Entwurf und Aufbau einer semantischen Repräsentation von Quelltext

Dokumentenart:	Exposé für eine Masterarbeit
Autor:	Felix Eurich
Matrikel-Nr.:	1756347
Studiengang:	Informatik Master
Betreuer:	M.Sc. Tobias Hey
Datum:	28. August 2019

1 Motivation

Anforderungen im Kontext der Softwareentwicklung spezifizieren Eigenschaften und Funktionalitäten eines Softwaresystems. Sie liegen in der Regel in natürlicher Sprache vor und müssen von den Entwicklern umgesetzt werden. Das Bindeglied zwischen einer Anforderung und deren Entsprechung im Quelltext ist demnach der jeweilige Entwickler. Ohne explizite Dokumentation darüber aus welcher Anforderung ein Quelltextfragment entstanden ist, oder wieso es abgeändert wurde, geht diese Information schnell verloren. Infolgedessen müssen Entwickler bei Wartungs- oder Erweiterungsarbeiten zuerst nach den entsprechenden Stellen im Quelltext suchen, bevor sich der Aufwand für die Arbeit abschätzen lässt, bzw. die Arbeit durchgeführt werden kann. In einer empirischen Studie von Mäder und Egyed [ME15] betrug die Zeitersparnis beim Bearbeiten von Wartungsaufgaben, wenn Informationen über den Zusammenhang von Anforderungen und Quelltext vorlagen, im Durchschnitt 24%. Auch im Kontext der Wiederverwendbarkeit von Programmteilen und dem Programmverständnis sind diese Informationen wertvoll [ACC+02]. Trotz der vielseitigen Vorteile verzichten die meisten Softwareprojekte aufgrund des hohen Aufwands auf eine explizite Dokumentation der Zusammenhänge.

Das Forschungsprojekt INDIRECT (Intent-driven Requirements-to-Code Traceability) setzt an diesem Punkt an und will die Zusammenhänge zwischen Anforderungen und Quelltext automatisiert erkennen und zur Verfügung stellen. Eine Herausforderung hierbei stellt die interne Struktur des Quelltextes dar. Änderungen oder Erweiterungen, die zur Umsetzung einer Anforderung notwendig waren, liegen nicht zwangsläufig nahe beieinander. Werden die Stellen im Quelltext individuell betrachtet, ist die Zuordnung zur Anforderung oft nicht trivial, da jeweils nur ein Teil der Informationen

vorliegt. Daher bietet es sich an, den Quelltext vorab in eine semantische Repräsentation zu überführen, die es erlaubt auf inhaltlich zusammenhängende Quelltextfragmente ganzheitlich zuzugreifen. Der Entwurf und die Umsetzung einer solchen Quelltextrepräsentation ist das Ziel meiner Arbeit.

2 Zielsetzung

Im Rahmen dieser Arbeit soll ein Verfahren entwickelt werden, das eine semantische Repräsentation von Quelltext aufbaut. Hierzu soll zunächst eine Quelltextrepräsentation entworfen werden, mit der die verschiedenen Intentionen und Zusammenhänge im Quelltext auf adäquate Weise dargestellt werden können. Anschließend soll ein Verfahren zur automatisierten Generierung dieser Repräsentation ausgearbeitet und umgesetzt werden. Grundlage hierfür bilden Analysen, die semantische Zusammenhänge im Quelltext identifizieren. Für den initialen Aufbau der Repräsentation soll die statische Quelltextstruktur analysiert werden. Darüber hinaus existieren viele weitere Ansätze, wie z.B. die Untersuchung der Versionsverwaltungsdaten oder der natürlichen Sprache im Quelltext, welche hilfreich im Bezug auf das Auffinden von semantischen Zusammenhängen sind. Infolgedessen soll das Verfahren flexibel um zusätzliche Analysen erweitert werden können. Für die Evaluation des Zusammenspiels mehrerer Analysen, soll darüber hinaus eine zweite Quelltextanalyse implementiert werden.

