Karlsruher Institut für Technologie

Vergleich von Reverse-Engineering-Ansätzen für Software-Architekturen

Bachelorarbeit von

Moritz Gstür

an der Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr.-Ing. Anne Koziolek
Zweitgutachter:	Prof. Dr. Ralf H. Reussner
Betreuender Mitarbeiter:	M.Sc. Yves R. Kirschner
Zweiter betreuender Mitarbeiter:	M.Sc. Snigdha Singh

29. Dezember 2020 – 29. April 2021

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 29. April 2021

. . . . . .

(Moritz Gstür)

Zusammenfassung

Diese Arbeit dient der Ermittlung der Vergleichbarkeit und des Funktionsumfanges von Reverse-Engineering-Ansätzen für Software-Architekturen. Ziel der Arbeit war insbesondere die Feststellung der Eignung der Reverse-Engineering-Ansätze für die Rückgewinnung Komponenten- sowie Microservice-basierter Software-Architekturen. Hierfür wurden 26 Reverse-Engineering-Ansätze hinsichtlich ihres Funktionsumfanges geprüft. Acht der geprüften 26 Reverse-Engineering-Ansätze erwiesen sich als geeignet hinsichtlich ihres Funktionsumfanges und konnten in Betrieb genommen werden. Diese acht Reverse-Engineering-Ansätze wurden auf 22 Fallstudien angewandt und die Ergebnisse zur Identifikation von Stärken, Problemen und Einschränkungen der Ansätze genutzt. Es konnte gezeigt werden, dass ein Vergleich der Ansätze aufgrund der Heterogenität der Ergebnisse nicht durchführbar ist. Die Notwendigkeit kompilierter Quelltextdateien sowie Inkompatibilitäten der Reverse-Engineering-Ansätze zu Fallstudien bildeten zudem eine Einschränkung der Anzahl generierbarer Ergebnisse. Des Weiteren konnten die benötigte Zeit und der Ressourcenverbrauch der Reverse-Engineering-Ansätze als einschränkende Faktoren identifiziert werden. Die Architekturanalyse einschließlich der Berechnung von Metriken einer Software-Architektur konnte als Schwerpunkt der untersuchten Reverse-Engineering-Ansätze identifiziert werden. Eine Rückgewinnung von Software-Architekturen erwies sich auf Ebene der Klassen und Pakete als möglich. Eine Erkennung der Abhängigkeiten von Microservices sowie eine eindeutige Identifikation bestehender Komponenten und ihrer Schnittstellen war mithilfe der Ansätze im Zuge dieser Arbeit nicht möglich.

Inhaltsverzeichnis

Zusammenfassung **i**

1. Einleitung **1**
1.1. Motivation . . . 1
1.2. Forschungsfragen und Ziel der Arbeit . . . 2

2. Grundlagen **3**
2.1. Software-Architektur . . . 3
2.1.1. Architekturerosion . . . 3
2.1.2. Ausgewählte Architekturstile . . . 4
2.2. Reverse-Engineering . . . 5
2.2.1. Statische Analyse . . . 5
2.2.2. Dynamische Analyse . . . 5
2.3. Forward-Engineering . . . 6
2.4. Round-Trip-Engineering . . . 6

3. Verwandte Arbeiten **7**
3.1. Evaluation von Reverse-Engineering-Ansätzen . . . 7
3.2. Vergleich von Reverse-Engineering-Ansätzen . . . 7

4. Methodik **9**
4.1. Stufen der Evaluation von Ansätzen . . . 9
4.1.1. Einrichtung und Inbetriebnahme . . . 9
4.1.2. Ablauf der Anwendung . . . 9
4.1.3. Beschreibung generierter Ergebnisse . . . 10
4.1.4. Einschränkungen und Probleme . . . 10
4.2. Auswahl der Reverse-Engineering-Ansätze . . . 10
4.2.1. Funktionsumfang . . . 10
4.2.2. Verfügbarkeit . . . 11
4.2.3. Einrichtung und Inbetriebnahme . . . 11
4.3. Verwendete Fallstudien . . . 12

5. Evaluation bestehender Ansätze **17**
5.1. Ansätze zur Gewinnung von Rohdaten der Architektur . . . 17
5.1.1. MoDisco . . . 17
5.1.2. jQAssistant . . . 19
5.2. Ansätze zur Architekturrückgewinnung und -analyse . . . 22
5.2.1. Buschmais SAR-Framework . . . 22

5.2.2. Sonargraph Explorer 26
5.2.3. Teamscale 30
5.2.4. Structure101 Studio 33
5.2.5. Rational Software Architect Designer 37
5.2.6. Enterprise Architect 40
5.3. Diskussion 43
5.3.1. Einrichtung und Inbetriebnahme 43
5.3.2. Ablauf der Anwendung 44
5.3.3. Generierte Ergebnisse 44
5.3.4. Einschränkungen und Probleme 45

