Softwarearchitektur und Agile Entwicklung
Architekturawareness durch Monitoring und automatischer Rekonstruktion

Leon Fausten

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

Der Chaos Report von 2014, der Standish Group, hat herausgefunden, dass 31,1% aller Software Projekte abgebrochen werden. Bei 52,7% der untersuchten Projekte werden die Kosten mit insgesamt 187%, im Vergleich zur ursprünglichen Planung, überschritten. Ein großes Problem ist, dass viele Projekte häufig erneut von vorne gestartet werden. Von 100 Projekten werden im Schnitt 94 Projekte neu gestartet. Ein gleiches Projekt kann dabei mehrfach neu gestartet werden. Die Studie besagt ebenfalls, dass im Durchschnitt nur 61% der Anforderungen und Funktionalitäten umgesetzt werden können. Für die Architektur sind besonders die nicht funktionalen Anforderungen wichtig, wenn diese nicht umgesetzt werden kann dies den Erfolg des Projektes beeinträchtigen. Eine Befragung hat ergeben, dass unvollständige (Platz 2, 12,3%) und sich ändernde Anforderungen (Platz 3, 11,8%) für 32,1% ein Grund für Fehlschläge sind.[1]

80% der Gesamtkosten eines Systems entstehen in der Wartung. Dies ist unter anderem auf das altern und dadurch verschlechtern der Architektur zurück zu führen. Deshalb sollte die Architektur bereits während der Wartung weiterentwickelt und eventuell restrukturiert werden. Für eine erfolgreiche Anpassung der Architektur muss die derzeitig implementierte Architektur bekannt sein.[2]

Softwarearchitekturen sind meistens sehr komplex und schwer im Detail vollständig zu verstehen. Dies liegt daran, dass sie aus vielen unterschiedlichen Bestandteilen bestehen und bei großen Projekten einen Umfang haben, der nicht auf Anhieb verständlich ist. Moderne Entwicklungs-Frameworks[1] behandeln die Architekturentwicklung nur sehr leichtgewichtig. Doch wie sollte eine optimale Möglichkeit aussehen, solch eine Architektur zu entwickeln und zu warten. Ein erster Schritt ist es dabei ein Bewusstsein für die Architektur zu schaffen, damit alle Projektbeteiligten immer genau wissen, woran sie arbeiten. Die Kenntnis der derzeit implementierten Architektur kann Ansatzpunkte für problematische Stellen liefern, die ausgebessert werden sollten. Es soll ein Bewusstsein geschaffen werden, dass die Architektur eine wichtige Rolle für den Erfolg eines Projektes spielt.
Innerhalb dieser Arbeit wird eine genauere Problemstellung erarbeitet und eine mögliche Wunschsoftware zur Unterstützung entworfen. Dabei werden neben Informationen aus einer zuvor geführten Umfrage und Ergebnisse aus relevanten Arbeiten, Ideen aus existierenden Anwendungen herangezogen. Zentral wird vor allem die automatisierte Unterstützung und Visualisierung mithilfe von Reverse Engineering betrachtet. Beim Reverse Engineering wird die Architektur anhand vom Quellcode und anderen Artefakten (Dokumentation, Datenfluss, ...) rekonstruiert. Eine manuelle Untersuchung, durch eine händisch durchgeführte Code Inspektion, ist meist praktisch nicht durchführbar und sehr teuer, da die Anwendungen sehr umfangreich sein können. [3]. [4][5][6]
Automatisierte Rekonstruktion ist aufwendig und schwierig, da Menschen aus Erfahrungswerten intuitiv erahnen, welche Bestandteile relevant für die Architektur sind. Automatisierte Tools arbeiten deshalb oftmals mit Clustering und der Erkennung von Mustern. Dabei können aber schnell Fehler entstehen. Deshalb ist eine Mischung aus automatisierter und manueller Rekonstruktion wünschenswert.[5]

I. Problemstellung

Innerhalb der Agilen Frameworks ist kein Vorgehen zur Architektur Entwicklung beschrieben. Wie in der Ausarbeitung zum Grundseminar[2] festgestellt, befinden sich teilweise sogar widersprüchliche Aussagen innerhalb des Agilen Manifesto. Dadurch entsteht die Architektur meistens zufällig, durch die unterschiedlichen Erfahrungen und Vorlieben der Entwickler. Einzelne Komponenten können dadurch ohne genau Konventionen und Richtlinien vollständig unterschiedlich aufgebaut sein. Die entstehende Gesamtarchitektur ist dadurch entsprechend chaotisch und meist zusätzlich noch unbekannt.

Da nicht alle nicht funktionalen Anforderungen mit äquivalent hoher Priorität umgesetzt werden können, muss von den Stakeholder genau festgelegt werden, welche Eigenschaften in welchem Umfang realisiert werden sollen. Die nicht funktionalen Eigenschaften, wie z.B. Ausfallsicherheit und Geschwindigkeit können nicht gleichermaßen erreicht werden.

Der aktuelle Stand der Architektur, während der Entwicklung, ist meist nur grob oder gar nicht bekannt. Dies ist meist der Fall, wenn mehrere Personen an der Entwicklung beteiligt sind. Dadurch kann ein einzelner nicht genau wissen, wie sich die Anwendung weiterentwickelt hat. Festzustellen, ob die Architektur der gewünschten Soll-Architektur entspricht ist dadurch schwer möglich.[5] Zu Projektbeginn wird die Architektur meist nur informal über Diagramme oder textuelle Beschreibungen kommuniziert. Dadurch driften die konzeptionelle Architektur und die implementierte Architektur meist auseinander. [6]

Falls keine oder nur eine initiale Architektur, die nie weiterentwickelt wurde existiert, kann nicht sichergestellt werden,

[1] z.B. Scrum, Kanban, ...

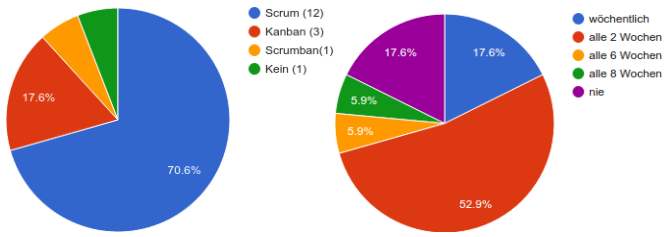
[2] siehe Ausarbeitung Grundseminar, http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master14-15-gsm/fausten/bericht.pdf

dass die Architektur überhaupt noch den Bedürfnissen und Wünschen der Stakeholder entspricht, da der aktuelle Stand unbekannt ist.

Es gibt den Ansatz in die eigentliche Programmiersprache eine Architektur Beschreibungssprache (ADL - Architectural Description Language) zu integrieren, wie z.B. in ArchJava [7] geschehen. Bei ArchJava wurde die ursprüngliche Java Sprache um die ADL erweitert. Alle Architekturentitäten sind first-class Elemente der Sprache. Das Architekturwissen wird in diesem Ansatz vollständig im Code abgebildet. Der Nachteil bei diesem Ansatz ist, dass eine hohe Abhängigkeit zu der dedizierten Sprache entsteht. Im Beispiel zu ArchJava, wurde diese seit 2005 nicht weiterentwickelt. Dies stellt ein großes Problem bei langlebigen Anwendungen dar. Ein anderer Ansatz ist, die Untersuchung mit unabhängigen externen Tools durchzuführen. Diese sollen keinen direkten Einfluss auf die Entwicklung haben und können auch ohne Probleme während der Entwicklung eingeführt, gewechselt oder abgeschafft werden.[6] Innerhalb dieser Arbeit wird nur die Verwendung von externen Tools weiter betrachtet, da das Risiko durch die starke Abhängigkeit bei einer direkten Integration in die Sprache zu hoch erscheint.

II. Resultate der Umfrage

Um aktuelle Informationen über die Entwicklung in der Praxis zu erhalten, wurde eine Umfrage mit 17 Teilnehmern zu Projekten ihrer Wahl durchgeführt. Jeder Teilnehmer und jede Teilnehmerin hat eigenständig ein Projekt, unabhängig von anderen Teilnehmern gewählt. Die Teilnehmer haben unterschiedliche Rollen. Project Owner, Scrum Master, Architekten und Entwickler haben an der Umfrage teilgenommen. In Abbildung 1a ist zu sehen, dass bei einem Großteil der Projekte Scrum verwendet wird. Aus diesem Grund macht es Sinn, die Anwendung in das Framework zu integrieren. Es sind Projekte unterschiedlicher Größe ausgewählt worden, angefangen bei 1,5 Entwicklern, bis hin zu 90 Entwicklern. Sechs der Entwickler sind seit Beginn an dem jeweiligen Projekt beteiligt.

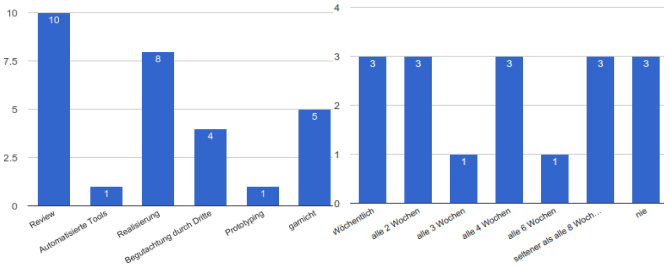


(a) Entwicklungsvorgehen (b) regelmäßige Debatte

Abbildung 1

Ebenfalls Sechs der 17 Teilnehmern (35,29%) war der aktuelle Projektstand unbekannt, sie konnten nicht abschätzen, wie viel der Arbeit bereits ungefähr erledigt war. Nur fünf Personen hatten Kenntnisse über die geplante Gesamtdauer des Projektes.

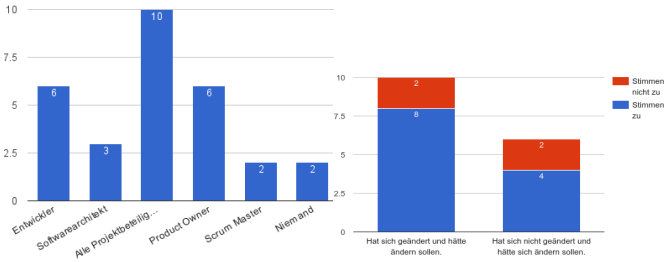
Über 70% der Befragten bietet sich bereits die Möglichkeit im Rahmen einer regelmäßigen Debatte (Abbildung 1b) mit einem Abstand von ein bis zwei Wochen über Änderungen und Erweiterungen zu diskutieren. Diese Debatte bietet sich an, um sie um einen Bestandteil der Architekturdebatte zu erweitern, wenn dies nicht bereits durchgeführt wird.



