Einstiegspunkte für Design und Codierung bei einer Wartungsaufgabe oder Fehlermeldung

Felix Fröhlich und Thorben Wiese

Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Department Informatik

Abstract. Bei der Entwicklung und Wartung von Software spielt die Identifikation des Nutzen von Programmabschnitten eine große Rolle, um entsprechende Funktionen einer Software zu ändern oder zu reparieren. Diese *Feature Locations* stellen einen Einstigespunkt in den Quelltext für Design- oder Code-Änderungen dar und können mithilfe verschiedener Technologien ermittelt werden. In dieser Seminararbeit stellen wir unterschiedliche Technologien und deren Verfahren vor und geben einen Überblick über geeignete Nutzungsfelder.

1 Einleitung

Die erste Quelle [1].
...
Ziel dieser Seminararbeit ist es, die verschiedenen Analyseverfahren und Technologien zur Erkennung von Features im Code zu beschreiben und zu vergleichen.

2 Begriffe

Für die Vorstellung der Analyseverfahren sollen zunächst einige Begriffe definiert und erklärt werden.

Feature

Ein Feature ist ein Software Artefakt, das eine spezifische Funktionalität implementiert [2]. Diese Funktionalität wird in natürlicher Sprache beschrieben und wird von einem Programmabschnitt wiedergespiegelt. Ein Feature besteht üblicherweise aus einen Namen, einer Bedeutung (Intension) und einer Erweiterung (Extension) [3].

Feature Location

Der Prozess der Feature Location beschreibt die Identifikation der Beziehung zwischen Features und deren Implementierung. Dabei liegt die Beschreibung des Features in natürlicher Sprache vor, die dann einem entsprechenden Codeabschnitt zugeordnet werden soll (Mapping) [1].

3 Analyseverfahren

In diesem Abschnitt sollen verschiedene Analyseverfahren vorgestellt werden, die das Finden von Feature Locations ermöglichen.

Program Dependence Analysis (PDA)

Die Analyse eines Programms auf interne Abhängigkeiten wird Abhängigkeitsanalyse (engl. Dependence Analysis) genannt. Sie umfasst in der Regel Kontrollflussabhängigkeiten und Datenabhängigkeiten innerhalb eines Programmes. Diese werden zur Übersetzungszeit mithilfe des Compilers festgestellt. Ziel dieser Analyse ist es zum Beispiel, zu überprüfen, ob ein Programm parallelisiert ausgeführt werden kann. Für die Analyse von Features im Quelltext ist diese Methode hilfreich, da durch sowohl Kontrollflussabhängigkeiten, als auch Datenabhängigkeiten auch semantische Verknüpfungen verschiedener Abschnitte des Codes hergestellt werden können [4].

Trace Analysis

Bei der Trace Analysis findet eine Auswertung der Traces (Spuren) statt. Das Erstellen von Spuren ist eine Form des Loggings bei der das Verhalten eines Programms aufgezeichnet wird. Traces unterscheiden sich vom Logging dadurch, dass sie eher von Entwicklern als Administratoren verwendet werden und deutlich weniger abstrahieren sondern eine Menge Ausgabedaten auf niedrigen Ausführungsebenen liefern. Meistens werden mehrere Spuren verwendet, wobei es eine Spur sämtliche wichtigen Ereignisse aufzeichnet und die anderen diese komplexe, quantitativ auffallende Spur nach bestimmten Kriterien filtern.

Latent Semantic Indexing (LSI)

Das Verfahren des Latent Semantix Indexing wird zum Indexieren von Abschnitten und Mustern eines Textes mithilfe von Singulärwertzerlegung der Ausdruck-Dokument-Matrix verwendet. Die Ausdruck-Dokument-Matrix (engl. Document-Term-Matrix) stellt die Frequenz der Ausdrücke innerhalb eines Dokumentes oder Textes dar. Die Singulärwertzerlegung dieser Matrix ist die Darstellung der ursprünglichen Matrix als Produkt dreier spezieller Matrizen, deren Singulärwerte auf bestimmte Eigenschaften der Matrix schließen lassen. LSI geht davon aus, dass Wörter, die in einem bestimmten Kontext verwendet werden, eine ähnliche Bedeutung haben. Dieses Verfahren lässt sich von natürlicher Sprache auch auf Quellcode übertragen, insbesondere auf Kommentare innerhalb des Codes [5].