6. Fazit **47**
6.1. Schlussfolgerung 47
6.2. Einschränkungen der Validität 48
6.3. Ausblick und zukünftige Arbeit 49

Literatur **51**

A. Anhang **57**

Abbildungsverzeichnis

5.1. Schematischer Ablauf einer Anwendung von MoDisco 18
5.2. jQAssistant Ergebnisgraph der Klassen der Fallstudie Spring PetClinic . . 20
5.3. Cypher-Abfrage des in Abbildung 5.2 dargestellten Ergebnisgraphen . . 20
5.4. Grafische Benutzeroberfläche des Buschmais SAR-Framework 23
5.5. Buschmais SAR-Framework Sehnendiagramm der Fallstudie Teammates 24
5.6. Buschmais SAR-Framework Kreispackung der Fallstudie Teammates . . 25
5.7. Ausschnitt der Kreispackung der Fallstudie Teammates 25
5.8. Explorationsansicht der Fallstudie Sagan erzeugt durch Sonargraph . . . 28
5.9. Graph einer zyklischen Abhängigkeit erzeugt durch Sonargraph 29
5.10. Auswahl vorkonfigurierter Metriken der Web-Anwendung Teamscale . . 32
5.11. Architektur-Editor der Web-Anwendung Teamscale 33
5.12. Levelized-Structure-Map aus der Anwendung Structure101 Studio 35
5.13. Grad der Strukturiertheit bestimmt durch Structure101 Studio 35
5.14. Abhängigkeiten von Paketen visualisiert durch Structure101 Studio . . . 36
5.15. Klassendiagramm erstellt mithilfe von Rational Software Architect Designer 39
5.16. Klassendiagramm erstellt mithilfe von Enterprise Architect 41

Tabellenverzeichnis

4.1. Eigenschaften der Reverse-Engineering-Ansätze 13
4.2. Eigenschaften der genutzten Fallstudien 15

5.1. Eckdaten der Anwendung des SAR-Frameworks auf Fallstudien 26
5.2. Anzahl und Ebenen der Metriken von Sonargraph Explorer 29

1. Einleitung

Dieses Kapitel dient der Beschreibung des Themas sowie der Erläuterung der Relevanz der vorliegenden Bachelorarbeit. Abschnitt 1.1 führt hierzu in Form einer Motivation an das Thema Reverse-Engineering von Software-Architekturen heran. Die Forschungsfragen sowie das Ziel der vorliegenden Bachelorarbeit sind in Abschnitt 1.2 zu finden.

1.1. Motivation

Die Entwicklung und der Einsatz moderner Software-Architekturen ist ein wichtiger Treiber von Software-Entwicklung. Die Anforderungen an Verfügbarkeit, Skalierbarkeit und Verteilung sowie deren Relevanz für heutige Software-Systeme nimmt zu. Der Stellenwert der Software-Architektur zur Sicherstellung eben genannter Anforderung in modernen Software-Systemen ist bekannt. Des Weiteren ermöglicht Wissen über die Software-Architektur eine Abstraktion und Komplexitätsreduktion bei der Arbeit mit bestehenden Software-Systemen. Für die Instandhaltung von Software-Systemen kann dieses Wissen eine Voraussetzung sein.

Jedoch besteht ein Risiko im Umgang mit Software-Systemen und ihren Software-Architekturen. Bestehende Software-Systeme leiden unter dem Problem der Software-Erosion. Neue Probleme erfordern neue Lösungen und verlangen so nach Anpassung der Software-Systeme. Software-Evolution und Software-Wartung führen tagtäglich zu einer Veränderung des bestehenden Quelltextes und der Dokumentation. Dies kann einen Verlust des Wissens über die aktuelle Software-Architektur, die verwendeten Komponenten und Dienste sowie ihre Relationen zur Folge haben.

Eine Lösung für dieses Problem ist der Ansatz der Architektur-Rückgewinnung durch Reverse-Engineering eines bestehenden Software-Systems. Diese Arbeit schlägt vor, diese Lösung hinsichtlich ihrer Realisierbarkeit und Praktikabilität zu analysieren und auf ihre Eignung zu prüfen. Hierzu werden Ansätze des Reverse-Engineerings auf Fallstudien angewandt und die Ergebnisse untersucht und bewertet. Auf dieser Grundlage findet zudem eine Ermittlung der Vergleichbarkeit unterschiedlicher Reverse-Engineering-Ansätze statt.

Durch eine Analyse und einen Vergleich der Reverse-Engineering-Ansätze sollen Informationen über ebendiese gesammelt werden. Hierdurch soll eine Ableitung von Empfehlungen für die Anwendung eines bestimmten Reverse-Engineering-Ansatzes ermöglicht werden. Im Besonderen die Ableitung von Empfehlungen für die Anwendung eines Reverse-Engineering-Ansatzes auf bestimmte Software-Architekturstile soll durchgeführt werden. Auch die Identifikation möglicher Schwächen oder Probleme eines Ansatzes soll geschehen, im Besonderen im Umgang mit bestimmten Software-Architekturstilen.