(a) Arten (b) Regelmäßigkeit

Abbildung 2. Evaluierung

Die meisten der Projekte werden mithilfe von Reviews evaluiert. Dies bietet eine gute Grundlage um darauf aufzubauen und in alle Projekte zu integrieren. Nur bei einem Teilnehmer werden automatisierte Tools zur Überprüfung verwendet. Die Verwendung solcher Tools ermöglicht eine regelmäßige Prüfung inklusive Monitoring. Da dies selten verwendet wird, kann dies zur Folge haben, dass sich die Architektur in eine falsche Richtung entwickelt, ohne das es rechtzeitig bemerkt wird. Nicht automatisierte Evaluierungsmethoden, bedeuten, dass nicht immer die vollständige Anwendung geprüft wird, sondern maximal nur Ausschnitte. Die Begutachtung durch wechselnde Dritte ist empfehlenswert, da die Personen die Anwendung unvoreingenommen betrachten können und neues Wissen und Erfahrung ins Team bringen können.
Sechs der Projekte werden alle ein bis zwei Wochen evaluiert (Abbildung 2b), meistens geschieht dies durch Reviews. Länger sollte der Abstand der einzelnen Evaluierungen nicht sein, damit Probleme frühzeitig erkannt werden und Lösungen gefunden werden können.



(a) Ansprechpartner bei Architekturfragen (b) Änderungen

Abbildung 3

In Abbildung 3a ist zu sehen, dass die meisten Teilnehmer der Umfrage einen Ansprechpartner bei Fragen zur Architektur haben. Drei Personen steht dabei ein professioneller Architekt zur Verfügung. In den meisten Fällen ist das gesamte Team für die Architektur zuständig. Zum Teil wird Scrum nicht korrekt durchgeführt, da der Scrum Master bei zwei Personen der Ansprechpartner ist. Außerdem sind zwei Personen vollständig auf sich selbst gestellt, obwohl deren Teams aus 7, bzw. 12 Entwicklern besteht. Dadurch, dass es in

den meisten Fällen keinen speziellen Ansprechpartner und Architekturverantwortlichen gibt, werden wichtige Aktionen, wie z.B. ein Review meist vernachlässigt. Vier der Teilnehmer sind mit der entwickelten Architektur nicht einverstanden. Bei einer Änderung der Architektur im Laufe der Entwicklung stimmen zwei Personen nicht zu, dass sie sich hätte ändern müssen. Ebenfalls zwei Personen zufolge hätte sie sich ändern sollen, obwohl sie es nicht tat (Abbildung 3b).
Für einen erfolgreichen Projektstart, ist wichtig, dass alle Teammitgliedern eine Einführung in die Architektur bekommen. Neben der Einführung ist eine gute Dokumentation wichtig, damit sie schnell eine Übersicht über die gewünschte und die aktuell entwickelte Architektur erhalten können. Elf der Teilnehmer waren nicht von Anfang an am Projekt beteiligt, nur zwei von ihnen haben eine Einführung in die Architektur bekommen. Die Restlichen mussten sich selbst einen Überblick schaffen, indem sie aktiv nachgefragt haben, den Code durchgesehen haben oder entstandene Dokumente betrachtet haben. Dies verdeutlicht umso mehr, dass eine aktuelle Dokumentation wichtig ist, die es ermöglicht einen schnellen Einblick in das derzeit implementierte System zu bekommen.
Die Dokumentation wird auf unterschiedlichste Arten durchgeführt. Dabei wird in sieben Fällen eine rein grafische, eine rein mündliche oder keine Art von Dokumentation erstellt. Diese haben das Problem, dass sie sehr kurzlebig sind. Grafiken ohne Text sind nach langer Zeit oder für unbeteiligte Personen meist schwer verständlich. Rein mündliche Dokumentationen werden oft fehlerhaft und unvollständig weitergegeben und geraten mit der Zeit in Vergessenheit. In acht Fällen wird eine lokal installierte Anwendung oder ein global zugängliches Websystem zur Dokumentation verwendet. Eine webbasierte Dokumentation hat den Vorteil, dass diese von überall zugreifbar ist, ohne zusätzliche Software installieren zu müssen, hat aber den Nachteil, dass lokale Änderungen nicht berücksichtigt werden. Dies ist aber nur relevant, wenn das Tool die Anwendung automatisch analysiert und nicht manuell gepflegt werden muss.

Zusammenfassend kann gesagt werden, dass in den meisten hier vorhandenen Projekten bereits eine Grundlage für eine Architekturdebatte vorhanden ist. Vorhandene Meetings und teilweise konkrete Versuche die Architektur explizit weiterzuentwickeln zeigen deutlich, dass eine Bereitschaft existiert das Vorgehen zu verfeinern. Der erste Schritt kann dabei die Einführung eines Tools sein, um den aktuellen Zustand der Architektur visuell aufzuarbeiten und zu monitoren.

III. Das Wunsch-System

Zur Lösung der Probleme ist eine softwaregestützte Lösung hilfreich. Diese soll durch den Vorgang leiten und mit nützlichen Tools unterstützen.

Innerhalb dieses Abschnittes soll eine theoretische Wunschanwendung erarbeitet werden, dazu werden Punkte erarbeitet, die mit bereits existierenden Anwendungen verglichen werden sollen.

1) **Gruppieren und Abstrahieren** Damit große Systeme händelbar werden, sollen diese bei der Modellierung in viele kleine Komponenten zerteilt werden können, die wiederum aus Komponenten bestehen können. Tickets aus dem Ticketsystem können konkreten Komponenten zugeordnet werden. Über das Versionierungstool, kann die Änderungshistorie von Komponenten betrachtet werden. Durch die Möglichkeit Details und ganze Komponenten auszublenden soll die Komplexität des Systems verringert werden. Bei einer Betrachtung kann sich so auf die in diesem Moment wichtigen Bestandteile besser konzentriert werden. Bei Bedarf können Details wieder mit eingeblendet werden. Einzelne Komponenten können z.B. gezielt ein/ausgeblendet werden.
2) **Referenz** Referenzen auf Pattern, Referenzarchitekturen oder ähnlichem können festgelegt werden oder durch eine Codeanalyse ermittelt werden. Dadurch kann sichergestellt werden, dass sich bereits bewährte Lösungen wiederverwendet werden.
Eine Modell-Datenbank kann bei der Suche nach ähnlichen Strukturen, wie der umzusetzenden, hilfreich sein. Diese Datenbank enthält hilfreiche Pattern, inklusive deren Vor- und Nachteile. Die Referenzen können z.B. zu Komponenten oder Tickets hinzugefügt werden. Bei der Suche nach einem Pattern kann man in der Modell-DB sehen, in welchen Projekten diese verwendet wurden und welche Ansprechpartner man bei Fragen kontaktieren kann.
3) **Sprachen unabhängig** Für eine möglichst großflächige Nutzbarkeit sollte das Tool nicht abhängig von speziellen Programmiersprachen sein, sondern bei möglichst allen weit verbreiteten Sprachen einsetzbar sein.
4) **Abhängigkeiten** Durch die unterschiedlichen Komponenten entstehen zwingend Abhängigkeiten zwischen ihnen, ansonsten könnten sie ein eigenständiges System bilden. Dadurch kann schnell erkannt werden, welche Komponenten bei Schnittstellenänderungen aktualisiert werden müssen. Außerdem werden potentielle Nadelöhre sichtbar. Zu viele unterschiedliche Abhängigkeiten zwischen Komponenten können zusätzlich auf einen falschen Aufbau der Komponenten hinweisen.
5) **Zugriff** Alle Informationen sind überall zugänglich. Damit effektiv immer auf Basis von aktuellen Daten gearbeitet werden kann, sind alle Informationen auch außerhalb der Meetings mit aktuellen Daten vorhanden. Am nützlichsten ist dabei eine direkte Integration in die Entwicklungsumgebung und als webbasiertes System. Das webbasierte System eignet sich gut für Meetings oder für ein Monitoring auf einem gut sichtbaren Bildschirm. Die Daten des Websystems beziehen sich dann immer auf den Stand auf dem Entwicklungsserver. Dadurch muss keine zusätzliche Anwendung gestartet werden oder die aktuelle Version auf einen Server übertragen werden, um die Änderungen während der Entwicklung zu sehen. Eine Mischung von lokaler Integration und Websystem ist wünschenswert.
6) **Entwicklung** Damit die Architektur nicht zufällig entsteht, sondern direkt gewünschte Eigenschaften unterstützt, sollte die Entwicklung aktiv von Beginn

an des Projektes unterstützt und dauerhaft begleitet werden. Ein initialer Architekturentwurf wird durch die Anwendung unterstützt und kann als Grundlage für den Projektstart dienen. Durch eine direkte Integration in die IDE (siehe Zugriff) kann eine Art von Model Driven Development umgesetzt werden, indem das Design dort grafisch modelliert werden kann. Das entsprechende Code Grundgerüst (Komponenten, Klassen, Interfaces, ..) kann daraufhin generiert werden. Details müssen manuell ergänzt werden.

7) **Bemerkungen, Hinweise und Alerts** An möglichst allen Stellen, können freitext Kommentare erstellt werden, um auf etwas aufmerksam zu machen oder Fragen zu stellen, die sich auf konkrete Stellen beziehen. Diese lassen sich priorisieren und gesammelt in einer Übersicht anzeigen. Es ist möglich unterschiedliche Typen auszuwählen (z.B. Frage, Hinweis, Empfehlung, u.s.w.). Diese sollen automatisiert z.B. in einem Meeting oder einem Zeitpunkt Erinnerungen anzeigen. Zusätzlich sollen automatisierte Meldungen bei der Verletzung von vorgegebenen Richtlinien angezeigt werden. Es können z.B. bei Interface Verletzungen (z.B. Erkennung von Casten auf konkrete Objekte) oder Differenzen zwischen modellierter Architektur und implementierter Architektur Warnungen erzeugt werden.
8) **Abstimmungen und Beschlüsse** Es ist möglich Abstimmungen zu starten, um die unterschiedliche Meinungen der Projektbeteiligten zu erfahren.
9) **Workflow** Die Anwendung unterstützt den Vorgang aktiv und kann ihn sogar leiten, indem z.B. Reviews nach vielen Änderungen oder einer bestimmter Zeitspanne empfohlen werden. Durch eine direkte Integration in die IDE (siehe Punkt Entwicklung) kann eine Art von Model Driven Development unterstützt werden.
10) **Review/ Evaluierung** Dies beinhaltet die Prüfung, ob die Ist-Architektur der Soll-Architektur entspricht. Bei der Soll-Architektur soll ermittelt werden, ob diese überhaupt noch mit den Projektzielen übereinstimmt oder angepasst werden muss. Dazu wird ermittelt, ob sich die Projektziele seit Beginn der Entwicklung geändert haben. Dies muss manuell durch das Projektteam und den Stakeholdern geschehen.
Für Reviews gibt es einen Review-Benutzertyp, dieser kann für jeden teilnehmenden Entwickler eingeführt werden. Jeder Entwickler muss maximal eine festgelegte Anzahl von Reviews pro Zeitraum (z.B. 2 Reviews pro Monat) durchführen. In der Anwendung existiert ein Button um ein Review anzufordern. Die Anwendung ermittelt dann einen zufälligen Reviewer. Dies hat den Vorteil, dass Wissen zwischen Teams einfacher ausgetauscht wird und Personen unterschiedlicher Erfahrung und Vorwissen ein Review durchführen. Dabei können auch Personen anderer Abteilungen gewählt werden. Neben dem Prüfen der Codequalität wird das Fachwissen breiter in der Firma verteilt.
11) **Kommunizieren/ Vorstellen** Das Vorstellen der Architektur den anderen und vor allem neuen Teammitgliedern ist sehr wichtig, damit alle das gleiche Verständnis der aktuellen Architektur besitzen. Dazu soll es eine Übersichtsseite mit wichtigen Entscheidungen und Fakten zum Projekt geben. Diese soll dazu dienen allen Projektbeteiligten einen groben Einblick über den aktuellen Stand zu präsentieren. Eine gute Übersicht kann dauerhaft auf einem gut sichtbarem Monitor dargestellt werden. Dadurch werden Diskussionen angeregt und automatisch alle, die daran vorbei laufen, auf den aktuellen Stand gebracht.
12) **Visualisierung** Für eine anschauliche Darstellung ist eine geeignete Visualisierung notwendig. Diese sollten die Komplexität und auch die Abhängigkeiten darstellen können. Die Visualisierung soll den unterschiedlichen Bedürfnissen angepasst werden, um so die Konzentration auf unterschiedliche Punkte lenken zu können. Komplexe Visualisierungen, wie Abhängigkeitsgraphen mit vielen Verknüpfungen, sollten durch Ausblenden einzelner Details und Bestandteilen vereinfacht werden können.
13) **Metriken und Analysen** Es lassen sich Metriken wie Methoden/Klassengrößen, ungenutzter Code, duplizierter Code und ähnlichem Berechnen. Analysen zu Interface Verletzungen, leeren Try-Catch Blöcken oder ähnlichem können durchgeführt werden.
14) **Pattern und Strukturen** Die Anwendung kann automatisiert Pattern und Strukturen erkennen. Die Pattern können manuell bestimmt werden.

