass 1, many relevant annotations will be missing. We now add those on top of the pass 1 annotations as double annotations. For the double annotations, we still prefer complete subsentences where possible (or other clearly delineated parts such as parentheses), but choose the shorter of two possibilities whenever we are unsure.

- Rate the knowledge type as true only if there is clear evidence that knowledge of that type is present in the stretch. If you have doubts, consult the type's definition below. If the doubts do not disappear, do not annotate that type.
- However, all text of the documentation must be marked with a type. (Only hand-written documentation, not the signature itself and not the placeholders [Something removed here] that indicate left-out nested documentation blocks).

Read (and re-read whenever needed) the following descriptions very carefully. They explain how to recognize each knowledge type.

2.2.2 Kleinere Änderungen

Zudem waren einige kleine Änderungen notwendig, die sich entweder aus dem Stil der Python-Dokumentation ergeben oder als sinnvoller bei der Bewertung von Einheiten ergeben haben. Folgende Semantische Änderungen gab es dabei:

- Informationen über bestimmten Input einer Funktion oder Methode, welcher zu einer „Exception“ führt (und nur dann), soll als „Directive“ und nicht als „Functionality and Behavior“ markiert werden.
- Die simple Nennung von gültigen Parametertypen wird als nicht als „Directive“ angesehen, sofern nicht Schlüsselwörter wie „must“ oder „have to“ etc. verwendet werden.
- Der Ausdruck „Changed in version x.y.“ soll als „Environment“ markiert werden. Der darauf anschließend Text kann ebenfalls „Environment“ sein, muss es aber nicht.
- Platzhalter der Form „[Something removed here]“ sollen nicht markiert und bewertet werden, da diese lediglich auf ausgelassene, verschachtelte Einheiten hinweisen (siehe Abschnitt „Extrahierer“)
- Der Wissenstyp „Structure“ wird treffender in „Structure and Relationship“ umbenannt.

3 Konzeption

In diesem Abschnitt werden grundlegende Faktoren und Vorgehenweisen erläutert, die während der Bachelorarbeit wichtig geworden sind. Diese betreffen sowohl die Beschaffenheit der Dokumentationseinheiten als auch die Stichprobenziehung und der daraus resultierende Zeitaufwand für die Gutachter.

3.1 Dokumentationseinheiten

Mit Dokumentationseinheiten werden die Teilabschnitte aus der Dokumentation bezeichnet, die ein Gutachter für die Bewertung zusammenhängend angezeigt bekommt. Diese Einheiten wurden anhand der HTML-Syntax in der Original-Dokumentation bestimmt. Es sind folgende Kategorien mit den dazugehörigen HTML- Syntaxen aufgetreten:

1. Methoden (engl. methods)
 - `<dl class="method"> Text </dl>`
 - `<dl class="classmethod"> Text </dl>`
 - `<dl class="staticmethod"> Text </dl>`
 - `<dl class="function"> Text </dl>`
2. Felder (engl. fields)
 - `<dl class="attribute"> Text </dl>`
 - `<dl class="data"> Text </dl>`
3. Module (engl. modules)
 - `<div class="section"> Text </div>`
4. Klassen (engl. classes)
 - `<dl class="class"> Text </dl>`
 - `<dl class="exception"> Text </dl>`
5. Beschreibungen (engl. describe)
 - `<dl class="describe"> Text </dl>`

Die Kategorien unterscheiden sich von denen aus der Originalstudie in der Form, dass Felder und Methoden unabhängig von einander geführt werden und zusätzlich noch die Beschreibungselemente hinzugekommen sind. Anders als in der Originalstudie wird die Kategorie Module in der späteren Analyse nicht ausgelassen. Die Einheiten unterscheiden sich nebst Inhalt auch stark in ihrer Textlänge. Während die Einheiten der Kategorien 1, 2, 4

und 5 tendenziell eine kleine Textlänge haben, sind die Module (3. Kategorie) in der Regel länger:

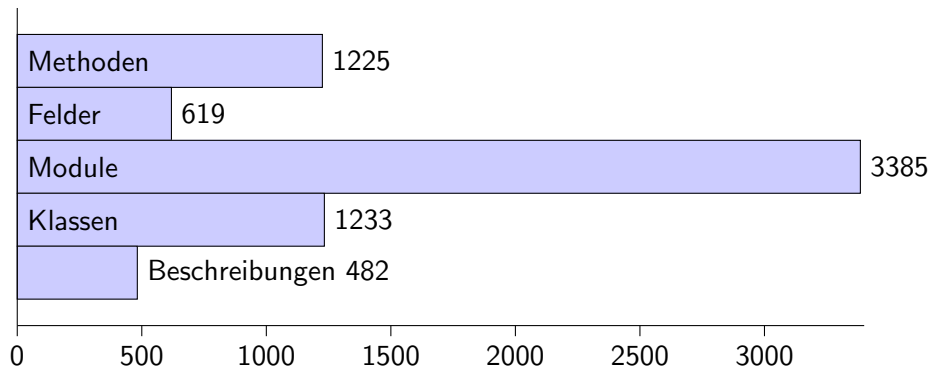


Abbildung 2: Durchschnittliche Textlängen je Kategorie

Sehr große Unterschiede gibt es zudem in der Häufigkeit verschiedener Typen. Die wenigsten Vorkommen gibt es von den Einheiten „describe“, „classmethod“ und „staticmethod“. „Methods“ treten dagegen am häufigsten auf. Trotz des geringen Auftretens der „describe“-Elemente haben wir uns dafür entschlossen, diese als eigene Kategorie zu behandeln, da diese bei den bisher behandelten Programmiersprachen (Java und .NET) nicht existent waren.

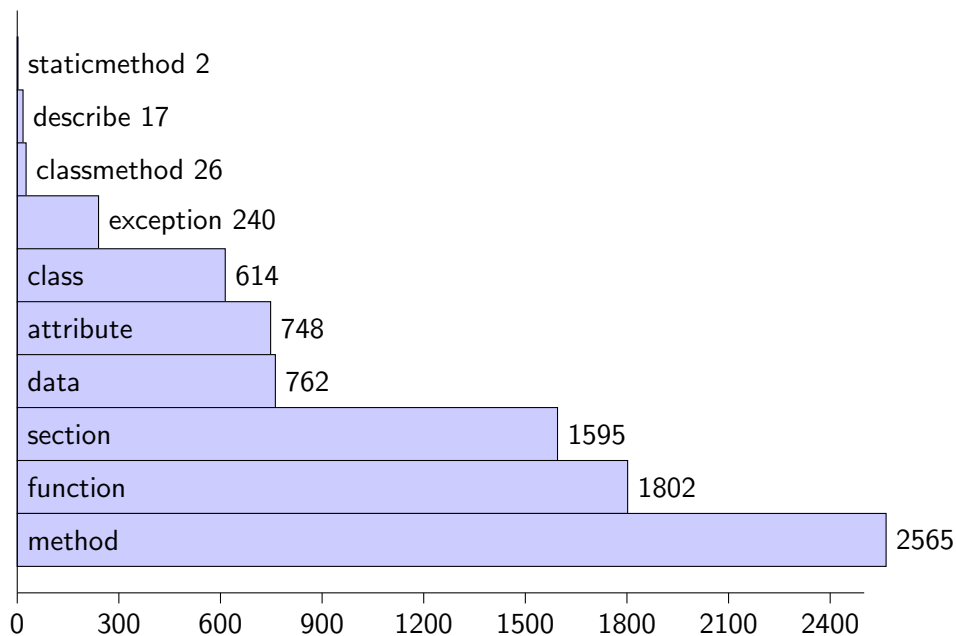


Abbildung 3: Gesamthäufigkeiten der Einheiten

Die Verteilung auf Kategorie-Ebene zeigt ebenfalls einen deutlichen Überschuss an Methoden, nämlich fast drei mal so viele wie es Felder gibt. Auf Grund der einelementigen Kategorien „Module“ und „Beschreibungen“ decken sich hier deren Häufigkeiten exakt mit denen der „section“ und „describe“-Einheiten, so dass auch hier die Beschreibungselemente den geringsten Anteil darstellen.

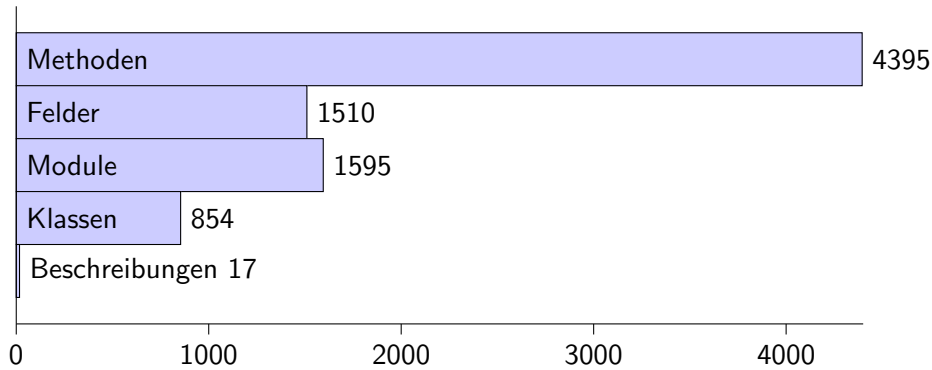


Abbildung 4: Gesamthäufigkeiten der Kategorien

Mit Hilfe dieser Informationen können die im nächsten Abschnitt beschriebenen Stichprobengrößen berechnet werden.

