Freie Universität Berlin

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. Elfriede Fehr

Berlin, 12. August 2014

Zusammenfassung

Walid Maalej und Martin P. Robillard haben im September 2013 einen Artikel [MR13] veröffentlicht, in dem sie die Dokumentationen der Programmiersprachen Java und .NET auf ihren Informationsgehalt hin untersucht und verglichen haben. Ihre Untersuchung wird im Rahmen dieser Bachelorarbeit auf die Dokumentation von Python[1] 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.
Die anschließende Auswertung der Ergebnisse und ein Vergleich dieser mit denen aus dem Originalartikel zeigt, dass die vorgenommenen Änderungen in der Konzeption sinnvoll waren. Die Struktur der Wissenstypen in der Dokumentation von Python ähnelt stark der bei Java und .NET, hat aber auch überraschende Abweichungen. Die erhobenen Daten bieten zudem eine Grundlage für weitergehende Untersuchungen zu den Informationstypen in der Python-Dokumentation wie beispielsweise die Reihenfolge des Auftretens oder die Wortmenge von Informationen.

[1] https://www.python.org/

Danksagungen

Zuerst 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 Holger Schmeisky und Franz Zieris von der AG Software Engineering. Dank 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 der Studienjahre danken. 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 **0**
1.1 Aufbau der Arbeit 0

2 Grundlagen **1**
2.1 Sprachgebrauch 1
2.2 Artikel von Maalej und Robillard 1
2.3 Kodier-Handbuch 3
2.3.1 Markierungen 3
2.3.2 Kleinere Änderungen 5

3 Konzeption **6**
3.1 Dokumentationseinheiten 6
3.2 Stichprobe 9
3.3 Goldstichprobe 9
3.4 Zeitaufwand 10
3.5 CADo-Tool vs. Eigenentwicklung 12

4 Entwicklung und Durchführung **14**
4.1 Extrahierer 14
4.1.1 BeautifulSoup4 15
4.1.2 Implementierung 15
4.1.3 HTML–Fehler 17
4.2 Typisierungswebsite 18
4.2.1 Anforderungen 18
4.2.2 Umsetzung 18
4.2.2.1 Navigationsleiste 19
4.2.2.2 Login 19
4.2.2.3 Einheit typisieren 19
4.2.2.4 Noch nicht typisierte Einheiten 21
4.2.2.5 Typisierte Einheiten 21
4.2.2.6 Persönliche Statistik 22
4.2.2.7 Kodierhandbuch 23
4.2.2.8 Bugtracker 23
4.2.2.9 Allgemeine Statistik 24
4.2.2.10 Zufällige Einheit 24
4.2.2.11 Logout 25
4.2.2.12 Hosting 25
4.2.2.13 Datenbankmodell 25
4.2.2.14 Datenanalyse 27
4.3 Gamification 27
4.4 Verteilung einer Stichprobe 28

5 Auswertungen **29**
5.1 Verwechslungen 29
5.1.1 Hilfsmittel 31
5.2 Ergebnis-Zusammenführung der Goldstichproben 32
5.3 Übereinstimmungen 32
5.3.1 Übereinstimmung pro Wissenstyp nach Gutachtern . 32
5.3.2 Übereinstimmung nach Wissenstyp 33
5.3.3 Übereinstimmung nach Einheiten 34
5.3.4 Rechenbeispiel 35
5.3.5 Übereinstimmung mit Goldstichprobe 35
5.3.6 Unstimmigkeiten zur Goldstichprobe 36
5.3.6.1 Gemeinsam unstimmig 37
5.3.7 Verträglichkeit von Einheiten 38
5.4 Ergebnis-Zusammenführung - Hauptstichprobe 40
5.5 Strukturen und Muster der Wissenstypen 40
5.5.1 Vorkommen und Verhältnis 41
5.5.2 positive Korrelation 43
5.5.3 Längenanalyse 44

6 Fazit **49**
6.1 Vergleich 49
6.2 Ausblick 50

7 Anhang **52**
7.1 Quelltext 52

Abbildungsverzeichnis

1 Wissenstypen in [MR13] 2
2 Durchschnittliche Textlängen je Kategorie 7
3 Gesamthäufigkeiten der Einheiten 8
4 Gesamthäufigkeiten der Kategorien 8
5 Stichprobengrößen 9
6 Goldstichprobengrößen 10
7 Einheiten pro Student inkl. Zeitaufwand 11
8 Zeitaufwand für die Goldstichprobe 11
9 Screenshot des CADo-Tools aus Gutachtersicht [MRb] . . . 13
10 Beispielhafte Struktur der HTML-Elemente 15
11 Login-Bereich 19
12 Einheit typisieren 20
13 Einheit typisieren (2) 21
14 Noch nicht typisierte Einheiten 21
15 Abgespeicherte Einheiten 22

16 Persönliche Statistik 23
17 Bugtracker 24
18 Datenbankmodell der Django-Applikation 26
19 Konfusionshäufigkeiten 29
20 Abschätzung der besseren Typen bei Konfusionen 31
21 Übereinstimmung nach Gutachtern 33
22 Übereinstimmung nach Einheit je Gutachter 35
23 Übereinstimmung nach Einheit je Kategorie 35
24 Unstimmigkeiten zur Goldstichprobe 37
25 Gemeinsam unstimmig 38
26 Verträglichkeit nach Gutachtern 39
27 Verträglichkeit nach Kategorien 40
28 Häufigkeiten der Wissenstypen 41
29 prozentuale Häufigkeiten der Wissenstypen 42
30 Verhältnis der Wissenstypen pro Dokumentationseinheiten . 42
31 Verhältnis der Wissenstypen pro Kategorie 43
32 absolute Kookurrenz von Wissenstypen 44
33 Kookkurrenz der Wissenstypen im Verhältnis zum Vorkommen beider Typen 44
34 Anzahl der Wörter nach Kategorie 45
35 Anzahl der Wörter nach Kategorie in einem Boxplot-Diagramm 46
36 Methoden: Anzahl der Wörter und Anzahl der Wissenstypen 46
37 Felder: Anzahl der Wörter und Anzahl der Wissenstypen . . 47
38 Module: Anzahl der Wörter und Anzahl der Wissenstypen . 47
39 Klassen: Anzahl der Wörter und Anzahl der Wissenstypen . 48
40 Beschreibungen: Anzahl der Wörter und Anzahl der Wissenstypen 48

1 Einleitung

Zu jeder Programmiersprache und Programmierschnittstelle gehört in der Regel eine Dokumentation über die bereitgestellten Funktionalitäten, auch Referenzhandbuch genannt. Während die Vor- und Nachteile der Programmiersprachen häufig disktutiert werden, findet man nur wenig Analyse zu den Dokumentationen. Dabei trägt eine Dokumentation nicht unwesentlich zur Qualität einer Programmiersprache bei. Entscheidend ist, wie gut die Entwickler mit den gebotenen Informationen zurechtkommen 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 Walid Maalej und Martin Robillard [MR13] beantwortet. Sie fanden bei der Analyse der JAVA und .NET Dokumentationen insgesamt zwölf gut unterscheidbare Wissenstypen, im Original heißen diese „knowledge types“.
Die Analyse über die Verteilung dieser Typen in der offiziellen Python-Dokumentation (Version 3.4.0) 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 zwölf 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 sie Grundlage für diese Bachelorarbeit bildet. Hierbei gehe ich insbesondere auf die verschiedenen Informationstypen (auch Wissenstypen genannt) ein. Im Anschluss erkläre ich, 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 die Studenten eine bestimmte Stichprobe an Dokumentationseinheiten erhielten, um diese zu typisieren, wird ebenso thematisiert. Zuletzt werden dann die ausgewerteten Ergebnisse vorgestellt und es wird ein Fazit gezogen.

2 Grundlagen

2.1 Sprachgebrauch

Die in dieser Bachelorarbeit verwendeten Personalpronomen sind, sofern nicht später anders definiert, wie folgt zu verstehen:

- „Ich“ steht für den Autor dieser Arbeit, Sven Wildermann, und meint ausschließlich diesen.
- „Wir“ steht für Prof. Dr. Lutz Prechelt und Sven Wildermann in enger Zusammenarbeit.

Aus sprachlichen Gründen wird teilweise Gebrauch des Passivs als Handlungsrichtung gemacht. Ist dies der Fall, bin ich als Akteur allein verantwortlich für die damit beschriebenen Entscheidungen und Handlungen.
Zudem werden teilweise Anglizismen und (englische) Zitate verwendet, da eine Übersetzung aller Termini die Vergleichbarkeit mit der Studie von Maalej und Robillard erschweren würde.

2.2 Artikel von Maalej und Robillard

Walid Maalej und Martin P. Robillard haben im Septemer 2013 den Artikel „Patterns of Knowledge in API Reference Documentation“ [MR13] veröffentlicht und besprechen darin zum einen eine Taxonomie von in Dokumentationen vorkommenden Wissenstypen und zum anderen die durchgeführte Analyse der Dokumentationen von Java SDK 6 and .NET 4.0. Zu diesem Zweck stellten sie zu jedem neu gefundenen Wissenstyp eine Frage auf und erhielten so über 100 verschiedene Fragestellungen.
Daraufhin wurden ähnliche Fragen zusammengefasst und schließlich die zwölf in Abbildung 1 gezeigten verschiedenen Wissenstypen ausfindig gemacht. Auf der Website zu dieser Studie wurde dann zudem der „Coding Guide“ [MRa] veröffentlicht, welcher für jeden Typ einen Fragenkatalog, Anmerkungen und Beispiele angibt.

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]

Die Vorgehensweise sah vor, dass die Gutachter für jeden Wissenstyp bestimmten, 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 zwei Gutachtern unabhängig bewertet. Die Einheiten wurden in drei verschiedene Kategorieren 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 schließlich 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 Prozent Konfidenzintervall und 2,5 Prozent Fehlerspanne gezogen, so dass insgesammt 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 diejenigen Fälle ausgewertet, 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 miteinander korrelieren, also ob z.B. der Typ „Structure“ häufig zusammen mit „Functionality and Behaviour“ auftritt. Ebenso wurde ausgewertet, 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.

2.3 Kodier-Handbuch

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

2.3.1 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 ergänzenden 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 handwritten 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.

2.3.2 Kleinere Änderungen

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

- Informationen über einen 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 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 anschließende 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 Vorgehensweisen erläutert, die während der Bachelorarbeit wichtig geworden sind. Diese betreffen sowohl die Beschaffenheit der Dokumentionseinheiten als auch die Stichprobenziehung und den daraus resultierende Zeitaufwand für die Gutachter.