IV. Relevante Arbeiten

Es existieren bereits einige Ideen zur softwaregestützten Architekturentwicklung. In diesem Abschnitt werden ein paar davon besprochen und deren Vor- und Nachteile erläutert. Es wird untersucht, auf welchen man aufbauen kann und welche gute Ideen und Ansätze für die eigene Anwendung liefern.

In dem Artikel Visual Tools for Software Architecture Understanding [5] wurden unterschiedliche Tools zur Visualisierung untersucht. Einige der Anwendungen, hatten dabei den Nachteil, dass diese sehr aufwendig zu konfigurieren waren. Dadurch sind diese nicht besonders nützlich, da für eine gute Visualisierung viel zeitaufwendiges Feintuning erforderlich ist. Für die Visualisierung ist wichtig, wer dessen Stakeholder sind, Entwickler benötigen andere Darstellungen als Manager. In diesem Projekt sind die Entwickler und Architekten die Adressaten. Ein Ergebnis deren Studie ist, dass sich die Darstellung von mehreren Attributen gleichzeitig, wie z.B. bei Tree Maps (z.B. Abbildung 6) am besten eignet um Informationen gebündelt darzustellen. Ein großes Problem ist die automatisierte Erkennung der Architektur:

> Major technical limitations to architecture analysis include the lack of reliable, easily reusable solutions for automated architecture extraction, architecture comparison (visual or not), and architecture pattern detection, despite sustained ongoing research.[5]

Die Rekonstruktion kann durch unterschiedliche Arten von Analysen durchgeführt werden.
statische Analyse: nur mit Hilfe des Quellcodes, ohne Ausführung der Anwendung
dynamische Analyse: mithilfe der Ausführung der Anwendung

A. Architektur Rekonstruktion

Durch die dauerhafte Weiterentwicklung ist eine manuelle Dokumentation schwierig und schnell veraltet. Deshalb ist eine automatische Rekonstruktion (SAR - Software Architecture Reconstruction) sehr hilfreich und empfehlenswert.

Die Rekonstruktion der Architektur aus Quellcode ist sehr komplex, da sich die Architektur in den meisten Sprachen nicht direkt ersichtlich im Code befindet und sich dauerhaft weiterentwickelt. Eine Architektur wird nicht allein durch Klassen und Pakete abgebildet, sondern auch durch den Daten- und Kontrollfluss. Der Sourcecode und die Anwendungsausführung sind eine der wenigen wirklich verlässlichen Informationen, die die aktuelle Architektur enthalten. Durch unterschiedlichste Sprachkomplexe, wie Polymorphismus, späte Bindung, dynamische Bindung zur Laufzeit, Delegation und Vererbung wird die Rekonstruktion sehr komplex und aufwendig. Eine reine automatisierte Rekonstruktion ist im Normalfall aber nicht ausreichend, da die Beweggründe für Architekturentscheidungen für das Verständnis meist unerlässlich sind. Eine reine Rekonstruktion enthält noch keine Bewertung der Architektur. Eine Bewertung kann mithilfe von Performance Untersuchungen, Pattern Erkennung oder ähnlichem vorgenommen werden. [8]

Eine Architekturrekonstruktion kann unterschiedlich angegangen werden. Eine Methode ist die Cluster-basierte Methode, eine weitere basiert auf der Erkennung von Pattern.

Bei dem Cluster-basierten Ansatzes werden Teile der Anwendung mithilfe einfach berechenbarer Metriken zusammengefasst. Diese Metriken können z.B. den Zusammenhalt beschreiben. Das Ergebnis ist eine Menge von Komponenten (z.B. Subsysteme oder Module) und deren Verbindungen untereinander. Der Cluster-basierte Ansatz, hat den Nachteil, dass die Abstraktion durch die Metriken sehr hoch wird. Außerdem können nicht alle Architektur relevanten Informationen mithilfe von Metriken entdeckt werden. Strukturen können erkannt werden, aber nicht deren Zweck. Erkenntnisse zum Beispiel zur Kopplung oder Kohäsion können durch die Cluster Methode erkannt werden. Der Vorteil dieser Methode ist, dass sich die Metriken im Vergleich zur Pattern-basierten Methode schnell berechnen lassen. [2]

Eine andere Möglichkeit der Architektur Rekonstruktion ist die Pattern-basierte Rekonstruktion. Pattern sind sich bereits bewährte Lösungsmuster für häufig vorkommende Problemstellungen. Neben Pattern, die eine Lösungsart empfehlen, gibt es auch Anti-Pattern, die Lösungen darstellen, die nicht gewählt werden sollen, weil sie spezielle Probleme mit sich bringen. Das Vorkommen von Anti-Pattern weist auf Code-Smell hin.

> A code smell is a surface indication that usually corresponds to a deeper problem in the system. [9]

Da die Pattern Erkennung erheblich mehr Daten, wie z.B. den Kontroll- und Datenfluss benötigt ist die Analyse langsamer im Vergleich zur Cluster-basierten Variante. Pattern-basierte Architekturrekonstruktion hat das Ziel vordefinierte Muster und Verhaltens Strukturen zu erkennen.[2]

Durch eine Analyse können Strukturen erkannt werden und bekannten Mustern (Pattern) zugeordnet werden. Sie erhöhen das Verständnis für die Software, da dadurch die Beweggründe des Entwicklers deutlicher werden. Pattern haben sich im Normalfall bereits mehrfach bewiesen, außerdem sind deren Vor- und Nachteile bekannt oder können nachgelesen werden. Eine Diskussion unter Entwicklern wird durch Pattern vereinfacht, da diese grob im Allgemeinen bekannt sind und nicht im Detail besprochen werden müssen. Die Architektur wird dadurch leichter verständlich. Arcelli und Fontana [3] haben herausgefunden, dass die meisten Tools, die die Möglichkeit zur automatisierten Erkennung von Pattern besitzen nur einen Bruchteil der Pattern der Gang of Four erkennen und bei mittelgroßen bis großen Systemen schlecht skalieren. Außerdem werden häufig falsche, nicht umgesetzte Pattern gefunden (false positive). Sie haben ein Eclipse Plugin (MARPLE - Metrics and Architecture Reconstruction) zur Architektur Rekonstruktion und zur Pattern Extraktion (DPD - Design Pattern Detection) entwickelt. Beide Verfahren gleichzeitig zu verwenden bietet den Vorteil, dass die Ergebnisse der Patternanalyse zur Architekturrekonstruktion verwendet werden können. [3] Neben der Detektion von Design Pattern, können auch Anti Pattern oder Pattern für Bad Smell gezielt gesucht werden. Diese beschreiben schlechte Lösungen für häufig auftretende Problemstellungen. [2]

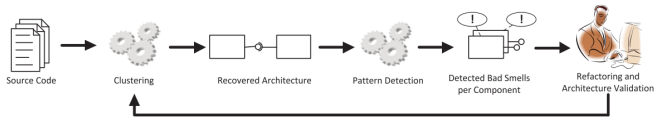


Abbildung 4. Empfohlenes Vorgehen beim Reengineering Prozess [2]

Von Detten und Becker empfehlen eine iterative Kombination von beiden Vorgehen (siehe Abbildung 4) zum Reengineering der Anwendung. Beim schrittweisen Anpassen der Architektur soll immer erst eine Cluster-basierte Rekonstruktion und anschließend, auf dessen Basis, eine Pattern-basierte Rekonstruktion stattfinden. Dies hat den Vorteil, dass die durchführende Person nach und nach eine bessere Kenntnis der Architektur erlangt und die zuvor durchgeführten Änderungen direkt sehen und überprüfen kann. Die Qualität der Software nimmt schrittweise zu. Für eine beschleunigte Durchführung der Pattern Erkennung kann diese nur auf bestimmte Bereiche eingegrenzt werden, wo z.B. die Cluster Erkennung besonders Probleme festgestellt hat. [2] Beim Monitoring werden beide Schritte nur einmal vollständig durchgeführt.

Neben dem Reengineering kann dies während der Entwicklung durchgeführt werden, um schnell einen Fortschritt zu sehen und durch die Berechnungen nicht zu lange von der Entwicklung abgelenkt zu werden.

Die Rekonstruktion nur mithilfe von Source Code (low-level Informationen) wird als Bottom-Up Prozess bezeichnet. Ausgehend vom Quellcode aus werden die Modelle rekonstruiert. Das Top-Down Verfahren ist ein anderer Ansatz. Bei diesem besitzt man am Anfang Informationen über Anforderungen oder Architekturstile. Ziel ist es die Architektur mithilfe von möglichen Hypothesen durch Vergleiche mit dem Source Code zu rekonstruieren. Ein hybrider Ansatz, der eine Kombination des Bottom-Up und des Top-Down Prozesses miteinander kombiniert ist ebenfalls möglich. Hierbei werden beide Prozesse durchgeführt und die Ergebnisse miteinander

verglichen. Dieser Ansatz wird oft zur Vermeidung von Architektur Erosion verwendet. Die konzeptionelle Architektur wird mit der realen Architektur verglichen. [8]

B. Technische Umsetzung und Bibliotheken

Hier werden zwei Open Source Anwendungen betrachtet, die die Analyse von Metriken und das Erkennen von Pattern durchführen können. Die Anwendungen werden teilweise in später vorgestellten Anwendungen eingesetzt.

