Softwarearchitektur und Agile Entwicklung
Experimentaufbau zur toolgestützten Architekturrekonstruktion

Leon Fausten

Hochschule für Angewandte Wissenschaften Hamburg, Deutschland
Email: leon.fausten@haw-hamburg.de

I. Einleitung

Um eine Anwendung am leben zu halten, muss diese regelmäßig gewartet werden. Dazu gehört die Korrektur von Fehlern und die Implementierung von neuen Funktionalitäten. Um Änderungen erfolgreich und effizient durchführen zu können, ist es erforderlich zu wissen, wo angesetzt werden muss. Dafür muss die Architektur der Anwendung bekannt sein.[1] Der Wartungsprozess ist der zeitaufwendigste und teuerste Abschnitt der Softwareentwicklung. Die meiste Zeit geht dabei verloren, ein Verständnis für die Software zu entwickeln. [2] Wenn die Erweiterungen und Wartungen durch Personen durchgeführt werden, die lange nicht, oder sogar noch nie an dem Projekt gearbeitet haben, müssen diese sich vorerst in das Projekt einarbeiten. Die Einarbeitung in ein großes Projekt kann sehr komplex und zeitaufwendig sein. Die zur Verfügung stehenden Unterlagen, um ein Projekt kennen zu lernen, sind in den meisten Fällen sehr unterschiedlich. Dies hängt sehr von der vorhandenen Dokumentation und der Art der Umsetzung ab. In manchen Projekten ist eine genaue schriftliche Dokumentation vorgegeben, andere verzichten vollständig auf diese und setzen nur auf rein mündliche Absprachen. Je nach Anwendungsfall und Adressat, werden unterschiedliche Informationen über die Architektur benötigt, dadurch wird eine allumfassende Dokumentation sehr groß und unübersichtlich. Dies ist in einem Dokument schwer umzusetzen, da sich dort, für alle Personen, alle relevanten Informationen befinden müssen. Bei solchen großen Dokumenten entsteht die Schwierigkeit, in der Masse der Informationen, die von den Einzelpersonen benötigten Informationen gezielt in akzeptabler Zeit zu finden. Je größer und ausführlicher die Dokumentation ist, desto schwieriger wird es die gesuchten Informationen zu finden.[3] Der Umfang, für viele unterschiedliche angepasste Dokumentationen, ist schwer abzustecken, außerdem müssen diese jeweils einzeln gepflegt werden. Die manuelle Pflege im Allgemeinen kostet viel Zeit und wird dadurch schnell vernachlässigt.[4] Die benötigten Informationen sollten mithilfe einer einfachen, schnell durchführbaren Suche gefunden werden können.[5]

Im Grundseminar[1] wurde beschrieben, welche Probleme die Agile Entwicklung für die Entwicklung der Architektur mitbringt. Die Architektur wird in vielen Fällen nicht mehr explizit geplant, sondern entsteht als ein Nebenprodukt der Implementation. Dadurch ist diese häufig nicht gut strukturiert und unübersichtlich. Dies fällt besonders bei Projekten mit vielen unterschiedlichen Entwicklern auf.

Im Grundprojekt[2] wurden die Möglichkeiten des Monitorings und der Visualisierung der Architektur diskutiert. Um stets die aktuelle Architektur monitoren zu können, ist eine automatische Architekturrekonstruktion notwendig. Die Ausarbeitung des Grundprojektes beinhaltet eine Vorstellung und kurze Analyse von Tools und Bibliotheken, die dies bewerkstelligen können. Diese können neben der Berechnung von Metriken zu Abhängigkeiten und ähnlichem, teilweise UML-Diagramme erzeugen, Aussagen über die Code Qualität machen oder bekannte Programmiermuster (Pattern) erkennen.

Eine alleinige manuelle schriftliche Dokumentation der Architektur ist nur in den wenigsten Fällen praktikabel. Die Erstellung und Wartung ist sehr aufwendig. In den meisten Fällen ist diese deshalb entweder nicht vorhanden oder veraltet und entspricht dadurch nicht mehr der realen Umsetzung. Schnelle Veränderungen in der Software können in den meisten Fällen nicht durch die Dokumentation abgebildet werden, weil den Entwicklern die entsprechende Zeit dazu fehlt.[6] Eine grafische Visualisierung der Anwendung ermöglicht ein besseres und schnelleres Verständnis des Softwaredesigns und der Funktionalitäten zu erhalten.[2] Interaktive Graphiken, mit denen sich manuell durch die Architektur navigiert werden kann sind hilfreich. In den vorherigen Arbeiten wurden Tools, die dies bewerkstelligen können hauptsächlich beschrieben, ohne dass dies praktisch evaluiert wurden.

Eine Evaluierung, ob eine toolgestützte Dokumentation durch eine automatische Rekonstruktion auch in der Realität hilfreich ist, steht noch aus. Es ist weiterhin offen, ob zusätzliche Tools einen merkbaren Vorteil bei der Entwicklung, im Vergleich zu Standard Entwicklungsumgebungen bringen. Innerhalb dieser Arbeit werden deshalb zu Anfang in Kapitel II bereits vorhandene Evaluierungen vorgestellt und bewertet. Anschließend wird in Kapitel III ein möglicher Experimentaufbau beschrieben. Dieser soll sowohl das Vorgehen, bestehend aus der Aufgabenstellung, der Vorstellung der zur Verfügung stehenden Toolchain, sowie die Art der Auswertung beinhalten. Nach der Vorstellung des Versuchsaufbaus, wird der Aufbau in einem eingeschränkten Selbstversuch in Kapitel IV durchgeführt und die Ergebnisse dessen vorgestellt. Am Ende dieser Arbeit (Kapitel V) werden der Experimentaufbau und die Ergebnisse des Versuches bewertet und, wenn möglich Änderungsvorschläge erarbeitet, welche in den Aufbau einfließen sollen. Mögliche weitere Aspekte für einen sinnvollen Einsatz der Tools werden genannt.

[1] http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master14-15-gsm/fausten/bericht.pdf

[2] http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master2015-proj/fausten.pdf

II. Ähnliche Arbeiten

Viele der vorhandenen Anwendungen wurden mit einem wissenschaftlichem Hintergrund entwickelt und wurden deshalb zum großen Teil nur innerhalb eines Experimentes evaluiert. Ergebnisse dieser Evaluationen werden in diesem Abschnitt gesammelt und zusammengefasst.

Kobayashi und andere [7] haben ihre entwickelte Technik *SArF Map* mithilfe mehrerer Fallstudien überprüft. Ihre Technik beinhaltet die Visualisierung von Funktionalitäten und Schichten mithilfe einer stadtähnlichen Darstellung. Die Darstellung wird mithilfe eines zuvor entwickelten Clustering-Algorithmus zur Abhängigkeitsanalyse durchgeführt. Die entstehenden Graphen sollen ebenfalls für Nicht-Entwickler verständlich sein und als gutes Mittel zur Kommunikation dienen können. Klassen werden als Häuser dargestellt und Verbindungen zwischen ihnen als Straßen. Straßenblöcke stellen Funktionalitäten dar. Die Stadtdarstellung beschreiben sie im Allgemeinen sehr positiv, da sie sehr intuitiv Verständlich und für viele unterschiedliche Metriken geeignet ist. Außerdem können mehrere Metriken zur gleichen Zeit betrachtet werden, ohne dafür zwischen Darstellungen wechseln zu müssen. Um den Nutzen zu überprüfen, wurden fünf Fallstudien mit Open Source und kommerziellen Java Anwendungen durchgeführt. In einem Beispiel konnte erfolgreich erkannt werden, dass ein Package weit verstreut, an vielen verschiedenen Stellen auftaucht, dies bedeutet, dass dieses Package weiter aufgeteilt werden sollte. Durch die Möglichkeit auf eine vorhandene Dokumentation zurückgreifen zu können oder sogar die Entwickler fragen zu können, konnte sichergestellt werden, dass die Anwendungen korrekt analysiert wurden. Die Fallstudien haben nur untersucht, ob die durchgeführten Analysen korrekte Ergebnisse liefern. Wie sie allerdings ebenfalls festgestellt haben, müssen weitere Studien zeigen, wie hilfreich diese Visualisierungen in der Praxis sind. Schwachstellen im Algorithmus, wie z.B. wenn sich mehrere Features überlappen, wurden festgestellt. Außerdem ist die Darstellung abhängig von der Qualität des gelieferten Abhängigkeitsgraphen. Der Graph kann zuvor berechnet oder mitgeliefert werden. Bei der Verwendung von Techniken, wie Reflection oder Dependency Injection ist es möglich, dass unvollständige Stadtdarstellungen generiert werden.[7]