3.1 Dokumentationseinheiten

Als Dokumentationseinheiten werden die Teilabschnitte aus der Dokumentation bezeichnet, die ein Gutachter für die Bewertung zusammenhängend angezeigt bekommt. Informationen über die Beschaffenheit dieser Einheiten sind relevant für die spätere Berechnung der Stichprobengröße.
Die 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"> Methodeninhalt </dl>
   - <dl class="classmethod"> Methodeninhalt </dl>
   - <dl class="staticmethod"> Methodeninhalt </dl>
   - <dl class="function"> Methodeninhalt </dl>
2. Felder (engl. fields)
   - <dl class="attribute"> Feldinhalt </dl>
   - <dl class="data"> Feldinhalt </dl>
3. Module (engl. modules)
   - Modulinhalt
4. Klassen (engl. classes)
   - <dl class="class"> Klasseninhalt </dl>
   - <dl class="exception"> Klasseninhalt </dl>
5. Beschreibungen (engl. describe)
   - <dl class="describe"> Beschreibungstext </dl>

Die Kategorien unterscheiden sich von denen aus der Originalstudie in der Form, dass Felder und Methoden unabhängig voneinander 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:

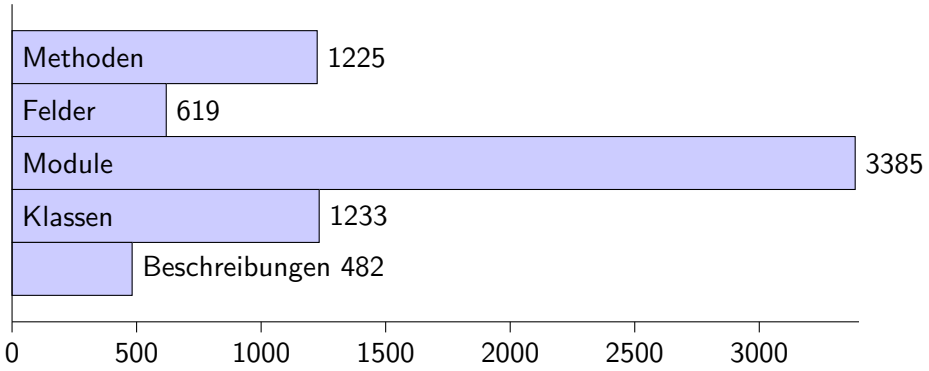


Abbildung 2: Durchschnittliche Textlängen je Kategorie

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

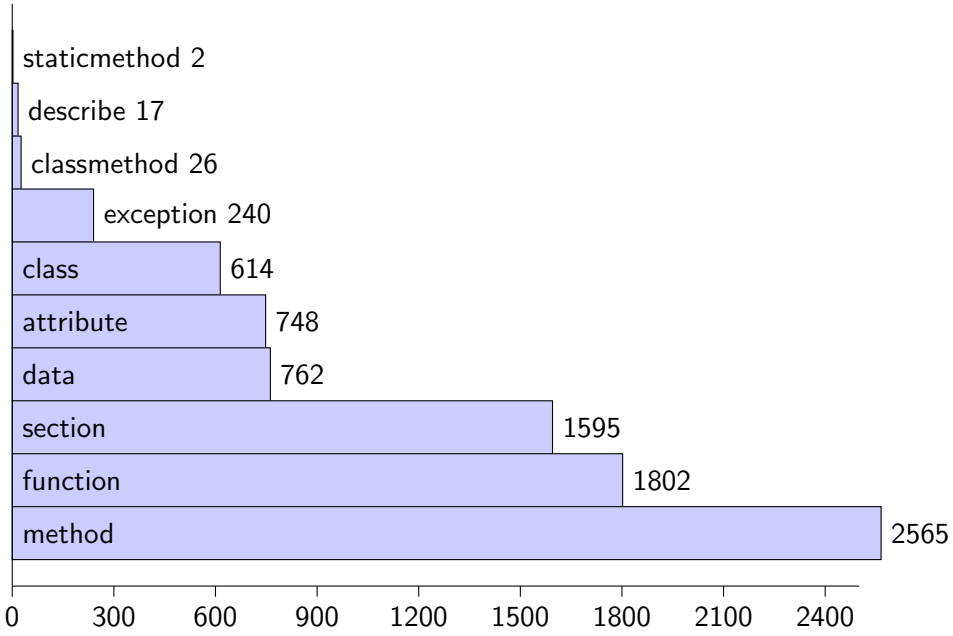


Abbildung 3: Gesamthäufigkeiten der Einheiten

Die Verteilung auf der Kategorieebene zeigt ebenfalls einen deutlichen Überschuss an Methoden: nämlich fast dreimal 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.

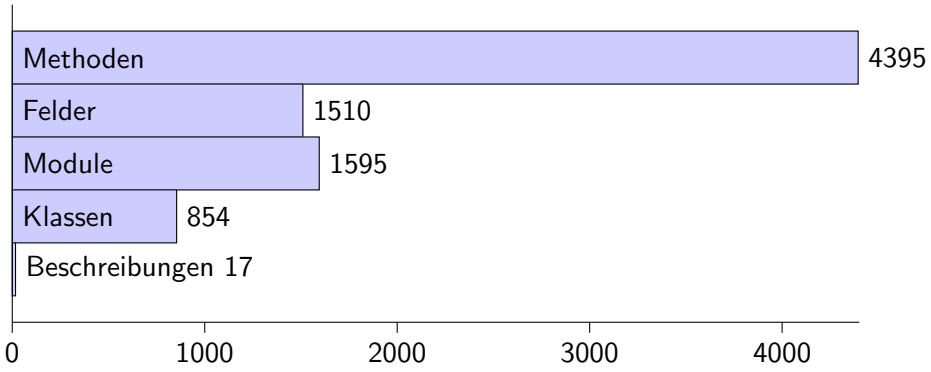


Abbildung 4: Gesamthäufigkeiten der Kategorien

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 voneinander 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 Prozent ist Z=1.96 [METM12]. Mit einer Fehlerrate von 5 Prozent ergibt sich für $n_0 = 384,16$, und dadurch ergeben sich dann die folgenden minimalen 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 so eine weitere Stichprobe, die Goldstichprobe, gezogen. Diese hat folgenden Zweck:

- Absicherung der Messung bezüglich Korrektheit der Gutachterergebnisse und
- Bestimmung eines Wertes für 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. Lutz Prechelt, AG Software Engineering
- Holger Schmeisky, AG Software Engineering
- Franz Zieris, AG Software Engineering
- Sven Wildermann, Verfasser dieser Bachelorarbeit

Wobei Holger Schmeisky und Franz Zieris zusammen eine Stichprobe bearbeitet haben.
Die Größe dieser Stichprobe wurde auf 2 Prozent 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 der Hauptstichprobe im Rahmen des Kurses „Forschungspraktikum“ an der Freien Universität Berlin für fünf ECTS [2] 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 mit in die Stichprobenziehung eingeflossen und haben dazu geführt, dass die Fehlerrate auf 5 Prozent 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

[2] European Credit Transfer System

mit zwei multipliziert werden, um die Gesamtanzahl der Bewertungen zu erhalten. Diese wurde dann auf die sieben 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	⌀7.6 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 BugTracker[3] umzugehen. Hierfür wurden zusätzlich insgesamt noch etwa zehn bis zwanzig Arbeitsstunden investiert, wobei der tatsächliche Aufwand je Gutachter stark abweichen kann.

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

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	⌀7.6 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 die Frage, wie diese Einheiten aus der Originaldokumentation verfügbar gemacht werden.

[3] Ein System zur Erfassung von Defekten und Verbesserungsvorschlägen

3.5 CADo-Tool vs. Eigenentwicklung

Im Rahmen der Forschung von Maalej und Robillard [MR13] wurde ein Tool mit dem Namen CADo[4] geschaffen, welches folgende Werkzeuge und folgende Fähigkeiten mit sich bringt (übersetzter Auszug) [MRb] :

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

Aus der Perspektive des Gutachters besitzt 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
- Statistiken ansehen

[4] Content Analysis for Software Documentation

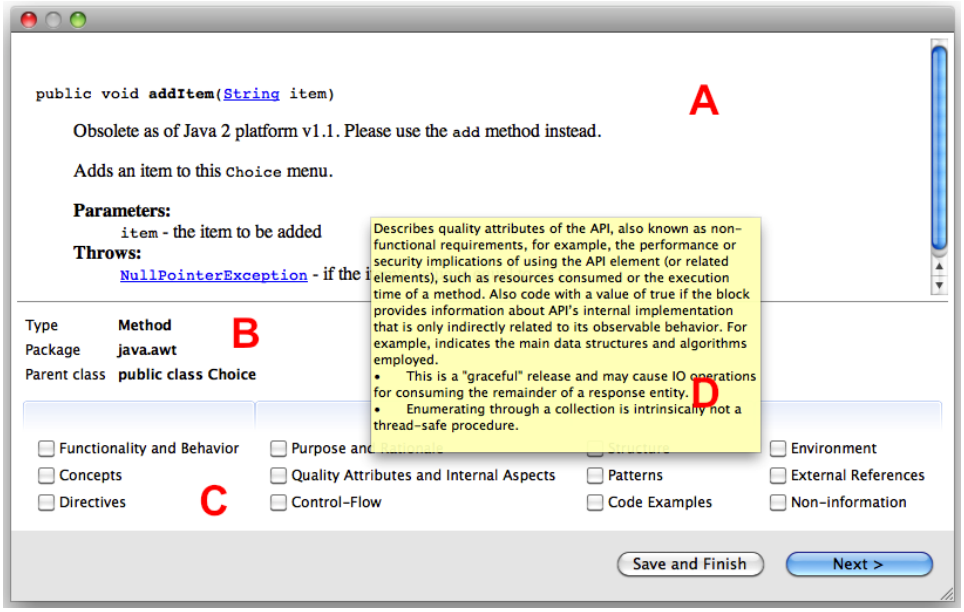


Abbildung 9: Screenshot des CADo-Tools aus Gutachtersicht [MRb]

Auf Grund der Vielzahl der für uns nützlichen Fähigkeiten stellte sich die Frage, ob dieses Werkzeug für diese Bachelorarbeit ebenfalls genutzt werden kann. Also wurde dieses Werkzeug heruntergeladen und getestet. Bereits beim Herunterladen gab es Schwierigkeiten, da der auf der Website veröffentlichte Link ungültig war. Nach Rückfragen bei den Autoren wurde dieser Misstand dann innerhalb weniger Tage von diesen behoben.
Der Test des Werkzeugs erwies sich dann ebenfalls als schwierig, da es in der alpha-Version[5] und ohne Benutzerhandbuch vorlag. Zudem gab es bei möglichen Fragen keinen einheitlichen Ansprechspartner und keine garantierte Antwortzeit der Entwickler, was ebenfalls zu einem höherem Risikio in der Benutzung geführt hätte.
Somit haben wir es als sinnvoller angesehen, die für diese Bachelorabeit 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 entwicklen, um zusätzlich noch Synergieeffekte bezüglich des Erlernens und

[5] frühes Entwicklungsstadium mit hoher Fehleranfälligkeit

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 2.3.1), da wir so nicht mehr an die Gegebenheiten des CADo-Tools gebunden waren.

4 Entwicklung und Durchführung

In diesem Kapitel wird zuerst erläutert, wie das Extrahieren der Dokumentationseinheiten aus der Original-Pythondokumentation gemeistert wurde. Anschließend wird die Konzipierung und Umsetzung des Werkzeugs für die Markierung der Dokumentationseinheiten besprochen.

4.1 Extrahierer

Um die Dokumentationseinheiten aus der HTML-Dokumentation von Python zu erhalten und in die Datenbank zu importieren, 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 verständlichere Syntax bietet, was zu einer leichteren und wartbareren Entwicklung führt.
Entsprechend der Definitionen von Dokumentationseinheiten sollten also die HTML-Schnipsel getrennt voneinander 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 Weglassen 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:

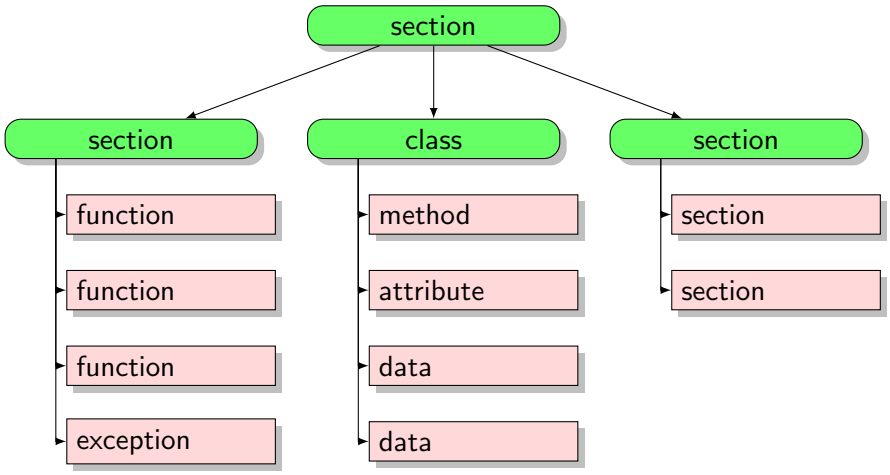


Abbildung 10: Beispielhafte Struktur der HTML-Elemente

4.1.1 BeautifulSoup4

Für das Parsen der HTML-Einheiten wurde BeautifulSoup4[6] verwendet, da es ein hierfür geschaffenes, mächtiges Werkzeug darstellt und gleichzeitig auf Python basiert. Somit konnte ich Synergieeffekte nutzen und mich während des Schreibens am Skript weiter in Python einarbeiten. Zudem bietet der Einsatz von BeautifulSoup einige Vorteile gegenüber dem Parsen mittels regulärer Ausdrücke. 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 Beispiele, und auch die Community hinter BeautifulSoup ist groß genug, um auf Internetportalen wie Stackoverflow[7] 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 vollständige 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.