3 Herangehensweise

Für die Umsetzung soll die PARSE Architektur [WT15] verwendet werden. Abbildung 1 veranschaulicht den Verarbeitungsprozess an einem abstrakten Beispiel. Die Verarbeitung läuft von links nach rechts ab, die Farben markieren semantisch zusammengehörende Quelltextteile. Zu Beginn soll Quelltext in eine Graphrepräsentation überführt werden (siehe Abschnitt 3.1). Die Suche nach semantischen Zusammenhängen im Quelltextgraphen übernehmen anschließend unabhängig arbeitende Agenten. Erste Ansätze wie aus den Agenten die semantische Repräsentation aufgebaut werden kann, sind in Abschnitt 3.2 aufgeführt. Die angestrebten Quelltextanalysen werden in Abschnitt 3.3 näher erläutert.

3.1 Graphrepräsentation von Quelltext

Quelltext soll anhand des Datenflusses und seiner Abhängigkeitsstruktur in einen Graph überführt werden. Klassen, Methoden, Variablen, Kontrollstrukturen und andere Quelltextfragmente werden durch Knoten repräsentiert. Die Kanten des Graphen beschreiben den Zusammenhang der Knoten im Bezug auf den Quelltext. Mögliche Beziehungstypen sind bspw. Ver-

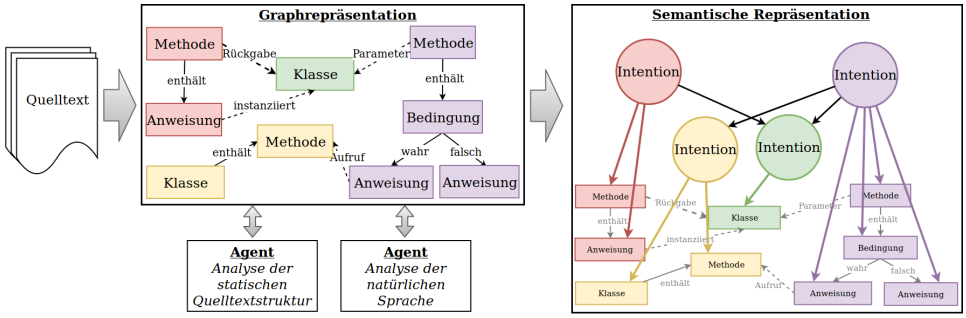

Abbildung 1: Der abstrakte Verarbeitungsprozess

weis, Modifikation, Aufruf, Erweiterung, Implementierung, Instanziierung und Rückgabe. Für die Extraktion der benötigten Informationen aus dem Quelltext, kann z.B. das Werkzeug JavaParser [1] oder MoDisco [2] eingesetzt werden.

3.2 Identifikation semantischer Zusammenhänge

Quelltextfragmente gelten im Rahmen dieser Arbeit als zusammenhängend, wenn sie der Erfüllung eines gemeinsamen Ziels oder Zwecks dienen. Im Gegensatz zu bestehenden Arbeiten auf dem Gebiet der Konzeptlokalisierung [SM11] [KDG07], beschränkt sich der hier angestrebte Ansatz nicht auf eine spezifische Quelltextinformation. Vielmehr soll er die Kombination verschiedener Informationen nutzen, um semantische Zusammenhänge zu identifizieren. Damit am Ende eine globale Lösung entsteht, die von allen Agenten profitiert und unabhängig von der Anzahl der aktiven Agenten ist, muss festgelegt werden, wie die Informationen aus den Analysen zusammenfließen. Eine Möglichkeit hierfür wäre, alle Agenten an einer gemeinsamen semantischen Repräsentation arbeiten zu lassen und gefundene Zusammenhänge mit Konfidenzen zu annotieren. Ein einzelner Agent wäre dann in der Lage, die von ihm erkannten Zusammenhänge mit den vorhandenen zu vergleichen und ggf. Korrekturen an der semantischen Repräsentation vorzunehmen. Alternativ wäre es auch denkbar, einen zusätzlichen Agenten zu entwickeln, der die Ergebnisse der verschiedenen Analysen zu einer Gesamtrepräsentation zusammenbaut.