Caserta und andere [2] haben eine sehr ausführliche Studie zur Visualisierung von Anwendungen durchgeführt und dabei zahlreiche Darstellungsmöglichkeiten und vorhandene Anwendungen betrachtet. Ziel ihrer Untersuchung war es herauszufinden, wie hilfreich diese im praktischen Einsatz sind. Sie haben festgestellt, dass man die Analysen und die daraus entstehenden Visualisierungen in drei Kategorien (statisch, dynamisch und evolutionär) unterteilen kann. Bei statischen Analysen wird nur der Source Code betrachtet, dieser ist bei jeder Art von Ausführung gültig. Dynamischen Analysen betrachten einzelne, spezielle Ausführungsstränge zur Laufzeit. Die evolutionären Analysen fügen eine Zeitkomponente zu der statischen Analyse hinzu. Dadurch ist erkennbar, wie eine Software weiterentwickelt wurde. In ihrer Untersuchung haben sie allerdings nur die statischen Analysen und die Evolution dieser betrachtet.
3D Darstellungen haben im Vergleich zu 2D Darstellungen den Vorteil, dass in ihnen mehr Informationen gleichzeitig präsentiert werden können. Dies erhöht aber die Gefahr der Informationsüberladung. Dadurch wird die Darstellung unter Umständen zu unübersichtlich. Die Visualisierung einer Klassenstruktur, inklusive der Darstellung von Abhängigkeiten (Methoden Aufrufe und ähnliches) stellt z.B. Informationen dar, für dessen Erkennung der Quellcode ansonsten vollständig durchgegangen werden müsste. Wenn allerdings zu viele Klassen auf einmal präsentiert werden, wird die Darstellung schnell sehr unübersichtlich.
Ein Baum Model (wie in Abbildung 1a) eignet sich gut um Pakete, Klassen und Methoden Strukturen darzustellen, werden aber bei großen komplexen Strukturen ebenfalls schnell sehr unübersichtlich. Ein Treemap Model kann Hierarchien ebenfalls gut widerspiegeln, nutzt den verfügbaren Platz, im Vergleich zu normalen Baum Strukturen, erheblich effektiver aus. Abbildung 1b zeigt eine äquivalente Treemap Struktur, im Vergleich zur Baumstruktur in Abbildung 1a. Die Darstellung wird erheblich kompakter, die Strukturen sind aber nur noch indirekt ersichtlich. In der zirkuläre Treemap, in Abbildung 1c, sind die Strukturen mithilfe von Kreisen verdeutlicht. Die Hierarchien kommen hier wieder deutlicher, dadurch dass übergeordnete Knoten ihre Kinder auch visuell beinhalten, zum Vorschein Der Platz wird aber nicht mehr so effizient, wie bei der Treemap, ausgenutzt. Abbildung 1d zeigt eine Sunburst Darstellung. Diese ist von innen nach außen aufgebaut und stellt dadurch die Strukturen ebenfalls vollständig und einfacher verständlich, als eine Treemap dar. Experimente [8] haben allerdings gezeigt, dass das Finden von Elementen in Treemaps und in Sunbursts keine relevanten Unterschiede bei der Geschwindigkeit aufweist. Die Sunburst Darstellung ist allerdings schwieriger zu verstehen. Die Darstellung als Stadt haben Caserta und andere ebenfalls untersucht. Diese Darstellungsart ist intuitiv verständlich. Der Betrachter hat außerdem die Möglichkeit in die Stadt rein zu zoomen um Details genauer betrachten zu können. Da nicht alle Eigenschaften auf die Stadtmetapher abgebildet werden können, müssen Informationen teilweise vereinfacht werden (z.B. Metriken in Bereiche zusammenfassen). Es wurde allerdings festgestellt, dass dies sogar das einfachere Verständnis fördert.
Baumartige Graphen eignen sich gut um Beziehungen zu visualisieren. Beziehungen können z.B. Vererbungen oder Methodenaufrufe sein. Die entstehenden Graphen können dabei sehr unübersichtlich, durch zahlreiche sich überlappende Beziehungen, werden. Alternativ werden Beziehungen häufig ebenfalls gerne mithilfe einer quadratischen Matrix dargestellt. Die Anzahl der Beziehungen zwischen einer Entität einer Reihe und einer Entität einer Zeile werden dort notiert. Die Anzahl repräsentiert die Anzahl der Verbindungen, die innerhalb eines Graphen zwischen zwei Punkten notwendig wären. UML-Klassendiagramm sind graph-basierte Darstellungen. Ihr Ziel ist es Vererbungen, Generalisierungen, Assoziationen, Aggregationen und Kompositionen darzustellen. Damit die Komplexität großer UML-Diagramme verringert wird, wird vorgeschlagen, die sich überlappenden Verbindungen zu reduzieren. Dies kann z.B. durch die Vereinigung von Verbindungen, mit identischem Ziel oder einem festgelegtem orthogonalem Layout geschehen. Für eine bessere Übersicht haben manche Tools Funktionalitäten, um z.B. nur Methodenaufrufe einer bestimmten Klasse darzustellen umgesetzt.
Statische Software Metriken sind berechnete Zahlenwerte. Sie können Aussagen über System Stabilität, Ressourcennutzung und Design Komplexität machen. Metriken zur Kopplung können z.B. im zuvor vorgestellten Stadt Design angezeigt

werden. Je stärker die Kopplung ist, desto näher sind zwei Gebäude. Diese Darstellung ist sehr intuitiv verständlich, kann aber bei Veränderungen verwirren, da es vorkommen kann dass sich das allgemeine Layout schnell ändert. Metriken, wie z.B. die Anzahl der Methoden oder der Klassengröße allgemein können an die Häuserhöhe oder Breite gebunden werden. Um die visuelle Komplexität zu verringern, wird empfohlen die unterschiedlichen Höhen auf maximal fünf zu begrenzen. Dies kann z.B. durch das Aufteilen in Bereiche erreicht werden.[9][10][2]

Panas und andere [11] nennen einen wichtigen Punkt bei individuellen Visualisierungen je nach Stakeholdertyp. Durch die unterschiedlichen Darstellungen wird die Kommunikation zwischen Gruppen schwierig, da alle nur die für sie gedachten Darstellungen kennen. Aus diesem Grund empfehlen sie eine allgemein gültige Visualisierung, die alle notwendigen Informationen enthält.

Im Allgemeinen ist zu erkennen, dass alle festgestellt haben, dass Visualisierungen helfen, es existiert aber eine große Gefahr, dass Darstellungen schnell zu groß und Komplex werden. Dies betrifft vor allem den Einsatz innerhalb großer Software Projekte. Für diese ist solch eine automatisierte Dokumentation und Visualisierung allerdings besonders wichtig, da diese nicht einfach ohne Hilfe überblickt werden können. Außerdem sind Anwendungen entsprechender Größe im Normalfall sehr lange in Verwendung und müssen dadurch dauerhaft gewartet und erweitert werden. Aus diesem Grund sind interaktive Darstellungen, die anfänglich eine stark vereinfachte Struktur darstellen und bei Bedarf Details schrittweise einblenden können sehr wichtig.