3.1 Statische Feature Location Technologien

In diesem Abschnitt sollen statische Technologien zum Finden von Features im Quelltext vorgestellt werden.

Statische Analyse

Die Analyse von Quelltext zu einem Zeitpunkt, bei dem das Programm kompiliert wird und noch nicht ausgeführt wurde, wird statische Analyse genannt. Hierbei werden mithilfe von zum Beispeil Datenfluss-Analyse und Kontrollgraphen alle Abhängigkeiten und Funktionsaufrufe innerhalb des Codes analysiert und es können unter anderem Fehler wie zum Beispiel Race-Conditions oder Buffer-Overflows identifiziert werden. Dieser Prozess wird häufig von automatisierten Tools durchgeführt [6].

Technologie Beispiele

Robillard et al. [35]
Shao et al. [40]

3.2 Dynamische Feature Location Technologien

Dynamische Analyse

Allgemein dynamische Analyse Hauptmerkmal der dynamischen Analyse ist ihre Durchführung während der Laufzeit eines Programms. Das bedeutet die Ausführung des Programms ist zwingend erforderlich und während der Laufzeit werden Informationen gesammelt um wahrscheinliche Features zu lokalisieren. Dabei können ausschließlich funktionale Features gefunden werden, da die dynamische Analysetechnik an eine eingeschränkte Sicht auf das Programm gebunden ist, die der Sicht des Anwenders entspricht.
Ähnlich wie bei der statischen Analyse unterscheidet man zwischen Plain-Output-Techniken, bei denen mögliche, zu einem Feature gehörende, Quellcodebestandteile ungeordnet zurückgeliefert werden und Guided-Output-Techniken, bei denen die ausgegebenen Komponenten durch zusätzliche Informationen in Zusammenhang gebracht werden und ihr Kontext genauer erläutert wird.

Technologie Beispiel - Plain Output

In ihrer wissenschaftlichen Arbeit Location Program Features by using Execution Slices benutzen die Autoren W. Eric Wong, Swapna S. Gokhale, Joseph R. Horgan und Kishor S.Trivedi ein dynamisches Plain-Output-Verfahren zur Analyse von Code [7]. Sie verwenden dabei Execution Slicing im Gegensatz zum Static Slicing. Beide Techniken werden unter Program Slicing zusammengefasst. Slices können verschiedene Bestandteile eines Programms sein wie z.B. Code-Blöcke oder Kontrollstrukturen. Auch sogenannte c- und p-uses können Bestandteile einer Slice sein.
C- und p - uses beschreiben beide den Pfad einer Variable im Programmfluss ohne, dass die Variable verändert wird, wobei c-uses Variablen zur Berechnung dienen und p-uses-Variablen für eine Prädikat oder eine Entscheidung.
Faktoren, wie Heuristiken, Test Cases, Detailgenauigkeit und Tool-Unterstützung