[6] Eine Python-Bibliothek zum Parsen von Inhalten auf Websiten
[7] www.stackoverflow.com

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

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

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

```
1     for file in files:
2         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 DOM[8]-Element samt Attribute entgegennimmt:

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

Der Aufruf zum Parsen aller Elemente der Form

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

und Abspeichern dieser in einer Liste funktioniert dann wie folgt:

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

Dieser Schritt wurde analog für alle zehn verschiedenen Elementtypen 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:

[8] Document Object Model

```
1 def get_offsets(soup_str, elems):
2     """soup_str is a soup element converted to a string
3      elems is a array of soup-elements
4     """
5     offsets=[]
6     for each in elems:
7         find_index = soup_str.find(str(each))
8         offsets.append(find_index)
9     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:

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

Wegen der bereits erwähnten verschachelten 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 Elementen durch folgende Platzhalter ersetzt:

```
1         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 aufeinanderfolgende Platzhalter wieder zu einem Platzhalter zusammengefasst:

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

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

4.1.3 HTML–Fehler

Bei der späteren Typisierung der Einheiten sind ein paar Syntaxfehler in der originalen Pythondokumentation aufgefallen. So kam es vor, dass das in

Kapitel 3.1 beschriebene dl-Element für Methoden direkt nach der Signatur wieder geschlossen und der zugehörige Beschreibungstext durch den Extrahierer dem übergeordneten Element zugeordnet wurde. Die Einheit für dieses Element enthielt also lediglich die Signatur und blieb deswegen unmarkiert.
Daher bietet es sich an, die gesamte Pythondokumentation nach solchen Syntaxfehlern zu durchsuchen und im Original zu korrigieren. Dies sollte der Vollständigkeit halber aber nicht nur für die Stichprobe geschehen und wurde deswegen im Rahmen dieser Bachelorarbeit nicht durchgeführt.

4.2 Typisierungswebsite

4.2.1 Anforderungen

Damit die Einheiten von den Studenten typisiert werden konnten, musste ein Werkzeug geschaffen werden, welches folgende Eigenschaften aufweist. Die mit einem Stern markierten Eigenschaften sind optional:

- Ein- und Auslogfunktion
- Betriebssystemunabhängige Online-Erreichbarkeit
- Anzeige der dem Student zugewiesenen Einheiten
- Markierung von Segementen 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 Community gibt, die im Zweifel bei Herausforderungen unterstützten können. Für die Beantwortung der anfänglichen Fragen haben wir das in Berlin stattfindende Meetup der Django User Group[9] besucht. Darüber hinaus gibt es eine Vielzahl von Möglichkeiten, mit anderen Django-Entwicklern zu kommunizieren, wie über die Mailingliste[10] oder den IRC-Chat (deutsch/weltweit). Die Frontend-Entwicklungen mit Hilfe von JavaScript und CoffeeScript im Abschnitt 4.2.2.3 stammen zu einem Großteil von Phil Stelzer.

[9] http://www.meetup.com/django-user-group-berlin/
[10] https://groups.google.com/forum/#!forum/django-de

4.2.2.1 Navigationsleiste

Die Navigationsleiste soll dem jeweiligen Benutzer alle Funktionen und Seiten anzeigen, die er im Rahmen des Werkzeugs benutzen kann. Links, die auf ein Ziel innerhalb des Werkzeugs führen, sind blau. Externe Links sind grau gestaltet. Administratorfunktionalitäten haben eine rote Linkfarbe und werden nur eingeloggten Administratoren angezeigt. Wenn ein Benutzer ohne Administrationsrechte dennoch auf einen Link gerät, für dessen Inhalt er keine Berechtigung besitzt, wird eine entsprechende Benachrichtigung angezeigt.

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 Zurücksetzten des Passwortes nicht benötigt wurde, wurde diese auch nicht implementiert.

Abbildung 11: 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 Gutachters bei dieser Einheit
3. Show Parent - zeigt das direkte Vaterelement der Dokumentationseinheit an (siehe Abbildung 10)

4. Show File - zeigt die gesamte HTML-Datei, aus welcher diese Dokumentationseinheit stammt, im Original an

Sobald der Text von dieser Einheit mit dem Mauscursor markiert wurde, erscheint ein Fenster für die Auswahl eines Wissenstyps. Die Position der Markierung ergibt sich aus dem Start- und Endoffset[11] und wird dann zusammen mit dem Wissenstyp mit Hilfe von JavaScript zwischengespeichert. Bim Klicken auf „save“ wird sie über Ajax als POST-request an den Controller[12] (in Django ist es die views.py) gesendet. Nach dem Speichern wird dann die nächste Einheit angezeigt.

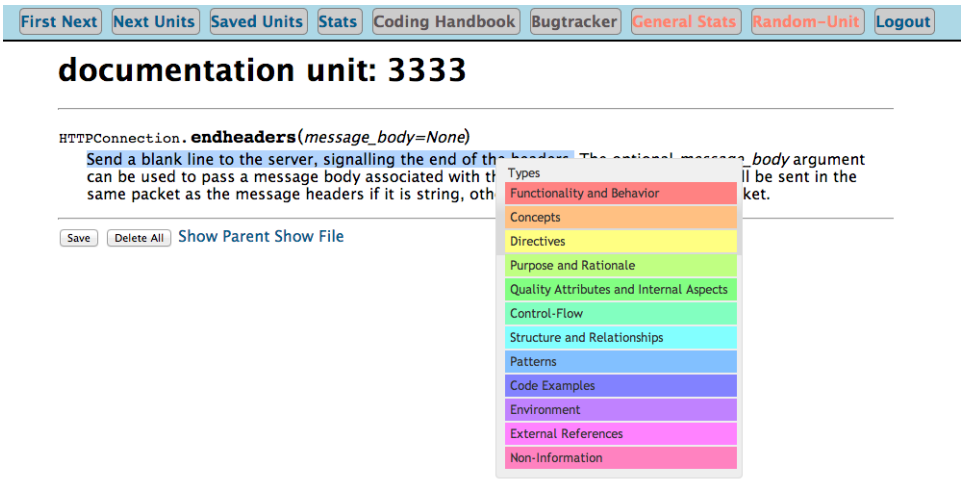


Abbildung 12: Einheit typisieren

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-Over[13] berührt wird, wird der damit verbundene, markierte Text rot und gestrichelt umrandet. Ein Klick auf dieses Element bewirkt das Löschen der angezeiten Markierung.

[11] realisiert mit Hilfe der JavaScript-Bibliothek „rangy“. https://code.google.com/p/rangy/

[12] Entsprechend des „Model View Controller“-Musters

[13] Bewegung des Maus-Cursor auf ein Element ohne es anzuklicken

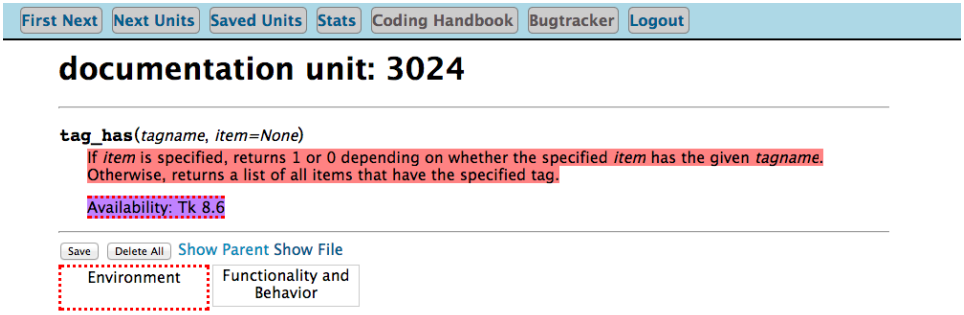


Abbildung 13: Einheit typisieren (2)

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, 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.

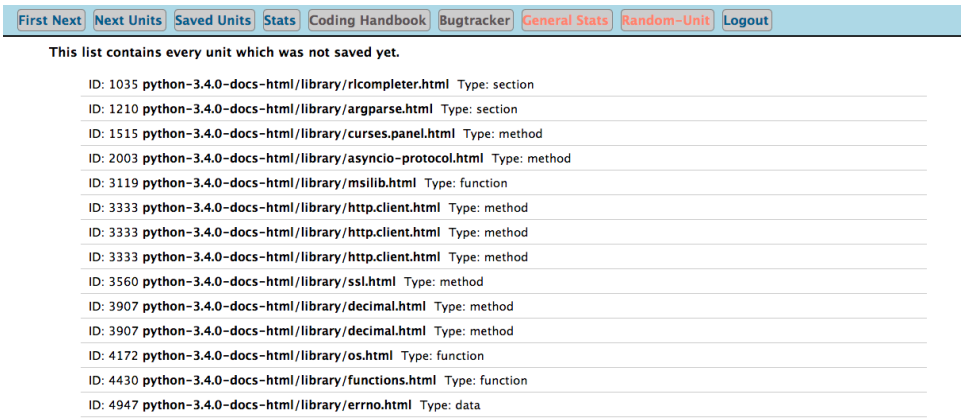


Abbildung 14: Noch nicht typisierte Einheiten

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 zu mehr als 25 Prozent und mehr als 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 einge-

führten Platzhalter herausgerechnet, 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 vorhergegangenen Speicherung ausgegeben und bei Hervorhebung noch die Angabe in Prozent, wie viel des Textes nicht markiert ist.

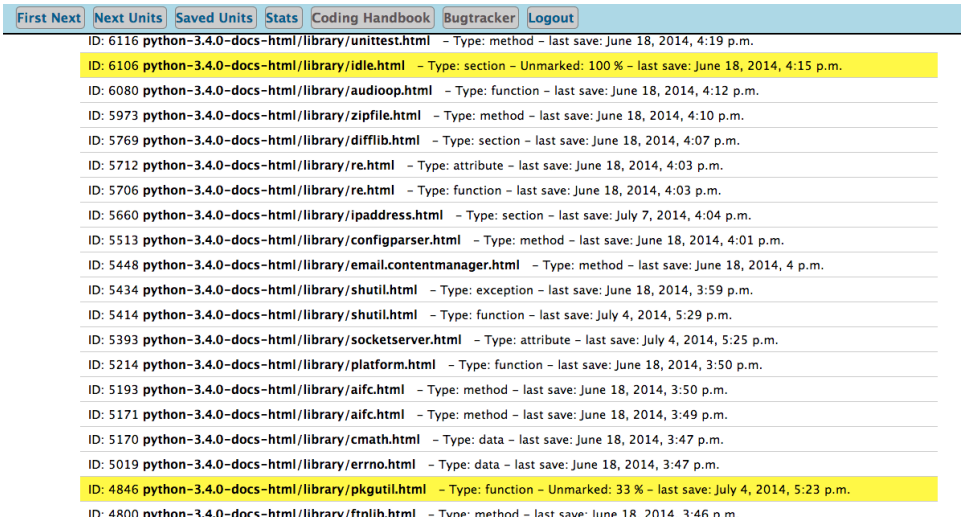


Abbildung 15: Abgespeicherte Einheiten

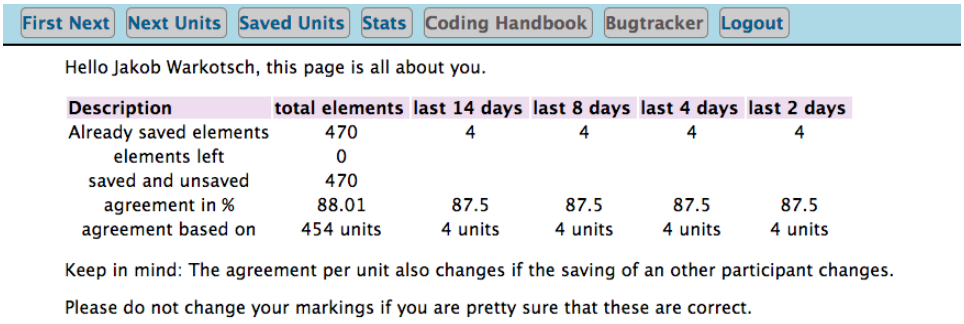
4.2.2.6 Persönliche Statistik

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

1. Anzahl insgesamt abgespeicherter Einheiten
2. Anzahl noch zu typisierender Einheiten (noch nicht abgespeichert)
3. Anzahl insgesamt zugewiesener Einheiten
4. Prozentuale 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 vierzehn, acht, vier und zwei Tage. Das hat den Vorteil, dass jeder Student schnell seinen eigenen Fortschritt für eine kurze Vergangenheit beobachten und dann auch ggf. erkennen kann, ob die Übereinstimmung mit anderen Gutachtern eher zu- oder abnimmt.



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

Abbildung 16: 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 Änderungskommentare sind so in der Regel mit der lösenden Änderung im Quelltext verknüpft.

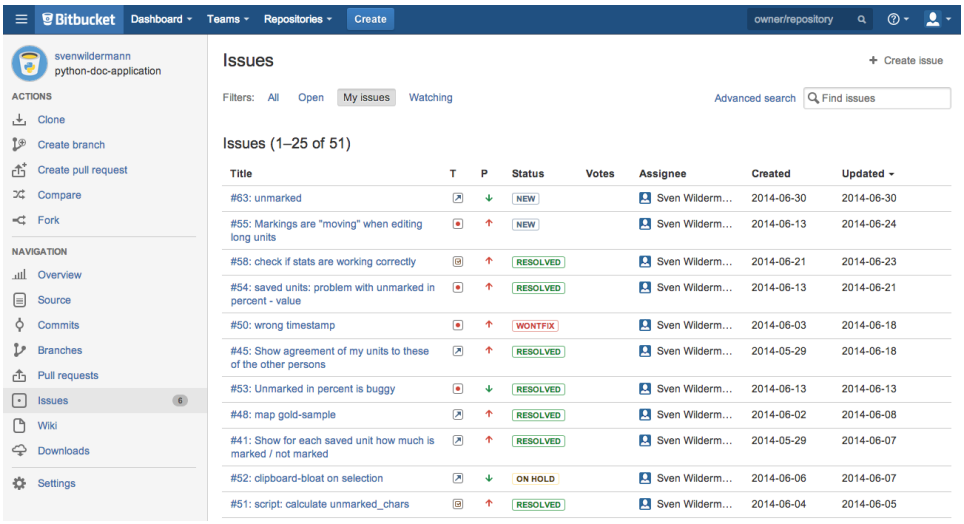


Abbildung 17: Bugtracker

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 Neuentwicklungen immer wieder anhand bisher unmarkierter Einheiten getestet werden 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 Hosting

Das gesamte Tool ist auf einem universitären Linux-Server des Instituts für Informatik gehostet. Über die Eingabe einer URL im Browser war es somit von überall aus erreichbar. Dabei kamen die Technologien Apache2, Django 1.6, GIT, Python3.2.3 und PostegreSQL 9.3.5 zum Einsatz. Entwicklungen während der Bachelorarbeit wurden jederzeit lokal auf einem Notebook betrieben, dann erst mit Hilfe des Versionskontrollsystems GIT an das Repository geschickt und schließlich auf dem Server wieder heruntergeladen. Der Link zum Repository war die einzige ausgehende Verbindung außerhalb des universitären Netzwerks, die akzeptiert wurde. Notwendige Änderungen an der Server-Datenbank wurden je nach Komplexität entweder manuell oder halbautomatisch mit Hilfe des Django-Werkzeugs „South“ [14]durchgeführt.

4.2.2.13 Datenbankmodell

Für die Datenhaltung habe ich ein Datenbankmodell konzipiert, welches ausgehend von den drei Grundtabellen die zusätzlichen Daten verknüpft. Die drei Grundtabellen sind: Die Wissenstypen (KnowledgeType), die Dokumentationseinheiten (DocumentationUnit) und die Benutzer (User). Die Tabelle „User“ wird bereits im Authentifizierungsmodul von Django mitgeliefert. So werden über „MappedUnit“ den Benutzern die Dokumentationseinheiten zugewiesen und über „MarkedUnit“ die Markierungen der Benutzer zu den Dokumentationseinheiten inklusive des Wissenstyps gespeichert. Ich habe es offen gelassen, ob auch Markierungen zu Elementen gemacht werden können, wenn diese nicht zugewiesen sind, so dass diese Relation nicht über „MappedUnit“ abgedeckt. Weiterhin werden über „AccessLog“ alle Zugriffe der Benutzer auf Dokumentationseinheiten gespeichert. In den Tabellen „Agreement“ und „Compatibility“ werden die Übereinstimmung und die Verträglichkeit von zugewiesenen Einheiten gespeichert. Ein Ausschnitt des Datenbankmodells ist in Abbildung 18 zu sehen.

[14]South ist ein Werkzeug für Datenbankmigrationen innerhalb von Django-Applikationen.

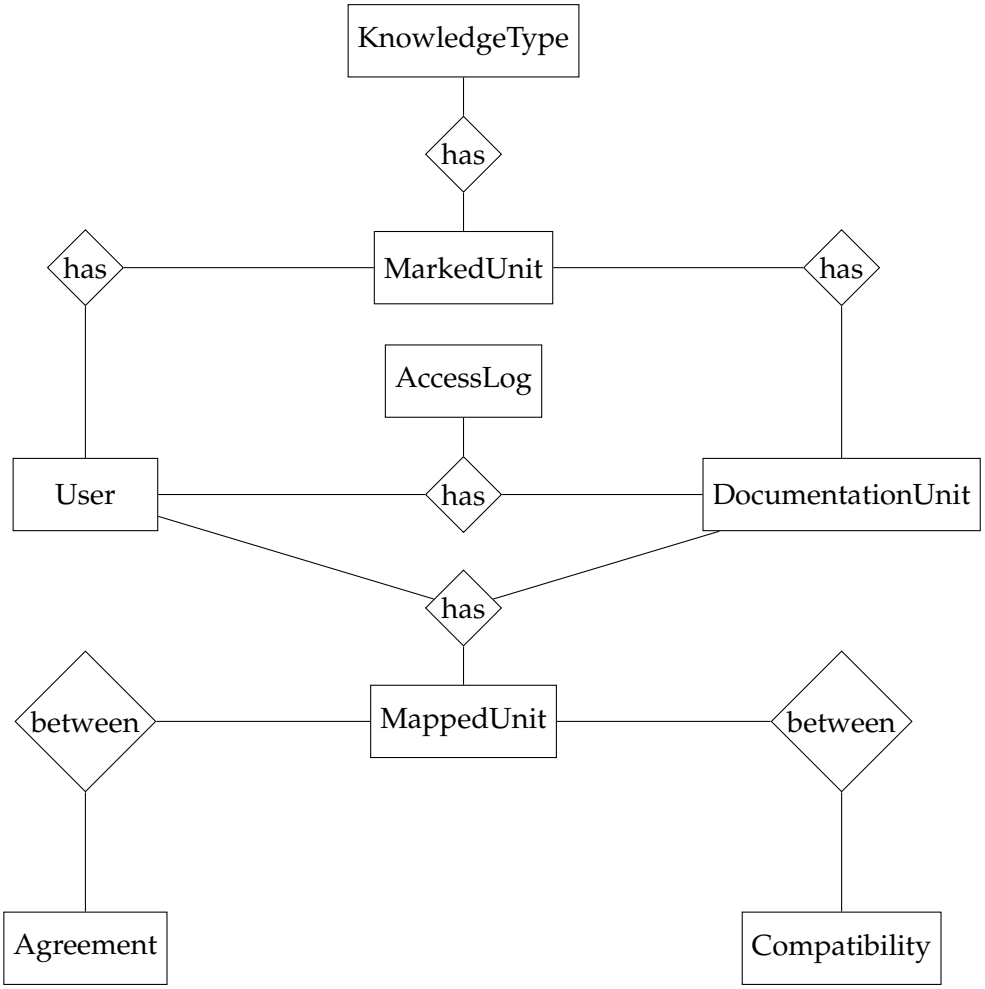


Abbildung 18: Datenbankmodell der Django-Applikation

4.2.2.14 Datenanalyse

Für die Analyse der erhobenen Daten wurden Django-Kommandos programmiert, die dann über die Terminalumgebung aufgerufen werden. Hierfür werden im Ordner „commands“ der Django-Applikation die Pythonmodule abgelegt, welche einen „BaseCommand“ implementieren. Für die Berechnung der Übereinstimmung zweier Einheiten, unter Zuhilfenahme einer Funktion aus der „views.py“, sieht dies beispielsweise wie folgt aus:

```
1  from django.core.management.base import BaseCommand
2  from extractor.models import MappingUnitToUser
3  from extractor.views import calculate_agreement
4
5  class Command(BaseCommand):
6      help = 'calculates the agreement of all units and
           save the results into the database'
7
8      def handle(self, *args, **options):
9          self.stdout.write("***START***")
10
11         all_units = MappingUnitToUser.objects.filter(
               user__groups__name='Students')
12         status_array = []
13         for each in all_units:
14             status = calculate_agreement(each.user,each.
                   documentation_unit.id)
15
16         self.stdout.write(str(len([x for x in
               status_array  if x == True])))
```

Für triviale Auswertungen wurden zudem Datenbankabfragen mit Hilfe SQL an die postgres-Datenbank getätigt. Ein Beispiel hierfür findet sich in Abschnitt 5.1.

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 detailierten 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 sechs Wochen gedauert hat, gab es drei 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 der Belohnung anderer informiert. Dabei wurden unterschiedliche Fortschritte prämiert:

1. Nach den ersten zwei Wochen wurde lediglich der fleißigste Teilneh-

mer 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 vollständig Einheiten typisiert wurden, ergab eine Prämierung des „fleißigsten Teilnehmers“ keinen Sinn mehr.

Insgesamt wurden Gutscheine im Wert von 30 Euro ausgezahlt.

4.4 Verteilung einer Stichprobe

Nachdem etwa die Hälfte der Zeit für das Typisieren der Hauptstichprobe verstrichen war, wurde bekannt, dass einer der Gutachter (C1) aus privaten Gründen nur die Hälfte seiner Stichprobe bearbeiten wird. Daher mussten die dem Gutachtern zugewiesenen 442 Einheiten $221 = \frac{442}{2}$ auf alle anderen Gutachter verteilt werden.
Um den Mehraufwand für die Studenten möglichst gering zu halten, haben wir uns dazu entschlossen, ebenfalls einen gleich großen Teil dieser Stichprobe zu typisieren. Zu den eigentlichen sieben Gutachtern wurden deswegen zwei weitere Gutachter-Accounts (später als C8 und C9 bezeichnet) hinzugefügt, so dass insgesamt neun Gutachter an der Hauptstichprobe mitgewirkt haben.
Dabei wurde allerdings darauf geachtet, dass wir (C8/C9) keine Einheiten erhalten, die ebenfalls in der Goldstichprobe vorhanden sind, um zu vermeiden, dass unsere Markierungen mit den Ergebnissen aus der Goldstichprobe verglichen werden. Dies hätte ansonsten zu einer Ergebnisverzerrung führen können. Im schlimmsten und unwahrscheinlichsten Fall wären so nämlich $42\ (= 2 * 21)$ von 168 Einheiten, also 25 Prozent, beim Vergleich mit der Goldstichprobe nicht mit den Ergebnissen von Studenten, sondern mit denjenigen von den Kursleitern verglichen worden.
Jeder Gutachter (mit Ausnahme von C1) hat somit 27 bzw. 28 Einheiten $(\frac{221}{8} = 27.625)$ zusätzlich bewertet.

5 Auswertungen

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

5.1 Verwechslungen

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 miteinander verwechselt werden. Mit Hilfe der folgenden SQL-Anweisung wurden alle aufgetretenen Konfusionen inklusive Häufigkeit des Auftretens ausgegeben:

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

Wenn jetzt noch die Reihenfolge ignoriert wird, also die Ergbenisse 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 19: Konfusionshäufigkeiten

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

sionen beziehen zu können, sind diese durchnummeriert. Für jedes dieser Paare wird einmal der Typ angegeben, welcher erfahrungsgemäß häufiger der richtige sein müsste (siehe Abbildung 20). Dabei wird berücksichtigt, dass es in der Regel besser sein sollte, in einem langen Segment erst den allgemeinen (stärkeren) und danach den speziellen (schwächeren) Fall zu typisieren (entsprechend des Kodier-Handbuchs). Dies ergibt die zweite und dritte Spalte der Tabelle in Abbildung 20. 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 eindeutig mit der Ersteinschätzung übereinstimmt, werden weitere zehn Elemente begutachtet. In Bezug auf die gesamte Stichprobe wirken sich diese Vorrangsregeln eher wenig aus, helfen aber dennoch weiter, um aus den Typisierungen einer Einheit von zwei Gutachtern ein Endergebnis zu schaffen, ohne große Teile der Markierungen weglassen zu müssen.

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	Functionality and Behaviour	12/20	3/20	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 20: Abschätzung der besseren Typen bei Konfusionen

Insgesamt können so 13 von 17 häufigen Konfusionen aufgelöst werden. Vier Konfusionen haben ein uneindeutiges Ergebnis. Keines der Ergebnisse widerspricht völlig den Erwartungen, auch wenn nicht alle Erwartungen mit Sicherheit bestätigt werden konnten.
Sobald eine der obigen Konfusionen auftritt und ein besserer Typ identifiziert wurde (grüne Zeilen), wird dieser dann als richtige Typisierung gewertet.

5.1.1 Hilfsmittel

Für diese Stichprobenziehung wurde eine Django-Funktion geschrieben, welche die betroffenen Benutzer und die ID der zehn Dokumentationseinheiten unter Angabe eines Konfusionspaares ausgibt. Indem noch zusätzlich die Option „–count“ übergeben wird, wird die Häufigkeit des angegebenen Konfusionspaares ausgegeben und keine Stichprobenziehung durchgeführt. Die zuletzt ausgebene Zahl gibt zum Zweck der Überprüfung die Anzahl der ausgebenen Zeilen an. Solange mehr als zehn Konfusionspaare der abgefragten Kombination vorhanden sind, wird eine „10“ ausgeben, andernfalls die Anzahl der verfügbaren Konfusionen. Beispielsweise erhält man bei der Eingabe

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

die Augabe

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

5.2 Ergebnis-Zusammenführung der Goldstichproben

Da die Goldstichprobe insgesamt dreimal von verschiedenen Gutachtern typisiert wurde, müssen deren für die weitere Verwendung zusammengeführt werden. Um aus drei Wahrheiten die „eine“ zu machen, wird wie folgt vorgegangen:
Es werden alle Paare (insgesamt drei) von Markierungen auf die Gesamtverträglichkeit über alle Zeichen berechnet. Die zwei höchsten Übereinstimmungswerte besitzen somit automatisch denselben Gutachter, so dass sich damit für diesen Gutachter die höchste Gesamtverträglichkeit der Einheit ergibt. Die Markierungen dieses Gutachters werden dann einem „Dummy-Benutzer“ kopiert.
Anschließend werden die Markierungen aller drei Gutachter mit dem des Dummys händisch verglichen und wo möglich noch kleinere Korrekturen vorgenommen, um die Qualität noch zusätzlich zu steigern. Diese sind folgender Natur:

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

Da die Wissenstypen „Structure and Relationship“ und „External References“ in der Regel offensichtlich sind, ist die händische Nachbearbeitung nicht diskussionswürdig.

5.3 Übereinstimmungen

5.3.1 Übereinstimmung pro Wissenstyp nach Gutachtern

Die Übereinstimmung der Wissenstypen nach Gutachtern wurde in der vorangegangen 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_is 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 21 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 Median ist dabei über alle Wissenstypen hinweg größer als 0,75 und insgesamt größer als 0,87.

Wissenstyp	**Min**	**Median.**	**Max.**	**Bester**	**Schlechtester**
Functionality	0,730	0,771	0,926	C9	C6
Concepts	0,792	0,858	0,963	C9	C5