3.2 Stichprobe

Um für alle Kategorien ein aussagekräftiges Ergebnis der späteren Begutachtungen erzielen zu können, werden die Stichproben unter Vorgabe von Konfidenzintervall und Fehlerspanne separat von einander pro Kategorie gezogen. Die minimale Anzahl der pro Kategorien zu ziehenden Einheiten wird mit dieser Formel berechnet [METM12]:

$$MIN = \frac{n_0}{1 + \frac{n_0 - 1}{Gesamtmenge}}$$

wobei n_0 wie folgt berechnet wird:

$$n_0 = \frac{Z^2 * 0.25}{e^2}$$

Dabei ist Z die Angabe des Konfidenzintervalls als z-score und e die tolerierte Fehlerrate. Bei einem Konfidenzintervall von 95% ist $Z=1.96$ [METM12]. Mit einer Fehlerrate von 5% ergibt sich für $n_0 = 384,16$ und dadurch dann die folgenden minimale Stichprobengrößen für die jeweiligen Kategorien:

Kategorie	Gesamtmenge	Stichprobengröße
Methoden	4395	651
Felder	1510	306
Module	1595	310
Klassen	854	265
Beschreibungen	17	16
Summe	8371	1548

Abbildung 5: Stichprobengrößen

3.3 Goldstichprobe

Zudem wurde eine so weitere Stichprobe, die Goldstichprobe gezogen. Diese hat folgenden Zweck:

- Absicherung der Messung der Korrektheit der Gutachterergebnisse und
- Bestimmung eines Wertes für sehr hohe Typisierungsqualität

Diese Stichprobe wurde dann von vier unabhängigen Gutachtern der Freien Universität Berlin (Institut für Informatik) jeweils vollständig typisiert. Diese Gutachter waren:

- Prof. Dr. Prechelt, AG Software Engineering
- Herr Schmeisky, AG Software Engineering
- Herr Zieris, AG Software Engineering
- Sven Wildermann, Verfasser dieser Bachelorarbeit

Wobei die Herren Schmeisky und Zieris zusammen eine Stichprobe bearbeitet haben.

Die Größe dieser Stichprobe wurde auf 2% der Gesamtmenge aller Einheiten festgelegt, damit bei Beibehaltung einer repräsentativen Größe der Aufwand im Rahmen blieb. Für die einzelnen Kategorien ergaben sich somit folgende Mengen:

Kategorie	Gesamtmenge	Stichprobengröße
Methoden	4395	88
Felder	1510	30
Module	1595	32
Klassen	854	17
Beschreibungen	17	1
Summe	8371	168

Abbildung 6: Goldstichprobengrößen

Dabei wurde der Wert für die Kategorie „Beschreibungen“ um eins erhöht, da diese Elemente sonst überhaupt nicht in der Goldstichprobe vorgekommen wären.

3.4 Zeitaufwand

Da die Gutachter im Rahmen des Kurses „Forschungspraktikum“ an der Freien Universität Berlin für fünf ECTS ¹ die Einheiten bewerten haben, sollte der Aufwand so verteilt werden, dass die benötigten Punkte erreicht werden, aber gleichzeitig der Zeitaufwand nicht überschritten wird (ein ECTS entspricht etwa 25 bis 30 Arbeitsstunden). Hierfür muss der Aufwand des Markierens pro Einheit geschätzt werden, so dass die Einheiten pro Gutachter festgelegt werden können. Diese Überlegungen sind auch schon in der Stichprobenziehung mit eingeflossen und haben dazu geführt, dass die Fehlerrate auf 5% gesetzt wurde. Da vor dem Start noch eine intensive Einarbeitung inklusive Hausarbeiten durchgeführt wurde, sind für die eigentliche Bewertung der Einheiten noch drei ECTS pro Student veranschlagt worden. Dadurch, dass jede Einheit von zwei Gutachtern bewertet werden sollte, mussten vorher die Stichprobengrößen mit zwei multipliziert werden, um die Gesamtanzahl der Bewertungen zu erhalten. Diese wurde dann auf die 7 Studenten verteilt (in der Tabelle wurde gerundet) und mit einem grob geschätzten, durchschnittlichen Zeitaufwand pro Einheit multipliziert:

Kategorie	Anzahl der Bewertungen	Einheiten pro Student	Zeitaufwand pro Einheit	Zeitaufwand gesamt
Methoden	1302	186	5 min	930 min
Felder	12	87	5 min	435 min
Module	620	89	15 min	1335 min
Klassen	530	76	10 min	760 min
Beschreibungen	32	5	3 min	15 min
Summe	3096	443	Ø7.6 min	57,92 h

Abbildung 7: Einheiten pro Student

Zusätzlich waren die Gutachter dazu angehalten, regelmäßig das Kodier-Handbuch zu lesen, um das übergreifende Verständnis nicht zu verlieren sowie gelegentlich mit dem BugTracker² umzugehen. Hierfür wurden zusätzlich insgesamt noch etwa 10 Arbeitsstunden investiert, wobei der tatsächliche Aufwand je Gutachter stark abweichen kann.

Analog kann auch der Zeitaufwand für die Bearbeitung der Goldstichprobe berechnet werden:

¹European Credit Transfer System

²Ein System zur Erfassung von Defekten und Verbesserungsvorschlägen

Kategorie	Anzahl der Einheiten	Zeitaufwand pro Einheit	Zeitaufwand gesamt
Methoden	88	5 min	440 min
Felder	30	5 min	150 min
Module	32	15 min	480 min
Klassen	17	10 min	170 min
Beschreibungen	1	3 min	3 min
Summe	168	Ø7.6 min	20,72 h

Abbildung 8: Zeitaufwand für die Goldstichprobe

Da die Goldstichprobe drei mal bearbeitet wurde, ergibt sich so ein Gesamtaufwand für die Bearbeitung dieser von 62,16 Stunden. Somit stellt sich als nächstes die Frage, wie diese Einheiten extrahiert und bearbeitet werden sollen.

3.5 CAdo-Tool vs. Eigenentwicklung

Im Rahmen der Forschung von Maalej und Robillard [MR13] wurde ein Tool mit dem Namen CAdo³ geschaffen, welches folgende Werkzeuge und Fähigkeiten mit sich bringt (übersetzter Auszug) [MRb] :

- API-Dokumentationen aus Online-Quellen extrahieren
- Ziehen von zufälligen, stratifizierten Stichproben
- Erstellung eines Codierungsschemas
- Zuweisen von Einheiten zu Gutachtern
- Berechnung der Übereinstimmung von Gutachtern

Aus Sicht der Kodierer birgt dieses Tool zusätzlich noch folgende Fähigkeiten [MRb]

- Online und offline login
- Laden der zugewiesenen Einheiten
- Einheiten typisieren (Codierungen hinzufügen)
- Kodierhandbuch anzeigen
- Darstellung der Dokumentation
- Kodiersitzungen zwischenspeichern und laden

³Content Analysis for Software Documentation

- Statistiken ansehen

Auf Grund der Vielzahl der für uns nützlichen Fähigkeiten, stellte sich die Frage, ob dieses Werkzeug für diese Bachelorarbeit ebenfalls genutzt werden kann. Also sollte das Werkzeug heruntergeladen und ausprobiert werden. Schon beim Herunterladen gab es Schwierigkeiten, da der auf der Website veröffentlichte Link ungültig war. Nach Rückfrage bei den Autoren wurde dieser Misstand dann schnell behoben. Der Test des Werkzeugs erwies sich dann ebenfalls als schwierig, da es noch in der alpha-Version⁴ und ohne Benutzerhandbuch vorlag. Zudem gab es bei möglichen Fragen keinen einheitlichen Ansprechpartner und keine garantierte Antwortzeit der Entwickler, was ebenfalls zu einem höherem Risiko in der Benutzung geführt hätte.

Somit war es sinnvoller, die für diese Bachelorarbeit benötigten Werkzeuge selbst zu entwickeln. Dies hatte noch weitere Vorteile:

- Grundverständnis des vorliegenden Programmcodes
- Änderungen des Grundkonzepts eher möglich
- schnellere Weiterentwicklung bei Bedarf
- weniger Kommunikationsaufwand

Wir haben uns entschlossen, diese Werkzeuge dann mittels Python zu entwickeln, um zusätzlich noch Synergieeffekte bezüglich des Erlernens und Verstehens der Programmiersprache Python auszunutzen. Da der Zugriff verteilt von den Rechnern der Gutachter erfolgen sollte, war die Entwicklung einer Website zu diesem Zweck sinnvoll, so dass Restriktionen bezüglich der Betriebssystemwahl ausgeräumt werden konnten.