III. Experimentaufbau

Zur Evaluierung der Tools wird ein Versuchsaufbau erarbeitet. Dieser wird anschließend in einem Selbstversuch evaluiert. Zu dem Aufbau gehört die Vorstellung unterschiedlicher Ausgangsszenarien, die Auswahl der während des Experiments zur Verfügung stehenden Tools, sowie die Art der Evaluierung im Anschluss.

A. Szenarien

Die Aufgabe in jedem Szenario ist die gleiche. Einem Entwickler wird die Aufgabe gegeben an einer bereits vorhandenen Anwendung zu arbeiten. Es sollen entweder Fehler behoben oder eine neue Funktionalität entwickelt werden. Die Entwickler müssen die Stellen im Code finden, an denen sie für Änderungen ansetzten müssen.

Im Folgenden werden drei unterschiedliche Ausgangssituationen beschrieben, die ebenfalls unter realen Bedingungen denkbar sind.

Szenario A: Der Entwickler kennt das Projekt und hat bereits früher an diesem gearbeitet. Das Projekt wurde seit mindestens einem halben Jahr nicht weiter entwickelt. Der Entwickler kennt die Strukturen dadurch nur noch grob. Er kann sich aber eventuell schneller wieder an den Aufbau und die Gründe dafür erinnern, nachdem er sich wieder etwas eingearbeitet hat.

Szenario B: Die Erweiterung wird von einem Entwickler durchgeführt, für den das Projekt neu ist. Für den Entwickler ist es möglich ursprüngliche am Projekt beteiligte Entwickler zu kontaktieren um nach Unterstützung zu fragen. Diese können ihm Tipps geben, um die Aufgabe zu lösen und die Struktur der Anwendung vorstellen. Der Erfolg dieses Szenarios ist sehr von der Unterstützungsbereitschaft der ursprünglichen Entwickler abhängig. Wenn die Testperson an einem Open Source Projekt mit einer großer Community entwickelt, ist es wahrscheinlicher, zeitnah die gewünschten Antworten zu bekommen, als bei einem Projekt mit einem einzelnen schwer erreichbaren Entwickler. Für die Testperson ist es am besten, wenn es sich um ein Projekt innerhalb des gleichen Arbeitsumfeldes, bei dem er die ursprünglichen Entwickler persönlich auf kurzem Weg ansprechen kann, handelt.

Szenario C: Die Erweiterung wird von einem Entwickler durchgeführt, für den das Projekt neu ist. Im Unterschied zu Szenario B hat dieser nicht die Möglichkeit mit einem früheren Entwickler Kontakt aufzunehmen, um diesen um Unterstützung zu beten. Die ursprünglichen Entwickler sind entweder nicht auffindbar oder wollen, bzw. können nicht helfen. Dadurch ist die Person vollständig auf sich alleine gestellt und muss sich alles selbst erarbeiten. Fragen zu Beweggründen oder zu komplexen Strukturen muss er sich eigenständig erschließen.

In allen Szenarien wird die vollständige, lauffähige Anwendung zur Verfügung gestellt. Bereits existierende Dokumentationen stehen in allen Szenarien zur Verfügung. In den Szenarien kann sich sowohl an diesen orientiert werden, sowie auch an der Dokumentation innerhalb des Quellcodes. Falls die Dokumentation der Anwendung sehr ausführlich ist, kann diese im Rahmen des Experiments gekürzt werden oder den Teilnehmern vorenthalten werden. Dadurch kann zusätzlich geprüft werden, ob dieser der realen Umsetzung entspricht. Dies hat den Grund, dass in vielen Projekten keine oder eine nicht aktuelle Dokumentation existiert. Oft ist die Dokumentation außerdem sehr kurz gehalten, mit nur minimal nützlichen Informationen für Weiterentwicklung und Wartung.

Jedes Szenario wird dabei mit zwei unterschiedlichen Aufbauten durchgeführt. Im ersten Aufbau stehen nur die Infos und Möglichkeiten, wie in der Szenariobeschreibung, zur Verfügung. Im zweiten Aufbau wird eine zuvor festgelegte Toolchain zur Verfügung gestellt. Diese soll es erleichtern, die notwendigen Ansatzpunkte für die erforderliche Änderungen zu finden. Die Probanden bekommen eine Einweisung in die Toolchain.

Geplant ist, nur Szenario C in einer praktischen Evaluation zu überprüfen, da diese den extrem Fall, ohne jegliche Hilfe durch Dritte darstellt. Dazu muss zuvor allerdings sichergestellt werden, ob diese Szenarien sich wirklich in der Art ähneln und die Szenarien A und B nur abgeschwächte Ausgangssituationen zu C sind. Es stellt sich die Fragen, ob in den beiden anderen Szenarien nicht andere Informationen benötigt werden, da das Kennenlernen der Anwendung unter Anleitung durch andere, bzw. durch eigene Kenntnisse geschieht.

B. Toolchain

Bei einer Reihe der Durchführungen dürfen die Testpersonen eine Toolchain zur Unterstützung verwenden, diese wird in diesem Abschnitt genauer vorgestellt. Die Testpersonen

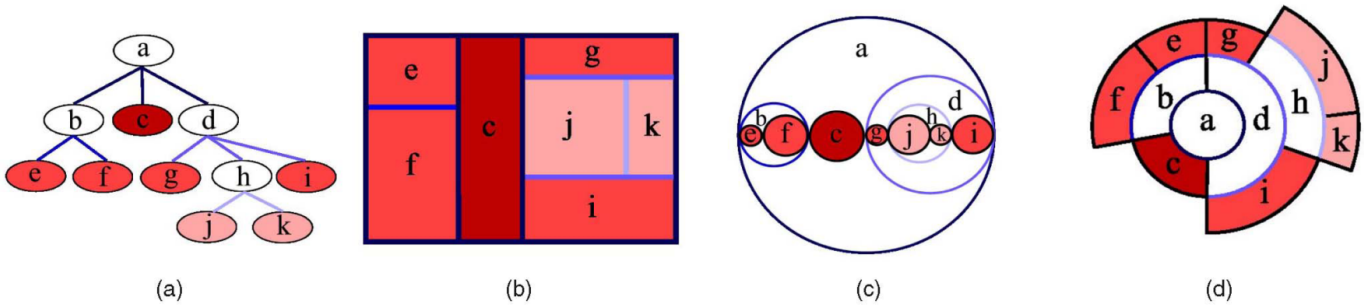

Abbildung 1: Möglichkeiten Strukturen zu visualisieren [2]

ohne Toolchain, arbeiten mit einer Standard IDE, wie Eclipse[3] oder IntelliJ[4]. Die Standard IDE steht den Testpersonen mit Toolchain ebenfalls zusätzlich zur Verfügung.

Es werden zwei unterschiedliche Tools zur Verfügung gestellt, zum einen Sonarqube[5] und die IDE Understand[6].

1) Sonarqube: Sonarqube [12] ist eine Open Source Plattform um Codequalität zu managen. Sonarqube ist mit vielen unterschiedlichen Plugins erweiterbar. Die Funktionalitäten und die eingesetzten Plugins, die für das Experiment eingesetzt werden, werden hier vorgestellt. Sonarqube besteht aus zwei Teilen, einem Server mit Webinterface, der die Informationen darstellt und zur Administration und Datenhaltung der Analysen dient. Der zweite Teile ist der *Sonar-Runner*, dieser führt die eigentlichen Analysen durch und sendet die Daten an den Server. Er muss auf dem Rechner ausgeführt werden, auf dem sich der Sourcecode befindet. Dies kann auf einem Buildserver, oder auch direkt auf den Entwicklerrechnern geschehen. Für weit verbreitete Entwicklungsumgebungen, wie z.B. Eclipse und IntelliJ existieren Plugins, die es ermöglichen die Analysen direkt aus diesen durchzuführen. Analysen können von beliebig vielen Clients eingereicht werden. Die im Beispiel verwendeten Grafiken wurden, wie im Grundprojekt, mit dem Open Source Android Mail Client K9 generiert.