Directives	0.803	0,885	0,929	C8	C6
Purpose	0,703	0,811	0,857	C8	C6
Quality	0,820	0,873	0,910	C1	C4
Control-Flow	0,893	0,947	0,964	C6	C8
Structure	0,633	0,763	0,817	C5	C4
Patterns	0,873	0,887	0,964	C8	C5
Code-Examples	0,963	0,985	1,0	C8	C9
Environment	0,890	0,945	1,0	C8, C9	C6
References	0,951	0,970	0,977	C4	C2
Non-Info	0,765	0,852	1,0	C8	C4
Gesamt	0,818	0,879	0,942	C8	C6

Abbildung 21: 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 Typisierungweise 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. Zudem wurde in der Originalstudie mit Hilfe dieser Werte die Sinnhaftigkeit der Informationstypen unter Beweis gestellt. Diese wurden in dieser Arbeit vorausgesetzt und stehen daher nicht zur Diskussion.

5.3.3 Übereinstimmung nach Einheiten

Zusätzlich kann die Übereinstimmung nach Einheiten berechnet werden. Für jede Einheit wird für jeden Wissenstypen geprüft, ob die Gutachter mindstens eine Markierung dieser Art getätigt haben. Eine Übereinstimmung liegt dann vor, wenn beide Gutachter sich einig sind, ob dieser Wissenstyp (mindestens einmal) vorkommt oder nicht. Da es insgesamt zwölf Wissenstypen gibt, entspricht jede Übereinstimmung in einem Wissenstyp $\frac{1}{12} = 8,33$ Prozent . Für jeden Gutachter wird so die Übereinstimmung über alle Einheiten berechnet und der Durchschnittswert in Abbildung 22 ausgegeben. Der Zeile des besten Gutachters ist grün, die des schlechtesten rot markiert. Analog wird der Durchschnittswert für die unterschiedlichen Kategorien in Abbildung 23 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.
Die Werte entsprechen genau denen, die den Gutachtern und Administratoren während der Typisierung unter den eigenen Statistiken angezeigt werden und wurden von dort übernommen. In Abbildung 22 ist erkennbar, dass Gutachter C9 am meisten mit den Typisierungen der anderen Gutachtern übereinstimmte. Wenn man die Gutachter C8 und C9 wegen der in Kapitel 4.4 beschriebenen Problematik bei der Analyse ausblendet, so ergibt sich Gutachter C7 sehr knapp vor C2 als Gutachter mit den höchsten Übereinstimmungswerten. Am wenigsten Übereinstimmung mit den anderen Gutachtern hat C6.
Die Übereinstimmung allein sagt aber noch wenig über die Güte der Typisierungsergebnisse aus. Wenn man davon ausgeht, dass es den Fall geben kann, dass ein Großteil der Gutachter sich in manchen Fällen (auf die selbe Weise) irrt und ein anderer Gutachter die Situation richtig einschätzt, so wird dieser

mit schlechten Übereinstimmungswerten abgestraft. Da die Unterschiede zwischen den Übereinstimmungswerten gering sind (maximal 6,2 Prozent), kann man von einer eher gleichmäßigen Qualität ausgehen.

Gutachter	Übereinstimmung
C1	0,873
C2	0,882
C3	0,855
C4	0,849
C5	0,870
C6	0,845
C7	0,883
C8	0,899
C9	0,907
Durchsch.	0.874

Abbildung 22: Übereinstimmung nach Einheit je Gutachter

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

Abbildung 23: 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 Wissenstyp 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 desselben.

5.3.5 Übereinstimmung mit Goldstichprobe

Um die Gutachter entsprechend der Qualität ihrer Leistungen einordnen zu können, werden die Typisierungen dieser mit den Ergebnissen aus der

Goldstichprobe anhand der Verträglichkeit (siehe Kapitel 5.3.7) verglichen. Aus den Verträglichkeitswerten ergibt sich dann direkt eine Platzierung für alle Gutachter der Hauptstichprobe — mit Ausnahme der Gutachter C8 und C9 wegen der in Kapitel 4.4 besprochenen Änderung bezüglich der Stichprobenverteilung. Daher wurden die Platzierungen für die Gutachter C8 und C9 aus den Übereinstimmungswerten in Abbildung 22 entnommen. Wegen des geringen Anteils dieser Bewertungen an der gesamten Stichprobe ist diese Vorgehensweise tolerierbar.
So erhält man einen ungefähren Wert für gute Typisierungsqualität. Die Markierungen von Gutachter C7 sind in 96,269 Prozent der Fälle mit dem Markierungen aus der Goldstichprobe verträglich. Anders ausgedrückt sind weniger als 4 von 100 Markierungen von Gutachter C7 nicht verträglich mit den Markierungen der Goldstichprobe.

Platzierung	Gutachter	Übereinstimmung
1	C9	-
2	C8	-
3	C7	96.269
4	C4	93.782
5	C2	93.374
6	C5	91.215
7	C3	90.589
8	C1	90.536
9	C6	89.753

5.3.6 Unstimmigkeiten zur Goldstichprobe

Die bisherigen Untersuchungen beziehen sich immer auf die Übereinstimmung zwischen den Wissenstypen oder den Gutachtern. Ebenso wichtig ist es, zu erkennen, wie genau die Unstimmigkeiten zur Goldstichprobe beschaffen sind.
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: Der Gutachter gibt an, dass der Wissenstyp in dieser Einheit vorhanden ist, obwohl dies laut Goldstichprobe nicht der Fall ist.

2. Falsch-Negativ: Der 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-positiven (FP) und falsch-negativen (FN) Bewertungen verursacht wurden. Da jede Einheit doppelt typisiert wurde, liegt eine Menge von 336 Typisierungen mit insgesamt $336 * 12 = 4032$ [15] Bewertungen zugrunde.
In Abbildung 24 wird deutlich, dass die meisten Unstimmigkeiten bezüglich des Wissenstyps „Structure and Relationship“ (83) entstanden sind und am wenigsten Uneinigkeit bei dem Wissenstypen „Code-Example“ herrscht. Das ist deswegen plausibel, weil Programmierbeispiele im Gegensatz zu zu den meisten anderen Typen einfach anhand der Syntax identifiziert werden können.

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
Durchschn.	46	0,539	0,461

Abbildung 24: Unstimmigkeiten zur Goldstichprobe

5.3.6.1 Gemeinsam unstimmig

Noch interessant ist der Fall, wenn sich zwei Gutachter bei einem Wissenstyp für eine Einheit der Goldstichprobe einig sind, aber dennoch andere Typen, als die aus dem Ergebnis der Goldstichprobe, gewählt haben. Daher wird überprüft, wie oft für jeden Wissenstyp genau dieser Fall in der Goldstichprobe auftritt. Auch hier wird zwischen falsch-postiven und falsch-negativen Werten unterschieden.
So erhält man in Abbildung 25 einen stochastischen Wert für die ungefähre Ungenauigkeit des Endergebnisses.
Dabei wird deutlich, dass es am meisten gemeinsame Unstimmigkeiten bei dem Wissenstyp „Structure and Relationship“ (14,2 Prozent) und am wenigsten bei den Wissenstypen „Control-Flow“ und „Environment“ (je 3

[15] Es gibt 12 Wissenstypen

Wissenstyp	Gesamt	Gesamt in Prozent	FP	FN
Functionality	11	0,065	0,545	0,455
Concepts	6	0,036	0,50	0,50
Directives	8	0,048	0,50	0,50
Purpose	3	0,018	0,667	0,333
Quality	14	0,083	0,857	0,143
Control-Flow	2	0,012	0,0	1,0
Structure	24	0,142	0,25	0,75
Patterns	4	0,024	0,50	0,50
Code-Examples	3	0,018	1,0	0,0
Environment	2	0,012	0,50	0,50
References	5	0,03	0,20	0,80
Non-Info	5	0,03	0,60	0,40
Durchschn.	7,25	0,043	0,51	0,49

Abbildung 25: Gemeinsam unstimmig

Prozent) gibt. So haben die studentischen Gutachter insgesamt 18 mal den Wissenstyp "Structure and Relationship" nicht gefunden obwohl dieser in der Einheit laut Goldstichprobe vorhanden ist.
Im Durchschnitt gibt es diese Art von Unstimmigkeiten in 4,3 Prozent aller Einheiten.

5.3.7 Verträglichkeit von Einheiten

Weiterhin wird die Verträglichkeit der Einheiten betrachtet und ausgewertet. Dazu werden zunächst aufeinanderfolgende Markierungen des selben Wissenstyps zu einer Markierung zusammengefasst. Zwei Markierungen sind aufeinanderfolgend wenn der Abstand zwischen diesen nicht mehr als drei Zeichen beträgt. Die Verträglichkeit bezieht sich auf die einzelnen Markierungen einer 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 denselben Wissenstypen oder eine Kombination aus den Konfusionspaaren besitzen.