1.2. Forschungsfragen und Ziel der Arbeit

Das Ziel dieser Arbeit ist die Feststellung der Vergleichbarkeit sowie der Vergleich von Reverse-Engineering-Ansätzen. Um dieses Ziel zu realisieren, werden Ansätze des Reverse-Engineerings ausgewählt und auf Fallstudien angewandt. Die Anwendung der Reverse-Engineering-Ansätze auf die Fallstudien führt zur Generierung von Ergebnissen. Diese Ergebnisse werden untersucht, interpretiert und bewertet. Auf Grundlage dieser Ergebnisse stellt die vorliegende Arbeit die Vergleichbarkeit der Reverse-Engineering-Ansätze fest. Darüber hinaus findet eine Identifikation von Stärken und Vorteilen, aber auch Problemen und Einschränkungen einzelner Reverse-Engineering-Ansätze statt. Ein Schwerpunkt des Vergleiches der Ansätze ist die Eignung zur Rückgewinnung von Software-Architekturen mit hoher Aktualität. Im Besonderen werden hierzu Reverse-Engineering-Ansätze ausgewählt und verglichen, die in der Lage sind komplexe Software-Architekturen zurückzugewinnen. Die vorliegende Arbeit setzt hierbei einen Fokus auf Komponenten- sowie Microservice-basierte Software-Architekturen. Die Rückgewinnung komponentenbasierter Software-Architekturen erfordert insbesondere die Verfeinerung explorierter Abhängigkeiten des Quelltextes. Die Rückgewinnung Microservice-basierter Software-Architekturen setzt eine Identifikation von Abhängigkeiten eigenständiger Dienste voraus. Die Zielerreichung dieser Arbeit wird durch die Beantwortung dreier Forschungsfragen gewährleistet.

Forschungsfrage 1: Sind die Ergebnisse von Reverse-Engineering-Ansätzen in eine einheitliche Form überführbar, sodass sie mithilfe einer Metrik verglichen werden können?

Forschungsfrage 2: Ist die Identifikation von Komponenten einer Software-Architektur mit Reverse-Engineering-Ansätzen möglich?

Forschungsfrage 3: Ist die Rückgewinnung von Abhängigkeiten eigenständiger Dienste mit Reverse-Engineering-Ansätzen möglich?

2. Grundlagen

Dieses Kapitel dient der Beschreibung von Grundlagen und der Vorstellung der Grundlagenliteratur. Zu Beginn des Kapitels werden in Abschnitt 2.1 die Grundlagen der Software-Architektur beschrieben. Hierauf folgt die Beschreibung des Reverse-Engineerings in Abschnitt 2.2. Das in Abschnitt 2.3 vorzufindende Forward-Engineering dient der Abgrenzung vom Reverse-Engineering sowie dem Verständnis des Round-Trip-Engineering in Abschnitt 2.4. Die genannten Grundlagen werden in nachfolgenden Kapiteln als bekannt angenommen.

2.1. Software-Architektur

Im Folgenden sei, gemäß Bass, Clements und Kazman [5], eine sogenannte Software-Architektur eine Menge von Strukturen eines Software-Systems. Ein genauerer Blick auf eine solche Struktur ermöglicht hierbei die Ableitung bestehender Relationen und Eigenschaften zwischen den einzelnen Elementen eines Software-Systems. Anzumerken ist dabei, dass nicht jede Struktur, die in einem Software-System vorzufinden ist, auch Teil der Software-Architektur ist. So muss eine Struktur als Teil der Software-Architektur stets auch zum Verständnis des Software-Systems und seiner Eigenschaften beitragen. Ist dies nicht der Fall, berücksichtigt die Software-Architektur eine solche Struktur nicht. Denn mehr noch als ein Beitrag zum Verständnis, ist die Software-Architektur als Abstraktion von dem darunter liegenden Software-System zu verstehen. Als eine solche Abstraktion dient die Software-Architektur der Vereinfachung komplexer Sachverhalte innerhalb des Software-Systems, was einen Umgang mit einem solchen Software-System ermöglicht. Des Weiteren wirft der Anstieg der Größe und Komplexität von Software-Systemen Probleme hinsichtlich dem Entwurf und der Spezifikation von Strukturen innerhalb ebendieser Software-Systeme auf [25]. Darüber hinaus ist die Instandhaltung dieser Systeme eine Herausforderung, die das Verständnis der Entwickler für die bestehende Software-Architektur eines Software-Systems voraussetzt [56]. Bestehendes Wissen ist im Besonderen dann vonnöten, wenn die Personen, die das System Instand halten, nicht die Personen sind, die das Software-System entworfen haben [15].

2.1.1. Architekturerosion

Ein inhärentes Risiko im Umgang mit Software-Systemen stellt der Verlust des Wissens hinsichtlich der Software-Architektur dar. Obwohl jedes Software-System eine Software-Architektur besitzt, kann das konkrete Wissen über diese verloren gehen. Dies kann als Folge von Abwanderung von Mitarbeitern, verlorener oder nie angelegter Dokumentation oder nicht vorhandenem Quelltext geschehen [5].

Eine Erosion der Software-Architektur kann zudem durch Zeitdruck oder fehlende Erfahrung hervorgerufen werden. Oft entstehen und bleiben Architektur-verletzende Abhängigkeiten unbemerkt. Ein erhöhter Kopplungsgrad und steigende Komplexität der Software-Architektur sind zu beobachten [34].

Darüber hinaus kann das Fehlen konkreten Wissens und Verständnisses durch die Nutzung sowie Wartung fremden Quelltextes begründet sein [56].

2.1.2. Ausgewählte Architekturstile

Dieser Abschnitt dient der Beschreibung ausgewählter Architekturstile. Ein Architekturstil ist als Rahmenwerk einer Software-Architektur zu verstehen. Dieses Rahmenwerk beschreibt die Elemente einer Software-Architektur und stellt Bedingungen an die Relationen dieser Elemente [25]. In dieser Arbeit werden die aufgeführten Architekturstile genutzt, um eine Klassifikation betrachteter Software-Architekturen durchzuführen.