Innerhalb einer Instanz können mehrere unterschiedliche Projekte analysiert werden. Für jedes Projekt lässt sich ein eigenes unabhängiges Dashboard einrichten. Projekte können miteinander verglichen werden.

Für die meisten Analysen reicht der reine Quellcode aus, für die Abhängigkeitsanalysen werden zusätzlich die kompilierten Quelldateien benötigt.

Der Aufbau der Anwendung kann mithilfe einer Treemap (Abbildung 2) dargestellt werden. Durch klicken auf die einzelnen Pakete kann man bis zur konkrete Datei zoomen. Die Größe der Felder wird durch die Anzahl der Codezeilen bestimmt, die Farbe visualisiert die Anzahl der duplizierten Codezeilen. Die Parameter sind beliebig konfigurierbar.

Abbildung 3 zeigt die Anwendung als Stadt[7]. In der Darstellung sind die Paketstrukturen ersichtlich. In der default

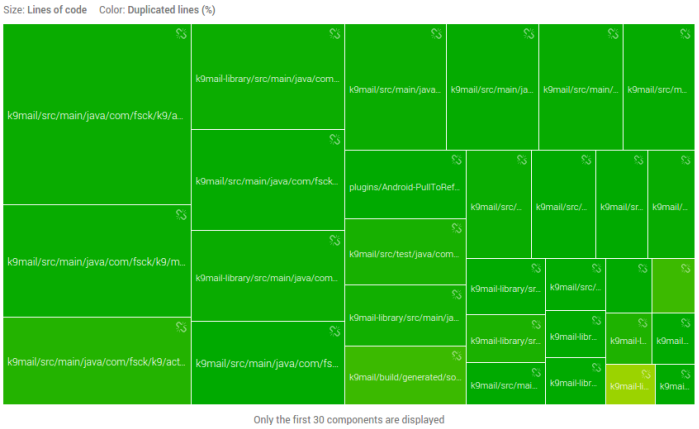

Abbildung 2: Sonarqube Treemap

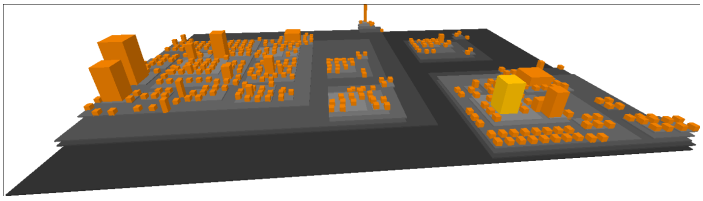

Abbildung 3: Sonarqube SoftVis3d City

Ansicht repräsentiert die Höhe der Häuser, die Anzahl der Codezeilen. Die Breite wird durch die Komplexität definiert. Wie auch bei der Treemap ist es möglich andere Parameter zur Visualisierung auszuwählen. Innerhalb der Visualisierung ist es möglich, durch die Auswahl eines Objektes, dessen Kind und Elternobjekte als Metainformationen zu erhalten. Andere Tools/Plugins bieten an dieser Stelle erweiterte Informationen an, diese sind allerdings nicht kostenfrei oder als Sonarqube Plugin erhältlich.

Mithilfe eines Balkendiagramms kann man sich Technische Schulden (Technical Depts) ansehen. Abbildung 4 zeigt dies beispielhaft für die Android K9 App. Die technischen Schulden, werden in Kategorien, wie z.B. Wiederverwendbarkeit oder Testbarkeit eingeteilt. In dem Diagramm ist z.B. ersichtlich, dass die Wartbarkeit Schulden von 49 Tagen hat. Dies heißt nach einer Schätzung, dass circa 49 Tage benötigt werden um diese zu beheben. Die Probleme werden nach definierten Codequalitätsrichtlinien gefunden. Für diese

[3]https://eclipse.org/home/index.php
[4]https://www.jetbrains.com/idea/
[5]http://www.sonarqube.org/
[6]https://scitools.com/
[7]http://softvis3d.com/#/download

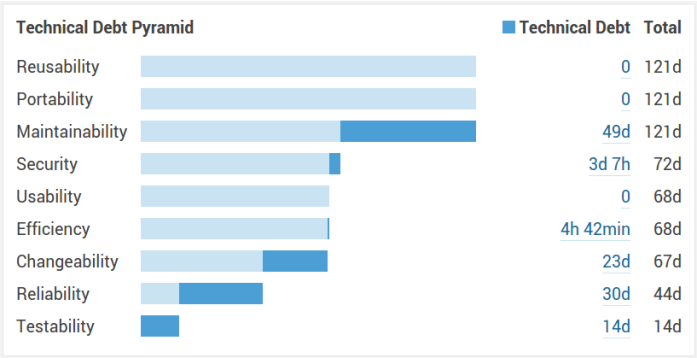

Abbildung 4: Sonarqube Technische Schulden

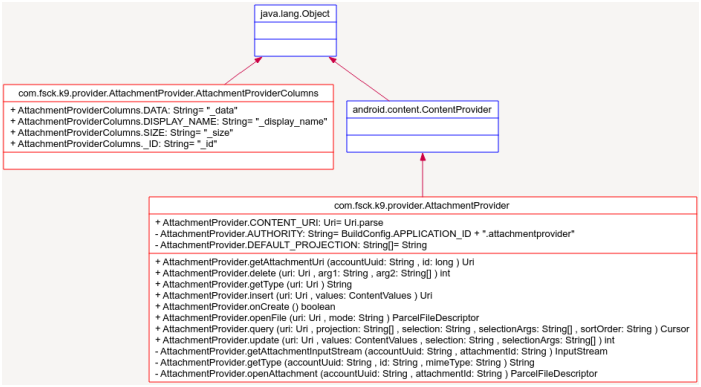

Abbildung 5: Understand Klassendiagramm

kann dort ebenfalls eine geschätzte Lösungsdauer angegeben werden. Durch drauf klicken auf den Wert, kann bis zur entsprechenden Codestellen gesprungen werden. Die Probleme werden zusätzlich nach Wichtigkeit priorisiert. Die höchste Priorität enthält Probleme, die zu einem Blockieren der Anwendung führen können. Die niedrigste stellt nur Informationen, wie z.B. gefundene TODO Kommentare dar.

Abhängigkeiten werden, wie zuvor in Kapitel II beschrieben, mithilfe einer quadratischen Matrix dargestellt. Für die K9-App werden allerdings keine Abhängigkeiten gefunden.

In der Praxis muss sich zeigen, ob diese Informationen helfen neue Features zu entwickeln oder ob sich diese eher für ein Monitoring eignen.

2) Understand: Understand [13] ist eine eigenständige IDE. Sie wurde von Grund auf neu Entwickelt, mit direkt integrierten Möglichkeiten zur Architekturrekonstruktion.

Mithilfe der Anwendung ist es möglich Klassendiagramme, wie in Abbildung 5 zu generieren. Im dargestellten Beispiel wurde die Analyse auf einer Datei ausgeführt. Es ist zu erkennen, dass in dieser Datei zwei Klassen definiert werden. Außerdem sieht man die öffentlichen, sowie privaten Variablen und Methoden. Die Vererbungshierarchie ist über die Baumdarstellung ersichtlich.