Diese Entscheidung führte dazu, dass wir die Typisierungen überhaupt anhand von Markierungen durchführen konnten (siehe Erläuterung in Kapitel [2.2.1](#)).

4 Entwicklung und Durchführung

Dieses Kapitel zusammenfassen

4.1 Extrahierer

Um die Dokumentationseinheiten aus der HTML-Dokumentation von Python zu erhalten, war es nötig, einen Extrahierer als Skript zu schreiben. Dieser wurde in Python3 mit Hilfe von BeautifulSoup4 angefertigt. BeautifulSoup wurde genutzt, da es im Gegensatz zu regulären Ausdrücken eine

⁴frühes Entwicklungsstadium ohne Garantie auf fehlerfreies funktionieren

verständlichere Syntax bietet, was zu einer leichteren und wartbareren Entwicklung führt.

Entsprechend der Definitionen von Dokumentationseinheiten sollten also die HTML-Schnipsel getrennt von einander in eine Datenbank importiert werden. Diese Einheiten sind allerdings häufig geschachtelt, so dass eine Methodendeklaration in der Regel innerhalb einer Klassenbeschreibung vorkommt, welche wiederum innerhalb einer Sektion anzutreffen ist. Um Dopplungen bei der Typisierung zu vermeiden, wurden deswegen Platzhalter der Form „[something removed here]“ an solchen Stellen eingebaut. Platzhalter haben einen wichtigen Vorteil gegenüber dem einfachen Weggelassen dieser Elemente: So sieht auch der Gutachter, dass hier etwas von der Original-Dokumentation abweicht und kann sich somit die entstandenen Lücken erklären. Dieser Fall tritt besonders häufig bei Sektionen auf, da diese in der Regel alle weiteren Elemente beinhalten. Aneinanderreihungen von Platzhaltern wurden wieder zu einem Platzhalter zusammengefasst.

Eine mögliche Struktur der Verschachtelung von Dokumentationseinheiten sieht ohne Beschränkung der Allgemeinheit so aus:

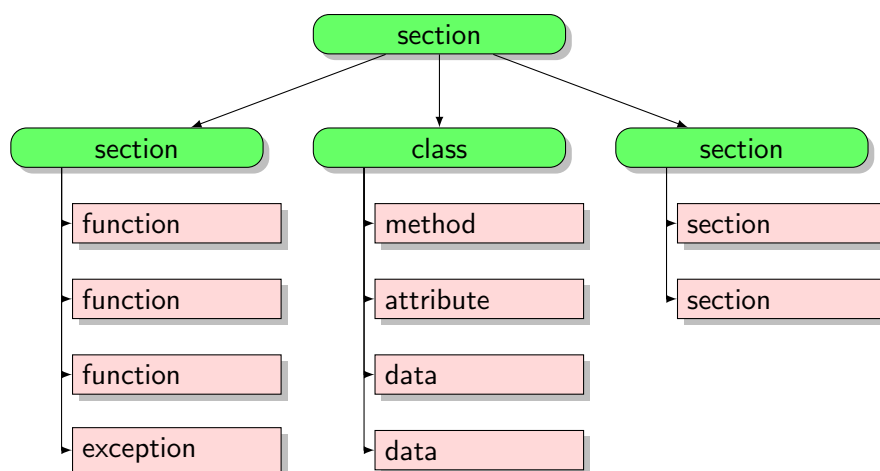


Abbildung 9: Beispielhafte Struktur der HTML-Elemente

4.1.1 BeautifulSoup4

Für das Parsen der HTML-Einheiten wurde BeautifulSoup4 verwendet, da es ein hierfür geschaffenes, mächtiges Werkzeug darstellt und gleichzeitig auf Python basiert. Somit konnten Synergieeffekte genutzt werden und sich während des Schreibens des Skriptes weiter in Python eingearbeitet werden. Zudem bietet der Einsatz von BeautifulSoup viele Vorteile gegenüber dem Parsen mittels regulären Ausdrücken. In erster Linie ist der Code lesbarer und verständlicher, sowohl für den Entwickler selbst als auch für Dritte. Außerdem gibt es eine ausführliche Dokumentation inklusive zahlreicher Bei-

spiele und auch die Community hinter BeautifulSoup ist groß genug, um auf Internetportalen wie [Stackoverflow](https://stackoverflow.com)⁵ Unterstützung erhalten zu können. Die Implementierung selbst wird im nachfolgenden Kapitel erklärt.

4.1.2 Implementierung

Bevor die nötigen Schritte mit BeautifulSoup4 durchgeführt werden konnten, war es nötig, die vollständig Dokumentation herunterzuladen und die notwendigen Dateien ausfindig zu machen. Unter dem Link <https://docs.python.org/3.4/archives/python-3.4.1-docs-html.zip> ist die gesamte Dokumentation in einem HTML-Format verfügbar. Interessant sind jedoch lediglich die Dateien in dem Unterordner „library“ innerhalb dieser zip-Datei, da Tutorials, Inhaltsverzeichnisse, FAQ und zusätzliche Informationen analog der Studie von Maalej und Robillard [MR13] ausgeschlossen wurden.

Der Extrahierer durchsucht anfangs alle Dateien in dem Unterordner library und fügt den vollständigen Pfad dieser in eine Liste:

```
mypath = "python-3.4.0-docs-html/library/"
files = get_list_of_filepath(mypath)
```

Aus jeder Datei wird dann ein BeautifulSoup-Objekt gemacht, welches die verschachtelte, innere HTML-Datenstruktur repräsentiert:

```
for file in files:
    soup = file_to_soup(file)
```

Dieser Schritt ist nötig, um im Anschluss mittels BeautifulSoup-API die einzelnen Dokumentationseinheiten extrahieren zu können. Hierfür wird der Befehl find-all genutzt. Um eine einfache und fehlerfreie Bedienung zu ermöglichen, wurde eine Funktion geschrieben, die das DOM⁶-Element samt Attribute entgegen nimmt:

```
def grab_elements(soup, elem, attr1, attr2):
    """grabs the different elemens with the given
    attributes out of a soup"""
    return soup.find_all([elem], attrs={attr1: [attr2]})
```

Der Aufruf zum parsen alle Elemente der Form

```
<dl class="method">
Inhalt des Elements
</dl>
```

und abspeichern dieser in einer Liste funktioniert dann wie folgt:

⁵[www.stackoverflow.com](https://stackoverflow.com)

⁶Document Object Model

```
methods = grab_elements(soup, "dl", "class", "method")
```

Dieser Schritt wurde analog für alle zehn verschiedenen Elementstypen ausgeführt. Um später Aussagen über die Lage der Texte innerhalb einer Einheit zu erhalten, wurden zudem die Startoffsets der Elemente berechnet und später zusammen mit dem Endoffsets in der Datenbank abgelegt, wobei sich das Endoffset jeweils sehr leicht errechnen lässt: $Ende = Start + Laenge$.

Für die Bestimmung der Startoffsets wurden die einzelnen Elemente in ihrer Datei mittels „find“ aus BeautifulSoup gesucht und die Rückgabe, also der Index an der Stelle des Auftretens, gespeichert. Auch hierfür gibt es eine eigene Funktion, um den Aufruf lesbarer zu gestalten:

```
def get_offsets(soup_str, elems):
    """soup_str is a soup element converted to a
    string
    elems is a array of soup-elements
    """
    offsets=[]
    for each in elems:
        find_index = soup_str.find(str(each))
        offsets.append(find_index)
    return offsets
```

Außerdem wurde dann für jedes Element das Vaterelement gesucht, zum Einen, um die Elemente später leichter wieder in die richtige Reihenfolge bringen zu können und zum Anderen, um den Gutachtern die Möglichkeit zu geben, sich den engeren Kontext, in dem die zu bewertende Einheit steht, genauer anzusehen:

```
for child in childs:
    parents.append(child.findParent())
return parents
```

Wegen der bereits erwähnten, verschachtelten HTML-Struktur der Elemente musste ein Weg gefunden werden, um zu verhindern, dass ein inneres Element zweimal von den Gutachtern typisiert wird. Also wurden alle Vorkommen von inneren Elementen in den äußeren Elemente durch folgende Platzhalter ersetzt:

```
placeholder = '[something removed here]'
```

Da so in vielen Fällen, z.B. bei der Aufzählung von Methoden und Attributen, seitenweise Platzhalter entstanden wären, wurden im Anschluss direkt aufeinander folgende Platzhalter wieder zu einem Platzhalter zusammengefasst:

```
def summarize_placeholders(parent, string):
```

```
""" removes multiple placeholders if they are next
    to each other """
for elem in parent.find_all("p"):
    while isinstance(elem.next_sibling,
                      Tag) and elem.next_sibling.
                        name == 'p' and elem.text
                        == string and elem.
                        next_sibling.text ==
                        string:
        elem.next_sibling.extract()
```

Die final zur Verfügung stehenden Informationen wurden dann in einer Datenbank abgespeichert.