2.1.2.1. Komponentenbasierte Software-Architektur

Nach Kienzle u. a. [45] sowie Bosch, Szyperski und Weck [9] ist eine Software-Komponente eine Einheit der Software-Architektur, die Implementierung kapselt und eine Menge von Schnittstellen bereitstellt. Eine Software-Komponente kann als Baustein einer komponentenbasierten Software-Architektur verstanden werden. Als ein solcher Baustein ist die Software-Komponente eigenständig, lose gekoppelt und kohäsiv. Insbesondere ist eine Software-Komponente wiederverwendbar und austauschbar. Diese Eigenschaften ermöglichen eine Modularisierung der Software-Architektur und erlauben die Durchführung einer Anpassung der Software-Architektur ohne Kenntnis der Implementierung einzelner Software-Komponenten.

2.1.2.2. Microservice-basierte Software-Architektur

Eine Microservice-basierte Software-Architektur ist eine Sonderform der komponentenbasierten Software-Architektur. Nach Fowler und Lewis [23] liegt eine Microservice-basierte Software-Architektur einer einzelnen Anwendung zugrunde, wenn diese ihrerseits aus kleinen Diensten, sogenannten Microservices, aufgebaut ist. Diese Microservices sind eigenständig und unabhängig voneinander lauffähig. Insbesondere gilt, dass einzelne Microservices in eigenen Prozessen oder auf unterschiedlichen physischen oder logischen Maschinen ausgeführt werden können. Diese Grad der Eigenständigkeit ermöglicht es, einzelne Microservices einer Anwendung in unterschiedlichen Programmiersprachen sowie mithilfe verschiedener Technologien zu realisieren. Die Kommunikation zwischen den Diensten findet mittels leichtgewichtiger Mechanismen statt. Diese ermöglichen den Nachrichtenaustausch zwischen den Microservices. Eine solchen Mechanismus mit hoher Aktualität stellt der Einsatz von HTTP dar.

2.2. Reverse-Engineering

Das verlorene Wissen über die Software-Architektur zurück zu erlangen, ist die Aufgabe des Reverse-Engineerings. Beim Prozess des Reverse-Engineerings handelt es sich um einen untersuchenden Prozess. Änderung oder Replikation bestehender Software-Systeme ist kein Teil des Prozesses. Das Ziel des Reverse-Engineerings ist die Identifikation von Strukturen, in Form von Elementen und Relationen, innerhalb des zu untersuchenden Software-Systems. Hierdurch ermöglicht das Reverse-Engineering insbesondere die Rückgewinnung des Entwurfes, der einem Software-System zugrunde liegt [15]. Gemäß der in Abschnitt 2.1 besprochenen Definition von Software-Architektur, soll durch Reverse-Engineering eine Rekonstruktion der zugrundeliegenden Software-Architektur eines Software-Systems ermöglicht werden.

Die Art und Form der Ausgabedaten unterschiedlicher Reverse-Engineering-Ansätze kann variieren. Sowohl die Generierung grafischer, als auch nicht-grafischer Ausgabedaten ist möglich. Die Ansätze der Gruppe des modellgetriebenen Reverse-Engineerings generieren Modelle auf Grundlage der Informationen des Software-Systems. Unabhängig von der spezifischen Ausprägung, sollte das Erreichen einer höheren Verständlichkeit, Nachvollziehbarkeit sowie Abstraktion durch die Ausgabedaten stets angestrebt werden [15].

2.2.1. Statische Analyse

Die statische Analyse bietet die Möglichkeit der Gewinnung von Informationen aus den Bestandteilen eines Software-Systems [29]. Diese Gewinnung von Informationen erfolgt durch eine Analyse statischer Ressourcen des Software-Systems. Diese Ressourcen umfassen nicht ausschließlich den Quelltext eines Software-Systems, sondern mitunter auch Konfigurationsdateien, Datenbanken und Dokumentation [10]. Die gewonnenen Informationen können im Prozess des Reverse-Engineerings für die Rückgewinnung der Software-Architektur eingesetzt werden. Ein Vorteil der statischen Analyse besteht darin, dass das Software-System für die Durchführung der Analyse nicht gestartet und verteilt werden muss.

2.2.2. Dynamische Analyse

Diese Art der Analyse von Software-Systemen zur Gewinnung von Informationen nutzt Laufzeitinformationen eines Software-Systems. Die Erfassung dieser Art von Informationen ist ausschließlich bei einem laufenden und gegebenenfalls verteilten Software-System möglich. Nach Granchelli u. a. [29] ist durch diese Art der Analyse unter anderem eine Gewinnung von Informationen verteilter Container sowie Protokoll-Dateien möglich. Einen Anwendungsfall dieser Analyse stellt beispielsweise die Exploration der Topologie verteilter Container-basierter Software-Systeme dar. Eine solche Topologie kann durch die Beobachtung von stattfindendem TCP-Verkehr der Container zur Laufzeit erzeugt werden [68]. Ein Vorteil der dynamischen Analyse ist folglich die Erkennung von Abhängigkeiten in einer Software-Architektur, die erst zur Laufzeit identifizierbar sind.