Abhängigkeiten können mithilfe drei verschiedener Darstellungen betrachtet werden. Abbildung 6 zeigt, von

Abbildung 6: Understand Abhängigkeiten Von

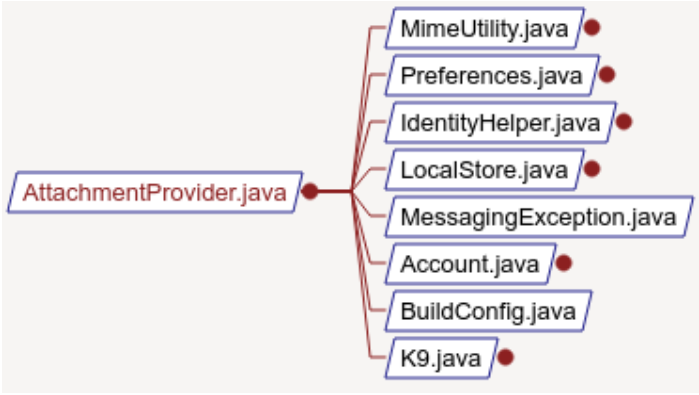

Abbildung 7: Understand Abhängigkeiten

welchen Bestandteilen die untersuchte Datei abhängig ist. Die andere Richtung wird in Abbildung 7 dargestellt. Hier wird präsentiert, welche Dateien von der untersuchten abhängig sind. Die dritte Möglichkeit ist die Kombination aus den beiden innerhalb eines Diagramms.

In dem Cluster Call By Butterfly Diagramm (Abbildung 8) können ebenfalls Abhängigkeiten betrachtet werden. Hier kann mithilfe von Doppelklicks rein und rausgezoomt werden. Bei der Start-Darstellung (kein Zoom) wird die Beziehung von Dateien dargestellt. In der detailliertesten Darstellung sind die konkreten Methodenaufrufe sichtbar (erkennbar oben links in der Abbildung). Je weiter in die Details reingezoomt wird, desto größer und dadurch unübersichtlicher wird der Graph.

Ähnlich zu Sonarqube kann eine Treemap generiert werden, diese kann nach Dateien, Klassen oder Methoden berechnet werden.
Da die Anwendung eine IDE ist, besteht der Vorteil. dass der Sourcecode direkt bearbeitet werden kann. Kompilieren der Android Anwendung ist aber nicht möglich.
Ein weiteres hilfreiches Feature ist, dass alle berechneten Metriken als CSV exportiert werden und dadurch außerhalb der Anwendung genutzt werden können.

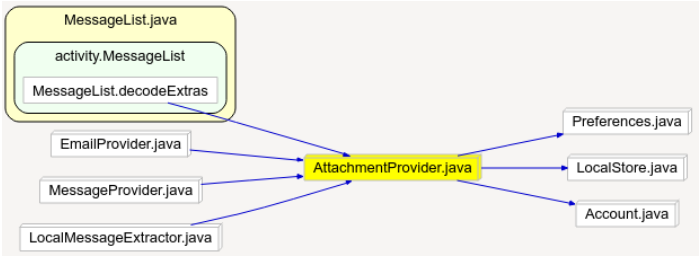

Abbildung 8: Understand Cluster Call Butterfly

C. Experimentevaluierung

Damit geprüft werden kann, ob die Toolchain unterstützend wirkt, muss dies evaluiert werden. Dies soll durch Beobachtungen und Interviews geschehen.

Innerhalb von Sonarqube kann das Piwik[8] Plugin zusätzlich integriert werden. Dies ist eine Analyse Plattform, ähnlich wie Google Analytics[9]. Piwik funktioniert aber auch bei rein internen Webseiten. Dadurch lässt sich untersuchen, welche Seiten besucht wurden und wie lange diese betrachtet wurden. Es kann leider nicht festgestellt werden, welche Grafiken genau betrachtet wurden, falls nur die Sammlung auf dem Dashboard betrachtet wird.

Zur Durchführung werden Fragen erarbeitet, welche bei Beendigung des Experiments in einem persönlichem Interview als Leitfaden dienen sollen.

1) Wie war der erste Eindruck der Tools?
2) Hat die Einführung in die Tools geholfen?
3) Wie haben Sie begonnen?
   a) War dies hilfreich oder würden Sie nächstes mal anders vorgehen?
4) Wie haben Sie die notwendigen Stellen für die Änderungen gefunden, wie sind Sie weiter vorgegangen?
5) Haben die Tools Sie eher abgelenkt oder unterstützt?
6) Welche Informationen haben Ihnen während der Bearbeitung gefehlt?
7) Welche Visualisierung, welche Funktionalität hätten Sie sich zusätzlich gewünscht?
8) Welche Fragen würden sie den ursprünglichen Entwicklern, der zu Bearbeitenden Anwendung, gerne weiterhin stellen, welche sind ihnen während der Bearbeitung gekommen?
9) Können Sie sich andere Einsatzmöglichkeiten für die Visualisierungen und Tools vorstellen?
10) Was hat Ihrer Meinung nach besonders zum Erfolg beigetragen?
11) Was hat Ihrer Meinung nach am wenigsten zum Erfolg beigetragen?
12) Welche Parameter finden Sie für die Darstellungen der Treemaps und der Stadtdarstellungen am hilfreichsten?

Die Fragen sollen als Leitfaden am Ende des Versuchs für ein persönliches Interview dienen. Diese können bei Bedarf im Gespräch angepasst und je nach Situation individualisiert und erweitert werden. Durch die Möglichkeit im Gespräch nachzuhaken, ist es möglich erst allgemeine Fragen zu stellen und diese bei Bedarf zu spezialisieren.

IV. Selbstversuch

In zwei kleinen Selbstversuchen soll überprüft werden, ob der Experimentaufbau sinnvoll ist. Dafür wird der Aufbau in einem theoretischen Experiment überprüft. Das Experiment wird, wie im Szenario C beschrieben, ohne Hilfe der ursprünglichen Entwickler durchgeführt. Das Programm und die Toolchain stehen, wie zuvor vorgestellt, zur Verfügung. Falls

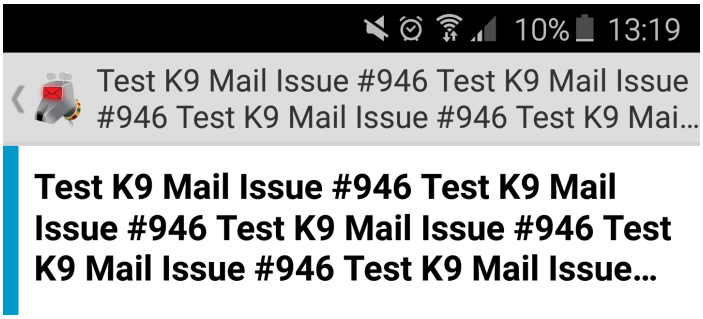

Abbildung 9: K9 Mail Titel Zeile

der Aufwand der Umsetzung zu hoch ist, soll statt der Umsetzung des gewünschten Features, bzw. Behebung des Bugs, nur geprüft werden, ob die Tools helfen, die richtigen Stellen zu finden, an denen angesetzt werden muss, um die gewünschte Funktionalität zu entwickeln oder zumindest hilfreiche Ansätze liefern kann.

Für beide Selbstversuche soll erneut, wie im Grundprojekt die Android App K9-Mail[10] dienen. Auf Github haben bereits 132 Personen Änderungen und Erweiterungen in über 5000 Commits seit Oktober 2008 beigesteuert [14].

A. Versuch A: Darstellung des Betreffs

Im Selbstversuch soll versucht werden eines der vorhandenen Issues[11] zu lösen. Dadurch ist die Durchführung möglichst nah an der Realität, es muss nicht extra ein künstlicher Anwendungsfall konstruiert werden.

1) Aufgabenstellung: Im konkreten Fall soll folgende Anfrage bearbeitet werden:

> Can't read the full subject on long email subjects #946[12]

Bei diesem Problem geht es darum, dass lange Betreffzeilen nicht angezeigt werden. Im Normalfall sollte diese bei einer Kürzung im Nachrichtenteil vollständig wiederholt angezeigt werden, dies ist in der Beschreibung nicht der Fall. Dieser Fehler kann nicht nachgestellt werden. Allerdings wird bei zu langen Betreffzeilen der Text innerhalb des Nachrichtenteils ebenfalls nach drei Zeilen gekürzt (siehe Abbildung 9 und 10). Konkret geht es in dem Fehler um das zweite Auftreten des Betreffs. Bei kurzen Texten, die in die oberen zwei Titelzeilen passen, wird er kein zweites mal angezeigt. Beim Nachstellen des Problems wurde zusätzlich festgestellt, dass trotz einer Begrenzung des oberen Titels auf zwei Zeilen ein drittes mal umgebrochen wird, wenn die Wörter mit einem Bindestrich getrennt sind. Bei zu langen Wörtern wird ebenfalls korrekt gekürzt.

