Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Messung der Informationstypen-Häufigkeiten in der Python-Dokumentation

Sven Wildermann
Matrikelnummer: 4567553
bachelorarbeit@wildermann.berlin

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

Berlin, 27. Juli 2014

Zusammenfassung

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

Danksagungen

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

Eidesstattliche Erklärung

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

27. Juli 2014

Sven Wildermann

Inhaltsverzeichnis

Sven Wildermann

Ideen für die Bachelorarbeit

1. Ein DOM-Knotenbaum einer typischen HTML-Datei einbauen
2. UML-Diagramm des Tools
3. Datenbank-Eintitäten aufzeigen
4. Möglicher Workflow (als Nicht-deterministischer Automat oder so)
5. Screenshot z.B. von Dokumentationseinheit1898 hinzufügen, um das Interface zu zeigen
6. Alle Links als Fussnoten anzeigen - sieht besser aus und ist besser für den offline gebrauch!
7. Nummerierung bei den Code-Schnipseln hinzufügen
8. Bei Bildern immer Bildunterschriften einfügen, die etwas zu dem gezeigten aussagen
9. Alle Kommentare im Code auf Rechtschreibfehler überprüfen
10. Related Work
11. Future Work

1 Einleitung

Zu jeder Programmiersprache gehört eine Dokumentation über die bereit gestellten 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 zum Erfolg oder Misserfolg einer Programmiersprache bei. Entscheidend ist, wie gut die Entwickler mit den gebotenen Informationen zu recht kommen und wie schnell Antworten gefunden werden können. Die Anzahl und Qualität der Programmbeispiele ist dabei mindestens genauso wichtig wie die Erläuterung von Funktionalitäten, Konzepten und Abhängigkeiten.
Um etwas über die Qualität von Dokumentationen aussagen zu können, muss erst einmal verstanden werden, welche Informationen zu welchen Teilen in dem jeweiligen Handbuch vorhanden sind. Der erste Teil dieser Frage wurde bereits von den Herren Maalej und Robillard [MR13] beantwortet. Sie fanden bei der Analyse der JAVA und .NET Dokumentationen insgesamt 12 gut unterscheidbare Wissenstypen, im Original heißen diese knowledge types.
Die Analyse über die Häufigkeit dieser Typen in der Dokumentation wird von Studenten innerhalb eines Forschungspraktikums am Institut für Informatik durchgeführt. Sie erhalten Ausschnitte aus der Dokumentation über eine hierfür entwickelte Website und geben dann an, welche der 12 Typen auf diese Einheit passen. Die genauen Regeln für die Bewertung dieser Einheiten gibt das Kodier-Handbuch an, welches im Wesentlichen aus der Originalstudie übernommen und auf Python angepasst wurde.

1.1 Aufbau der Arbeit

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

2 Grundlagen

Warum diese Version in Python - also 3-4-1? Welche Elemente werden geparst? Warum diese?

2.1 Artikel von Maalej und Robillard

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

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

Abbildung 1: Wissenstypen in [MR13]

jeden Wissenstyp bestimmen, ob dieser in der vorgelegten Einheit vorkommt oder nicht. Hierzu konnten für jede Einheit die verschiedenen Wissenstypen mit Hilfe von „CheckBoxes“ angekreuzt werden. Jede Einheit wurde von 2 Gutachtern unabhängig bewertet. Die Einheiten wurden in drei verschiedene 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 Informatinosgehalts bei .NET dann aber nicht analysiert. Damit beschränkt sich die Original-Studie also auf Typen und Mitglieder. Die Stichprobe über die verbleibenden vier Kategorien (zwei je Programmiersprache) wurden mit 95% Konfidenzintervall und 2,5% Fehlerspanne gezogen, so dass 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 Auswertungen über die Fälle getroffen, in denen sich zwei Gutachter uneinig waren. Zudem konnten so Aussagen darüber getroffen werden, welche Wissenstypen bei welcher Einheitskategorie wie häufig vorkommen und ob bestimmte Wissenstypen mit einander korrelieren, also ob z.B. der Typ “Structure“ häufig zusammen mit „Functionality and Behaivor“ auftritt. Ebenso wurde analysiert, ob und wie ein Zusammenhang zwischen der Anzahl der gefundenen Wissenstypen und der Länge der Einheiten besteht.
Besonders die vorgestellten Wissenstypen und das Kodier-Handbuch [Coding Guide] waren für diese Folgestudie sehr nützlich und wurden deswegen übernommen und angepasst.

