AUTOMATISCHES AUFLÖSEN VON ABKÜRZUNGEN IN QUELLTEXT

Dokumentenart:	Exposé für eine Masterarbeit
Autor:	GILBERT GROTEN
Matrikel-Nr.:	1772069
Studiengang:	Informatik Master
Betreuer:	TOBIAS HEY
Datum:	26. Juni 2020

1 Motivation

Sowohl für die manuelle, als auch für die automatische Analyse und Wartung von Software ist die Lesbarkeit des Quelltextes von äußerster Wichtigkeit. Dies beinhaltet die Wahl von ausdrucksstarken Bezeichnern [10] [8]. Trotz allgemeinem Bewusstsein um diesen Umstand sind Abkürzungen im Quelltext leider immer noch sehr verbreitet. Diese hindern die maschinelle Erfassung des Textes, oder im Falle von mangelndem Projekt- oder Domänenwissen, sogar das Verständnis von menschlichen Betrachtern. Somit können die Rückverfolgbarkeit von Anforderungen und die Wartung oder Weiterentwicklung der Software erschwert werden. Die Interpretation des Textes, sowohl durch Menschen, als auch durch Werkzeuge, kann zudem durch falsche Domänen-Annahmen behindert werden. Gerade wenn Abkürzungen in der Domäne mehrdeutig sind, kann dies zu zeitaufwändigen Missverständnissen führen.

Um die Analyse und Wartung von Abkürzungen enthaltender Software oder die maschinelle Erfassung von Zuordnungen und Zusammenhängen zu erleichtern, wäre ein Verfahren zur automatischen Auflösung von Abkürzungen zu lexikalischen Wörtern hilfreich. Zwar existieren solche Werkzeuge bereits, doch ihre Qualität ist noch nicht ausreichend für die Praxis [2] [3]. Insbesondere für die Verwendung als Vorverarbeitungsschritt für automatische Software-Analysen sind Fehler problematisch, da diese Mängel sich in der Weiterverarbeitung potenzieren.

In Quelltextbeispiel 1 ist ein menschlicher Leser mit etwas Programmiererfahrung in der Lage zu erkennen, was die einzelnen Abkürzungen bedeuten. Dieser kann den Quelltext zurückverfolgen, und somit „cn“ als Akronym für „class name“, „str“ als Präfix von „string“ oder „cmp“ als das Wort „compare“, bei dem einige Buchstaben weggelassen wurden, identifizieren.

```
public static class CompareAndCopy
{
    //computing whether two words are common in all characters
    bool cmpstr(std::string str, std::string cn)
    {
        return str == cn;
    }

    //for the common purpose of standard replacement of words
    bool cmpCpStr(std::string* str, std::string* class_name)
    {
        if (!cmpstr(*str, *class_name))
        {
            *str = *class_name;
            return true;
        }

        return false;
    }
}
```

Quelltextbeispiel 1: Abkürzungen und deren Auflösungskandidaten im Quelltext.

Ein automatisches Analyseverfahren, welches keine Auflösungstechnik für Abkürzungen nutzt, würde den abgekürzten Bezeichnern keine Information entnehmen können. Auch bei automatischer Auflösung der Abkürzungen sind einige Schwierigkeiten zu beachten. So hält sich die Methode „cmpstr“ nicht an den Standard der Binnenmajuskelschreibweise (eng. „camel case“), und es muss anderweitig herausgefunden werden, ob und wo dieser Bezeichner in zwei Abkürzungen zu spalten ist. Zudem bieten die Kommentare einige irreführende Auflösungskandidaten. Hier wird deutlich, dass die Frage nach der besten Quelle für Auflösungskandidaten wichtig ist. In diesem Beispiel befinden sich die richtigen Auflösungskandidaten in Quelltext-Bezeichnern. Insbesondere in denen mit Koreferenzen auf die Bezeichner, welche die entsprechenden Abkürzungen enthalten. In anderen Fällen, wie zum Beispiel bei domänenspezifischen Abkürzungen, könnte dies wiederum anders sein. Oftmals sind richtige Auflösungen überhaupt nur in der Dokumentation zu finden [2]. Dennoch könnte der Quelltext passende, aber eben unrichtige, Auflösungskandidaten enthalten.

2 Zielsetzung

Ziel dieser Arbeit ist es, ein Verfahren zu entwickeln, welches das Erkennen und Auflösen von Bezeichnern ermöglicht, welche Abkürzungen enthalten.