SoMoX (Software Model Extractor) [10] kann aus vorhandenen Anwendungen Modelle und Strukturen extrahieren und ermöglicht dadurch Analysen der Codequalität. Die Analysen basieren auf der Auswertung unterschiedlicher zuvor berechneter Metriken. Für die Auswertungen können Einstellungen zu den Metriken vorgenommen werden, um sie an das jeweilige Projekt anpassen zu können. Mithilfe einer Blacklist kann Code ausgewählt oder per regulärem Ausdruck bestimmt werden, der bei der Analyse ausgeschlossen werden soll. Die Ergebnisse können als Graph erzeugt werden. Es wurde an dem Karlsruhe Institute für Technologie entwickelt und unterstützt die Sprachen C/C++, Delphi und Java. Die Software wurde so gebaut, dass sie einfach um weitere Sprache erweitert werden kann.

Reclipse [11] ist ein Tool zur Pattern Erkennung. Es wurde an der Universität Paderborn entwickelt. Statische und dynamische Code Analysen werden unterstützt. Zur statischen Analyse werden Patternbeschreibungen in einer speziellen Spezifikationssprache verwendet. Mithilfe von Graph-Matching Methoden werden mögliche Pattern Kandidaten gefunden. Eine anschließend durchgeführte dynamische Analyse kann die Liste der gefundenen Kandidaten verfeinern.

Reclipse beherrscht die Erkennung von Struktur und Verhaltens Mustern. Suchbare Pattern werden mithilfe einer DSL beschrieben. Diese können grafisch modelliert werden.

Beide Tools analysieren den Source Code nicht direkt, sondern verwenden deren Generalisierten Abstrakten Syntax Baum (GAST). SoMoX berechnet Metriken, wie Kopplung oder Kohäsion als Input für den eigentlichen Clustering Algorithmus. Bei erreichen von festgelegten Grenzwerten werden Elemente zu Komponenten zusammengefasst. [2]

V. Vorhandene Software - Toolchain

Dieser Abschnitt stellt bereits vorhandene Anwendungen und Bibliotheken vor. Dazu wurde ein Beispiel Projekt zur Analyse verwendet. Als Testprojekt wurde der Open Source Android Mail Client K9-Mail[3] verwendet. Die Anwendung besteht aus 66 862 Zeilen Code in 367 Dateien mit 4 443 Funktionen. Davon sind 3,8% (3 548 Zeilen) duplizierter Code.[4]

Es wird nur ein Ausschnitt an vorhandenen Systemen gezeigt, viele der Anwendungen wurden nur zu Forschungszwecken entwickelt, waren nie produktiv im Einsatz und wurden seit Jahren nicht weiterentwickelt. Einige der Anwendungen existieren nicht in direkt ausführbarer Form. Deshalb werden nur deren Ideen, die bei der aktuellen Problemstellung hilfreich sein können erwähnt.

Punkte, der Wunschanwendungen, die die vorgestellten Anwendungen im Ansatz oder Teilweise lösen, werden genannt. Pro Anwendung werden zu Beginn ein paar Fakten kurz tabellarisch präsentiert.

A. Sonarqube

Open Source
Sprachen: mehr als 20 durch Plugins
Funktionalitäten durch Plugins erweiterbar
Websystem, Plugins für IntelliJ, Eclipse
wird noch Weiterentwickelt
Löst die Punkte: Sprachenunabhängig, Zugriff, Visualisierung, Kommunizieren/ Vorstellen, Metriken und Analysen, andere Punkte durch Plugin größtenteils lösbar

Sonarqube [12] ist eine Plattform um Codequalitäts Analysen durchzuführen und zu präsentieren. Durch die Möglichkeit sie durch Plugins zu erweitern ist die Anwendung kompatibel zu vielen Programmiersprachen und Anwendungsfällen. Außerdem können zahlreiche unterschiedliche Analysen durchgeführt werden. Einige der im folgenden vorgestellten Anwendungen bieten ebenfalls Plugins für Sonarqube an und stellen deshalb eine gute Kombination dar.
Auf der Startseite bietet Sonarqube ein frei konfigurierbares Dashboard an, welches an die persönlichen Bedürfnisse angepasst werden kann. Neben den Plugins können über eine Rest-Schnittstelle Daten angefragt werden.
Die Anwendung ist aus einem Server und beliebig vielen Clients aufgebaut. Der Server speichert die Ergebnisse der Analysen und bietet ein Webinterface um diese darzustellen. Die Clients werden auf den Systemen mit den Anwendungen ausgeführt. Sie führen die Analysen durch und übertragen sie an den Server. Der Client kann entweder ein Plugin für die Entwicklungsumgebung sein oder ein separater Sonar-Runner. Für den Test wurde der Sonar-Runner verwendet, da dieser unabhängig von der IDE ist. Der Sonar-Runner wird als einfacher Terminalbefehl ausgeführt, dadurch kann dieser auch fest in den Buildprozess integriert werden.

Per Default sind bereits, ohne zusätzliche Plugins, einige Analysen möglich, wie z.B. Test Coverage, Komplexität und das Aufspüren von Duplikaten im Code. Das SQALE Plugin bietet die Darstellung eines Sunbirst (siehe Abbildung 5). Hier wird die Anwendung in verschiedene Oberkategorien, wie z.B. Testbarkeit, Sicherheit, Wartbarkeit, Zuverlässigkeit unterteilt. Diese werden erneut nach Kriterien, wie Exception Handling oder Lesbarkeit unterteilt, und deren Kinder ebenfalls. Die Größe der einzelnen Felder wird durch Kriterien des äußersten Rings, wie z.B. ein öffentlicher Konstruktor(Verständlichkeit - Wartbarkeit) oder falscher Zugriff auf Daten bestimmt. [12]

[3] https://github.com/k9mail/k-9

[4] gemessen mit Sonarqube, siehe V-A

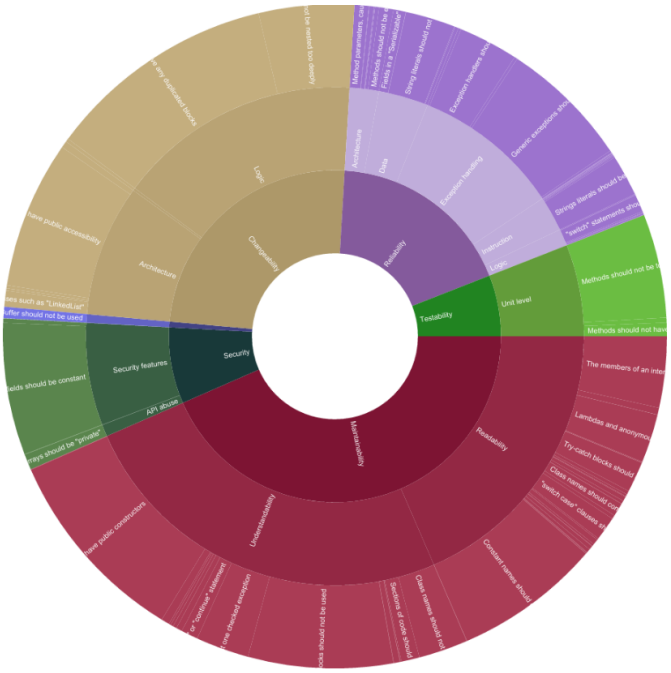


Abbildung 5. Sonarqube SQALE Sunbirst

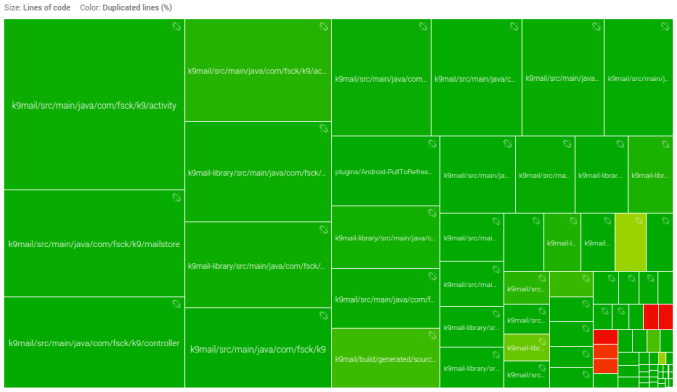


Abbildung 6. Sonarqube Treemap nach Codezeilen (Größe) und dupliziertem Code (Farbe)

B. *Sonargraph*

erwerbliche, zeitbasierte Lizenz, Preis abhängig von der Projektgröße
statische Analysen
Sprachen: Java (C#, C/C++)
lokal installierte Anwendung für Windows, OS X, Linux
wird noch Weiterentwickelt
Löst die Punkte: Gruppieren und Abstrahieren, Abhängigkeiten, Zugriff, Visualisierung, Metriken und Analysen, Pattern und Strukturen

Sonargraph [13] ist ein Produkt-Paket der Firma Hello2Morrow. Die beinhaltenden Anwendungen führen rein statische Analysen (nur anhand von Quellcode, ohne Ausführung) durch. Die Anwendungen können mit einer zeitlich beschränkten Lizenz erworben werden. Die Kosten sind abhängig von der Größe des zu untersuchenden Projekts.

Es besteht aus verschiedenen Programmen, dem Sonargraph-Explorer, Sonargraph-Architekt und Sonargraph Quality. Der Explorer unterstützt bereits die weiteren Programmiersprachen C# und C/C++. Durch das Parsen der Anwendung wird ein Model erzeugt. Metriken, wie die Duplikation von Code, Abhängigkeiten und andere werden berechnet. Benutzerdefinierte Qualitätsrichtlinien können überprüft werden. Der Architekt ermöglicht den Vergleich einer definierten Architektur mit der implementierten Architektur. Eine durchgehende Prüfung der Architektur ist möglich. Bei nicht einhalten zuvor festgelegter Schwellwerte für Metriken können Meldungen aktiviert werden. Komplexe Schritte einer Refaktorisierung können vor der Realisierung simuliert werden. Durch diese Simulation kann geprüft werden, ob das gewünschte Resultat erreicht wird. Es ist möglich mehrere Lösungen miteinander zu vergleichen ohne das diese Umgesetzt werden müssen.

Sonargraph-Quality beinhaltet alle Funktionalitäten des Sonargraph-Architekt. Kann allerdings zusätzlich genauere Analysen durchführen und den Verlauf von mehreren Analysen betrachten. Durch die Kenntnis mehrerer Analysen können Trends in Metriken berechnet werden, welche Probleme seit dem letzten Build gelöst wurden und welche hinzugekommen sind. Queries ermöglichen die Suche nach speziellen Strukturen (z.B. Bad Smells).

Sonargraph kann in Sonarqube integriert werden. Die Analysedaten werden dazu von der Sonargraph Installation zu Sonarqube, mit entsprechend installiertem Plugin, übertragen.