2.3. Forward-Engineering

Das Forward-Engineering bezeichnet einen Prozess, der abstrakte Information eines Software-Systems in eine konkrete Implementierung überführt [15]. Folglich kann das Forward-Engineering als Umkehrung des Reverse-Engineerings verstanden werden. Analog zu den Ressourcen des Reverse-Engineerings ist auch die Nutzung unterschiedlicher Ressourcen für den Prozess des Forward-Engineerings möglich. Neben Anforderungen umfassen ebendiese Ressourcen insbesondere Modelle. Ein solches Modell dient der Abstraktion eines Software-Systems. Die initiale Erstellung eine Software-Systems auf Grundlage definierter Anforderungen oder Modelle ist demnach ein Forward-Engineering-Prozess.

2.4. Round-Trip-Engineering

Das Round-Trip-Engineering bezeichnet die Verknüpfung des Reverse-Engineerings mit dem Forward-Engineering [70]. Das Round-Trip-Engineering dient dabei dem Abgleich einer Abstraktion eines Software-Systems mit der konkreten Implementierung. Zur Unterstützung eines Software-Entwicklungsprozesses wird angestrebt das Abgleichen von Informationen in Echtzeit durchzuführen. Eine Durchführung in Echtzeit dient der Verhinderung inkonsistenter Zustände. In vorliegender Arbeit ist insbesondere das modellgetriebene Round-Trip-Engineering von Relevanz. Hierbei wird durch Reverse-Engineering ein bestehendes Software-System in ein Modell überführt. An diesem Modell können im Verlauf der Software-Entwicklung Änderungen vorgenommen werden. Eine solche Änderung wird durch das Forward-Engineering auf das Software-System übertragen. Nach Abschluss des Prozesses befinden sich sowohl das Modell als auch das Software-System in einem konsistenten Zustand. Analog ist genannter Prozess bei Änderungen des Software-Systems möglich.

3. Verwandte Arbeiten

Dieses Kapitel dient der Vorstellung verwandter Arbeiten. Im Besonderen soll hierdurch eine Erläuterung der Relevanz sowie die Nachvollziehbarkeit gewählter Methodiken der vorliegenden Arbeit gewährleistet werden.

3.1. Evaluation von Reverse-Engineering-Ansätzen

Für die vorliegende Arbeit wurde die Methodik der Untersuchung und Bewertung von Reverse-Engineering-Ansätzen anhand von Ergebnissen nach Anwendung auf Fallstudien gewählt. Eine Beschreibung dieses Vorgehens ist in Kapitel 4 zu finden. Dieses Vorgehen wurde gewählt, da Hersteller verfügbarer Reverse-Engineering-Ansätze dies zur Bewertung, Untersuchung und Evaluation einzelner Ansätze einsetzen. Dieses Vorgehen findet sich exemplarisch in Publikationen zu den Ansätzen Bauhaus [22], Archimetrix [54], MicroArt [28] und MICROLYZE [47]. In diesen Arbeiten konnten insbesondere zwei Einschränkungen der Validität präsentierter Ergebnisse festgestellt werden. Die erste Einschränkung der Validität stellt die Voreingenommenheit der Hersteller der Ansätze bei der Auswahl möglicher Fallstudien dar. Dies kann in vorliegender Arbeit aufgrund der Unabhängigkeit zu Herstellern und Reverse-Engineering-Ansätzen ausgeschlossen werden. Die zweite Einschränkung ist die geringe Anzahl genutzter Fallstudien. Dies kann zu einer Verzerrung der Ergebnisse aufgrund fehlender Heterogenität der Fallstudien führen. In den Publikationen zu Archimetrix [54], MicroArt [28] und MICROLYZE [47] wurde nur je eine Fallstudie genutzt. Eine Publikation zum Ansatz Bauhaus [22] verwendete vier Fallstudien. In vorliegender Arbeit kommen insgesamt 22 Fallstudien zur Generierung von Ergebnissen mithilfe der zu evaluierenden Reverse-Engineering-Ansätze zum Einsatz.

3.2. Vergleich von Reverse-Engineering-Ansätzen

Neben den genannten Publikationen orientiert sich die vorliegende Arbeit an Publikationen mit dem Ziel des Vergleiches mehrerer Reverse-Engineering-Ansätze. Von den Autoren Gueheneuc, Mens und Wuyts [32] ist hierbei eine Publikation eines Rahmenwerkes für einen solchen Vergleich zu nennen. Diese Publikation diente der Identifikation von Kategorien und Fragestellungen für die Erstellung der in Abschnitt 4.1 beschriebenen Stufen der Evaluation einzelner Ansätze. Auf eine Beschreibung der genutzten Algorithmen und Implementierung wurde in vorliegender Arbeit verzichtet, da sowohl Umfang als auch Bearbeitungszeit diese Analysen nicht zuließen.