Oder anders ausgedrückt:
Seien 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 M1 und M2 verträglich, wenn:

1. $(B_1 >= B_2)$ & $(E_1 <= E_2)$ ||
2. $(E_1 >= B_2$ & $B_1 <= B_2)$ & $((E_1 - B_2) >= \frac{E_1 - B_1}{2})$

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

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 26 die durchschnittliche Verträglichkeit pro Gutachter dargestellt wird, zeigt Abbildung 27 diese pro Kategorie.
Von beiden Werten ist die Verträglichkeit nach Zeichen aussagekräftiger als die Verträglichkeit nach Markierungen, da so die Verträglichkeiten anhand der Länge gewichtet werden. So erhält eine kurze Markierung nur geringes Gewicht bei der Berechnung und analog dazu eine lange Markierung entsprechend viel Gewicht.
Insgesamt ist erkennbar, dass die Markierungen im Durchschnitt zu mehr als 80 Prozent verträglich zueinander sind. Dass die Verträglichkeitswerte gemessen an der Zeichenanzahl durchgehend höher sind als die nach Anzahl der Markierungen, macht deutlich, dass sich Gutachter bei größeren Segmenten eher einig sind als bei kleineren.

	Verträglichkeit	
Gutachter	(1) nach Zeichen	(2) nach Markierungen
C1	0,894	0,873
C2	0,894	0,879
C3	0,875	0,856
C4	0,776	0,758
C5	0,902	0,878
C6	0,851	0,831
C7	0,892	0,874
C8	0,965	0,938
C9	0,917	0,887
Durchsch.	0.885	0,863

Abbildung 26: Verträglichkeit nach Gutachtern

Kategorie	Verträglichkeit (1) nach Zeichen	(2) nach Markierungen
Methoden	0,914	0,862
Felder	0,774	0,769
Module	0,876	0,841
Klassen	0,786	0,772
Beschreibungen	0,861	0,854
Durchschn.	0,842	0,819

Abbildung 27: Verträglichkeit nach Kategorien

5.4 Ergebnis-Zusammenführung - Hauptstichprobe

Da jede Einheit aus der Hauptstichprobe ebenfalls 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 durch ein Skript (Django-Kommando) automatisiert werden.
Das Markierungsresultat wird für die Markierungen der Gutachter G_i und G_j wie folgt bestimmt:
Die zusammengefassten Markierungen (siehe Kapitel 5.3.7) werden nach ihrem Auftreten aufsteigend sortiert. Nacheinander werden alle Markierungen, die sich zu mindestens 50 Prozent überschneiden, miteinander verglichen. Sobald ein Gutachter bei diesem Vergleich die Markierung mit dem „besseren“ Wissenstyp (laut Konfusionsergebnis) getätigt hat, bekommt er einen „Pluspunkt“ und seine Markierung wird in das Endresultat übernommen (erster Fall).
Wenn es für eine Markierung entweder eine Übereinstimmung der Wissenstypen gibt oder sie so verschieden sind, dass diese nicht zu den aufgelösten Konfusionspaaren gehören, dann wird die Markierung des Gutachters mit mehr „Pluspunkten“ übernommen (zweiter Fall).
Falls die „Pluspunkte“ kein eindeutiges Ergebnis liefern, werden die Gutachter mit den besseren Ergebnissen im Vergleich zur Goldstichprobe bevorzugt (dritter Fall).
Wenn die Segmentierungsweisen (also die Start und Endpunkte der Markierungen) beider Gutachtern stark voneinander abweichen, entstehen im Endergebnis unter Umständen mehr Markierungen als im Durchschnitt beider Gutachter. Insgesamt wurden so 5737 Markierungen auf 1548 Einheiten identifiziert, also im Durchschnitt 3,7 Markierungen pro Einheit.

5.5 Strukturen und Muster der Wissenstypen

In diesem Abschnitt wird das Endergebnis der Hauptstichprobe auf die Strukturen und Muster der vorkommenden Wissenstypen untersucht.

5.5.1 Vorkommen und Verhältnis

Abbildung 28 zeigt die Häufigkeit aller aufgetretenen Wissenstypen über alle Markierungen in absoluten Zahlen an. In Abbildung 29 wird diese Darstellung noch einmal mit relativen Zahlen im Bezug zur Gesamtmenge aller Markierungen wiederholt. In Abbildung 30 wird das Verhältnis der Wissenstypen über alle Dokumentationseinheiten dargestellt, zum Beispiel enthalten 72 Prozent aller Einheiten den Wissenstyp „Functionality and Behaviour“. Diese Informationen werden dann in Abbildung 31 noch einmal für jede Kategorie einzeln dargestellt. Aus dieser Tabelle wird beispielsweise deutlich, dass die Einheiten der Kategorie „Methoden“ eindeutig mehr Wissen zu dem Wissenstyp „Functionality and Behavior“ enthalten als alle anderen Einheiten (mindestens 30 Prozent mehr) und dass in Modulen viel häufiger Programmierbeispiele vorkommen als in allen anderen Kategorien (mindestens dreieinhalbmal so viel).

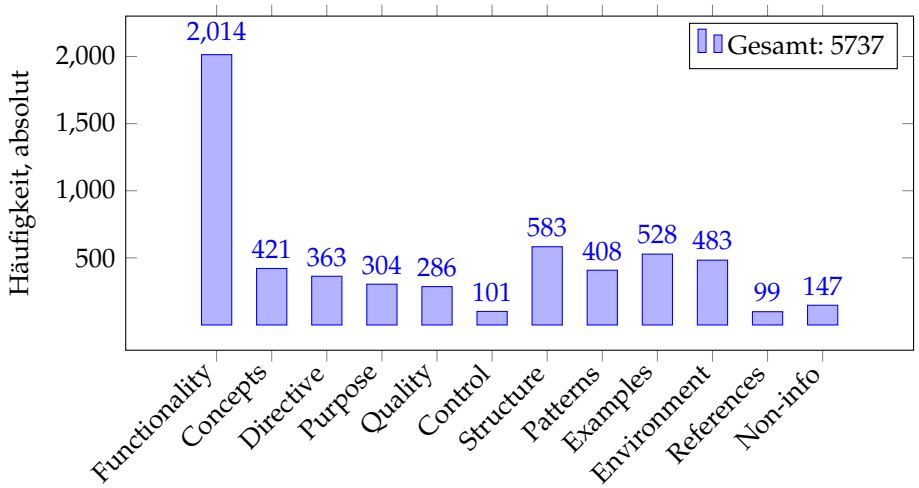


Abbildung 28: Häufigkeiten der Wissenstypen

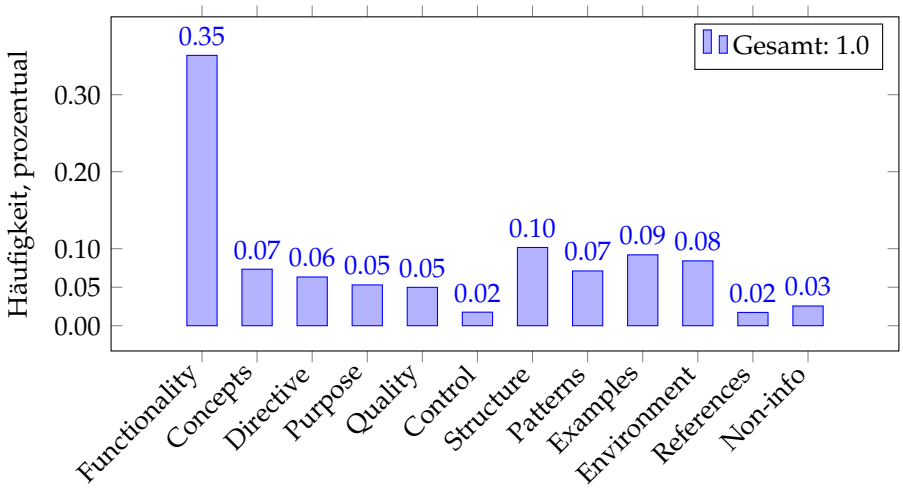


Abbildung 29: prozentuale Häufigkeiten der Wissenstypen

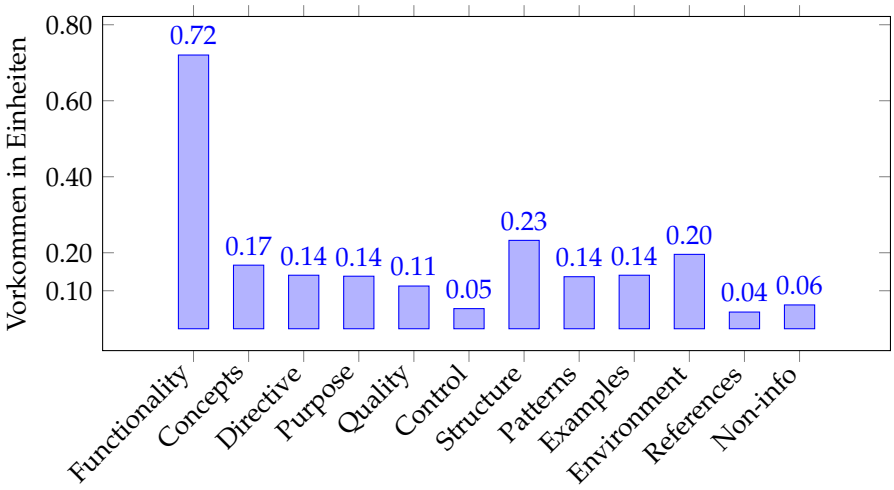


Abbildung 30: Verhältnis der Wissenstypen pro Dokumentationseinheiten

Wissenstyp	Methoden	Felder	Module	Klassen	Beschreibungen
Functionality	0.94	0.59	0.44	0.64	0.88
Concepts	0.08	0.09	0.38	0.24	0.00
Directives	0.18	0.03	0.15	0.17	0.00
Purpose	0.10	0.05	0.25	0.22	0.00
Quality	0.08	0.04	0.26	0.12	0.00
Control	0.08	0.0	0.05	0.05	0.00
Structure	0.18	0.13	0.32	0.38	0.13
Patterns	0.12	0.03	0.28	0.15	0.06
Examples	0.10	0.05	0.37	0.08	0.06
Environm.	0.19	0.20	0.18	0.23	0.00
Referenc.	0.02	0.03	0.12	0.03	0.00
Non-Info.	0.02	0.02	0.17	0.08	0.00

Abbildung 31: Verhältnis der Wissenstypen pro Kategorie

5.5.2 positive Korrelation

Die Auswertung in Kapitel 5.5.1 zeigt die Verteilung der verschiedenen Wissenstypen, jedoch kein direktes Muster von gemeinsam auftretenden Wissenstypen. Fraglich ist, ob es beispielsweise den Wissenstyp „Quality Attributes and Internal Aspects“ meist in Verbindung mit „Functionality and Behavior“ gibt.
Daher wird in Abbildung 32 die Korrelation zwischen zwei verschiedenen Wissenstypen in absoluten Zahlen und in Abbildung 33 im Verhältnis zum Vorkommen beider Typen im Endergebnis angezeigt. Die Berechnung dieser Werte erfolgte jeweils innerhalb eines Django-Kommandos. Wenn ein Wissenstyp mehr als einmal in einer Dokumentationseinheit vorgekommen ist, wurde dieser (analog zum Artikel von Maalej und Robillard [MR13]) nur einmal gewertet.
Grundsätzlich sind dabei alle 66 möglichen Kombinationen aufgetreten. Am meisten korrelieren die Wissenstypen „Patterns“ mit „Examples“ (23 Prozent) sowie „Concepts“ mit „Structure“ (21 Prozent), am wenigsten korrelieren „Control“ mit „References“ (3 Prozent) sowie „Control“ mit „Non-Information“ (4 Prozent).
Im Durchschnitt gibt es zwischen zwei Wissenstypen eine Korrelation von 12 Prozent. Signifikat nach oben abweichende Werte können als „positive Korrelation“, nach unten als „negative Korrelation“ bezeichnet werden, um die Vergleichbarkeit mit der Originalstudie zu erhöhen.

-	Concepts	Directives	Purpose	Quality	Control	Structure	Patterns	Examples	Environm.	Referenc.	Non-Info
Functionality	159	188	172	134	73	259	165	145	233	55	62
Concepts		72	90	66	14	128	78	81	74	32	39
Directives			65	67	32	89	67	61	85	13	32
Purpose				63	19	99	82	71	74	23	33
Quality					14	83	69	56	70	21	34
Control						32	25	15	22	5	7
Structure							99	90	114	34	49
Patterns								100	73	19	34
Examples									60	18	37
Environm.										28	29
Referenc.											12

Abbildung 32: absolute Kookurrenz von Wissenstypen

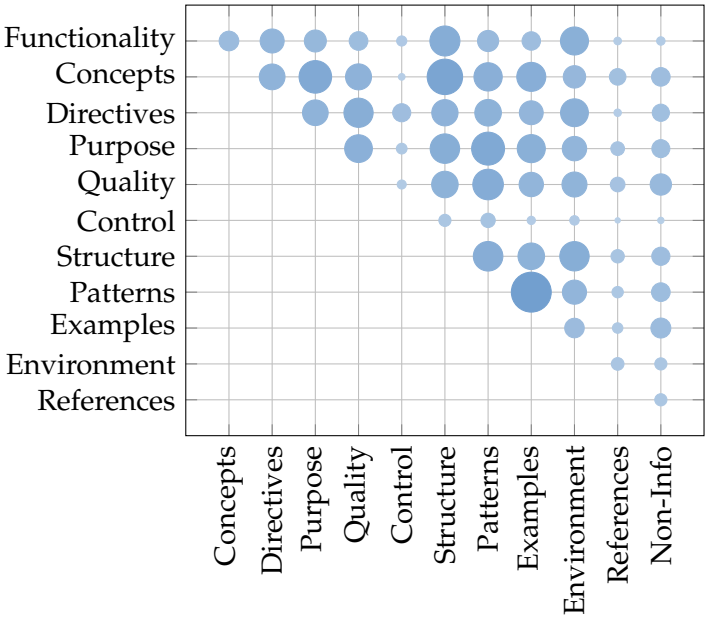


Abbildung 33: Kookkurrenz der Wissenstypen im Verhältnis zum Vorkommen beider Typen

5.5.3 Längenanalyse

Weiterhin kann die Beziehung zwischen der Anzahl der verschiedenen Wissenstypen in einer Dokumentationseinheit und der Länge dieser Einheit betrachtet werden.