4.2 Typisierungswebsite

4.2.1 Anforderungen

Damit die Einheiten von den Studenten typisiert werden konnten, musste ein Werkzeug geschaffen werden, welches mindestens folgende Eigenschaften aufweist:

- Ein- und Auslogfunktion
- Betriebssystemunabhängige Online-Erreichbarkeit
- Anzeige der dem Student zugewiesenen Einheiten
- Markierung von Segmenten und Zuweisung zu Informationstypen
- Anzeige der gesetzten Markierungen
- Anzeige der Anzahl noch verbleibender und schon gespeicherter Einheiten
- Anzeige der eigenen Übereinstimmung mit anderen Studenten

4.2.2 Umsetzung

Für die Umsetzung der genannten Anforderungen wurde das Webframework „Django“ verwendet - da es auf „Python“ basiert und es eine große Gemeinschaft gibt, die im Zweifel bei Herausforderungen unterstützen können. Für die Beantwortung der anfänglichen Fragen wurde das in Berlin stattfindende Meetup der Django User Group⁷ genutzt. Darüber hinaus gibt es eine Vielzahl von Möglichkeiten, mit anderen Django-Entwicklern zu kommunizieren, z.B. über die Mailingliste oder den IRC-Chat (deutsch / weltweit).

⁷<http://www.meetup.com/django-user-group-berlin/>

4.2.2.1 Navigationsleiste

Unterschiedliche Rollen, Unterschiedliche Farben

4.2.2.2 Login

Der Login soll allen Gutachtern ermöglichen, sich mit einem individuellen Account anzumelden, um geschützten Zugriff auf die Dokumentationseinheiten und die persönliche Statistik zu erhalten. Weiterhin sollen Administratoren sich über diese Maske anmelden können, um Zugriff auf Administratorfunktionalitäten zu erhalten.

Der Login-Bereich wurde mit Hilfe des bereits in Django vorhandenen Authentifizierungsmoduls gefertigt. Angepasst wurde das Design an das der restlichen Website. Da eine Funktion für das Zurücksetzen des Passwortes nicht benötigt wurde, wurde diese auch nicht implementiert.



Abbildung 10: Login-Bereich

4.2.2.3 Einheit typisieren

Nach dem Login wird der Gutachter direkt zu der nächsten zu typisierenden Einheit weitergeleitet. Auf dieser werden neben der Navigation noch die Identifikationsnummer der Dokumentationseinheit, der Text der Einheit und Steuerungselemente angezeigt. Die Steuerungselemente sind im Einzelnen:

1. Save - speichert die aktuelle Markierung
2. Delete All - löscht alle gesetzten Markierungen des Gutachter bei dieser Einheit
3. Show Parent - zeigt das direkte Vatorelement der Dokumentationseinheit an, siehe Abbildung 9
4. Show File - zeigt die gesamte HTML-Datei, aus welcher diese Dokumentationseinheit stammt, im Original an

Sobald Text von dieser Einheit mit dem Mauscursor markiert wurde, erscheint ein Fenster für die Auswahl eines Wissenstypes. Die Position der Markierung wird dann zusammen mit dem Wissenstyp mit Hilfe von Javascript zwischengespeichert und dann beim Klicken auf „save“ über Ajax als POST-request an den Controller⁸ (in Django ist es die views.py) gesendet. Nach dem Speichern wird dann die nächste Einheit angezeigt.

⁸entsprechend des „Model View Controller“-Musters

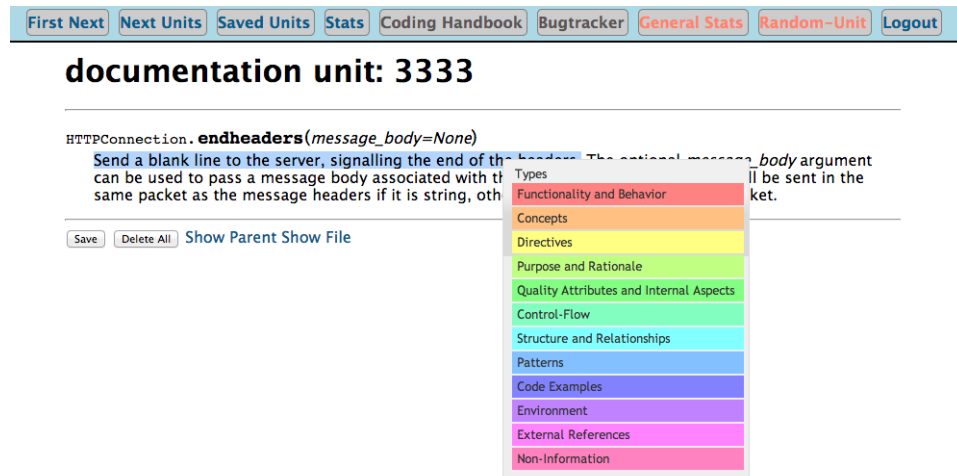


Abbildung 11: Einheit typisieren

Die gesetzten Markierungen werden mit JavaScript farblich dargestellt. Bei Doppelmarkierungen wird die Farbe angezeigt, die zuletzt für die Auswahl gewählt wurde. Außerdem erscheint unterhalb der Einheit eine Auflistung aller Markierungen. Sobald ein Element aus dieser Auflistung mit Mouse-Over⁹ berührt wird, wird der damit verbundene, markierte Text rot und gestrichelt umrandet. Ein Klick auf dieses Element bewirkt das Löschen der angezeigten Markierung.

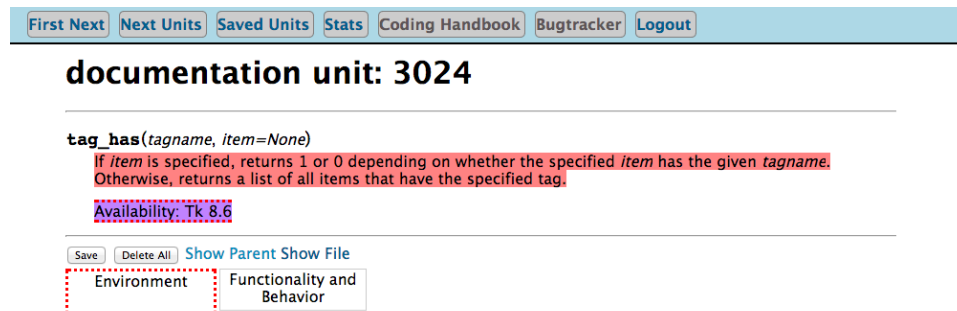


Abbildung 12: Einheit typisieren (2)

4.2.2.4 Noch nicht typisierte Einheiten

Für den Fall, dass der Gutachter die zur Zeit nächste Einheit noch nicht bewerten will, weil ihm diese für den Augenblick zu lang erscheint oder ihm gerade zu schwierig ist, kann er sich alle Einheiten auflisten, die noch typisiert werden müssen und davon eine auswählen, die er als nächstes bearbeiten möchte. Dabei wird dem Gutachter die Identifikationsnummer, die Datei

⁹Bewegung des Maus-Cursor auf ein Element ohne es anzuklicken

aus welcher die Einheit kommt und die Kategorie angezeigt. Der Klick auf eine Zeile führt dann zu der Maske, in welcher die Einheit typisiert werden kann.

First Next	Next Units	Saved Units	Stats	Coding Handbook	Bugtracker	General Stats	Random-Unit	Logout
This list contains every unit which was not saved yet.								
ID: 1035	python-3.4.0-docs-html/library/rlcompleter.html	Type: section						
ID: 1210	python-3.4.0-docs-html/library/argparse.html	Type: section						
ID: 1515	python-3.4.0-docs-html/library/curses.panel.html	Type: method						
ID: 2003	python-3.4.0-docs-html/library/asyncio-protocol.html	Type: method						
ID: 3119	python-3.4.0-docs-html/library/msilib.html	Type: function						
ID: 3333	python-3.4.0-docs-html/library/http.client.html	Type: method						
ID: 3333	python-3.4.0-docs-html/library/http.client.html	Type: method						
ID: 3333	python-3.4.0-docs-html/library/http.client.html	Type: method						
ID: 3560	python-3.4.0-docs-html/library/ssl.html	Type: method						
ID: 3907	python-3.4.0-docs-html/library/decimal.html	Type: method						
ID: 3907	python-3.4.0-docs-html/library/decimal.html	Type: method						
ID: 4172	python-3.4.0-docs-html/library/os.html	Type: function						
ID: 4430	python-3.4.0-docs-html/library/functions.html	Type: function						
ID: 4947	python-3.4.0-docs-html/library/errno.html	Type: data						

Abbildung 13: Noch nicht typisierte Einheiten

4.2.2.5 Typisierte Einheiten

Zudem kann sich der Gutachter alle Einheiten auflisten lassen, die er bereits typisiert hat. Dabei werden alle Einheiten gelb hervorgehoben, die a) zu mehr als 25% nicht markiert sind und b) mindestens 100 Zeichen nicht markiert sind. Dies wurde so eingeführt, damit sehr kurze Einheiten, bei denen die nicht zu typisierende Überschrift fast genauso lang ist wie der eigentliche Text, nicht fälschlich hervorgehoben werden. Bei den Textlängen wurden außerdem die eingeführten Platzhalter heraus gerechnet, da auch diese nicht markiert werden sollten. So hat der Gutachter eine schnelle Übersicht über Einheiten, die er (in der Regel versehentlich) unvollständig abgespeichert hat. Weiterhin wird für jede Einheit der Zeitstempel der letzten Speicherung ausgegeben und bei Hervorhebung noch die Angabe in Prozent, wie viel des Textes nicht markiert ist.