2) Durchführung und Ergebnisse: Der Fehler, des falschen Umbrechens, bei Bindestrichen in der Titelleiste ist eigentlich nicht Teil des Issues, soll im Versuch aber zu Anfang gelöst werden. Hauptaufgabe ist es die Stellen zu finden, an denen angesetzt werden muss. Dazu wurde eine TreeMap auf Basis

[8] http://piwik.org/what-is-piwik/
[9] http://www.google.com/analytics/
[10] https://github.com/k9mail/k-9
[11] https://github.com/k9mail/k-9/issues
[12] https://github.com/k9mail/k-9/issues/946

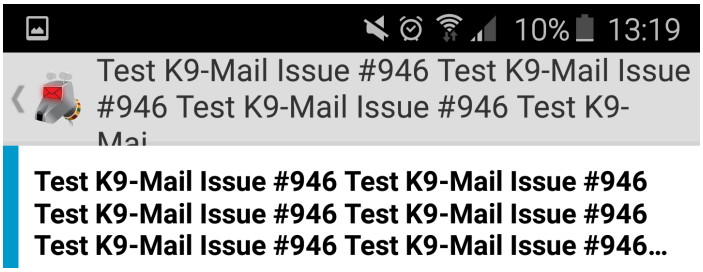

Abbildung 10: K9 Mail Titel Zeile

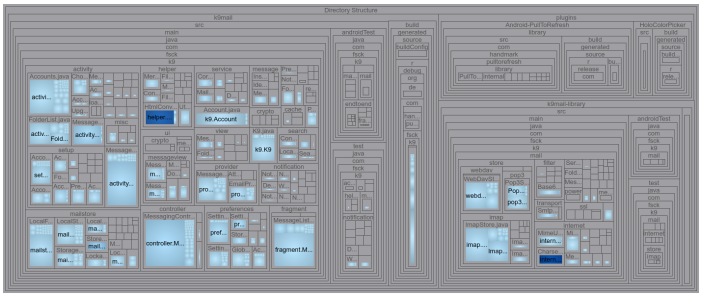

Abbildung 11: Understand TreeMap der gesamten Anwendung

der Klassen und Interfaces, gruppiert anhand der Ordnerstruktur generiert. Als Ergebnis wurde eine recht unübersichtliche Darstellung geliefert (Abbildung 11).

Bei genauerer Betrachtung konnte eine Gruppierung nach dem View Ordner gefunden werden. Sofern die Benennung korrekt ist, sollten hier weitere Hinweise zu finden sein. In Abbildung 12 ist die TreeMap nur für die View Komponente abgebildet. Diese reduzierte Darstellung ist bereits erheblich übersichtlicher.

In dieser Ansicht kann wiederum die MessageTitleView.java Datei gefunden werden. Auf Basis der TreeMap kann von hier aus nicht weitergearbeitet werden. Weitere Details sind in dieser Ansicht nicht darstellbar. Die IDE bietet allerdings an, sich z.B. eine Übersicht über die Deklarationen anzeigen zu lassen (Abbildung 13). Hier kann erkannt werden, dass die Klasse tatsächlich von einem Widget erbt. Das heißt, dass sie für die Darstellung zuständig ist. Des weiteren ist erkennbar, dass die Klasse eine globale Variable MAX_LINES beinhaltet. Diese wurde testweise verändert. Entsprechend wird korrekterweise nach mehr, bzw. weniger Zeilen ausgeblendet.

Abbildung 12: Understand TreeMap der View Gruppierung

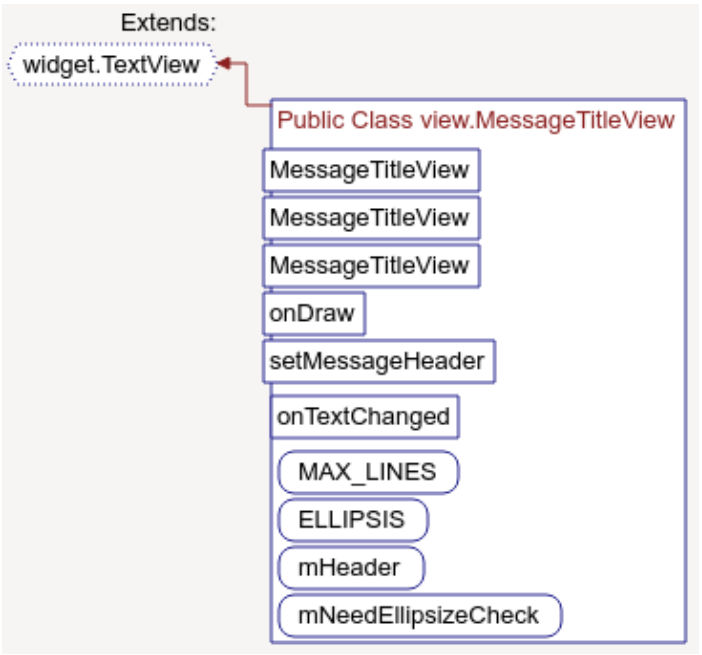

Abbildung 13: Understand Deklarationen

```
/**
 * Check to see if we need to hide the subject line in {@link MessageHeader} or not.
 */
@Override
public void onDraw(Canvas canvas) {
    /*
     * Android does not support ellipsize in combination with maxlines
     * for TextViews. To work around that, check for ourselves whether
     * the text is longer than MAX_LINES, and ellipsize manually.
     */
    if (mNeedEllipsizeCheck) {
        if (getLayout() != null && mHeader != null) {
            if (getLayout().getLineCount() > MAX_LINES) {
                int lineEndIndex = getLayout().getLineEnd(MAX_LINES - 1);
                setText(getText().subSequence(0, lineEndIndex - 2) + ELLIPSIS);
            } else {
                mHeader.hideSubjectLine();
            }
            mNeedEllipsizeCheck = false;
        }
    }
    super.onDraw(canvas);
}
```

Abbildung 14: Source Code Verwendung der Variable MAX_LINES in der Klasse Message Title View

Das Feld, in dem sich die Titelleise befindet wird allerdings nicht vergrößert, dadurch sind bei mehreren Zeilen nur circa 2,5 Zeilen im Vordergrund sichtbar. Dieser Versucht hat aber gezeigt, dass diese Datei für die Darstellung der Titelleise zuständig ist.

Ab hier wird in der Code betrachtet. Dazu wird geprüft, wo die zuvor gefundene Variable verwendet wird (Abbildung 14). Konkret wird die Anzahl der zulässigen Zeichen mithilfe der Zeile *int lineEndIndex = getLayout().getLineEnd(MAX_LINES - 1);* berechnet. Innerhalb der IDE ist es nicht möglich sich die konkreten Methoden anzusehen, da die entsprechenden Klassen Bestandteil des Android SDKs sind, welches nicht gefunden wird. Bei IntelliJ ist dies ohne weiteres möglich. Das Problem scheint also in der gefundenen Zeile zu liegen

und handelt sich somit sogar um ein Bug im Android System selbst und wird deshalb nicht weiter verfolgt.