2.2 Kodier-Handbuch

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

2.2.1 Markierungen

Der wichtigste Unterschied zu dem originalen Kodier-Handbuch [MRa] ist, dass die Gutachtern nicht nur pro Einheit bewerten sollen, ob bestimmte Wissenstypen vorhanden sind, sondern Markierungen in einer Einheit vornehmen und pro Markierung einen Typ festlegen. Wie mit einem Textmarker

soll jede Einheit vollständig bearbeitet werden und abschließend jedes Zeichen (mit wenigen Ausnahmen) markiert sein. Dies führt zu ergänzenten Regeln über :

- Die Art Markierungen vorzunehmen
- Der Länge von Markierungen
- Die Notwendigkeit von Doppelmarkierungen in einem Segment

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

> You will be presented with documentation blocks extracted from API reference documentation (Javadocs and the like). For each block, you will be also presented with the name of its corresponding package/namespace, class, method, or field. Your task is to read each block carefully and evaluate where the block contains knowledge of the different types described below. Apply the following rules when doing so:
>
> - Consider the documentation initially one paragraph at a time. If the paragraph contains only information of one knowledge type, mark the whole paragraph with that type in one stretch. Never mark more than one paragraph at once.
> - If multiple knowledge types mix within the paragraph, mark a contiguous stretch of one or more sentences with one type and the next stretch with another.
> - If necessary, treat subsentences connected with conjunctions such as „and“, „or“, „but“, or with colon or semicolon like complete sentences.
> - A sentence (or such subsentence) as a whole is never marked with more than one type, but sometimes phrases within the sentence will require a separate marking with a different type. Double-marking the same text with two types is allowed (and required) in this case. To create such annotations uniformly, we work in two passes:
>
>   – Pass 1: Prefer longer segments of a complete sentence or several. Annotate subsentences only rarely. If a sentence contains knowledge of more than one type (which happens quite often), look if one of them is clearly dominant for the overall role of the sentence in the documentation block. If so, annotate only that dominant type to the whole sentence and do not annotate any of the other types yet.

  - Pass 2: After pass 1, many relevant annotations will be missing. We now add those on top of the pass 1 annotations as double annotations. For the double annotations, we still prefer complete subsentences where possible (or other clearly delineated parts such as parentheses), but choose the shorter of two possibilities whenever we are unsure.

- Rate the knowledge type as true only if there is clear evidence that knowledge of that type is present in the stretch. If you have doubts, consult the type's definition below. If the doubts do not disappear, do not annotate that type.

- However, all text of the documentation must be marked with a type. (Only hand-written documentation, not the signature itself and not the placeholders [Something removed here] that indicate left-out nested documentation blocks).

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

2.2.2 Kleinere Änderungen

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

- Informationen über bestimmten Input einer Funktion oder Methode, welcher zu einer „Exception“ führt (und nur dann), soll als „Directive“ und nicht als „Functionality and Behavior“ markiert werden.

- Die simple Nennung von gültigen Parametertypen wird als nicht als „Directive“ angesehen, sofern nicht Schlüsselwörter wie „must“ oder „have to“ etc. verwendet werden.

- Der Ausdruck „Changed in version x.y.“ soll als „Environment“ markiert werden. Der darauf anschließend Text kann ebenfalls „Environment“ sein, muss es aber nicht.

- Platzhalter der Form „[Something removed here]“ sollen nicht markiert und bewertet werden, da diese lediglich auf ausgelassene, verschachtelte Einheiten hinweisen (siehe Abschnitt „Extrahierer“)

- Der Wissenstyp „Structure“ wird treffender in „Structure and Relationship“ umbenannt.