First Next	Next Units	Saved Units	Stats	Coding Handbook	Bugtracker	Logout
ID: 6116	python-3.4.0-docs-html/library/unittest.html	- Type: method - last save: June 18, 2014, 4:19 p.m.				
ID: 6106	python-3.4.0-docs-html/library/idle.html	- Type: section - Unmarked: 100 % - last save: June 18, 2014, 4:15 p.m.				
ID: 6080	python-3.4.0-docs-html/library/audioop.html	- Type: function - last save: June 18, 2014, 4:12 p.m.				
ID: 5973	python-3.4.0-docs-html/library/zipfile.html	- Type: method - last save: June 18, 2014, 4:10 p.m.				
ID: 5769	python-3.4.0-docs-html/library/diffib.html	- Type: section - last save: June 18, 2014, 4:07 p.m.				
ID: 5712	python-3.4.0-docs-html/library/re.html	- Type: attribute - last save: June 18, 2014, 4:03 p.m.				
ID: 5706	python-3.4.0-docs-html/library/re.html	- Type: function - last save: June 18, 2014, 4:03 p.m.				
ID: 5660	python-3.4.0-docs-html/library/ipaddress.html	- Type: section - last save: July 7, 2014, 4:04 p.m.				
ID: 5513	python-3.4.0-docs-html/library/configparser.html	- Type: method - last save: June 18, 2014, 4:01 p.m.				
ID: 5448	python-3.4.0-docs-html/library/email.contentmanager.html	- Type: method - last save: June 18, 2014, 4 p.m.				
ID: 5434	python-3.4.0-docs-html/library/shutil.html	- Type: exception - last save: June 18, 2014, 3:59 p.m.				
ID: 5414	python-3.4.0-docs-html/library/shutil.html	- Type: function - last save: July 4, 2014, 5:29 p.m.				
ID: 5393	python-3.4.0-docs-html/library/socketserver.html	- Type: attribute - last save: July 4, 2014, 5:25 p.m.				
ID: 5214	python-3.4.0-docs-html/library/platform.html	- Type: function - last save: June 18, 2014, 3:50 p.m.				
ID: 5193	python-3.4.0-docs-html/library/aifc.html	- Type: method - last save: June 18, 2014, 3:50 p.m.				
ID: 5171	python-3.4.0-docs-html/library/aifc.html	- Type: method - last save: June 18, 2014, 3:49 p.m.				
ID: 5170	python-3.4.0-docs-html/library/cmath.html	- Type: data - last save: June 18, 2014, 3:47 p.m.				
ID: 5019	python-3.4.0-docs-html/library/errno.html	- Type: data - last save: June 18, 2014, 3:47 p.m.				
ID: 4846	python-3.4.0-docs-html/library/pkgutil.html	- Type: function - Unmarked: 33 % - last save: July 4, 2014, 5:23 p.m.				
ID: 4800	python-3.4.0-docs-html/library/fractions.html	- Type: method - last save: June 18, 2014, 3:46 p.m.				

Abbildung 14: Abgespeicherte Einheiten

4.2.2.6 Persönliche Statistik

In der Statistik (kurz: Stats) werdem dem angemeldeten Benutzer folgende Werte angezeigt:

1. Insgesamt abgespeicherte Einheiten
2. noch zu typisierende Einheiten (noch nicht abgespeichert)
3. Insgesamt zugewiesene Einheiten
4. Übereinstimmung der Markierungen mit den anderen Gutachtern (nur für die Gutachter der Hauptstichprobe)
5. Angabe der Anzahl der Einheiten, auf die sich die Berechnung der Übereinstimmung stützt

Die 5. Angabe ist deswegen wichtig, da die Berechnung bei den schnellen Gutachtern sich auf weniger Einheiten stützt als schon markiert wurden. Das ist der Tatsache geschuldet, dass die Berechnung der Übereinstimmung pro Einheit erst erfolgen kann, wenn zwei Gutachter diese abgespeichert haben.

Diese Angaben werden einmal im Gesamten gemacht und dann anhand der letzten Speicherungsdaten für die letzten 14, 8, 4 und 2 Tage. Das hat den Vorteil, dass jeder Student schnell seinen eigenen Fortschritt für eine kurze Vergangenheit beobachten kann und dann auch ggf. erkennen kann, ob die Übereinstimmung mit anderen Gutachtern eher zu- oder abnimmt.

First Next	Next Units	Saved Units	Stats	Coding Handbook	Bugtracker	Logout
----------------------------	----------------------------	-----------------------------	-----------------------	---------------------------------	----------------------------	------------------------

Hello Jakob Warkotsch, this page is all about you.

Description	total elements	last 14 days	last 8 days	last 4 days	last 2 days
Already saved elements	470	4	4	4	4
elements left	0				
saved and unsaved	470				
agreement in %	88.01	87.5	87.5	87.5	87.5
agreement based on	454 units	4 units	4 units	4 units	4 units

Keep in mind: The agreement per unit also changes if the saving of an other participant changes.

Please do not change your markings if you are pretty sure that these are correct.

Abbildung 15: Persönliche Statistik

4.2.2.7 Kodierhandbuch

Aus der Websitenavigation heraus kann direkt das Kodierhandbuch in einem neuen Tab geöffnet werden. Der Link verweist direkt auf die Website, die dieses enthält. Dieser direkte Link soll dazu führen, dass nochmaliges nachlesen möglichst einfach gemacht wird und die Gutachter nicht erst noch nach dem Link suchen müssen. Der Link ist in der Navigation grau markiert, um zu verdeutlichen, dass es sich dabei um eine externe URL handelt.

4.2.2.8 Bugtracker

Ebenso kann der Bugtracker (gehosted bei bitbucket.org) schnell aus der Navigation heraus erreicht werden. Die Gutachter wurden angehalten, Schwierigkeiten in der Bedienung oder auftretende Fehler direkt dort zu melden, so dass alle Aufgaben bezüglich Verbesserung der Website zentral verwaltet werden können. Gegenüber der üblichen e-Mail-Kommunikation hat dies den Vorteil, dass Gutachter erkennen können, wenn ein Fehler bereits gemeldet wurde, so dass Dubletten vermieden werden können. Außerdem können Gutachter noch zusätzliche Informationen zu den einzelnen Aufgaben liefern und für die Erledigung einer Aufgabe stimmen.

Weiterhin kann der interessierte Gutachter sich hierüber den gesamten Quelltext der Website ansehen und so Ursache und Lösung der einzelnen Thematiken nachvollziehen, denn Lösungsmeldungen sind in der Regel mit der lösenden Änderung im Quelltext verknüpft.

Title	T	P	Status	Votes	Assignee	Created	Updated
#63: unmarked	📄	↓	NEW		Sven Wilderm...	2014-06-30	2014-06-30
#55: Markings are "moving" when editing long units	📄	↑	NEW		Sven Wilderm...	2014-06-13	2014-06-24
#58: check if stats are working correctly	📄	↑	RESOLVED		Sven Wilderm...	2014-06-21	2014-06-23
#54: saved units: problem with unmarked in percent - value	📄	↑	RESOLVED		Sven Wilderm...	2014-06-13	2014-06-21
#50: wrong timestamp	📄	↑	WONTFIX		Sven Wilderm...	2014-06-03	2014-06-18
#45: Show agreement of my units to these of the other persons	📄	↑	RESOLVED		Sven Wilderm...	2014-05-29	2014-06-18
#53: Unmarked in percent is buggy	📄	↓	RESOLVED		Sven Wilderm...	2014-06-13	2014-06-13
#48: map gold-sample	📄	↑	RESOLVED		Sven Wilderm...	2014-06-02	2014-06-08
#41: Show for each saved unit how much is marked / not marked	📄	↑	RESOLVED		Sven Wilderm...	2014-05-29	2014-06-07
#52: clipboard-bloat on selection	📄	↓	ON HOLD		Sven Wilderm...	2014-06-06	2014-06-07
#51: script: calculate unmarked_chars	📄	↑	RESOLVED		Sven Wilderm...	2014-06-04	2014-06-05

Abbildung 16: Bugtracker

4.2.2.9 Allgemeine Statistik

Die allgemeine Statistik ist eine reine Administratorfunktionalität und zeigt im wesentlichen die persönliche Statistik für alle Gutachter übersichtlich an. Die Übereinstimmung unter den Gutachtern der Goldstichprobe wurde nicht berechnet und deswegen auch nicht angezeigt. Außerdem weist diese Statistik noch die insgesamt zugewiesenen, gespeicherten (auch ohne Markierungen), markierten und nicht gespeicherten Einheiten an. Dabei werden Einheiten von Test- und Administratoraccounts mitberechnet. Da die einzelnen Gutachter hier mit Vornamen aufgelistet werden, wird auf die Einbindung eines Screenshots aus Datenschutzgründen verzichtet.