3.3 Quelltextanalysen

Dieser Abschnitt geht auf die beiden Quelltextanalysen ein, welche in dieser Arbeit umgesetzt werden sollen.

[1] `https://javaparser.org/`
[2] `https://www.eclipse.org/MoDisco/`

Analyse der statischen Quelltextstrukturen Softwareprojekte besitzen in der Regel eine innere Struktur, die bei der Entwicklung berücksichtigt wurde. Einige intuitive Ansätze zur Interpretation dieser Strukturen sind im folgenden kurz skizziert.

Konzeptionell ähnliche Funktionalitäten sind in vielen Fällen nahe beieinander platziert, z.B. innerhalb einer Klasse oder gekapselt im selben Paket. Daher kann die Berücksichtigung der räumlichen Distanz zwischen Quelltextfragmenten helfen, eine Aussage über deren Zusammenhang zu treffen. Auch eine Analyse des Aufrufverhaltens von Methoden kann Hinweise darüber liefern, welche Stellen im Quelltext zusammenarbeiten. Wird eine private Methode bspw. exklusiv von einer anderen Methode aufgerufen, sind sie vermutlich einem gemeinsamen Konzept zuzuordnen. Auf die gleiche Weise kann auch der Datenfluss analysiert werden. Klassen, die viel miteinander interagieren, sonst aber kaum Abhängigkeiten aufweisen, arbeiten mit hoher Wahrscheinlichkeit zusammen an einer Funktionalität. Solche im Quelltext implizit vorhandenen Informationen, sollen bei dieser Analyse betrachtet werden.

Analyse der natürlichen Sprache Bei dieser Analyse sollen Kommentare, Klassen-, Methoden-, Variablennamen und weitere natürlichsprachige Elemente im Quelltext untersucht werden. Genau wie bei der zuvor beschriebenen Analyse, ist das Ziel die Identifikation semantisch zusammenhängender Quelltextstellen. Hierfür können Quelltextfragemente auf Wortmengen abgebildet werden, die dann als Grundlage für eine Ähnlichkeitsbewertung dienen. Bei der Vorverarbeitung der Worte kann auf bereits bestehende Werkzeuge von INDIRECT zurückgegriffen werden. Für die Bestimmung der Ähnlichkeit zweier Wortmengen, kommen unterschiedliche Ansätze auf Basis von z.B. Zeichenketten, Korpora oder Wissensquellen in Frage [GF13]. Eine vielversprechende Möglichkeit stellen Vektor basierte Ansätze, wie z.B. in [KDG07] verwendet, dar. Hierfür könnten die Wortmengen auf Vektoren abgebildet und der Winkel zwischen zwei Vektoren als Ähnlichkeitsmaß verwendet werden. Des Weiteren können bestehenden Funktionalitäten von INDIRECT, wie z.B. der *conceptualizer* ausgenutzt werden, um Entitäten, Aktionen und Konzepte zu erkennen und in den Vergleich mit einfließen zu lassen.

4 Evaluation

Im Rahmen der Evaluation soll geprüft werden, wie akkurat die semantische Beschreibung des Quelltextes ist. Hierzu bietet sich eine Nutzerstudie an, bei der die Teilnehmer die als zusammengehörend erkannten Quelltextfragmente hinsichtlich ihrer Sinnhaftigkeit bewerten. Alternativ dazu wäre es auch möglich, Teilnehmer die Quelltextfragmente gruppieren zu lassen und mit

dem Ergebnis des Systems zu vergleichen. Neben der Evaluation des Gesamtsystems wäre es zudem interessant, den Einfluss einzelner Agenten auf die Lösung zu untersuchen. Hierfür könnte die Repräsentation jeweils einmal mit und einmal ohne einen spezifischen Agenten generiert und am Ende verglichen werden.