Im nächsten Schritt suchen wir die Stellen, die das Kürzen des Betreffs im Nachrichtenbereich bewirken. In dem gefundenen Code Abschnitt wird im alternativ Fall, wenn der Text der Titelzeile nicht gekürzt werden muss *mHeader.hideSubjectLine();* ausgeführt. Hierbei handelt es sich um die zusätzliche Betreffzeile innerhalb der Nachricht, die ausgeblendet wird. Die entsprechende Stelle verweist auf die MessageHeader Klasse. Hier wird der entsprechende Text für den Subheader verwendet. Durch manuelles Hinzufügen von Text kann überprüft werden, ob der Text zuvor bearbeitet wird und deshalb trotzdem, wie zuvor, gekürzt wird. Dies ist nicht der Fall, da nach drei Zeilen ebenfalls gekürzt wird. Deshalb handelt sich sich sehr wahrscheinlich um eine Festlegung innerhalb des Frontends des Views. Durch eine Referenz auf das konkrete Objekt in der Oberfläche kann folgende Zeile gefunden werden: *android:maxLines="3"*. Wenn diese entfernt wird, ist es möglich den gesamten Betreff, ohne Kürzung, innerhalb der Nachricht anzuzeigen.

3) Bewertung der Ergebnisse: In diesem Selbstversuch wurden relativ schnell Ansätze zur Lösung gefunden. Trotzdem wird nicht klar, ob die Tools einen entsprechenden Vorteil liefern können. Zu Beginn wurde die Stelle des ersten Problems mithilfe der Tools schnell gefunden, beim zweiten Problem wurde die Tools allerdings nicht eingesetzt. In diesem Fall haben die Tools nur zum Teil geholfen. Es ist schwer gezielt vorzugehen, da nicht bewusst ist, welche Tools den Einstieg erleichtern. Außerdem wird die Android Entwicklung nicht vollständig unterstützt.

Bei der TreeMap konnten die Strukturen schnell erkannt werden. Bei Projekten mit unklaren Benennung und bei Problemen, bei denen unklar ist, in welchem Bereich diese liegen hilft dieser Ansatz allerdings nicht.

B. Versuch B: Einschränken der Suche

Im Rahmen des zweiten Versuches soll eine selbst konstruierte Anforderung umgesetzt werden. Die Erweiterung bezieht sich nur die die Logik und Datenhaltung und soll nicht auf der Oberfläche umgesetzt werden.

1) Aufgabenstellung: Die Aufgabe ist es, eine Suche zu entwickeln, die die bereits vorhandene, rein textbasierte Suche um eine Möglichkeit auf eine Einschränkung für einen Zeitraum hinzuzufügen (Angabe von maximalen und minimalem Von-Datum). Die Entwicklung soll sich hierbei nur auf die entsprechende Logik konzentrieren, innerhalb der App-Oberfläche soll diese Funktionalität nicht eingebaut werden.

2) Durchführung und Ergebnisse: Bei dieser Durchführung soll am Anfang, die bereits vorhandene Suche analysiert werden. Diese soll anschließend entsprechend erweitert, bzw. angepasst werden. Damit dies geschehen kann müssen die entsprechenden Stellen zu Anfang gefunden werden. Die TreeMap liefert in diesem Fall keinen Ansatzpunkt, da keine Komponente direkt mit der Suche in Verbindung gebracht werden kann. Als Alternative wird global nach "Search" und ähnlichen Synonymen gesucht um einen Startpunkt zu finden. Es wird eine Search Klasse gefunden, diese erbt von der Message List Klasse. Diese beinhaltet mehrere Suchmethoden, mit unterschiedlichen Parametern. Um klar zu machen, ob dies die gesuchte Methode ist, sehen wir uns mithilfe eines Butterfly Diagramms (Abbildung 15) an, wie die Methode aufgerufen wird und welche Methoden von ihr wiederum aufgerufen werden. Dies liefert allerdings keine hilfreichen Hinweise.

Durch den erneuten Einbau von Debugausgaben, kann festgestellt werden, dass keine dieser Methoden bei der Suche aufgerufen wird. Dies zeigt, dass das Tool nicht nur nützliche Hinweise liefert. Außerdem wurde um die Stelle zu finden, die Suche verwendet. Dies bietet auch jede IDE an.

Im nächsten Ansatz wird von der Oberfläche ausgegangen, indem nach der Methode gesucht wird, welche von dem Suche-Button ausgeführt wird. Die Understand IDE enthält allerdings nicht den Android Designer für die Oberflächen, weshalb wieder auf Android Studio zurückgegriffen werden muss. Dadurch kann der Suchen-Button gefunden werden und auch die benötigte Methode, welche allerdings nur das Suchfenster öffnet. Dadurch kann im nächsten Schritt allerdings auch die Methode gefunden werden, welche aufgerufen wird, wenn die Suche ausgeführt wird. Dieser Ansatz wird nicht weiter verfolgt, da dazu die zu untersuchenden Tools nicht verwendet werden. Dazu ist die Standard Version von Android Studio ausreichend.

3) Bewertung der Ergebnisse: Auch im zweiten Versuch haben die Tools keinen merkbaren Vorteil gebracht. Im Endeffekt wurde bei der Hauptaufgabe, dem Finden der entsprechenden Stellen erneut die Android Studio Version von IntelliJ verwendet und nicht die Funktionalitäten der Understand IDE. Nachdem die entsprechende Stelle gefunden wurde, hätte wieder auf die Tools zurückgegriffen werden können um zu erfahren, in welcher Sequenz diese aufgerufen werden.

V. Fazit und Ausblick

Sonarqube wurde in diesem Versuch nicht verwendet und eignet sich deshalb vermutlich in den wenigsten Fällen um bei einer aktiven Entwicklung zu unterstützen. Als Monitoring Tool ist dies eher hilfreich, um eine hohe Codequalität sicherzustellen. Sonarqube kann durch automatisierte, regelmäßige Prüfungen sicherstellen, dass ein definierter Standard eingehalten wird und auf Stellen aufmerksam machen, die Probleme verursachen können.

Des weiteren wurden nur wenige Tools von Understand verwendet. Es ist gut vorzustellen, dass bei Problem mit unterschiedlichen Methodenaufrufen und Abhängigkeiten, die Sequenz- und Kontrollflussdiagramme unterstützend wirken können. Um die Understand IDE effektiver zu verwenden, ist es notwendig, diese genauer kennenzulernen. Da es sich bei dieser IDE aber um ein hauptsächlich kommerziell eingesetztes Tool handelt, ist keine große Community vorhanden, die bei Fragen unterstützen kann. Ein weiterer Nachteil ist, dass zwar in der Understand IDE entwickelt, aber nicht kompiliert werden kann. Deshalb wird zusätzlich noch IntelliJ benötigt. Es sollte außerdem herausgefunden werden, ob die fehlenden Bibliotheken entsprechend eingebunden werden können und dadurch eine vollständige Android Entwicklung ohne IntelliJ möglich wird. Wenn dies nicht der Fall ist, ist ein effektiver Einsatz schwer möglich. Wenn dies nicht der Fall ist, sollte für

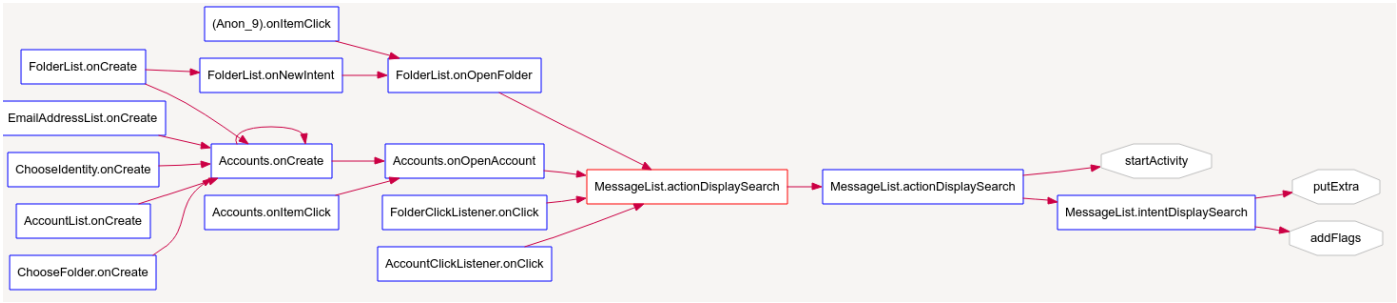

Abbildung 15: Butterfly Diagramm Suche

die konkrete Evaluierung, mit Testpersonen, ein Projekt gefunden werden, welches vollständig durch Understand unterstützt wird.