4.2.2.10 Zufällige Einheit

Der „Random Unit“-Button ist ebenfalls eine reine Administratorfunktionalität und dient lediglich dem Testen. Grund für die Einführung dieses Buttons war, dass bei Neuentwicklungen immer wieder anhand bisher unmarkierter Einheiten getestet werden sollte und für die Zuweisung jedes mal die Terminalumgebung genutzt werden musste.

Im Wesentlichen ist hinter dieser Funktion ein Bereich eingetragen, aus dem die Identifikationsnummern für die Dokumentationseinheiten stammen können. Aus diesem Bereich wird dann zufällig eine Nummer ausgewählt und als Einheit dem angemeldeten Benutzer zugewiesen. Dabei wurde keine Rücksicht darauf genommen, ob eine Einheit bereits markiert wurde oder nicht. Sollte eine Einheit erneut zugewiesen werden, die bereits dem Nutzer zugewiesen ist, taucht diese doppelt in den Listenansichten auf, wird aber trotzdem nur einmal bewertet. Da Markierungen von Administratoren sowieso nicht in der Auswertung berücksichtigt werden, ist dies unproblematisch.

4.2.2.11 Logout

Die Logout-Funktionalität ermöglicht allen Gutachtern das sichere Abmelden von der Website. Umgesetzt wurde dieses analog zu der Login-Funktion mit dem in Django verfügbaren Authentifizierungsmodul. Nach dem Logout wird der Benutzer wieder auf die Login-Seite umgeleitet. Im ausgeloggten Zustand kann nur der Login-Bereich besucht werden, alle anderen Funktionalitäten (mit Ausnahme der externen Websites) sind nicht erreichbar.

4.2.2.12 Datenbankmodell

4.2.2.13 Hosting

4.3 Gamification

Um die Motivation der Gutachter zu erhöhen, wurden spielerische Elemente eingesetzt, auch Gamification genannt. Zum einen hatte jeder der Gutachter Zugriff auf seine persönliche Statistik und erhielt somit einen detaillierten Fortschrittsbalken. Zum anderen wurden alle 14 Tage Auszeichnungen in Form von Gutscheinen für einen großen deutschen Versandhandel vergeben. Da die Typisierung aller Einheiten in etwa 6 Wochen gedauert hat, gab es 3 Termine für Auszeichnungen. Der 14-tägige Rhythmus wurde allen Teilnehmern vorab bekannt geben. Bei jeder Auszeichnung wurden auch die nicht ausgezeichneten Teilnehmer über den Grund und die Höhe des Gutscheins informiert. Dabei wurden unterschiedliche Fortschritte prämiert:

1. Nach den ersten zwei Wochen wurde lediglich der fleißigste Teilnehmer mit einen Gutschein belohnt, da die qualitative Auswertung noch nicht implementiert war.
2. Zwei Wochen später wurde sowohl der fleißigste Teilnehmer als auch der Teilnehmer mit der besten Übereinstimmung zu anderen Teilnehmern prämiert.
3. Am Ende erhielten die beiden Teilnehmer mit den besten Übereinstimmungen jeweils einen Gutschein. Da alle Einheiten bewertet wurden, ergab eine Prämierung des „fleißigsten Teilnehmers“ keinen Sinn mehr.

Insgesamt wurden Gutscheine im Wert von 30 EUR ausgezahlt.

5 Auswertungen

In diesem Abschnitt werden alle Ergebnisse der Auswertungen aufgezeigt und die dafür notwendigen Schritte erklärt.

5.1 Konfusionen

Bei der Überprüfung der Verträglichkeit zweier Markierungen gibt es immer wieder den Fall, dass Gutachter A den Typ X, Gutachter B hingegen den Typ Y für sinnvoller hält. Daher wird überprüft, ob es Typen gibt, die besonders häufig mit einander verwechselt werden. Mit Hilfe der folgenden SQL-Anweisung wurden alle aufgetretenen Konfusionen inklusive Häufigkeit des Auftretens ausgegeben:

```
SELECT atype_id, btype_id, COUNT(*)
FROM extractor_confusions
GROUP BY atype_id, btype_id
ORDER BY COUNT(*) DESC;
```

Wenn jetzt noch die Reihenfolge ignoriert wird, also die Ergebnisse der Tupel (X,Y) mit denen von (Y,X) addiert werden, erhält man folgende Konfusionen, absteigend sortiert nach Häufigkeiten des Auftretens.

Nr.	Typ A	Typ B	Anzahl
1	Functionality and Behaviour	Structure and Relationships	324
2	Functionality and Behaviour	Purpose and Rationale	232
3	Functionality and Behaviour	Concepts	209
4	Functionality and Behaviour	Directives	161
5	Functionality and Behaviour	Non-Information	147
6	Functionality and Behaviour	Qual. Attributes, Intern. Aspects	124
7	Functionality and Behaviour	Environment	110
8	Concepts	Structure and Relationships	87
9	Purpose and Rationale	Structure and Relationships	83
10	Functionality and Behaviour	Patterns	78
11	Patterns	Code Examples	72
12	Functionality and Behaviour	Control-Flow	65
13	Concepts	Purpose and Rationale	59
14	Structure and Relationships	Patterns	55
15	Purpose and Rationale	Patterns	54
16	Qual. Attributes, Intern. Aspects	Structure and Relationships	51
17	Concepts	Qual. Attributes, Intern. Aspects	51

Abbildung 17: Konfusionshäufigkeiten

Aufgelistet werden die Konfusionen hier, sobald die untere Schranke von 50 Vorkommen erreicht wird. Um sich später leichter auf einzelne Konfusionen

beziehen zu können, sind diese durchnummeriert. Für jedes dieser Paare wird einmal der Typ angegeben, welcher erfahrungsgemäß häufiger der richtige sein müsste. Dabei wird berücksichtigt, dass es besser sein sollte, in einem langen Segment erst den allgemeineren und danach den spezielleren Fall zu priorisieren (entsprechend des Kodier-Handbuchs). Dies ergibt die zweite und dritte Spalte der Tabelle in Abbildung 18. Anschließend werden von jedem dieser Konfusionspaare zufällig zehn Stück aus der Stichprobe gezogen und exemplarisch begutachtet. Für jeden dieser Fälle muss dann entschieden werden, welche Kodierung angemessen ist. Dabei gibt es drei Fälle:

- Typ X ist angemessener
- Typ Y ist angemessener
- Beide Typen sind gleich (gut oder schlecht) angemessen

Für die gesamte Stichprobe bekommt dann ein Typ X Vorrang, wenn Typ X mindestens zu $\frac{3}{5}$ und TYP Y höchstens zu $\frac{1}{5}$ besser ist und andersherum. Wenn das Ergebnis aus dieser Betrachtung nicht mit der Ersteinschätzung übereinstimmt, werden weitere zehn Elemente begutachtet. In Bezug auf die gesamte Stichprobe wirken sich diese Vorrangsregeln nur wenig aus, helfen aber dennoch weiter, um aus den Typisierungen einer Einheit von zwei Gutachtern eine endgültige zu erschaffen.

Nr.	Erfahrungsgemäß besserer Typ	Zustimmung	Widerspruch	Neutral
1	Structure and Relationships	7/20	10/20	3/20
2	Functionality and Behaviour	7/10	2/10	1/10
3	Concepts	7/10	2/10	1/10
4	Directives	6/20	12/20	2/20
5	Functionality and Behaviour	7/10	2/10	1/10
6	Qual. Attributes, Intern. Aspects	9/20	10/20	1/20
7	Environment	3/20	12/20	5/20
8	Structure and Relationships	9/10	1/10	0/10
9	Structure and Relationships	13/20	6/20	1/20
10	Patterns	8/10	2/10	0/10
11	Code Examples	9/10	0/10	1/10
12	Control-Flow	8/10	2/10	0/10
13	Concepts	8/10	1/10	1/10
14	Patterns	8/10	2/10	0/10
15	Purpose and Rationale	8/10	2/10	0/10
16	Qual. Attributes, Intern. Aspects	7/10	2/10	1/10
17	Concepts	6/10	2/10	2/10

Abbildung 18: Abschätzung der besseren Typen bei Konfusionen

Insgesamt können so 12 von 17 häufigen Konfusionen aufgelöst werden. Vier Konfusionen haben ein uneindeutiges Ergebnis. In einem der 17 Fälle entspricht dem Ergebnis nicht der Erwartungen und sollte daher separat diskutiert werden.

Sobald eine der obigen Konfusionen auftritt und ein besserer Typ indentifiziert wurde (grüne Zeilen), wird dieser dann als richtige Typisierung gewertet.