Für die Länge der Einheiten wird analog zur Originalstudie die Anzahl der Wörter gemessen. Wörter sind hierfür als durch Leerzeichen getrennte Zeichenketten definiert. In Abbildung 34 wird die Beziehung zwischen der Länge der Einheiten und der Kategorie in Tabellenform angegeben und in Abbildung 35 mit Hilfe eines Boxplot-Diagramms dargestellt.
In den Abbildungen 36 bis 40 werden für je eine Kategorie die Anzahl der verschiedenen typisierten Wissenstypen mit den Längen der Einheiten dargestellt. Während „Minimum“ und „Maximum“ die „Antennen“ (auch

„Whisker“ genannt) der Boxplots angeben, wird das Rechteck durch die Werte „unteres Quartil“ und „oberes Quartil“ gegeben. Die zusätzliche Linie innerhalb des Rechtecks bezeichnet den „Median“.
In der Studie von Walid Maalej und Martin Robillard wurden die „Whisker“ anders berechnet, so dass zusätzlich noch „Ausreißer“ als Punkte dargestellt werden konnten. Da die Berechnung dieser „Antennen“ allerdings nicht einheitlich geregelt ist und sich in der Originalstudie kein Hinweis darauf finden lässt, wie diese berechnet wurde, werden hier die Extrema dafür festgelegt.
Es ist dabei eine deutliche proportionale Korrelation zwischen Länge der Einheiten und gefundenen Wissenstypen erkennbar. Am deutlichsten ausgeprägt ist diese für die Einheiten der Kategorien „Methoden“ und „Klassen“. Die Ergebnisse für die Kategorie in Abbildung 40 „Beschreibungen“ zeigen vor allem, dass diese Einheiten nicht nur recht kurz sind (wie bereits in Abbildung 35 gezeigt), sondern auch nur wenig unterscheidbare Informationen enthalten; ähnlich verhält es sich mit den Einheiten der Kategorie „Felder“. Die Einheiten ohne zugeordnete Wissenstypen haben zwei unterschiedliche Ursachen: Zum einen gibt es immer wieder Einheiten, welche lediglich eine Überschrift und nur den Satz „[something removed here]“ besitzen, da der Inhalt vollständig in untere Kategorien durch den Extrahierer aufgeteilt wurde. Zum anderen gab es wenige Einheiten, die keine Informationen über die API, sondern über die Benutzung der GUI [16] beinhalteten und deswegen auch nicht typisiert wurden.

	unt. Quartil	**Median**	**Durschn.**	**ober. Quartil**	**Min**	**Max**
Methoden	126	226	371.30	430.5	38	3333
Felder	58	131	185.59	230	9	2292
Module	298	739.5	1062.25	1438.5	41	9439
Klassen	137	263	479.46	539	22	2841
Beschreibungen	43	65.5	120.31	110	37	823
Gesamt	117	238.5	488.88	521	9	9439

Abbildung 34: Anzahl der Wörter nach Kategorie

[16] Graphical User Interface (engl.) = Grafische Benutzeroberfläche

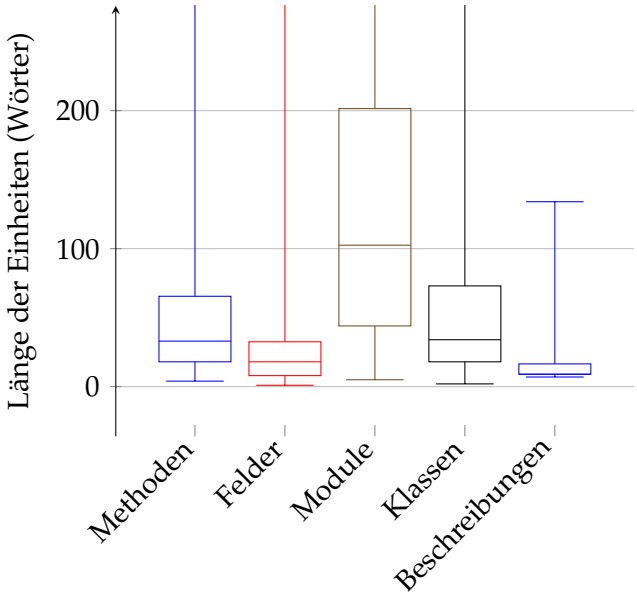


Abbildung 35: Anzahl der Wörter nach Kategorie in einem Boxplot-Diagramm

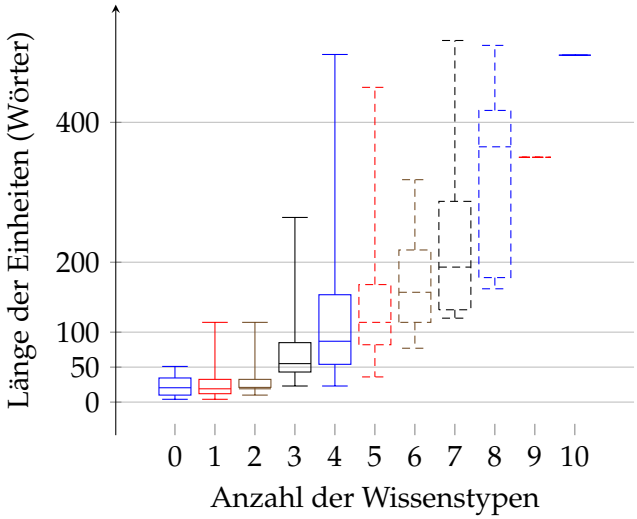


Abbildung 36: Methoden: Anzahl der Wörter und Anzahl der Wissenstypen

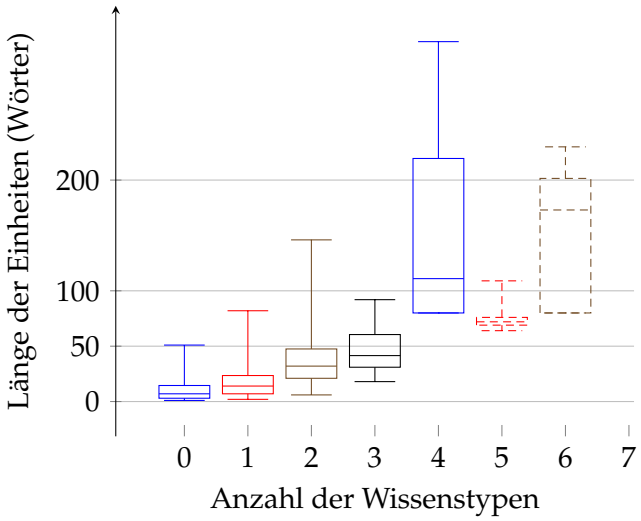


Abbildung 37: Felder: Anzahl der Wörter und Anzahl der Wissenstypen

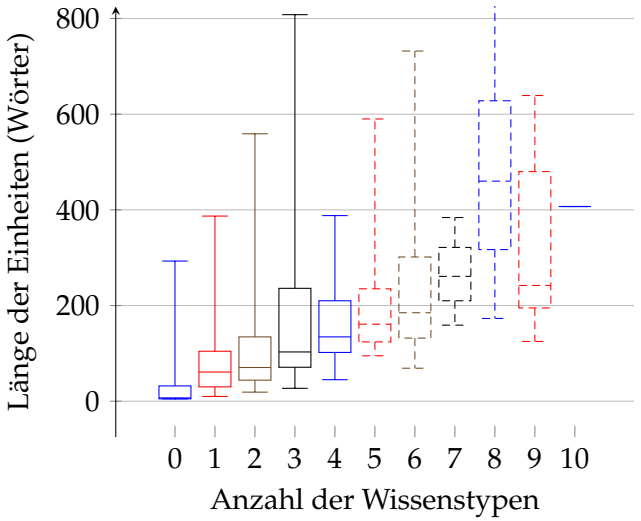


Abbildung 38: Module: Anzahl der Wörter und Anzahl der Wissenstypen

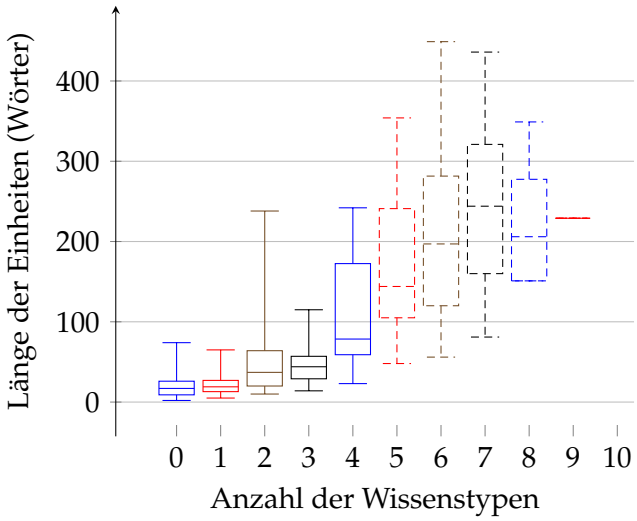


Abbildung 39: Klassen: Anzahl der Wörter und Anzahl der Wissenstypen

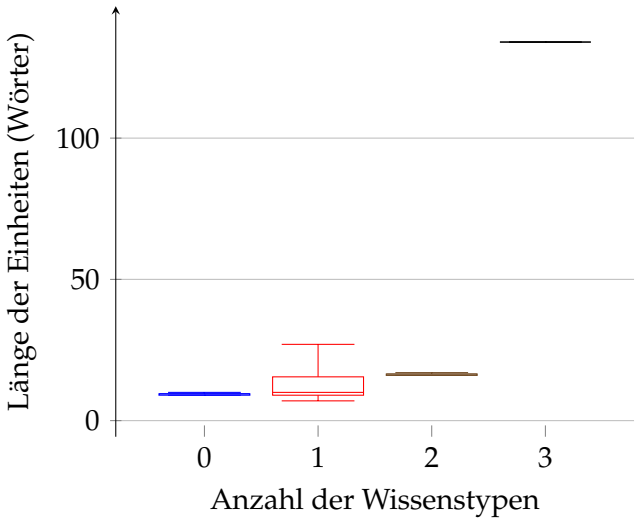


Abbildung 40: Beschreibungen: Anzahl der Wörter und Anzahl der Wissenstypen

6 Fazit

Das Ziel dieser Arbeit war das Übertragen der Informationtypen-Analyse von Walid Maalej und Martin Robillard [MR13] auf die Python-Dokumentation. Mit Hilfe der erhobenen Daten und der darüber ausgeführten Auswertungen können nun fundierte Aussagen über die Art und Verteilung von Informationen in der Dokumentation getroffen werden. Außerdem können diese Daten mit den Ergebnissen für die Dokumentationen in JAVA und .NET verglichen werden. Insgesamt kann die Durchführung dieser Arbeit als Erfolg gewertet werden, da über eine reine Folgestudie hinaus noch einige Verbesserungen im Ablauf vorgenommen wurden, die sich auf die Qualität der Ergebnisse ausgewirkt haben.

6.1 Vergleich

Es ist zu beachten, dass es teilweise Unterschiede in den Berechnungen dieser Arbeit und der Originalstudie gibt. Da in dieser Arbeit die Zuordnung der Wissenstypen nicht über das Setzen von Haken in „CheckBoxes“, sondern über das Markieren von Zeichenketten geschieht, sind die Ergebnisse nicht nur detailreicher, sondern auch weniger zufällig. Daher entfallen die Auswertungen darüber, wie wahrscheinlich es ist, dass sich zwei Gutachter zufällig einig sind.
Dennoch gibt es einige Auswertungen, die sowohl in dieser Arbeit als auch in der anderen Studie vorgenommen wurden. Die Berechnung der Übereinstimmung von Typisierungen verschiedener Gutachter wurde analog ausgeführt und zeigt, dass in dieser Arbeit die Übereinstimmungen zwischen den Gutachtern im Median um fast sechs Prozent höher sind (82 Prozent zu 87,9 Prozent). Die Werte für die minimale und die maximale Übereinstimmung zwischen zwei Gutachtern sind sogar fast zehn Prozent höher. Das führt zu dem Schluss, dass die in dieser Arbeit gewählte Methode zur Erfassung von Informationstypen zu größerer Sorgfalt geführt hat, obgleich die Dauer für die Typisierung pro Einheit anstieg.
Die Berechnungen für die Unstimmigkeiten zwischen den Gutachtern sind in dieser Arbeit erheblich anders durchgeführt worden als in der Originalstudie und bieten somit keine direkte Vergleichsmöglichkeit. Man erkennt aber sehr wohl, dass es bei beiden Studien nur wenige Unstimmigkeiten bezüglich des Wissenstypen „Code Examples“ gibt und eher viel Uneinigkeit darüber, wann die Wissenstypen „Structure and Relationship“ und “Functionality and Behaivor“ zutreffend sind.
Die Verteilung der verschiedenen Wissenstypen über alle Einheiten zeigt einige Parallelen. So sind allen drei Dokumentationen (Java, .NET, Python) viele Einheiten mit dem Wissenstyp „Functionality and Behavior“ vorhanden. Zu etwa 20 Prozent kommt in allen untersuchten Dokumentationseinheiten auch der Wissenstyp „Structure and Relationship“ vor. Einen

entscheidenden Unterschied gibt es bei der Menge an Einheiten mit dem Wissenstyp „Non-Information“. Während bei Java und .NET in etwa der Hälfte der Dokumentationseinheiten dieser Typ identifiziert wurde, waren es bei Python nur etwa sechs Prozent.
Wenn man die nach Kategorie aufgeteilten Wissenstypen beider Studien vergleicht, erkennt man bei Java und Python denselben Wert für die Kategorie „Klassen“ und den Wissenstyp „Functionality and Behaivor“. In beiden Dokumentationen liegt der Wert bei 64 Prozent. Fast identisch sind zudem die Werte dieser Kategorie für die Typen „Structure and Relationship“ und „Directives“. Gleichzeitig findet man in etwa jeder zweiten Methodenbeschreibung in JAVA und .NET „Non-Information“. Bei Python ist dies gerade einmal in jeder fünfzigsten Methode der Fall.
Analog zur Originalstudie wurde auch in dieser Arbeit eine starke Korrelation der Typen „Patterns“ und „Examples“ gefunden. Überlagert man die Grafiken, sind ähnliche Muster erkennbar. Dazu gehören beispielsweise die geringe positive Korrelation der Typen „Non-Information“ und „References“ zu allen anderen Wissenstypen sowie die hohe Korrelation der Wissenstypen „Examples“ und „Patterns“ zu allen übrigen Wissenstypen.
Der proportionale Zusammenhang zwischen der Anzahl der Wörter in einer Dokumentationseinheit und der Anzahl der Wissenstypen konnte in der vorangegangen Studie ebenso gezeigt werden wie in der vorliegenden. Dabei fiel das Ausmaß des Zusammenhangs bei der Python-Dokumentation über alle Kategorien hinweg deutlicher aus, als es bei Java und .NET der Fall war. Gleichzeitig ist zu bemerken, dass der Plaintext[17] in dieser Arbeit mit Hilfe des in BeautifulSoup definierten Befehls