Die Testanwendung wird als Open Source Anwendung entwickelt, dadurch arbeiten viele unterschiedliche Entwickler an dem Projekt. Da in Projekten dieser Art häufig Nachfragen anderer Entwickler entstehen, wird auf eine gute Lesbarkeit, durch aussagekräftige Namen und Strukturen geachtet. Außerdem wird für den Großteil der Codeeinreichungen, bevor diese in den Haupt Entwicklungsstrang überführt werden, ein Review durch andere Entwickler durchgeführt. Dies bewirkt, dass die Mehrheit der Nachfragen entweder überflüssig werden oder schnell beantwortet werden können.[15]
Im Gegensatz dazu, ist es bei Closed Source Projekten häufig der Fall, dass wenige Personen den Code sehen und die Entwickler gegen eine festgelegte Deadline arbeiten. Dadurch wird der Code bei Zeitdruck schnell geschrieben, statt dass auf sauberen und lesbaren Code geachtet wird.

Es konnte nicht herausgefunden werden, wie ein erfolgreicher Einstieg aussehen kann, um sich in eine neue Anwendung einzuarbeiten. Die Tools konnten nicht erfolgreich dazu beitragen. Ein nicht rein toolbasiertes Vorgehen ist deshalb empfehlenswert. Es ist erforderlich, dass erst ein Grundverständnis für die Anwendung aufgebaut wird. Die angebotenen Tools sind nur in ausgewählten Abschnitten unterstützend. Für ein Grundverständnis der Anwendung, ist es erforderlich, dass zu Beginn die Dienstleistungen, welche die Software anbietet verstanden werden. Dies ist notwendig, um zu verstehen, wozu die Anwendung verwendet werden kann. Dies ermöglicht es während der Entwicklungen genauere Vermutungen zu treffen, um diese Anschließend zu überprüfen. Anschließend sollen die Punkte gefunden werden, welche die gesuchte Dienstleistung auslösen. Dazu kann entweder von Oberfläche (z.B. welche Methode wird beim Drücken eines Knopfes ausgelöst) oder von der API aus, vorgegangen werden. Ab hier ist in unterschiedlichen Phasen ein Einsatz der Tools und Visualisierungen denkbar. Es kann z.B. ein Sequenzdiagramm erzeugt werden, welches den Aufruf der gefundenen Punkte darstellt. Dies ermöglicht es auf einen Blick zu sehen, wie welche Methoden, mit welchen Daten aufgerufen werden. Ohne das Diagramm müsste jede Methode im Code einzeln betrachtet werden. Wenn eine Methode gefunden wird, bei der Änderungen erforderlich sind, muss zuvor geprüft werden, ob diese ebenfalls an anderen Stellen zum Einsatz kommt. Dies kann innerhalb fachlich vollständig unabhängigen Dienstleistungen sein. Diagramme, wie das Butterfly Diagramm in Abbildung 15 können helfen, die Stellen zu finden, an denen diese Methoden verwendet werden. Für die gefundenen Abhängigkeiten darf sich bei einer Änderung der gefundenen Methode, im Normalfall nichts ändern. Dadurch können unerwünschte Nebeneffekte verhindert werden. Eine genauere Beschreibung eines solchen Vorgehens, mit den möglichen Einsatzzwecken der Tools sollte in zukünftigen Arbeiten entwickelt werden.

Die bereits vorhandenen Tools konnten in diesem Rahmen nur eingeschränkt unterstützen. Ein vollständiger Workflow, für die hier verwendeten Anwendungsfälle, kann mit ihnen noch nicht abgebildet werden. Dies zeigt, dass in diesem Themenbereich noch einiges an zukünftiger Arbeit steckt. Dazu zählt unter anderem neue Darstellungsformen zu entwickeln, die auch bei großen Projekten nicht zu unübersichtlich werden.

Eine weitere andere mögliche Betrachtungsweise dieses Problems für neue Projekte ist es, nicht erst aus bereits existierendem Code eine Architektur zu rekonstruieren, sondern von Beginn an eine geplante Architektur zu entwickeln. Dazu muss herausgefunden werden, wie die Planung einer Architektur und ein agiles Vorgehen erfolgreich zusammenspielen können.

Referenzen

[1] S. Ducasse and D. Pollet, “Software architecture reconstruction: A process-oriented taxonomy,” IEEE Transactions on Software Engineering, vol. 35, no. 4, 2009, pp. 573–591. [Online]. Available: http://rmod.lille.inria.fr/archives/papers/Duca09c-TSE-SOAArchitectureExtraction.pdf

[2] P. Caserta and O. Zendra, “Visualization of the Static Aspects of Software: A Survey,” IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 7, jul 2011, pp. 913–933. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/20733234http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5557869

[3] R. Kazman and S. J. Carrière, “Playing Detective: Reconstructing Software Architecture from Available Evidence RICK,” Automated Software Engineering, vol. 6, no. 2, 1999, pp. 107–138. [Online]. Available: http://link.springer.com/10.1023/A:1008781513258

[4] M. T. Su, “Capturing exploration to improve software architecture documentation,” in Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA ’10. New York, New York, USA: ACM Press, 2010, p. 17. [Online]. Available: http://dl.acm.org/citation.cfm?id=1842752.1842758http://portal.acm.org/citation.cfm?doid=1842752.1842758

[5] C. Stoermer, L. O’Brien, and C. Verhoef, “Practice patterns for architecture reconstruction,” in Ninth Working Conference on Reverse Engineering, 2002. Proceedings. IEEE Comput. Soc, 2002, pp. 151–160. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1173073

[6] R. Koschke, “Architecture Reconstruction,” in Technical Report CMU/SEI-2002-TR-034, 2009, no. November, Third Edition, pp. 140–173. [Online]. Available: http://www.sei.cmu.edu/reports/02tr034.pdfhttp://link.springer.com/10.1007/978-3-540-95888-8{_}6

[7] K. Kobayashi, M. Kamimura, K. Yano, K. Kato, and A. Matsuo, “SArF map: Visualizing software architecture from feature and layer viewpoints,” in 2013 21st International Conference on Program Comprehension (ICPC). IEEE, may 2013, pp. 43–52. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6613832

[8] J. T. Stasko, R. Catrambone, M. Guzdial, and K. Mcdonald, “An evaluation of space-filling information visualizations for depicting hierarchical structures,” Int. J. Hum.-Comput. Stud., vol. 53, no. 5, 2000, pp. 663–694.

[9] R. Wettel and M. Lanza, “Visualizing Software Systems as Cities,” in 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis. IEEE, jun 2007, pp. 92–99. [Online]. Available: http://www.inf.unisi.ch/projects/evospaces/publications/Wettel07b.pdfhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4290706

[10] ——, “Program Comprehension through Software Habitability,” in 15th IEEE International Conference on Program Comprehension (ICPC’07), 2007, pp. 231–240. [Online]. Available: http://www.inf.unisi.ch/phd/wettel/download/Wettel07a-icpc.pdf

[11] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc, “Communicating Software Architecture using a Unified Single-View Visualization,” in 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007). IEEE, 2007, pp. 217–228. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4276318

[12] Sonarsource, “Sonarqube,” 2015. [Online]. Available: http://www.sonarqube.org

[13] Scitools, “Understand,” 2015. [Online]. Available: https://scitools.com/

[14] K9mail, “k-9,” 2015. [Online]. Available: https://github.com/k9mail/k-9

[15] O. A. Samoladas Ioannis, Stamelos Ioannis, Angelis Lefteris, “Open Source Software Development Should Strive for EVEN GREATER CODE MAINTAINABILITY,” Communications of the ACM, vol. 47, no. 10, 2004, pp. 83–87.