5.1.1 Diskussion: Konfusion Nr. 7

5.1.2 Hilfsmittel

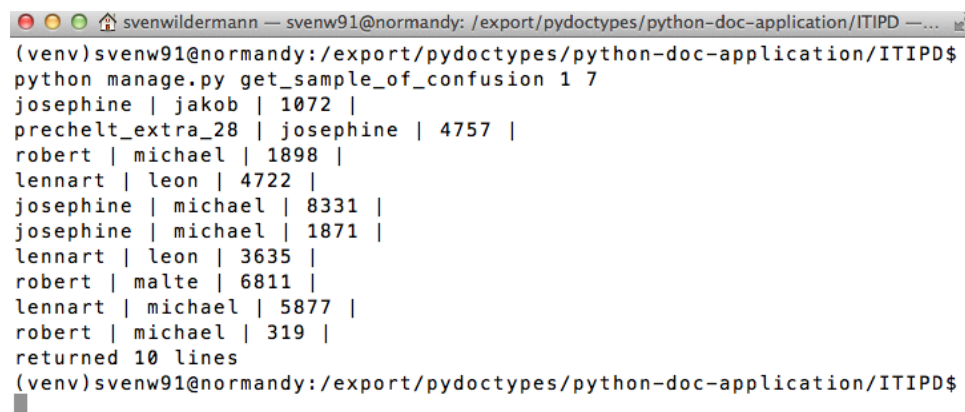
Für diese Stichprobenziehung wurde eine Django-Funktion geschrieben, welche die betroffenen Benutzer und die ID der zehn Dokumentationseinheiten unter Angabe eines Konfusionspaares ausgibt. In dem noch zusätzlich die Option „-count“ übergeben wird, wird die Häufigkeit des angegebenen Konfusionspaares ausgegeben und keine Stichprobenziehung durchgeführt. Aufruf und (mögliches) Ergebnis dieser Funktion sehen beispielhaft für Nummer 1 (Wissenstypen 1 und 7) aus wie in Abbildung 19 gezeigt..

Die zuletzt ausgegebene Zahl gibt zum Zweck der Überprüfung die Anzahl der ausgegebenen Zeilen an. Solange mehr als 10 Konfusionspaare der abgefragten Kombination vorhanden sind, wird eine 10 ausgegeben, andernfalls die Anzahl der verfügbaren Konfusionen. Beispielsweise erhält man bei der Eingabe

```
python manage.py get_sample_of_confusion 4 11
```

die Ausgabe

```
lennart | jakob | 4318 |
lennart | sven_extra_28 | 4377 |
josephine | leon | 7830 |
returned 3 lines
```



```
(venv)svenw91@normandy:/export/pydoctypes/python-doc-application/ITIPD$
python manage.py get_sample_of_confusion 1 7
josephine | jakob | 1072 |
prechelt_extra_28 | josephine | 4757 |
robert | michael | 1898 |
lennart | leon | 4722 |
josephine | michael | 8331 |
josephine | michael | 1871 |
lennart | leon | 3635 |
robert | malte | 6811 |
lennart | michael | 5877 |
robert | michael | 319 |
returned 10 lines
(venv)svenw91@normandy:/export/pydoctypes/python-doc-application/ITIPD$
```

Abbildung 19: Funktion für die Konfusions-Stichprobe

5.2 Zusammenführen der Goldstichproben

Da die Goldstichprobe insgesamt drei mal von verschiedenen Gutachtern typisiert wurde, müssen diese für die weitere Verwendung zusammengeführt werden. Um aus drei Wahrheiten die „eine“ zu machen, wird wie folgt vorgegangen:

Es werden alle möglichen Tupel auf die Gesamtverträglichkeit über alle Zeichen berechnet. Zwei dieser drei Tupel besitzen somit automatisch den selben Gutachter, so dass diese damit die höchste Gesamtverträglichkeit für diese Einheit besitzt. Die Markierungen dieses Gutachtern werden dann einem „Dummy-Benutzer“ kopiert.

Anschließend werden die Markierungen der Gutachter mit dem des Dummys manuell verglichen und wo nötig noch kleinere Korrekturen vorgenommen. Diese sind vor allem folgender Natur:

- Es gibt eine Markierung, die die anderen zwei Gutachter getätigt haben (also nicht in den Markierungen des Dummys vorkommt). Daher wird eine Markierung des Dummys durch die der anderen Gutachter ersetzt oder hinzugefügt (je nach Übereinstimmung der Segmentierungen).
- Mindestens einer der Gutachter hat richtig ein Segment mit dem Wissenstyp „Structure and Relationship“ markiert. Wenn dieses in der aktuellen Fassung des Dummys fehlt, wird es hinzugefügt.
- Mindestens einer der Gutachter hat richtig ein Segment mit dem Wissenstyp „External References“ markiert. Wenn dieses in der aktuellen Fassung des Dummys fehlt, wird es hinzugefügt.

Somit erhält man ein endgültiges, fast ideales Ergebnis der Goldstichprobe. Ein ähnlicher Vorgang muss jetzt mit der Hauptstichprobe geschehen.

5.2.1 Zusammenführen der Stichproben

Da jede Einheit aus der Hauptstichprobe von zwei Gutachtern bewertet wurde, müssen auch diese Ergebnisse zusammengeführt werden. Diese Stichprobe ist mit 1548 Einheiten allerdings erheblich größer als die Goldstichprobe mit 168 Einheiten, deswegen soll dies ohne manuelle Zuarbeit geschehen. Gleichzeitig gibt es zwei unterschiedliche Abstraktionshöhen mit unterschiedlichen Vorgehensweise:

1. Vergleich des Vorkommens von Wissenstypen (ein boolescher Wert pro Wissenstyp ergibt zwölf Werte). Dabei werden alle Werte in das Endresultat übernommen, die übereinstimmen.
2. Vergleich aller einzelnen Markierungen auf Verträglichkeit und Übernahme in das Endresultat nach einem noch zu definierenden Algorithmus. Siehe Abschnitt [5.2.1.1](#).

5.2.1.1 Markierungsergebnis bestimmen

Überprüfe für alle Markierungen in der Einheit, wie oft jeder Gutachter bei allen auftretenden Konfusionen dieser Einheit den besseren Wissenstypen gewählt hat. Übertrage alle verträglichen Markierungen des besseren Gutachters auf einen Dummy.

Seien M_1 und M_2 zwei unterschiedliche Markierungen, die sich zumindestens 50 Prozent überlappen.

5.3 Auswertung der Typisierungen

5.3.1 Übereinstimmung pro Wissenstyp nach Gutachtern

Die Übereinstimmung der Wissenstypen nach Gutachtern wurde in der vorangegangenen Studie wie folgt gemessen [MR13]:

„We calculated the agreement score for C_i by dividing the number of ratings which agree with the second coder by the total number of units coded by C_i . For example, if C_i coded 500 units and 400 of their Functionality ratings were the same as the ratings of the co-coder, C_i 's agreement score for Functionality would be $400/500 = 0.80$. We calculated the overall agreement by summing all agreements over all variables and dividing by the number of units times 12 (the number of variables).“

Diese Auswertung habe ich analog auf diese Arbeit übertragen und mittels eines Django-Kommandos umgesetzt. Ein Wissenstyp in einer Einheit ist vorhanden, sobald mindestens eine Markierung mit diesem in dieser Einheit gefunden wurde. Die Konfusionen habe ich dabei nicht aufgelöst, um die Vergleichbarkeit mit der Originalstudie zu erhalten. Abbildung 20 zeigt die Übereinstimmung nach Gutachtern. Die ersten drei Spalten geben den kleinsten, den durchschnittlichen und den größten Wert aller Gutachter für einen Wissenstypen an. Die letzten beiden Spalten identifizieren die Gutachter mit der besten und der schlechtesten Übereinstimmung. Der durchschnittliche Wert ist dabei über alle Wissenstypen hinweg größer als 0,75 und insgesamt größer als 0,87.

Wissenstyp	Min	Durschn.	Max.	Bester	Schlechtester
Functionality	0,730	0,786	0,926	C9	C6
Concepts	0,792	0,855	0,963	C9	C5
Directives	0.803	0,876	0,929	C8	C6
Purpose	0,703	0,788	0,857	C8	C6
Quality	0,820	0,863	0,910	C1	C4
Control-Flow	0,893	0,943	0,964	C6	C8
Structure	0,633	0,751	0,817	C5	C4
Patterns	0,873	0,903	0,964	C8	C5
Code-Examples	0,963	0,984	1,0	C8	C9
Environment	0,890	0,950	1,0	C8, C9	C6
References	0,951	0,969	0,977	C4	C2
Non-Info	0,765	0,857	1,0	C8	C4
Gesamt	0,818	0,877	0,942	C8	C6

Abbildung 20: Übereinstimmung nach Gutachtern

5.3.2 Übereinstimmung nach Wissenstyp

Die Übereinstimmung nach Wissenstypen wurde in der Studie von Maalej und Robillard wie folgt berechnet [MR13]:

„We measure agreement for a variable by dividing the number of units for which the two coders agreed (i.e., both rated True or both rated False) over the total number of units. This measure of raw agreement is one among many alternatives for measuring agreement by variable [25]. The advantage of the raw agreement measure is that it is simple to interpret. The disadvantage is that it does not take into account that coders might agree by chance.“

Auf Grund der veränderten Typisierungsweise in dieser Arbeit ist eine Übereinstimmung durch Zufall sehr viel unwahrscheinlicher, weshalb diese Auswertung für diese Arbeit eher uninteressant ist und somit nicht durchgeführt wird.