Hierzu muss zunächst erkannt werden, welche Bezeichner Abkürzungen enthalten. Gegebenenfalls müssen diese Bezeichner daraufhin aufgeteilt werden, da sie aus mehreren Abkürzungen zusammengesetzt sein können. Dazu muss analysiert werden, welche Abkürzungstypen wie und wo im Quelltext vorkommen können. Um die gefundenen Abkürzungen auflösen zu können sollen Auflösungskandidaten gesucht werden. Diese zu finden erfordert eine geeignete Auswahl der Wissensquellen (z.B. Quelltext oder Dokumentation), in welchen nach Kandidaten gesucht wird. Abschließend soll unter den gefundenen Auflösungskandidaten der geeignetste Kandidat ausgewählt werden.

3 Vorgehen

Zunächst müssen Abkürzungen und Bezeichner, welche diese enthalten, erkannt werden. Existierende Verfahren setzen hierzu unter anderem Wörterbücher ein, um einen Bezeichner darin zu suchen [9]. Wird er darin gefunden, wird davon ausgegangen, dass es sich um keine Abkürzung handelt. Weitere Wissensquellen, wie den Quelltext oder andere zugehörige Softwareartefakte, werden erst bei der Suche nach Auflösungskandidaten eingesetzt. Allerdings könnten bereits in diesem Schritt mehr Bezeichner ausgeschlossen werden, wenn weitere Wissensquellen zur Erkennung eingesetzt würden. Außerdem sind einige noch nicht zu Rate gezogene Wissensquellen denkbar, um darin nach Abkürzungen oder Auflösungskandidaten zu suchen. So zum Beispiel domänenspezifische Literatur, oder gar Dokumentation oder Quelltext anderer Software aus der gleichen Domäne.

Nicht jedes nicht lexikalische Wort enthält Abkürzungen. Bezeichner können sowohl gänzlich aus einer Abkürzung oder mehreren Teilbezeichnern bestehen. Für die Aufspaltung solcher Bezeichner existieren bereits Werkzeuge [7] [4], hier gibt es allerdings noch Verbesserungspotential [6]. Da zusammengesetzte Bezeichner zu mehreren adjazenten Wörtern aufgelöst werden sollten, wäre z.B. ein Werkzeug denkbar, welches als Spaltungs- und Auflösungskandidaten häufig auftretende N-Gramme berücksichtigt [1].

Für die Auflösung der Abkürzungen selbst sind bereits verschiedene Ansätze verfolgt worden. Diese nutzen verschiedene Wissensquellen, meist Software-Artefakte, um darin nach potentiellen Auflösungen einer Abkürzung zu suchen [2]. Auch sind die Verfahren nicht für alle Abkürzungstypen identisch. Oft werden Annahmen darüber getroffen, welche Wissensquelle allgemein zu priorisieren ist, oder für einen bestimmten Abkürzungstyp die besten Auflösungskandidaten enthalten dürfte [5]. Diese basierten bislang allerdings vorrangig auf Vermutungen. Inzwischen gibt es einige Beobachtungen bezüglich der Verteilung von Abkürzungen, beziehungsweise Abkürzungstypen und deren Auflösungen über verschiedene Wissensquellen [2], auf die man sich in Zukunft bei diesen Entscheidungen stützen kann.

Auch bislang nicht genutzte linguistische Information könnte die Auf-

lösungsfindung unterstützen. Zum Beispiel ist es möglich, dass die Begriffe, welche die Auflösung eines Bezeichners darstellen, nicht adjazent in den Wissensquellen zu finden sind (zum Beispiel „strCmp(s1, s2); //This method compares two strings.“ - im Kommentar steht „string“ nach „compare“, anstatt umgekehrt). Dieser Fall scheint kaum erforscht zu sein. Laut Newman et al. [2] könnte die Berücksichtigung solcher Auflösungskandidaten die Ausbeute eines Verfahrens deutlich erhöhen.

Vorangegangene Arbeiten nutzen nur begrenzt [3] oder gar keine [5] statische Analyse des Programms, um Auflösungen zu finden. Diese könnte helfen, über den Aufrufgraphen Zusammenhänge zwischen Bezeichnern zu erkennen. So könnten Auflösungskandidaten in den Quelltext-Bezeichnern gefunden werden, oder semantische Zusammenhänge erschlossen, die auf die Semantik von Wissensquellen abgebildet werden könnten. Für beides könnte Koreferenzanalyse des Quelltextes, sowie der natürlichsprachlichen Wissensquellen genutzt werden.

Mit Genauigkeiten von höchstens 78% [2] sind die bereits existierenden Werkzeuge bei weitem noch nicht zuverlässig genug für die Praxis. Gerade Akronyme stellen für viele Werkzeuge ein großes Problem dar [2]. Eine Möglichkeit dieses Problem anzugehen wäre die Nutzung bekannter N-Gramme [1], um nach häufig adjazent auftretenden Wörtern mit den entsprechenden Anfangsbuchstaben zu suchen.