3 Konzeption

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

3.1 Dokumentationseinheiten

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

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

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

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

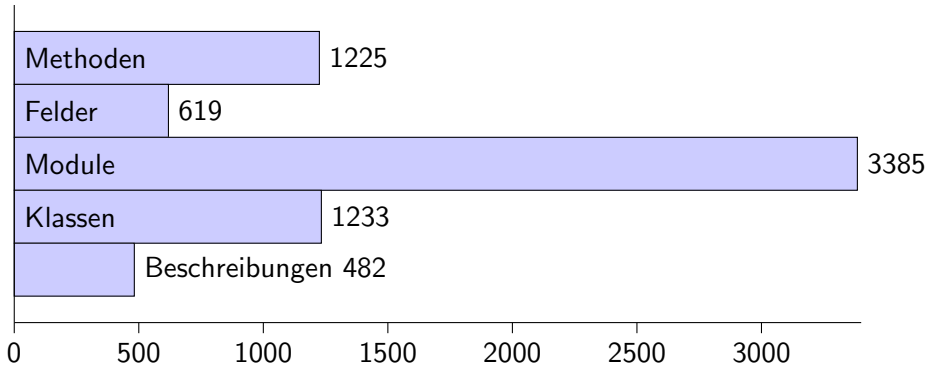

Abbildung 2: Durchschnittliche Textlängen je Kategorie

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

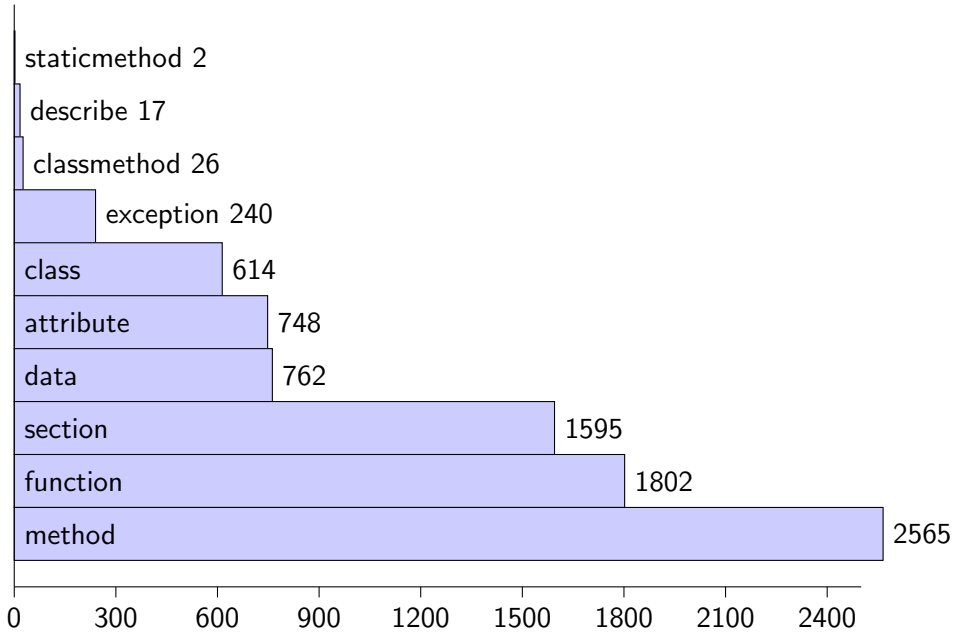

Abbildung 3: Gesamthäufigkeiten der Einheiten

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

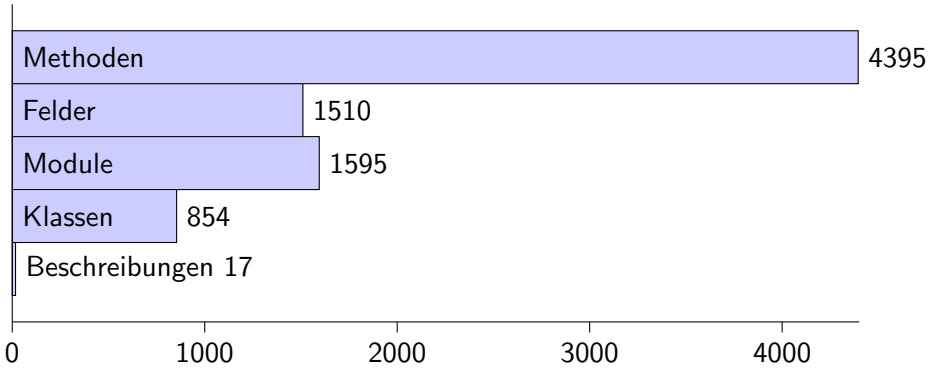

Abbildung 4: Gesamthäufigkeiten der Kategorien

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

3.2 Stichprobe

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

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

wobei n_0 wie folgt berechnet wird:

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

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

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

Abbildung 5: Stichprobengrößen

3.3 Goldstichprobe

3.4 Zeitaufwand

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

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	⌀7.8 min	57,92h

Abbildung 6: Einheiten pro Student

Zusätzlich waren die Gutachter dazu angehalten, regelmäßig das Kodier-Handbuch zu lesen, um das übergreifende Verständnis nicht zu verlieren so-

[1] European Credit Transfer System

wie gelegentlich mit dem BugTracker[2] umzugehen. Hierfür wurden zusätzlich insgesamt noch etwa 10 Arbeitsstunden invenstiert ,wobei diese geschätze Zahl je nach Gutachter stark nach oben und unten abweicht.

3.5 CADo-Tool vs. Eigenentwicklung

Im Rahmen der Forschung von Maalej und Robillard [MR13] wurde ein Tool geschaffen, welches folgende Werkzeuge und 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 Sicht der Kodierer birgt dieses Tool 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

Warum haben wir nicht das CADO-Tool benutzt sondern selbst etwas entwickelt? Welche Entscheidungen wurden hiervon beinträchtigt -> Markierungen statt Blockweise

4 Durchführung

4.1 Extrahierer

Um die Dokumentationseinheiten aus der HTML-Dokumentation von Python zu erhalten, war es nötig, einen Extrahierer als Skript zu schreiben.