sind ausschlaggebend für den Erfolg einer dynamischen Analysetechnik.
Besonderem Augenmerk gilt dabei der Menge der Tests, die dem Programm als Eingabe übergeben werden. Die Autoren heben hervor wie wichtig die richtigen Testmengen sind und unterscheiden zwischen aufrufenden (invoking) tests, die sich explizit auf ein bestimmtes Feature und seine Funktionalität fokussieren und ausschließenden Tests, die alle anderen Test-Sets enthalten, die nicht auf das Feature fokussiert sind worunter aber auch die zu finden sind, welche das gesuchte Feature und weitere Komponenten umfassen d.h. Tests bei denen das Feature nur ein Bestandteil von vielen ist.
Eine für die Autoren erfolgreiche Herangehensweise war zuerst die Menge aller aufrufenden Tests bilden von denen einige auch die Funktionalität anderer Features testen. Dann bildet man die Menge aller ausschließenden Tests, wobei diese jedoch auch Abschnitte des Quellcodes enthalten können, die vom gesuchten Feature benutzt werden. Die Differenz der beiden ist die Teilmenge, die zur Lokalisierung des gesuchten Features genutzt wird.
Die aufrufenden Tests sollen sich dabei sehr stark unterscheiden, während die ausschließenden Tests sehr ähnlich sein sollen.
Ohne Tool-Unterstützung wäre es sehr Aufwending Slices, die wie erwähnt aus Code-Blöcken, Kontrollstrukturen, c- oder p-uses bestehen, für jeden einzelnen Testfall zu sammeln. Die Autoren verwendeten deshalb xVue. Dieses Programm zählt für jeden Test wie oft die angegebenen execution slices verwendet wurden und ordnet dadurch jedem Test die dazugehörigen Slices zu. Weiterhin teilt es die Tests, sofern dies noch nicht geschehen ist, in aufrufende und ausschließenden Tests.
Die Autoren erläuterten das Verfahren an einer Fallstudie mit der wissenschaftlichen Software SHARPE, die aus 35 412 Zeilen C-Code besteht. Sie fokussierten sich auf 5 Features und sammelten aufrufende und ausschließende Tests mit dem heuristischen Verfahren, dass bereits beschrieben wurde.
xVue lieferte im Bezug auf Code-Blöcke und Kontrollstrukturen als Slices gute Ergebnisse, was die Lokalisierung der Features im Code betraf. Die dann zurückgelieferten Ergebnisse wurden von Experten bezüglich SHARPE verifiziert. Die Zuordnung von p- und c-uses ließ sich nicht vollständig verifizieren, da die Software derart komplex ist, das selbst die Experten nicht sämtliche Ergebnisse nachvollziehen konnten.
Nach der Auswertung der Studie kamen die Autoren zu dem Schluss, dass die, auf execution slices basierende Methode als ein Einstiegspunkt für Feature Location genutzt werden kann und stellten ihre Eignung dieser Analysetechnik besonders für komplexe System fest in denen sich implementierte Features über mehrere Module erstrecken. Dennoch steht und fällt das Verfahren mit den Test-Mengen mit denen das Programm gestartet wird und die Autoren stellten die Vermutung auf, dass auch in ihrer Fallstudie nicht der gesamte relevante Code gefunden wurde bezüglich eines Features. Auch wenn die Verifizierung der c- und p-uses nicht vollständig durchgeführt werden konnte, waren die SHARPE-Experten doch überrascht über die Detailgenauigkeit mit der xVue Features lokalisiert hatte.

Die Autoren hoben auch die einfache Anwendung des Verfahrens hervor, denn die Berechnung der execution slices war ja durch das Tool vereinfacht und automatisiert worden. Einzig allein die aufrufenden und ausschließenden Tests mussten durch den Anwender definiert werden. Die Identifizierung von c- und p-uses bot den weiteren Vorteil, dass diese Information zur Code Coverage genutzt werden konnte. Das heuristische Verfahren zur Ermittlung der Test-Mengen, welches sie genutzt hatten bewerteten sie als nützlich für den untersuchten Code, doch generell müssen die Heuristiken jeweils an die Art der Implementation angepasst werden.
Die Autoren kommen zu dem Schluss, dass sich ihr Verfahren durchaus für die Lokalisierung von relevanten Code bezüglich eines Features eignet. Es ist besonders nützlich um in großen Code-Bereichen die relevanten Zeilen zu finden, die ein solches Feature implementieren. Dennoch hat das Verfahren die typische Schwäche der dynamischen Analyse: es ist schwierig alle Komponenten im Code, die einem Feature zuzuordnen sind, zu entdecken und die Wahrscheinlichkeit ist groß, dass einige Komponenten nicht entdeckt werden. Der große Vorteil der Methode besteht in der Tool-Unterstützung, wodurch sie sehr einfach durchzuführen ist.

Technologie Beispiel - Guided Output

In dem Paper Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace von Dapeng Liu, Andrian Marcus, Denys Poshyvanyk und Vaclav Rajlich werden die Techniken LSI und Execution-Trace-Analyse zusammen genutzt.