5.3.3 Übereinstimmung nach Einheiten

Zusätzlich kann die Übereinstimmung nach Einheiten berechnet werden. Für jede Einheit wird für jeden Wissenstyp geprüft, ob die Gutachter mindestens eine Markierung dieser Art getätigt haben. Eine Übereinstimmung liegt dann vor, wenn bei beide Gutachter sich einig sind, ob dieser Wissenstyp (mindestens einmal) vorkommt oder nicht. Da es insgesamt 12 Wissenstypen gibt, entspricht jede Übereinstimmung in einem Wissenstyp $\frac{1}{12} = 8,33\%$. Für jeden Gutachter wird so die Übereinstimmung über alle Einheiten berechnet und der Durchschnittswert in Abbildung 21 ausgegeben. Der Zeile für des

besten Gutachtern ist grün, die des schlechtesten ist rot markiert. Analog wird der Durchschnittswert für die unterschiedlichen Kategorien in Abbildung 22 angezeigt.

Diese Übereinstimmung prüft somit nicht nur, ob sich zwei Gutachter darin einig sind, dass ein Wissenstyp vorhanden ist, sondern auch, ob sie sich darin einig sind, dass dieser nicht vorhanden ist.

Gutachter	Übereinstimmung
C1	0,873
C2	0,882
C3	0,855
C4	0,848
C5	0,870
C6	0,845
C7	0,882
C8	0,898
C9	0,907
Durchsch.	0.866

Abbildung 21: Übereinstimmung nach Einheit je Gutachter

Kategorie	Übereinstimmung
Methoden	0,845
Felder	0,876
Module	0,805
Klassen	0,839
Beschreibungen	0,916

Abbildung 22: Übereinstimmung nach Einheit je Kategorie

5.3.4 Rechenbeispiel

Um die Bedeutung dieser Berechnungen zur Übereinstimmung zu veranschaulichen, folgt ein Beispiel:

Eine Einheit wird von beiden Gutachtern mit genau einem Wissenstypen markiert. Gutachter A vergibt den Wissenstyp 1 und Gutachter B vergibt den Wissenstyp 2. Somit sind sich Gutachter A und B darüber einig, dass die anderen zehn Wissenstypen (drei bis zwölf) nicht vorhanden sind. Dies ergibt eine Übereinstimmung in dieser Einheit von $\frac{10}{12} = 83,33\%$.

Die Übereinstimmung misst somit nicht nur die Einigkeit über das Vorhandensein eines Wissenstypen sondern auch über das Fehlen dessen.

5.3.5 Unstimmigkeiten zur Goldstichprobe

Solange sich zwei Gutachter einig sind, wird das Ergebnis der Markierungen als gültiges Endresultat anerkannt. Wenn jedoch zwei Gutachter den falschen Wissenstypen wählen, zum Beispiel weil sie beide den Wissenstypen X mit Wissenstyp Y verwechseln, wird dieses Ergebnis verfälscht. Um solche Tendenzen der Gutachter aufzeigen zu können, vergleiche ich die gefundenen Wissenstypen (als zwölf boolsche Werte) der Gutachter mit den Ergebnissen aus der Goldstichprobe. Dabei gibt es zwei unterschiedliche Unstimmigkeiten:

1. Falsch-Positiv: Gutachter gibt an, dass der Wissenstyp in dieser Einheit vorhanden ist, obwohl dies laut Goldstichprobe nicht der Fall ist.
2. Falsch-Negativ: Gutachter hat keine Markierung mit diesem Wissenstyp getätigt, obwohl in der Goldstichprobe mindestens eine davon vorhanden ist.

Im folgenden wird für jeden Wissenstyp dargelegt, wie oft die Gutachter die Einheiten der Goldstichprobe falsch markiert haben sowie welche Anteile durch die falsch-positiv (FP) und falsch-negativen (FN) Bewertungen verursacht wurden. Da jede Einheit doppelt typisiert wurde, liegt eine Menge von 336 Typisierungen mit insgesamt $336 * 12 = 4032$ ¹⁰ Bewertungen zu Grunde.

Wissenstyp	Gesamt	FP	FN
Functionality	51	0,255	0,745
Concepts	33	0,667	0,333
Directives	47	0,596	0,404
Purpose	34	0,765	0,235
Quality	45	0,378	0,622
Control-Flow	17	0,412	0,588
Structure	83	0,361	0,639
Patterns	25	0,560	0,440
Code-Examples	9	0,667	0,333
Environment	16	0,625	0,375
References	15	0,333	0,667
Non-Info	40	0,850	0,15
Gesamt	415	0,539	0,461

5.3.6 Verträglichkeit von Einheiten

Weiterhin wird die Verträglichkeit der Einheiten betrachtet und ausgewertet. Die Verträglichkeit bezieht sich auf die einzelnen Markierungen einer

¹⁰Es gibt 12 Wissenstypen

Einheit und ist wie folgt definiert.

Definition: Eine Markierung ist verträglich, wenn es eine andere Markierung gibt, die

1. diese umschließt (größer oder gleich groß ist) **oder**
2. zu mindestens 50 Prozent Teil dieser Markierung ist

und beide Markierungen den selben Wissenstypen oder eine Kombination aus den Konfusionspaaren besitzen.

Oder anders ausgedrückt:

Sei B_i der Beginn, E_i das Ende und W_i der Wissenstyp der Markierung i , „&“ das Zeichen für das logische „und“, „||“ das Zeichen für das logische „oder“ und $confusion(i, j)$ eine Funktion, die angibt, ob die Wissenstypen i und j zu den aufgelösten Konfusionen gehören. Dann heißen zwei Markierungen M_1 und M_2 verträglich, wenn:

1. $(B_1 \geq B_2) \ \& \ (E_1 \leq E_2) \ ||$
2. $(B_1 \leq E_1 \geq B_2) \ \& \ ((E_1 - B_2) \geq \frac{E_1 - B_1}{2})$

$\& \ (W_1 = W_2 \ || \ confusion(W_1, W_2))$

Die Verträglichkeiten für jede Einheit werden auf zwei Weisen berechnet:

1. nach Anzahl der Markierungen. Dabei wird die Anzahl der verträglichen Markierungen durch die Anzahl der gesamten Markierungen beider Gutachter pro Einheit geteilt.
2. nach der Menge der verträglichen Zeichen. Hier wird die Anzahl aller Zeichen von verträglichen Markierungen durch die Anzahl aller Zeichen der Markierungen beider Gutachter geteilt.

Während in Abbildung 23 die durchschnittliche Verträglichkeit pro Gutachter dargestellt wird, zeigt Abbildung 24 diese pro Kategorie.

Gutachter	Verträglichkeit	
	(1) nach Zeichen	(2) nach Markierungen
C1	0,819	0,786
C2	0,848	0,823
C3	0,818	0,788
C4	0,704	0,674
C5	0,860	0,829
C6	0,783	0,750
C7	0,841	0,809
C8	0,922	0,895
C9	0,863	0,820
Durchsch.	0.828	0,797

Abbildung 23: Verträglichkeit nach Gutachtern

Kategorie	Verträglichkeit	
	(1) nach Zeichen	(2) nach Markierungen
Methoden	0,879	0,800
Felder	0,725	0,719
Module	0,791	0,730
Klassen	0,721	0,698
Beschreibungen	0,861	0,854

Abbildung 24: Verträglichkeit nach Kategorien

6 Fazit

Das Fazit sollte am Ende ruhig einen ganz eigenen kurzen Abschnitt bekommen

7 Anhang

7.1 Technologien

7.1.1 Django

7.1.2 Python

7.1.3 BeautifulSoup4

7.1.4 Coffeescript

7.1.5 Postgres

7.1.6 Ajax

7.1.7 R

7.2 Kodierhandbuch - vollständig

7.3 Glossar

1. knowledge type / Informationstypen
2. API - in Implementierung des Extrahierer
3. DOM-Element

Literatur

- [METM12] Martin Monperrus, Michael Eichberg, Elif Tekes, and Mira Mezini. What should developers be aware of? an empirical study on the directives of api documentation. Empirical Software Engineering, 17(6):703–737, 12 2012.
- [MRa] Walid Maalej and Martin P. Robillard. Coding guide.
- [MRb] Walid Maalej and Martin P. Robillard. Coding guide.
- [MR13] Walid Maalej and Martin P. Robillard. Patterns of knowledge in api reference documentation. IEEE Transactions On Software Engineering, 39(9):1264–1282, 09 2013.
- [PW] Lutz Prechelt and Sven Wildermann. Coding guide for python.