Bereits 1997 verglichen Bellay und Gall [7] vier verschiedene Reverse-Engineering-Ansätze miteinander. Später erschienene Vergleichsarbeiten stammen von Arcelli u. a. [1], mit zwei verglichenen Ansätzen, sowie Gorton und Zhu [26], mit fünf Ansätzen der

Architekturrückgewinnung. Die vorliegende Arbeit evaluiert insgesamt acht Reverse-Engineering-Ansätze. Hierdurch sollen Rückschlüsse auf den momentanen Stand der Technik ermöglicht werden. Ein Problem, das durch die vorliegende Arbeit gelöst werden soll, ist die fehlende Aktualität der genannten Arbeiten. Insbesondere hinsichtlich der Rückgewinnung moderner Software-Architekturen liegt ein Mangel an Aktualität genannter Publikationen vor. Diese Aktualität wird in vorliegender Arbeit durch geeignete Auswahl von Reverse-Engineering-Ansätze und Fallstudien gewährleistet. Eine genaue Beschreibung des Auswahlprozesses geeigneter Reverse-Engineering-Ansätze und Fallstudien ist Kapitel 4 zu entnehmen. Analog zu den Evaluationen einzelner Ansätze konnte bei diesen Publikationen nur eine geringe Anzahl genutzter Fallstudien festgestellt werden. In den Publikationen von Bellay und Gall [7] sowie Gorton und Zhu [26] wurde je nur eine Fallstudie zur Evaluation genutzt. Beide verwendeten Fallstudien wurden von Industriepartnern zur Verfügung gestellt. Insbesondere handelte es sich nicht um quelloffene Fallstudien. In der Publikation von Arcelli u. a. [1] findet keine genaue Beschreibung verwendeter Ressourcen statt. Die vorliegende Arbeit adressiert dieses Problem durch Nutzung 22 quelloffener Fallstudien. Diese weisen neben unterschiedlichen Software-Architekturstilen auch unterschiedliche Projektgrößen sowie verschieden Projektstrukturen auf.

4. Methodik

Dieses Kapitel dient der Erläuterung der angewandten Methodik für die Evaluation der Ansätze des Reverse-Engineerings von Software-Architekturen. Zu Beginn wird hierfür das Vorgehen bei der Evaluation der Ansätze anhand eines Vier-Stufen-Planes beschrieben. Auf die Beschreibung des Vorgehens der Evaluation folgt die Erläuterung des Auswahlprozesses der zu evaluierenden Ansätze des Reverse-Engineerings. Den Abschluss dieses Kapitels bildet die Beschreibung des Auswahlverfahrens der verwendeten Fallstudien.

4.1. Stufen der Evaluation von Ansätzen

Dieser Abschnitt dient der Erläuterung der Vorgehensweise der Evaluation einzelner Ansätze. Die durchgeführten Evaluationen sind in Kapitel 5 zu finden. Es wird ein Vorgehen verwendet, das eine Untersuchung und Bewertung der Ansätze auf unterschiedlichen Abstraktionsebenen und hinsichtlich unterschiedlicher Kriterien erlaubt. Im Sinne der Reproduzierbarkeit sind die untersuchten Stufen in chronologischer Reihenfolge, von Einrichtung bis zur Beschreibung von Ergebnissen, in den Evaluationen der Ansätze vorzufinden.

4.1.1. Einrichtung und Inbetriebnahme

Die Stufe der Einrichtung und Inbetriebnahme stellt die erste Stufe der Evaluation dar. Sie dient als Blaupause einer Einrichtung und Inbetriebnahme des zu untersuchenden Ansatzes. Dies ermöglicht die erneute Ableitung von Ergebnissen zu einem späteren Zeitpunkt und die Nachvollziehbarkeit vorgestellter Ergebnisse. Die Einrichtungen und Inbetriebnahmen der Ansätze wurden ausschließlich auf einem Computer durchgeführt. Der verwendete Computer besitzt einen Intel Core i7-3770k Prozessor mit 4,2 Gigahertz Taktrate sowie 16 Gigabyte Arbeitsspeicher. Genutzt wurde zudem das 64-Bit Betriebssystem Windows 10 Pro. Eine Einrichtung und Inbetriebnahme, anhand der jeweiligen Evaluation eines Ansatzes, kann folglich nur für oben genanntes Betriebssystem wie beschrieben durchgeführt werden.

4.1.2. Ablauf der Anwendung

Die Stufe des Ablaufes der Anwendung eines Ansatzes stellt die zweite Stufe der Evaluation dar. Ist der zu untersuchende Ansatz gemäß Unterabschnitt 4.1.1 eingerichtet und betriebsbereit, wird in dieser Stufe eine Evaluation des Ablaufes der Anwendung des Ansatzes durchgeführt. Darüber hinaus dient diese Stufe der Beschreibung von Parametern, die für die Anwendung des Ansatzes benötigt werden. Dieser Stufe der Evaluation liegen

die Beobachtungen der Anwendung eines Ansatzes auf Fallstudien zugrunde. Die für die Evaluation genutzten Fallstudien sind in Abschnitt 4.3 aufgeführt. Die Anwendungen auf die Fallstudien wurden gemäß den Vorgaben der Ersteller der Ansätze vorgenommen, sofern solche Vorgaben existierten.

4.1.3. Beschreibung generierter Ergebnisse

Die dritte Stufe der Evaluation ist die Beschreibung der generierten Ergebnisse nach Anwendung eines Ansatzes gemäß Unterabschnitt 4.1.2. Diese Stufe dient der Beschreibung der Form und des Inhaltes generierter Ergebnisse und Ausgabeartefakte. Die in der Evaluation beschriebenen oder dargestellten Artefakte und Ergebnisse sind als repräsentativ für einen Ansatz bei erfolgreicher Anwendung zu verstehen. Ergebnisse von nicht vollständig oder korrekt durchführbaren Anwendungen, aufgrund von Fehlern oder Inkompatibilität zu Fallstudien, werden innerhalb dieser Stufe nicht näher betrachtet.