```
1   findAll(text=True)
```

bereits im Extrahierer ausfindig gemacht wurde und dieser so möglicherweise mehr (Leer-)zeichen enthält als angezeigt werden. Daher werden unter Umständen auch mehr als die sichtbaren Wörter gezählt. Die Überprüfung dieses Verdachts bedarf einer weiteren Datenanalyse.
Zusammengefasst konnten sowohl wichtige Gemeinsamkeiten als auch Unterschiede zwischen den Dokumentationen von Java, .NET und Python gefunden werden. Vor allem der viel geringe Anteil an Nicht-Information bei gleichzeitig höherem Anteil an „Functionality and Behavior“-Wissenstypen in der Python-Dokumentation kann als Vorteil dieser gegenüber den anderen gewertet werden.

6.2 Ausblick

Mit den erhobenen Daten können über diese Bachelorarbeit hinaus noch viele andere Auswertungen getätigt werden. Beispielsweise kann untersucht werden, welche Markierungen sich in der Regel überlappen und in welcher

[17] Technischnische Bezeichnung für Klartext

Syntax dies geschieht. Damit erhält man dann eine Aussage darüber, welche Informationstypen zusammen in einem Satzgebilde verfügbar sind.
Weiterhin kann betrachtet werden, in welcher Reihenfolge bestimmte Wissenstypen auftauchen. Das ist ein wichtiger Schritt, um die Semantik hinter der Syntax zu verstehen. Man kann beispielsweise durch die Korrelation von „Examples“ und „Pattern“ davon ausgehen, dass in der Regel erst „Pattern“ (wie setze ich XY um) beschrieben wird und im Anschluss daran ein Programmierbeispiel folgt. Diese Vermutungen können dann durch Analysen bestätigt oder widerlegt werden.
Die Menge der Information je Informationstyp kann durch die vorliegenden Daten sehr genau analysiert werden. Während in der Originalstudie von Walid Maalej und Martin Robillard bekannt wird, ob ein Wissenstyp vorhanden ist, kann man jetzt sogar aussagen, wie viel davon vorhanden ist. Bei den Übereinstimmungswerten ist diese Analyse zu Teilen bereits mit eingeflossen.
Um die bis hierhin genannten Auswertungen in Relation zu anderen Dokumentationen auswerten zu können, sollte diese Art der Auswertung auch mit JAVA und .NET-Dokumentationen (und weiteren) wiederholt werden. Nur so kann langfristig der objekte Unterschied zwischen den Dokumentationen gänzlich verstanden werden.

Sofern weitere Analysen bezüglich Dokumentationen für Programmiersprachen folgen, kann möglicherweise schon bald auf Grundlage dieser ein Leitfaden für gute Dokumentationen veröffentlicht werden. Ziel dieses sollte es dann sein, den Informationsgehalt der Dokumentationen möglichst zu steigern und geringwertige Informationen zu vermeiden. Schließlich kann Programmcode nur so gut sein wie das Verständnis des Programmierers und dieses wiederum nur so gut wie die vorliegenden Informationen zu der Programmiersprache.

7 Anhang

7.1 Quelltext

Die entwickelten Werkzeuge (Website, Extrahierer, Django-Kommandos) stehen als GIT-Repository auf bitbucket.org zur Verfügung:
https://bitbucket.org/svenwildermann/python-doc-application.
Der letzte Commit[18] vor Abgabe dieser Arbeit fand am 9. August 2014 statt und besitzt folgenden HASH-Wert, wobei die ersten sieben Zeichen bereits für eine Identifizierung genügen:

```
34cbc4765d485f3d76cc33e89622b5509bc026bc
```

Die Angabe dieses Wertes bestätigt den Stand der Programmierung zum Zeitpunkt der Abgabe. Weiterentwicklungen nach Abgabe der Bachelorarbeit sind, speziell für weiterführende Auswertungen über den Rahmen dieser Arbeit hinaus, möglich und ggf. auch nötig.

[18] Befehl des Versionskontrollsystems „GIT“ für das Bekanntgeben von Änderungen

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. http://cado.informatik.uni-hamburg.de/coding-guide/.

[MRb] Walid Maalej and Martin P. Robillard. Content Analysis for Software Documentation. http://cado.informatik.uni-hamburg.de/tool/.

[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. http://www.inf.fu-berlin.de/w/SE/ThesisPythonDokuInfotypenCodingHandbook.