Um eine Anwendung zu Analysieren muss ein Eclipse Projekt oder ein IntelliJ Projekt importiert werden. Das zu untersuchende Projekt kann auch selber definiert werden.

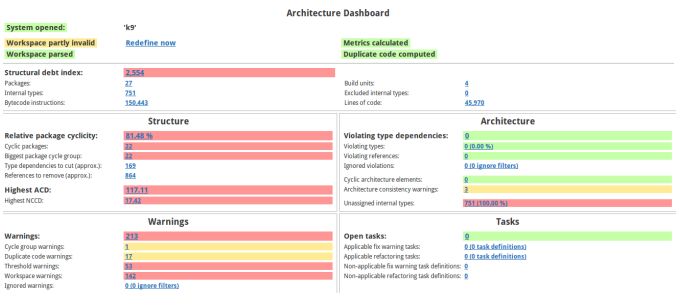


Abbildung 7. Sonargraph Dashboard

Das Importieren aus einem Projekt scheint in den meisten Fällen nicht auszureichen, da die Warnung angezeigt wird, das der Workspace teilweise ungültig ist. Die Fehler bestehen darin, dass beim Importieren definierte Pfade nicht existieren. Auf Anhieb ist nicht ersichtlich, wie diese Fehler zu beheben sind und ob diese relevant für die Analysen sind, da trotzdem Metriken berechnet werden können. Das Sonargraph-Architekt Dashboard (Abbildung 7) ist auf dem ersten Blick sehr unübersichtlich. Es werden einem nur Metriken als Zahlen präsentiert, ohne jegliche Grafiken.

In Abbildung 8 wird ein Abhängigkeitsgraph als gerichteter Graph zwischen Paketen dargestellt. Anfänglich werden alle Beziehungen dargestellt, es ist allerdings möglich einzelne Pakete zu selektieren, damit werden nur Abhängigkeiten markiert, die mit dem Paket in Verbindung stehen. Durch

diese Darstellung kann festgestellt werden, wo zyklische Abhängigkeiten bestehen, die eventuell aufgelöst werden sollten.

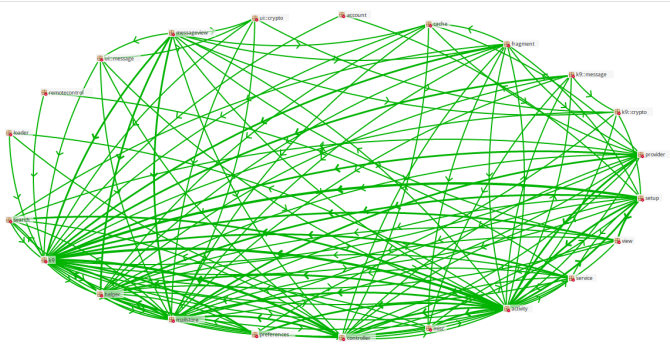


Abbildung 8. Sonargraph Zyklische Paketabhängigkeiten

Abhängigkeiten können in einem zusätzlichen Architektur Bereich betrachtet werden. Dort sind die obersten Pakete und deren Abhängigkeiten untereinander sichtbar. Durch klicken auf einzelne Pakete ist sichtbar, welche Abhängigkeiten ein Paket hat. [13]

C. Codecity

statische Analysen
Java
Eclipse Plugin
Neuentwicklung aus einer rein akademisch nutzbaren Version
wird noch Weiterentwickelt
Löst die Punkte: Visualisierung, Metriken und Analysen

CodeCity [14] stellt eine Anwendung als 3-Dimensionale Stadt dar. Die Höhe und Farbe der Gebäude kann durch unterschiedliche Kriterien bestimmt werden. Dazu gehört die Anzahl der deklarierten Methoden, die Codezeilen, Anzahl der Autoren und Commits und einige weitere. Es wird eine reine Cluster-basierte Analyse durchgeführt.

Die aktuelle Version ist eine Nachimplementierung einer auf Smalltalk basierenden Version von CodeCity[5]. Die damalige Version durfte nur zu rein akademischen Zwecken genutzt werden und ist als eigenständige Anwendung entwickelt worden. Über das neue Eclipse Plugin ist es schwer genauere Informationen über die Lizenz und den aktuellen Stand zu erhalten. Bis auf die Eclipse Plugin Seite gibt es keine offizielle Seite mit Informationen.

Abbild 9 zeigt das Ergebnis für die Android App K9-Mail. Die Fläche der Gebäude stellt dabei die Anzahl der deklarierten Felder innerhalb der Klasse dar. Die Höhe und Farbe wird in diesem Fall durch die Anzahl der deklarierten Methoden bestimmt. Auf jedem Gebäude ist ein Hover-Effekt mit dem alle berechneten Informationen zur Klasse dargestellt werden.

Diese Art der Darstellung stellt die Strukturen sehr gut dar, und kann deutlich machen, welche Klassen häufig Probleme verursachen, weil diese z.B. sehr oft von unterschiedlichen Personen bearbeitet wurden. Durch die Darstellung nach der

[5]http://www.inf.usi.ch/phd/wettel/codecity.html

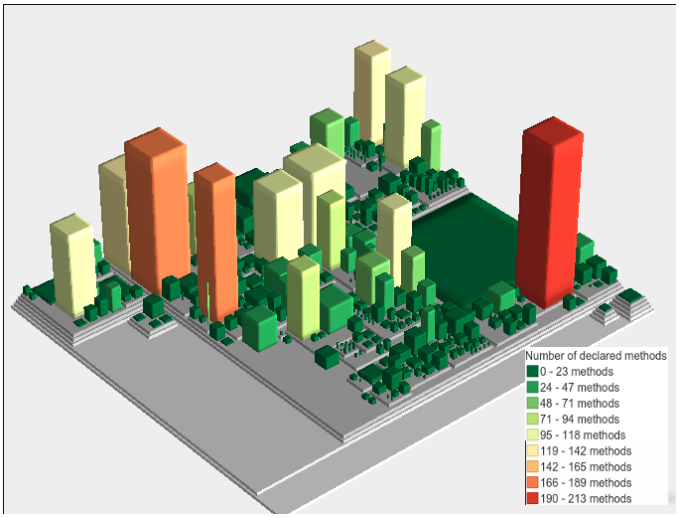


Abbildung 9. Codecity Methoden Deklarationen

zyklischen Komplexität können schnell Klassen erkannt werden, die Performance Probleme verursachen können.

Um die Stadt darzustellen muss diese innerhalb von Eclipse generiert werden. Anschließend wird sie im Browser interaktiv dargestellt. Innerhalb des Browsers gibt es Einstellungen zu den Darstellungskriterien. Es kann definiert werden, wonach sich die Länge und Breite der Häuser orientiert, welche Eigenschaft die Höhe bestimmt und was durch die Färbung dargestellt werden soll. Dies bietet die Möglichkeit drei Unterschiedliche Kriterien innerhalb eines Graphen darzustellen. Ein Nachteil ist allerdings, dass bei Änderungen die Darstellung jedes mal erneut generiert werden muss. Dies dauert in dem verhältnismäßig kleinem Beispielprojekt bereits ein paar Sekunden. Über den Browser können die Daten nicht aktualisiert werden. Die Integration von Metriken in diesem Tool aus dem Versionskontrollsystem ist sehr hilfreich und wird nicht von vielen anderen Tools unterstützt. [14]

D. IntelliJ Code Analyse Funktionen

statische Analysen
Java
Eigenständige erwerbbare IDE, kostenfreie Community-Version vorhanden
wird noch Weiterentwickelt
Löst die Punkte: Zugriff, Abhängigkeiten, Metriken und Analysen

Viele moderne Entwicklungsumgebungen bieten Funktionen zur Codeanalyse. Die Codeanalyse von IntelliJ [15] untersucht den Code neben Compilierfehler auf unerreichbaren Code, nie ausführbaren Code, nicht auflösbare Methoden, Memory Leaks und auch Rechtschreibfehler. Für die Analysen können Analyseprofile ausgewählt werden, diese können pro Projekt individuell definiert werden. Analysen können auf dem gesamten Projekt durchgeführt werden, auf einzelne Dateien oder Ordner. Compilierfehler werden, wie bei den meisten IDEs direkt beim Schreiben analysiert und angezeigt. Die manuell durchgeführten Analysen werden als Liste präsentiert, sortiert nach Fehlerart (z.B. Compilerfehler, Fehlerbehand-

lung). Eine grafische Darstellung gibt es nicht. Diese Liste kann allerdings trotzdem hilfreich sein, um Schwachstellen zu finden und auszubessern.

Die IDE bietet auch die Möglichkeit einer Abhängigkeitsanalyse. Die Abhängigkeiten werden als Liste mit drei Fenster dargestellt. In der ersten ist die normale Struktur der Anwendungen dargestellt. Hier wählt man eine entsprechende Datei aus, die analysiert werden soll. In dem zweiten Fenster erscheinen daraufhin die davon abhängigen Dateien. Bei der Auswahl einer Datei aus dem zweiten Fenster, werden im Dritten daraufhin die genaue Abhängigkeit dargestellt (z.B. anlegen eines Objektes).

Diese ist im Vergleich zu den anderen hier vorgestellten Tools sehr unübersichtlich und liefert nur bei sehr genauer Betrachtung nützliche Informationen. Um sich einen groben Überblick über die Anwendung zu verschaffen eignet sich diese Möglichkeit nicht. Grundlegende Probleme können so aber durch ein durcharbeiten der Liste behoben werden.

E. CodeCrawler

Open Source
Java Anwendung für Tomcat
seit 2005 nicht weiterentwickelt
Löst die Punkte: Sprachen unabhängig, Zugriff, Kommunizieren/ Vorstellen, Metriken und Analysen

CodeCrawler [16] ist eine webbasierte Anwendung um Quellcode zu durchsuchen. Bei der Analyse jeder einzelnen Datei wird ein Suchindex aufgebaut und semantisch wichtige Informationen werden gesammelt. Die indexierte Analyse kann über eine Webschnitte durchsucht werden. Die Ergebnisse werden nach Relevanz oder dem letztem Änderungsdatum sortiert und verweisen auf die entsprechenden Stellen im Quellcode. Die Relevanz wird durch semantische Faktoren, wie etwa Klassennamen sind relevanter als Variablennamen bestimmt. Die Suche wird durch zusätzliches Wissen über die Programmiersprache erweitert. Dadurch wird eine intelligentere Suche ermöglicht. Bei der Suche kann explizit nach Klassennamen, Dateinamen oder ähnlichem gesucht werden. Die Suche wird mithilfe eines Queries gebildet und kann dadurch beliebig komplex werden. CodeCrawler kann die Codebasis selbstständig zeitgesteuert aus einem Repository herunterladen und die Analysen und Indizierungen aktualisieren. Es wird allerdings nur CVS, WebDAV und lokale Dateien unterstützt.