Ausgangspunkt für die Evaluation ist ein Datensatz, der ausreichend dokumentiert ist. Die Arbeit von Zogan et al. [ZSMA17] liefert einen Überblick über die genutzten Datensätze von anderen Arbeiten in diesem Forschungsbereich. Eine erste Begutachtung der Datensätze bringt einige vielversprechende Kandidaten hervor. Beispiele hierfür sind iTrust [3], eine Anwendung zur Verwaltung von Krankenakten, oder JodaTime [4], eine Java Bibliothek für die Arbeit mit Datums- und Zeitangaben. Beide Projekte sind quelloffen und an JodaTime wird derzeit noch aktiv entwickelt. Einen weitereren vielversprechenden Datensatz stellt das Open-Source Projekt TASKANA[5] dar. Hierbei handelt es sich um eine Java basierte Anwendung für die Aufgabenverwaltung in Unternehmen. Der Vorteil dieses Datensatzes wäre, dass es sich um ein reales Projekt handelt, an dem aktuell gearbeitet wird. Somit wird die Gefahr einer Ergebnisverfälschung aufgrund von akademisch motivierten Daten minimiert. Neben den Quelltexten und der Historie ist auch das Ticketsystem von TASKANA einsehbar, sodass die ursprünglichen Anforderungen ermittelt werden können. Es wäre demzufolge ein Vergleich zwischen den Ergebnissen dieser Arbeit und der Musterlösung möglich.

[3] `https://sourceforge.net/projects/itrust/`
[4] `https://github.com/JodaOrg/joda-time`
[5] `https://github.com/Taskana/taskana`

Literatur

[ACC+02] Antoniol, G. ; Canfora, G. ; Casazza, G. ; Lucia, A. D. ; Merlo, E.: Recovering traceability links between code and documentation. In: *IEEE Transactions on Software Engineering* 28 (2002), Oct, Nr. 10, S. 970–983. http://dx.doi.org/10.1109/TSE.2002.1041053. – DOI 10.1109/TSE.2002.1041053. – ISSN 0098–5589

[GF13] Gomaa, Wael H. ; Fahmy, Aly A.: A survey of text similarity approaches. In: *International Journal of Computer Applications* 68 (2013), Nr. 13, S. 13–18

[KDG07] Kuhn, Adrian ; Ducasse, Stéphane ; Gîrba, Tudor: Semantic clustering: Identifying topics in source code. In: *Information and Software Technology* 49 (2007), Mar, Nr. 3, S. 230–243. http://dx.doi.org/10.1016/j.infsof.2006.10.017. – DOI 10.1016/j.infsof.2006.10.017. – ISSN 0950–5849

[ME15] Mäder, Patrick ; Egyed, Alexander: Do developers benefit from requirements traceability when evolving and maintaining a software system? In: *Empirical Software Engineering* 20 (2015), Apr, Nr. 2, S. 413–441. http://dx.doi.org/10.1007/s10664-014-9314-z. – DOI 10.1007/s10664–014–9314–z. – ISSN 1573–7616

[SM11] Scanniello, G. ; Marcus, A.: Clustering Support for Static Concept Location in Source Code. In: *2011 IEEE 19th International Conference on Program Comprehension*, 2011, S. 1–10

[WT15] Weigelt, S. ; Tichy, W. F.: Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* Bd. 2, 2015, S. 819–820

[ZSMA17] Zogaan, W. ; Sharma, P. ; Mirahkorli, M. ; Arnaoudova, V.: Datasets from Fifteen Years of Automated Requirements Traceability Research: Current State, Characteristics, and Quality. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, S. 110–121