3.3 Textuelle Feature Location Technologien

Allgemein textuelle Analyse, Tools beschreiben, Beispiele

4 Vergleich

Die statische Analyse untersucht den Quelltext vor der Ausführung und hat somit Zugriff auf sämtliche Bestandteile des Programms. Dadurch können Features jeglicher Art gefunden werden, seien sie nun funktional oder nicht nichtfunktional. Eine mehrmalige Wiederholung der statischen Analyse eines Programms ist idempotent und liefert immer wieder das gleiche Ergebnis.
Während also die statische Analyse einen größeren Feature-Raum untersuchen kann bezüglich ihrer Quantität und Diversität, ist die dynamische Analyse an das Programm zu Ausführung gebunden und damit auch an spezifische Eingabeparameter, die den Verlauf bestimmen. Somit kann das Auffinden von Features stark eingeschränkt sein und selbst ein gefundenes Feature lässt sich nur schwer generalisieren [20].
Betrachtet man Verfahren zu statischen und dynamischen Analyse als binären

Klassifikator, dann liefern statische Verfahren mit größerer Wahrscheinlichkeit eine Menge false-positives. Diese resultieren nicht nur aus den quantitativ höheren Featureraum sondern auch aus der Feststellung, dass die Wahrscheinlichkeit eines gefundenen Features nie genau bestimmt sondern nur approximiert werden kann. Statische Verfahren finden also mit größerer Wahrscheinlichkeit Features, die eigentlich gar keine sind, die sogenannte Überapproximiation.
Dynamische Verfahren dagegen, auf einen kleineren Feature-Raum beschränkt, liefern dagegen mehr false-negatives. Existierende Features werden in diesem Fall nicht als solche erkannt und falsch klassifiziert was man als Unterapproximation bezeichnet.
Worin sich dynamische und statische Verfahren weiterhin unterscheiden ist die Vorarbeit, die bezüglich der Eingabedaten geleistet werden muss. Während statische Verfahren, wie bereits erwähnt, bei mehrmaliger Durchführung das gleiche Ergebnis liefern sollten, sind dynamische Verfahren nur so gut in der Identifizierung von Features im Quellcode wie die Eingabeparameter mit denen das Programm gestartet wird. Diese müssen mit verschiedenen Heuristiken genau evaluiert werden, da eine ungenaue Testmenge in völlig unbrauchbare Ausgaben münden kann. Es ist in dieser Hinsicht mehr Aufwand nötig als bei statischen Analyse und der Anwender, der die Testfälle auswählt, muss qualifiziert und erfahren sein.

5 Fazit

Für welchen Zweck welches Analyseverfahren und welche Technologie Ergebnis wahrscheinlich: Alles gar nicht so schlecht, je nach Bedarf muss eine Technologie ausgewählt werden oder eventuell mit einer anderen kombiniert werden.

Literaturverzeichnis

[1] Rubin J, Chechik M. A Survey of Feature Location Techniques. University of Toronto, Department of Computer Science; 2013.

[2] IEEE. Std. 829;. Available from: https://standards.ieee.org/findstds/standard/829-2008.html.

[3] Chen K, Rajlich V. Case Study of Feature Location Using Dependence Graph. Wayne State University Department of Computer Science;. Available from: http://www.cs.wayne.edu/~severe/publications/Chen.IWPC.2000.FeatureLocation.pdf.

[4] Stafford J. A Formal, Language-Independent, and Compositional Approach to Interprocedural Control Dependence Analysis. University of Colorado;. Available from: ftp://ftp.sei.cmu.edu/pub/jas/Stafford-thesis.pdf.

[5] Deerwester S, Dumais S, Furnas G, Landauer T, Harshman R. Indexing by Latent Semantic Analysis. University of Colorado;. Available from: http://lsa.colorado.edu/papers/JASIS.lsi.90.pdf.

[6] Open Web Application Security Project (OWASP). Static Code Analysis. Open Web Application Security Project (OWASP);. Available from: https://www.owasp.org/index.php/Static_Code_Analysis.

[7] Wong WE, Gokhale SS, und Kishor S Trivedi JRH. Location Program Features by using Execution Slices. University of California;. Available from: https://www.researchgate.net/profile/Kishor_Trivedi2/publication/3795664_Locating_Program_Features_by_using_Execution_Slices/links/0fcfd50dc6cac8f70e000000/Locating-Program-Features-by-using-Execution-Slices.pdf.