CodeCrawler kann zusätzlich zur Visualisierung der Daten verwendet werden. Die Abhängigkeiten können ähnlich wie in Abbildung 8 als Graph dargestellt werden. Die Komplexität einer Anwendung wird durch Kästchen unterschiedlicher Größe präsentiert. Die Breite ergibt sich aus der Anzahl der Attribute, die Höhe aus der Anzahl der Methoden und die Farbe anhand der Codezeilen. In Abbildung 10 ist eine System Hotspot Sicht dargestellt. Die Größe wird hier nach der Anzahl der Methoden bestimmt und die Farbe nach der Stärke der Einbindung in die Hierarchie. [17] Die Anwendung wurde für Entwickler mit Kenntnissen zu Programmiersprachen und Regulären Ausdrücken entwickelt.

Die Anwendung wird als Websystem auf einem Tomcat Server deployed. Es wurde allerdings seit 2005 nicht weiterentwickelt.

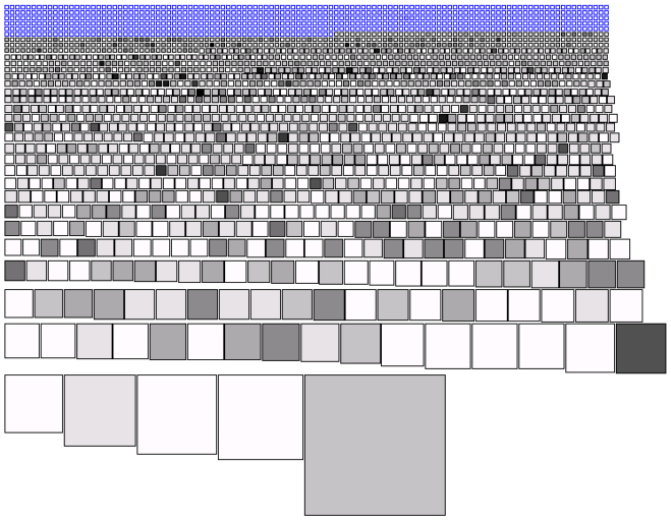


Abbildung 10. CodeCrawler System Hotspot [17]

F. SHrIMP/Creole

Open Source
Sprachen: Java#
lokal installierte Anwendung für Windows, OS X und Linux, Plugin für Eclipse, Sonar, Intellij, Web Anwendung
letztes Update 2013
Löst die Punkte: Gruppieren und Abstrahieren, Abhängigkeiten, Zugriff, Visualisierung, Metriken und Analysen

SHrIMP (Simple Hierarchical Multi-Perspective)[18] und Creole [19] sind die gleiche Anwendung. Creole bezeichnet dabei die Version als Eclipse Plugin.
In der Hauptansicht verwendet die Anwendung eine hierarchische Ansicht um Strukturen darzustellen. In die Ansicht kann hereingezogenen werden, um Details zu betrachten. Dadurch werden irrelevante Informationen ausgeblendet und für diesen speziellen Bereich vorhandene Kontextinformationen eingeblendet. Die Software Hierarchien werden durch verschachtelte Graphen dargestellt (Pakete beinhalten weitere Pakete, Klassen, Interfaces, usw.[6]). Je nach Konfiguration, kann der Graph auch als Vererbungshierarchie dargestellt werden. Zusätzliche Beziehungen, wie z.B. extends (Vererbung) oder implements werden über Pfeile repräsentiert. SHriMP kann mit JavaDoc interagieren und entsprechende Stellen in den Graphen automatisch verlinken und präsentieren. Das Tool wurde so designed, dass es einfach für die Zusammenarbeit mit anderen Tools angepasst werden kann.[20] [21]
Durch die Integration von Source Code und Dokumentation innerhalb einer Ansicht wollen sie die Effektivität beim durchsuchen und kennenlernen der Architektur erleichtern. Zu entsprechendem Sourcecode ist direkt ersichtlich, wie die umliegende Architektur aussieht und zur visuellen Architektur ist ersichtlich, wie die Implementierung aussieht.[21]

[6]Eltern - Kind Beziehung

G. Understand

Kommerziell
statische Analysen
Eigenständige IDE
Sprachen: Java, Ada, Cobol C, C#, uvw.
Linux, OS X, Windows
wird noch weiterentwickelt
Löst die Punkte: Gruppieren und Abstrahieren, Sprachen unabhängig, Abhängigkeiten, Zugriff, Entwicklung, Visualisierung , Metriken und Analysen

Understand ist eine von Grund auf neu entwickelte Entwicklungsumgebung. Sie muss kommerziell erworben werden. Die IDE bietet typische Informationen zu verwendeten Methoden, Klassen und Variablen (Wo und Wie werden sie verwendet) wie andere IDEs an. Metriken, wie Codezeilen oder Komplexität werden automatisiert erzeugt. Qualitätsreports werden erzeugt, die z.B. duplizierten Code, oder eventuell problematische Abschnitte hervorheben. Visuell kann die Anwendung mithilfe von Treemaps (wie Abbildung 6), UML Klassen Diagramme, Sequenzdiagramme, Kontrollfluss Diagramme, Hierarchiegraphen (siehe Abbildung 11), Deklarationsgraphen (z.B. wer ruft eine Methode auf, Was wird innerhalb der Methode aufgerufen, Welche Parameter werden in einer Methode verwendet) und Abhängigkeitsgraphen dargestellt werden. [22]

Abbildung 11 zeigt den Architekturgraphen der Paketstruktur. Dieser kann einen schnellen Einblick auf die Strukturierung der Anwendung geben. Der in Abbildung 12

Abbildung 11. Understand Architektur Graph

dargestellte Abhängigkeitsgraph, zeigt alle Verknüpfungen unter den einzelnen Programmteilen. Er kann beliebig detailliert dargestellt werden. Initial werden nur die Pakte dargestellt, in diese Ansicht kann bis zur untersten Ebene gezoomt werden. Es ist sehr Vorteilhaft, dass einzelne Nodes komplett, inklusive aller ihrer Verknüpfungen ausgeblendet werden können. Ein

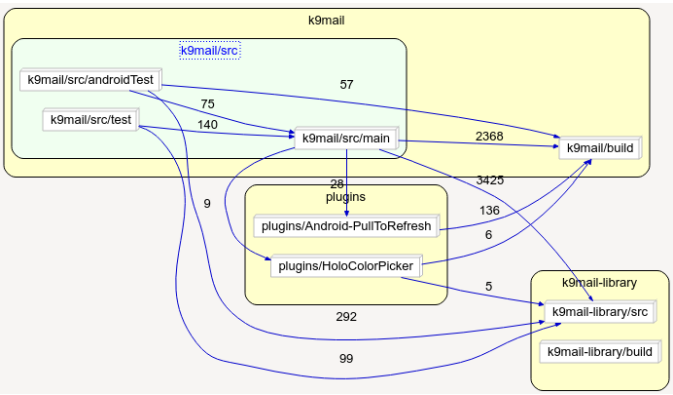


Abbildung 12. Understand Abhängigkeitsgraph

weiteres interessantes Feature ist der Vergleich von Soll und Ist-Architektur um Verstöße festzustellen.

Diese Anwendung scheint die Modellierung von Anwendungen und deren Überprüfung/Monitoring bereits gut zu unterstützen und zahlreiche reichlich Analyse und Visualisierungsmöglichkeiten zur Verfügung.

H. gql4jung/ JUNG

Open Source
Sprachen: Java
letzte Weiterentwicklung 2010
Löst die Punkte: Abhängigkeiten, Visualisierung, Metriken und Analysen

Die Anwendung gql4jung [23] verwendet das Framwork JUNG (Java Universal Network/Graph Framework) [24]. JUNG bietet eine erweiterbare Sprache an, um Daten, die als Graph abgebildet sind zu modellieren, analysieren und zu visualisieren. Die Bibliothek unterstützt unterschiedliche Darstellungsarten, wie z.B. gerichtete und ungerichtete Graphen. JUNG ist ebenfalls unter einer Open Source Lizenz veröffentlicht worden. An jeden Graphen, jeder Entität und jede Beziehung können Metadaten angehängt werden. Es wurde seit 2010, mit der Veröffentlichung von Version 2.0.1, nicht weiterentwickelt. Algorithmen zur Graphen Theorie (z.B. Pfadsuche), Data Mining (Suchen von Mustern in großen Datensätzen, Erkennung von Trends), statistische Analysen und weitere sind bereits in der Anwendung implementiert. Für die Visualisierung kann ein mitgelieferter Algorithmus verwendet werden. Das Framework unterstützt aber auch die Entwicklung einer eigenen Darstellung und stellt dazu Möglichkeiten zum Filtern der Informationen zur Verfügung.

Das gql4jung Projekt erweitert das JUNG Framework um eine Query Sprache für Graphen. Die Queries werden in XML definiert. Dadurch können Queries zum finden von (Anti-) Pattern definiert erzeugt werden. Es werden Graphen im GraphML [25] Format unterstützt, außerdem können Graphen aus Java Code extrahiert werden. GraphML ist ein Open Source Format auf XML Basis. Das Format wurde ebenfalls seit 2007 nicht weiterentwickelt. Zum Extrahieren aus bereits compiliertem Java Code wird der Dependency Finder [26] verwendet. Dies ist ein Tool, dass aktuell weiterentwickelt wird und zur Analyse von compiliertem Java Code verwendet werden kann. Neben einer Comandozeilen basierten Version, existiert auch eine grafische Oberfläche. Teil dieser Anwendung ist das Tool JarJarDif, dass Unterschiede zwischen zwei Implementierungen feststellen kann.

I. PMD

Open Source (BSD Lizenz)
Plugins für Maven, Eclipse, NetBeans, InteliJ, ...
Sprachen: Java, JavaScript, PLSQL, Apache Velocity, XML, XSL
wird noch Weiterentwickelt
Löst die Punkte: Referenzen, Metriken und Analysen

PMD [27] kann Quellcode Analysen durchführen um mögliche Problemstellen ausfindig zu machen. Es wird nach möglichen Bugs gesucht, indem unter anderem nach leere try-catch Blöcken, ungenutzten Variablen und Parametern oder privaten Methoden gesucht wird. Eine überflüssige Verwendung von

Strings, if Anweisungen oder Schleifen kann entdeckt werden, sowie duplizierter Code. Die Analysen werden mithilfe von Regelwerken durchgeführt. Um die Anzahl der Ergebnisse zu begrenzen kann nach einzelnen speziellen Regeln (z.B. nur nach ungenutztem Code suchen) analysiert werden. Eine weitere Möglichkeit ist aus einem Regelwerk einzelne Regeln gezielt auszuschließen. Bei Ausführung kann eine HTML Liste mit Änderungsvorschlägen erstellt werden. Bei der Analyse muss ein entsprechendes Regelwerk angegeben werden. In dem Dokument werden Änderungsvorschläge, wie

> Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.

,

> Avoid if (x != y) ..; else ..;

oder

> No need to check for null before an instanceof

erstellt.