4.1.4. Einschränkungen und Probleme

Die vierte Stufe der Evaluation dient der Beschreibung von Einschränkungen und Problemen eines Ansatzes. Zugrunde liegen dieser Stufe Eigenschaften einzelner Fallstudien, die eine Anwendung eines Ansatzes auf ebendiese Fallstudien erschwert oder unmöglich gemacht haben. Des Weiteren dient diese Stufe der Zusammenfassung von Problemen, die im Verlauf der Evaluation eines Ansatzes aufgetreten sind. Die beschriebenen Einschränkungen und Probleme sind reproduzierbar. Nicht reproduzierbares Verhalten hinsichtlich Einschränkungen oder Problemen wird in den nachfolgenden Evaluationen der Ansätze nicht betrachtet.

4.2. Auswahl der Reverse-Engineering-Ansätze

Dieser Abschnitt dient der Erläuterung des durchgeführten Verfahrens zur Auswahl geeigneter Ansätze für die Evaluation. Hierfür wurden Ansätze durch verwandte Arbeiten, aktuelle Publikationen sowie Recherche im Internet zum Thema Reverse-Engineering identifiziert. In Tabelle 4.1 sind die identifizierten Ansätze aufgelistet. Mithilfe der verwandten Arbeiten konnten dabei die Ansätze Bauhaus [22], Archimetrix [54], MicroArt [28], MICROLYZE [47], Enterprise Architect [59] sowie Imagix 4D [38] identifiziert werden. Bei den aufgelisteten Ansätzen handelt es sich sowohl um quelloffene sowie kostenfreie als auch um kostenpflichtige Reverse-Engineering-Ansätze. Die Eignung dieser Ansätze für die Evaluation wurde geprüft. Die Prüfung der Ansätze wurde in drei Schritten durchgeführt. Diese Schritte sind in den nachfolgenden Unterabschnitten näher beschrieben.

4.2.1. Funktionsumfang

Den ersten Schritt der Prüfung der Eignung eines Ansatzes für die Evaluation, stellte eine Analyse und Bewertung des angebotenen Funktionsumfanges dar. Informationen, die für diesen Schritt der Prüfung benötigt wurden, wurden aus verfügbaren Dokumen-

ten der Hersteller der Ansätze entnommen. Hierzu zählten neben wissenschaftlichen Publikationen auch die Web-Seiten und Dokumentation der Ansätze sowie Bild- und Videomaterial. Für die Bewertung wurde eine Mindestanforderung an den Funktionsumfang definiert. Diese Mindestanforderung besagte, dass ein Ansatz als ungeeignet gilt, wenn keine Rückgewinnung von Software-Architekturen möglich ist oder er ausschließlich die Rückgewinnung von Software-Architekturen in Form von Klassen- und Paketdiagrammen ermöglicht. Ziel dieser Mindestanforderung war die Identifikation von Ansätzen, die Komponenten- oder Microservice-basierte Software-Architekturen zurückgewinnen konnten. Die Umsetzung der Mindestanforderung führte zu dem Ergebnis, dass 10 der 26 Ansätze für die Evaluation nicht geeignet waren. Die Ansätze Ghidra und CodeInspect dienen dem Reverse-Engineering kompilierter Dateien mit dem Ziel den Quelltext zurück zu erhalten. Eine Rückgewinnung der Software-Architektur ist mit diesen Ansätzen nicht möglich. Die Ansätze UML Lab, IntelliJ IDEA Ultimate, Class Visualizer, eUML2, Papyrus Software Designer, jGRASP und Visual Paradigm erlauben eine Rückgewinnung der Software-Architektur ausschließlich in Form von Klassen- und Paketdiagrammen. Diese Ansätze gelten gemäß der definierten Mindestanforderung als ungeeignet. Da es sich bei den in Abschnitt 4.3 beschriebenen Fallstudien um Projekte mit Java-Quelltext handelt, wurde der Ansatz Axivion Suite als ungeeignet eingestuft. Axivion Suite unterstützt ausschließlich die Programmiersprachen C und C++ und ist somit nicht kompatibel zu den ausgewählten Fallstudien.

4.2.2. Verfügbarkeit

Dieser Schritt der Prüfung dient der Ermittlung der Verfügbarkeit der übrigen 16 Ansätze. Ein Ansatz gilt als verfügbar, wenn das Quelltext-Repository des Ansatzes oder eine Quelle des Ansatzes in ausführbarer Form bekannt ist. Bei den fünf Ansätzen Archimetrix, MicroArt, MoDisco, Buschmais SAR-Framework und jQAssistant handelt es sich um quelloffene Projekte deren Quelltext-Repositories bekannt sind. Die Hersteller der übrigen 11 Ansätze wurden per E-Mail angeschrieben und um ihre Mithilfe gebeten, indem sie ihre Ansätze für begrenzte Zeit zur Verfügung stellen. Die Hersteller der Ansätze Archium, Bauhaus und Sonargraph lehnten diese Anfrage ab. Durch die Hersteller der Ansätze Imaging, Imagix 4D, MICROLYZE, Lattix Architect, Rational Software Architect Designer, Enterprise Architect und Teamscale wurde die Anfrage nicht beantwortet. Die Ansätze Rational Software Architect Designer, Enterprise Architect, Teamscale sowie Sonargraph konnten aufgrund verfügbarer alternativer Varianten oder Testversionen dennoch als verfügbar eingestuft werden. Der Hersteller des Ansatzes Structure101 Studio stimmte der Anfrage zu und stellte den Ansatz unter akademischer Lizenz mit vollem Funktionsumfang zur Verfügung.