4 Evaluation

Einige Orakel, also Listen von Abkürzungen mit manuell bestimmten Auflösungen, welche aus einem ebenfalls frei verfügbaren Quelltext stammen, stehen zur freien Verfügung. Diese können verwendet werden, um die Genauigkeit, Präzision, Ausbeute und das F1-Maß des entwickelten Werkzeuges zu bestimmen. Die gleichen Datensätze zu verwenden bietet außerdem den Vorteil der Vergleichbarkeit.

Außerdem sollte beachtet werden, dass die Evaluation alle Informationen enthält, welche für einen Vergleich mit anderen Werkzeugen, oder für die weitere Forschung, von Nutzen sein könnten. So sollten neben allgemeinen Qualitätsangaben auch nach einzelnen Abkürzungstypen oder Wissensquellen aufgeschlüsselte Ergebnisse dokumentiert werden.

Literatur

[1] Y. Lin, J.-B. Michel, E. A. Lieberman, J. Orwant, W. Brockman und S. Petrov, „Syntactic annotations for the google books NGram corpus,“ S. 6,

[2] C. D. Newman, M. J. Decker, R. S. Alsuhaibani, A. Peruma, D. Kaushik und E. Hill, „An Empirical Study of Abbreviations and Expansions in Software Artifacts," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ISSN: 2576-3148, Sep. 2019, S. 269–279. DOI: 10.1109/ICSME.2019.00040.

[3] Y. Jiang, H. Liu, J. Q. Zhu und L. Zhang, „Automatic and Accurate Expansion of Abbreviations in Parameters," *IEEE Transactions on Software Engineering*, S. 1–1, 2018, Conference Name: IEEE Transactions on Software Engineering, ISSN: 1939-3520. DOI: 10.1109/TSE.2018.2868762.

[4] E. Hill, D. Binkley, D. Lawrie, L. Pollock und K. Vijay-Shanker, „An empirical study of identifier splitting techniques," *Empirical Software Engineering*, Jg. 19, Nr. 6, S. 1754–1780, 1. Dez. 2014, ISSN: 1573-7616. DOI: 10.1007/s10664-013-9261-0. Adresse: https://doi.org/10.1007/s10664-013-9261-0 (besucht am 09.06.2020).

[5] A. Corazza, S. Di Martino und V. Maggio, „LINSEN: An efficient approach to split identifiers and expand abbreviations," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, ISSN: 1063-6773, Sep. 2012, S. 233–242. DOI: 10.1109/ICSM.2012.6405277.

[6] B. Dit, L. Guerrouj, D. Poshyvanyk und G. Antoniol, „Can Better Identifier Splitting Techniques Help Feature Location?" In *2011 IEEE 19th International Conference on Program Comprehension*, ISSN: 1092-8138, Juni 2011, S. 11–20. DOI: 10.1109/ICPC.2011.47.

[7] E. Enslen, E. Hill, L. Pollock und K. Vijay-Shanker, „Mining source code to automatically split identifiers for software analysis," in *2009 6th IEEE International Working Conference on Mining Software Repositories*, ISSN: 2160-1860, Mai 2009, S. 71–80. DOI: 10.1109/MSR.2009.5069482.

[8] Y. Singh, P. K. Bhatia und O. Sangwan, „Predicting software maintenance using fuzzy model," *ACM SIGSOFT Software Engineering Notes*, Jg. 34, Nr. 4, S. 1–6, 6. Juli 2009, ISSN: 0163-5948. DOI: 10.1145/1543405.1543425. Adresse: https://doi.org/10.1145/1543405.1543425 (besucht am 23.06.2020).

[9] E. Hill, Z. P. Fry, H. Boyd, G. Sridhara, Y. Novikova, L. Pollock und K. Vijay-Shanker, „AMAP: automatically mining abbreviation expansions in programs to enhance software maintenance tools," in *Proceedings of the 2008 international working conference on Mining software repositories*, Ser. MSR '08, Leipzig, Germany: Association for Computing Machinery, 10. Mai 2008, S. 79–88, ISBN: 978-1-60558-024-1. DOI: 10.1145/1370750.1370771. Adresse: https://doi.org/10.1145/1370750.1370771 (besucht am 09.06.2020).

[10] N. Anquetil und T. Lethbridge, „Assessing the relevance of identifier names in a legacy software system,“ in *CASCON*, 1998. DOI: 10.1145/783160.783164.