An den Beispielen kann man sehen, dass sowohl Vorschläge für ein besseres Verständnis (unnötige Negation), aber auch Vorschläge für Performance Verbesserungen (überflüssige Abfragen und Vergleiche) gemacht werden. Bei jeder Meldung ist ein Link hinterlegt, der auf eine Github Seite mit einer Erläuterung und Beispielen verweist. PMD wird aktuell noch gepflegt und weiterentwickelt und bietet für viele IDEs ein Plugin an. Dadurch ist die Verwendung unkompliziert und kann ohne Umwege in die Entwicklung integriert werden.

J. Archimetrix - Iterative Architecture Recovery and Reengineering

Open Source
Sprachen: Java, C++, Delphi
Löst die Punkte: Gruppieren und Abstrahieren, Abhängigkeiten, Pattern und Strukturen seit 2014 nicht Weiterentwickelt

Archimetrix [28] wurde an der Uni Paderborn entwickelt. Neben der Architektur Rekonstruktion durch die Clusterbasierte Methode bietet es die Möglichkeit der Pattern Erkennung an. Defizite innerhalb von Komponenten und der Architektur können erkannt werden. Bei der Analyse gefundene Defizite werden nach der Stärke der Architektur Beeinflussung sortiert. Ein schrittweise Behebung der Probleme, mit häufiger Aktualisierung der Analyse wird empfohlen. Zum Beheben der Probleme werden Empfehlungen angezeigt, bestimmte Reengineering Strategien können dabei automatisiert durchgeführt werden. Resultate der automatisierten Änderungen können zuvor in einer Vorschau betrachtet werden.

Zur Architektur Rekonstruktion geht die Anwendung in zwei Schritten vor. Im ersten Schritt wird ein generalisierter abstrakter Syntax Baum (GAST) mithilfe des Parsers SISSy (Structural Investigation of Software Systems) [29] erzeugt. Zusätzlich zum Parsen des Codes kann Sissy problematische Pattern identifizieren (z.B. Interface-Verletzungen). Mithilfe von SISSy können außerdem Metriken, wie z.B. zum Zugriff auf externe Daten oder der durchschnittlichen Komplexität aller Methoden einer Klasse berechnet werden. Im zweiten Schritt wird mit SoMoX (siehe IV-B) die Architektur rekonstruiert.

K. Web of Patterns - The Pattern X-Ray

Open Source
Architektur Rekonstruktion und Pattern Erkennung
Sprachen: Java, C++, Delphi
seit 2007 nicht weiterentwickelt
Löst die Punkte:Gruppieren und Abstrahieren, Pattern und Strukturen

Das Pattern X-Ray ist als Eclipse Plugin entwickelt worden und ein Bestandteil des Web of Patterns Projekt [30]. Es beinhaltet ein sprachen neutrales Format, um Design Pattern zu beschreiben. Das Format basiert auf RDF[7] und OWL[8]. Die Pattern werden mithilfe des Apache Jena Frameworks[9] eingelesen. Die Pattern Definitionen können von einem Repository heruntergeladen werden. Die Ergebnisse werden nach Ähnlichkeiten zusammengefasst und können als XML exportiert und gespeichert werden. Um Pattern zu finden werden deren gespeicherten Definitionen in Constraints umgewandelt, diese werden versucht mithilfe des GAST zu lösen. Eine Bewertung der Pattern durch die Benutzer ist möglich. Die Bewertungen werden auf den Server übertragen und sind von allen einsehbar. Mithilfe eines Wizzards können eigene Pattern definiert werden, die Formal korrekt im RDF Format generiert werden. Diese können wieder für andere Veröffentlicht werden.

Seit 2007 wurde dieses Projekt nicht weiterentwickelt.

Die Idee eine Community für Pattern einzurichten ist eine gute Idee um verschiedene Meinungen über Lösungen zu erfahren und weiterzuentwickeln.

L. Weitere Tools

Es existieren zahlreiche weitere Tools, die die Architekturentwicklung unterstützen und versuchen die Entwicklung allgemein und die Anwendung selbst effizienter und weniger Fehleranfällig zu machen. Doxygen [31] ist hauptsächlich dazu gedacht Dokumentation anhand von Kommentaren zu erzeugen. Es kann aber auch aus unkommentiertem Quellcode Strukturen extrahieren. Dieses Tool eignet sich gut um schnell Dokumentation anhand von JavaDoc (oder äquivalentem in anderen Sprachen) zu generieren.

Weitere, während der Recherche gefundenen Tools, die allerdings nicht weiter untersucht wurden sind JDepend [32], Structure101 [33], Ndepend [34], SolidSX [35], Enterprise Architect [36], Rational Software Architect [37], Bauhaus [38], Sotograph [39], Swagkit [40], Lattix [41] und Marple - Metrics and Architecture Reconstruction Plug-in for Eclipse [42]

Der Großteil dieser Tools konzentriert sich auf die Rekonstruktion von Architektur und nicht auf deren initiale Entwicklung und Weiterentwicklung. Die genannten Tools Enterprise Architect und Rational Software Architect stellen eine Ausnahme dar, diese unterstützend die reine Modellierung der Architektur, dafür aber nicht deren Rekonstruktion.

[7] http://www.w3.org/RDF/

[8] http://www.w3.org/TR/owl-features/

[9] http://jena.apache.org/

VI. Eigener Lösungsansatz

Im Rahmen des Hauptseminars und des Hauptprojektes soll mit der Umsetzung einer eigenen Anwendung begonnen werden. Dazu müssen verschiedene Entscheidungen getroffen werden. Am nützlichsten ist ein Plugin für eine IDE, wie IntelliJ oder Eclipse. Die Einbindung in Monitoring- und Builtsystemen, wie Sonarqube oder Jenkins ist sehr hilfreich.

Da alle Anforderungen auf einmal nicht umsetzbar sind, soll sich vorerst auf ein paar Anwendungsfälle konzentriert werden. Die resultierende Anwendung soll anfänglich vor allem das Bewusstsein für die Architektur fördern. Deshalb soll sie im ersten Schritt hauptsächlich als Monitoring Tool dienen, Metriken berechnen und eine grobe Architekturrekonstruktion bieten.

Da die Hauptfunktionalität der Darstellung des aktuellen Zustandes dienen soll, wird die Anwendung so gebaut, dass sie als eigenständiges System laufen kann. Die Analysen können entweder mithilfe eines Buildservers oder lokal auf einem PC der Entwickler durchgeführt werden. Um den Entwicklungsaufwand einzuschränken, soll vorerst nur Java als einzige Programmiersprache unterstützt werden. Für diese Sprache existieren bereits viele Tools und Bibliotheken, die eventuell wiederverwendet werden können.

A. Vorgehen

Am Anfang der Entwicklung soll eine Möglichkeit zum Parsen des Sourcecodes umgesetzt werden, um auf dem Resultat grobe Analysen durchführen zu können. Der Sourcecode wird in eine andere Darstellung, wie z.B. dem generalisiertem Syntax Baum umgewandelt. Dazu muss praktisch untersucht werden, welche vorhandenen Bibliotheken dazu eingesetzt werden können. In dieser Phase wird nur der Source Code oder der compilierte Code als Input dienen. Hier können die Tools aus gq4ljung (JUNG, Dependency Finder) verwendet werden. Es handelt sich um einen reinen Bottom-Up Prozess, Nach dem erfolgreichen Parsen der Daten sollen Metriken mithilfe eines Cluster-basierten Verfahren berechnet werden. Hierbei kann z.B. dass vorgestellte SoMoX (siehe Kapitel IV-B) verwendet werden. Es soll mit Metriken zu dupliziertem Code, zyklomatischer Komplexität und Abhängigkeiten begonnen werden. Die Ergebnisse werden zunächst als Liste ausgegeben und erst im anschließenden Schritt visualisiert. Bei der Visualisierung soll mit einer TreeMap begonnen werden. Diese Darstellung erscheint für den Anfang am simpelsten und für viele unterschiedliche Metriken einsetzbar zu sein, um ein schnelles Resultat zu erzeugen. Für die Abhängigkeiten muss eine graphbasierte Darstellung implementiert werden. Diese Visualisierungen sollen zunächst nicht interaktiv sein. Für eine universell einsetzbare Visualisierung soll diese innerhalb eines Browsers als Webpage dargestellt werden. Dadurch ist es möglich, diese sowohl für ein Monitoring System zu verwenden, wie auch lokal auf den Rechnern der Entwickler. Die Darstellung soll in das vorhandene Sonarqube (siehe Kapitel V-A) Tool integriert werden. Dadurch kann sich auf die Entwicklung des Plugins konzentriert werden. Der Aufwand für die Entwicklung von nicht direkt relevanten Aufgaben (wie Absicherung, Webserver, Administration) wird dadurch minimiert. Die Analysen werden per Skript durchgeführt und an Sonarqube übermittelt.

Im zweiten Schritt sollen die berechenbaren Metriken erweitert werden. Die Visualisierungen sollen durch weitere Darstellungsmöglichkeiten und Funktionalitäten ergänzt werden. So kann bei der graphbasierten Darstellung hinzukommen, dass einzelne Nodes ausgeblendet und wieder eingeblendet werden können. Die Darstellung der TreeMap kann während des Betriebs durch andere Kriterien bestimmt werden.

Im dritten Schritt soll die Pattern-basierte Analyse integriert werden. Dazu muss zunächst eine geeignete Repräsentation für die Vergleichspattern gefunden werden. Dies kann ähnlich wie das Format im Web of Patterns Tool aussehen. Die Pattern Analyse kann mit dem Reclipse Tool umgesetzt werden (siehe Kapitel IV-B).

B. Weitere Entwicklung- und Forschungsmöglichkeiten

Die derzeit geplanten Features bilden nur sehr begrenzt die Möglichkeiten der Softwareanalyse und Architekturrekonstruktion dar. Nur wenige der am Anfang erarbeiteten Wunschfunktionen stehen zur Verfügung. Wie die Analyse bereits vorhandener Anwendungen gezeigt hat, sind noch einige weitere Analysen und Visualisierungen, wie z.B. die Sunbirst Variante möglich.

Die Einbeziehung weiterer Informationsquellen ist ebenfalls sehr wünschenswert. Dies kann z.B. die Versionskontrolle sein. Dadurch kann festgestellt werden, welche Dateien besonders häufig geändert werden oder wie viele unterschiedliche Personen an den gleichen Komponenten gearbeitet haben.

Eine weitere Quelle, wie das Ticketsystem kann eingebunden werden um Informationen zu Tickets, Personen, Diskussionen und z.B. einem Gesprächsverlauf direkt mit dem Quellcode zu verknüpfen.

Die Planung, Weiterentwicklung und Evaluierung der Architektur wurde in dieser Arbeit nur am Rande betrachtet, ist für eine erfolgreiche und effiziente Architektur aber erforderlich. Die vorgestellten Tools haben zum Großteil nur die Rekonstruktion beherrscht, eine Möglichkeit der Modellierung gab es zusätzlich nur bei Sonagraph. Die nicht weiter vorgestellten Tools Enterprise Architect und Rational Software Architect beherrschen die Architektur Modellierung, bieten aber nicht die Möglichkeit der automatisierten Rekonstruktion.