[2] Ein System zur Erfassung von Defekten und Verbesserungsvorschlägen

Dieser wurde in Python3 mit Hilfe von BeautifulSoup4 angefertigt. Entsprechend der Definitionen von Dokumentationseinheiten sollten also die HTML-Schnipsel getrennt von einander in die 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, habe ich deswegen Platzhalter der Form „[something removed here]“ an solchen Stellen eingebaut. Platzhalter haben einen wichtigen Vorteil gegenüber dem einfachen Weggelassen dieser Elemente: So sieht auch der Gutachter, dass hier etwas von der Original-Dokumentation abweicht und kann sich somit die entstandenen Lücken erklären. Dieser Fall tritt besonders häufig bei Sektionen auf, da diese alle weiteren Elemente beinhalten.
Eine typische Struktur für die Verschachtelung von Dokumentationseinheiten sieht so aus: **BILD VON EINER TYPISCHEN STRUKTUR MALEN - TODO**
Warum aber wurde BeautifulSoup eingesetzt und nicht etwa reguläre Ausdrücke?

4.1.1 BeautifulSoup4

Für das Parsen der HTML-Einheiten habe ich BeautifulSoup4 verwendet, da es ein hierfür geschaffenes, mächtiges Werkzeug darstellt und gleichzeitig auf Python basiert. Somit konnten Synergieeffekte genutzt werden und ich mich weiter in Python einarbeiten während ich dieses Skript schrieb. Zudem bietet der Einsatz von BeautifulSoup viele Vorteile gegenüber dem Parsen mittels regulären Ausdrücken. In erster Linie ist der Code lesbarer und verständlicher, sowohl für den Programmierer 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 Unterstützung erhalten zu können. Die Implementierung selbst wird im nachfolgenden Kapitel erklärt.

4.1.2 Implementierung

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

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

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

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

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

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

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

Der Aufruf zum parsen alle Elemente der Form

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

und abspeichern dieser in einer Liste funktioniert dann wie folgt:

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

Dieser Schritt wurde analog für alle zehn verschiedenen Elementsytpen ausgeführt. Um später Aussagen über die Lage der Texte innerhalb einer Einheit zu erhalten, wurden zudem die Startoffsets der Elemente berechnet und später zusammen mit dem Endoffsets in der Datenbank abgelegt, wobei sich das Endoffset jeweils sehr leicht errechnen lässt: $Ende = Start + Laenge$.
Für die Bestimmung der Startoffsets wurden die einzelnen Elemente in ihrer Datei mittels find aus BeautifulSoup gesucht und die Rückgabe, also der Index an der Stelle des Auftretens, gespeichert. Auch hierfür gibt es eine eigene Funktion, um den Aufruf lesbarer zu gestalten:

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

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

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

Wegen der bereits erwähnten, 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 Elemente durch folgende Platzhalter ersetzt:

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

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

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

Die final zur Verfügung stehenden Informationen konnten dann in der Datenbank abgespeichert werden.

4.2 Typisierungswebsite

4.2.1 Anforderungen

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

- Ein- und Auslogfunktion
- Betriebssystemunabhängige Online-Erreichbarkeit
- Anzeige der dem Student zugewiesenen Einheiten
- Markierung von 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

4.2.3 Ergebnis

4.3 Gamification

5 Fazit / Ergebnisse

5.1 Auswertung der Typisierungen

5.2 Herausforderungen

6 Anhang

6.1 Technologien

6.1.1 Django

6.1.2 Python

6.1.3 BeautifulSoup4

6.1.4 Coffeescript

6.1.5 Postgres

6.1.6 Ajax

6.1.7 R

6.2 Kodierhandbuch - vollständig

6.3 Glossar

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

Literatur

[METM12] Martin Monperrus, Michael Eichberg, Elif Tekes, and Mira Mezini. What should developers be aware of? an empirical study on the directives of api documentation. *Empirical Software Engineering*, 17(6):703–737, 12 2012.

[MRa] Walid Maalej and Martin P. Robillard. Coding guide.

[MRb] Walid Maalej and Martin P. Robillard. Coding guide.

[MR13] Walid Maalej and Martin P. Robillard. Patterns of knowledge in api reference documentation. *IEEE Transactions On Software Engineering*, 39(9):1264–1282, 09 2013.

[PW] Lutz Prechelt and Sven Wildermann. Coding guide for python.

Abbildungsverzeichnis