Funktionalitäten, die die Zusammenarbeit aktiv Fördern wurden bisher nicht betrachtet. Dazu gehört die einfache Möglichkeit, an sämtlichen Stellen für alle verfügbar Annotationen hinzufügen zu können. Ebenfalls zu diesem Bereich gehört die Bewertung von Pattern, mit Verweisen auf mögliche Ansprechpartner und deren Vor- und Nachteilen. Eine Diskussion über das Anfordern eines Reviews kann in zukünftigen Arbeiten einbezogen werden.

Durch die Wahl von Java als derzeit einzige unterstütze Sprache sind die Verwendungsmöglichkeiten für die Anwendung sehr eingeschränkt. Die Erweiterung auf andere weit verbreitete Sprachen würde die Nützlichkeit der Anwendung sehr erhöhen und den möglichen Nutzerkreis stark erweitern.

VII. Zusammenfassung und Bewertung

Die Architektur einer Anwendung ist sehr wichtig und trägt zu einem großen Teil zum Erfolg eines Projektes bei.

Existierende Entwicklungsframeworks und Tools unterstützen Tätigkeiten in Bezug auf die Architekturentwicklung nur sehr gering. Architekturdebatten benötigen ebenfalls einen technischen Abschnitt mit Implementierungsdetails. Agile Frameworks, wie Scrum konzentrieren sich allerdings primär auf fachliche Aufgaben.

Die Unterstützung der Architekturentwicklung ist sehr hilfreich und wie die Umfrage gezeigt hat, von den Entwicklern gewünscht. Durch eine Unterstützung können die Fehlerquellen verringert werden. Derzeit werden allerdings in nur sehr wenigen Projekten automatisierte Tools verwendet. Derzeit existieren nur wenige Produkte, die Entwicklung von Anfang, bis zum Ende unterstützen (Planung, Entwicklung, Wartung). Diese sind meist sehr komplex und teuer in der Anschaffung, bei sehr unterschiedlichem Leistungsumfang.

Bereits die reine Rekonstruktion ist sehr hilfreich um das Bewusstsein für die Architektur zu stärken oder auf möglichst simple Weise den aktuellen Stand kommunizieren zu können. Einfach berechenbare Metriken können als Grundlage für eine Diskussion dienen und Hinweise bieten, um die Qualität der Software zu erhöhen. Derzeit werden solche Tools allerdings selten produktiv verwendet, sondern dienen meist rein Forschungszwecken. Dadurch sind viele Anwendungen veraltet und wurden seit Jahren nicht gepflegt und weiterentwickelt. Vorhandene Tools bieten meist nur eingeschränkte, sehr unterschiedliche Funktionen. Die Verwendung ist in einigen Fällen komplex und erfordert eine längere Einarbeitungsphase. Ausnahme bietet das Understand Tool (siehe Kapitel V-G). Dieses stellt gute Analysemöglichkeiten, integriert in einer IDE zur Verfügung. Diese IDE ist allerdings nur kommerziell erwerbbar und bietet keine Community Version oder ähnliches für einzelne Entwickler an. Die Sonargraph Anwendung bietet die Möglichkeit um Refaktorisierungen zu simulieren. Dies kann sehr hilfreich bei großen, schnell unübersichtlichen Änderungen sein. Eine Schrittweise Durchführung, ohne das die Lauffähigkeit der Anwendung beeinträchtigt wird, ist dadurch möglich. Ebenfalls kann geprüft werden, ob die Änderungen theoretisch den gewünschten Erfolg bieten. Dies spart Zeit und Kosten.

Die Entwicklung einer weiteren Anwendung scheint sinnvoll, dabei sollten die Kenntnisse aus früheren Untersuchungen verwendet werden und wenn möglich verwendete Bibliotheken aktualisiert werden. Dadurch können individuelle Anforderungen umgesetzt werden. Zu den neuen Anforderungen zählt unter anderem die Verwendung von unterschiedlichen Informationsquellen neben dem Source Code, wie dem Versionskontrollsystem oder dem Ticketsystem. Möglichkeiten, die Zusammenarbeit aktiv durch soziale Funktionalitäten zu unterstützen, bringen einen großen Vorteil, um das Wissen zu verbreiten und gleichzeitig festzuhalten. Dazu gehört das Anfordern von Reviews, Verweise auf im Unternehmen ähnliche Implementierungen, inklusive Ansprechpartner, Abstimmungen und Bewertungen von Pattern. Das Fachwissen wird im Unternehmen besser verbreitet und bleibt nicht nur innerhalb einzelner Abteilungen oder Projekten. Eine Voraussetzung ist allerdings, dass genügend Mitarbeiter dazu beitragen.

Da der Aufwand für alle Anforderungen sehr hoch ist, muss die Anwendung Schrittweise umgesetzt und erweitert werden.

REFERENZEN

[1] Standish Group, "CHAOS Report," Tech. Rep., 2014. [Online]. Available: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf

[2] M. von Detten and S. Becker, "Combining clustering and pattern detection for the reengineering of component-based software systems," in Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS - QoSA-ISARCS '11. New York, New York, USA: ACM Press, 2011, p. 23. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-79960533364\&partnerID=40\&md5=375602d02a2ac11320cc460b050ade38$\backslash$nhttp://portal.acm.org/citation.cfm?doid=2000259.2000265

[3] F. Arcelli Fontana and M. Zanoni, "A tool for design pattern detection and software architecture reconstruction," Information Sciences, vol. 181, no. 7, Apr. 2011, pp. 1306–1324. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0020025510005955

[4] M. Von Detten, "Archimetrix: A tool for deficiency-aware software architecture reconstruction," in Proceedings - Working Conference on Reverse Engineering, WCRE, 2012, pp. 503–504.

[5] A. Telea, L. Voinea, and H. Sassenburg, "Visual Tools for Software Architecture Understanding: A Stakeholder Perspective," IEEE Software, vol. 27, no. 6, Nov. 2010, pp. 46–53. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5518754

[6] J. Van Eyck, N. Boucké, A. Helleboogh, and T. Holvoet, "Using code analysis tools for architectural conformance checking," Proceeding of the 6th international workshop on SHAring and Reusing architectural Knowledge - SHARK '11, 2011, p. 53. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1988676.1988687

[7] J. Aldrich, "Using types to enforce architectural structure," in 7th IEEE/IFIP Working Conference on Software Architecture, WICSA 2008, 2008, pp. 211–220.

[8] 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

[9] M. Fowler, "Code Smell," 2006. [Online]. Available: http://martinfowler.com/bliki/CodeSmell.html

[10] Karlsruhe Institute für Technologie, "SoMoX," 2015. [Online]. Available: https://sdqweb.ipd.kit.edu/wiki/SoMoX

[11] Uni Paderborn, "Reclipse," 2015. [Online]. Available: https://code.google.com/p/reclipse-emf/

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

[13] Hello2Morrow, "Sonargraph," 2015. [Online]. Available: https://www.hello2morrow.com/products/sonargraph

[14] Codetrails, "Codecity," 2015. [Online]. Available: https://marketplace.eclipse.org/content/codecity

[15] JetBrains, "ntelliJ IDEA 14.1.0 Help - Code Analysis," 2015. [Online]. Available: https://www.jetbrains.com/idea/help/code-analysis.html

[16] University Illinois-Urbana Champaign, "CodeCrawler," 2005. [Online]. Available: http://codecrawler.sourceforge.net/

[17] M. Lanza and S. Ducasse, "CodeCrawler–An Extensible and Language Independent 2D and 3D Software Visualization Tool," in Tools for Software Maintenance and Reengineering, 2005, pp. 74–94. [Online]. Available: http://scg.unibe.ch/archive/papers/Lanz05bCCBookChapter.pdf

[18] Chiselgroup, "SHriMP," 2013. [Online]. Available: http://thechiselgroup.org/2003/07/06/shrimp-views/

[19] ——, "Creole," 2013. [Online]. Available: http://thechiselgroup.org/2003/07/06/creole/

[20] M.-a. Storey, C. Best, and J. Michand, "Shrimp views: An interactive environment for exploring java programs," Program Comprehension, 2001. ..., 2001, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs$\backslash$_all.jsp?arnumber=921719

[21] M.-a. Storey, "SHriMP Views: An Interactive Environment for Exploring Multiple Hierarchical Views of a Java Program," 2002.

[Online]. Available: https://chiselgroup.files.wordpress.com/2012/07/10-1-1-94-8538.pdf

[22] Scitools, "Understand," 2015. [Online]. Available: https://scitools.com/

[23] J. Dietrich, "gql4jung," 2010. [Online]. Available: https://code.google.com/p/gql4jung/

[24] T. J. F. D. Team, "JUNG - Java Universal Network/Graph Framework," 2010. [Online]. Available: http://jung.sourceforge.net/

[25] Graph Drawing Steering Committee, "GraphML," 2007. [Online]. Available: http://graphml.graphdrawing.org/

[26] J. Tessier, "Dependency Finder," 2015. [Online]. Available: http://depfind.sourceforge.net/

[27] PMD, "PMD," 2015. [Online]. Available: https://pmd.github.io/

[28] Uni Paderborn, "Archimtrix," 2013. [Online]. Available: https://code.google.com/p/archimetrix/wiki/Introduction

[29] SQools, "SISSy," 2012. [Online]. Available: http://sissy.fzi.de/sissy/

[30] J. Dietrich, "The Web of Patterns Project," 2007. [Online]. Available: http://www-ist.massey.ac.nz/wop/

[31] Doxygen, "Doxygen," 2015. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/index.html

[32] I. Clarkware Consulting, "JDepend," 2009. [Online]. Available: http://clarkware.com/software/JDepend.html

[33] Structure101, "Structure101," 2015. [Online]. Available: http://structure101.com/

[34] P. Smacchia, "ndepend," 2014. [Online]. Available: http://www.ndepend.com/

[35] SolidSource IT, "SolidSX," 2013. [Online]. Available: http://www.solidsourceit.com/products/SolidSX-source-code-dependency-analysis.html

[36] Systems Sparx, "Enterprise Architect," 2015. [Online]. Available: http://www.sparxsystems.com/products/ea/

[37] IBM, "IBM Rational Software Architect," 2015. [Online]. Available: https://www.ibm.com/developerworks/downloads/r/architect/

[38] Uni Stuttgart, "Bauhaus," 2006. [Online]. Available: http://www2.informatik.uni-stuttgart.de/iste/ps/bauhaus/save/index-english.html

[39] Hello2Morrow, "Sotograph," 2015. [Online]. Available: https://www.hello2morrow.com/products/sotograph

[40] Swag, "Swag Kit," 2015. [Online]. Available: http://www.swag.uwaterloo.ca/swagkit/index.html

[41] Lattix, "Lattix Architect," 2015. [Online]. Available: http://lattix.com/

[42] Uni Milano Bicocca, "MARPLE," 2013. [Online]. Available: http://essere.disco.unimib.it/reverse/Marple.html