4.2.3. Einrichtung und Inbetriebnahme

Der letzte Schritt der Prüfung umfasst die Einrichtung und Inbetriebnahme der übrigen zehn Ansätze. Ansätze die nicht erfolgreich eingerichtet und in Betrieb genommen werden können, werden als ungeeignet für die Evaluation angesehen. Der Ansatz Archimetrix konnte nicht eingerichtet und in Betrieb genommen werden, da ein hierfür vorausge-

setztes Projekt nicht mehr verfügbar ist. Die Einrichtung und Inbetriebnahme gemäß der Dokumentation des Herstellers des Ansatzes MicroArt war nicht erfolgreich durchführbar. Die Ansätze MoDisco, Buschmais SAR-Framework, jQAssistant, Structure101 Studio, Rational Software Architect Designer, Enterprise Architect, Teamscale sowie Sonargraph konnten erfolgreich eingerichtet und in Betrieb genommen werden. Eine Beschreibung der erfolgreichen Einrichtungen und Inbetriebnahmen ist den Evaluationen in Kapitel 5 zu entnehmen.

4.3. Verwendete Fallstudien

Die Evaluation der in Abschnitt 4.2 beschriebenen Ansätze umfasst die Anwendung der Ansätze auf Fallstudien. Dieser Abschnitt stellt die genutzten Fallstudien vor. Des Weiteren dient dieser Abschnitt der Erläuterung des Auswahlprozesses der Fallstudien auf Grundlage ihrer Eigenschaften. In Summe wurden 22 Fallstudien für die Anwendung der Ansätze in den Evaluationen ausgewählt. In Tabelle 4.2 ist eine Auflistung der Fallstudien zu finden. Die genannte Tabelle dient zudem der Darstellung von Eigenschaften der Fallstudien, die im Auswahlprozess berücksichtigt wurden. Die initiale Identifikation geeigneter Fallstudien geschah mithilfe verwandter Arbeiten, aktueller Publikationen sowie Recherche im Internet. Durch verwandte Arbeiten sowie aktuelle Publikationen konnte eine Identifikation und Auswahl der Fallstudien Acme Air [36], TeaStore [46], Sock Shop [67] sowie Spring Cloud Event Sourcing Example [6] vorgenommen werden [46][28]. Bei den Fallstudien TimeSheetGenerator [57], Meet & Eat Server [31] sowie Meet & Eat Data [30] handelt es sich um Software-Projekte, an deren Entwurf und Implementierung der Autor der vorliegenden Arbeit beteiligt war. Das hierdurch zugängliche Wissen über diese Fallstudien sowie zugrundeliegende Software-Architekturen dient im weiteren Verlauf dieser Arbeit der Unterstützung der Evaluationen der Reverse-Engineering-Ansätze. Darüber hinaus wirkte der Autor der vorliegenden Arbeit an der Entwicklung sowie Fehlerbehebung der Fallstudie Artemis [14] mit.

Eine Voraussetzung im Auswahlprozess für alle Fallstudien war die Quelloffenheit. Dies wurde festgelegt, um eine Verifikation der Ergebnisse der Ansätze auf Grundlage des Quelltextes vornehmen zu können.

Eine Eigenschaft, die bei der Auswahl zudem berücksichtigt wurde, war die Software-Architektur. Aufgrund der Wichtigkeit und Verwendungshäufigkeit in moderner Software-Entwicklung, wurden im Besonderen Fallstudien mit Microservice-basierten sowie komponentenbasierten Software-Architekturen ausgewählt. Wie in Tabelle 4.2 zu sehen ist, handelt es sich bei 18 der 22 Fallstudien um Projekte mit komponentenbasierter Architektur. Hierzu zählen insbesondere Fallstudien mit Modul- oder auch Dienst-basierter Architektur. Eine Spezialisierung dieser 18 komponentenbasierten Fallstudien stellen die sechs Fallstudien TeaStore, Piggy Metrics, Spring PetClinic, Sock Shop, Microservice Sample und Spring Cloud Event Sourcing Example dar. Diese sechs Fallstudien weisen eine Microservice-basierte Software-Architektur auf. Neben der Gruppe der komponentenbasierten Fallstudien, wurden Meet & Eat Server sowie JabRef als Vertreter monolithischer Schichtenarchitektur ausgewählt. Die beiden Fallstudien Meet & Eat Data sowie Commons Lang wurden ausgewählt, da es sich bei ihnen um Software-Bibliotheken handelt.

Tabelle 4.1.: Eigenschaften der Reverse-Engineering-Ansätze

Reverse-Engineering-Ansatz	Quelloffen	Quelltextanalyse	Bytecode-Analyse	Container-Analyse
Archium [2]				X
Axivion Suite [4]		X		
Bauhaus [55]		X		
Archimetrix [54]	X	X		
jGRASP [3]		X		
Ghidra [51]	X		X	
Papyrus Software Designer [21]	X	X		
eUML2 [58]		X		
Class Visualizer [41]	X		X	
CodeInspect [50]			X	
IntelliJ IDEA Ultimate [40]		X		
UML Lab [71]		X		
MicroArt [28]	X			X
Visual Paradigm [65]		X	X	
Sonargraph [35]		X	X	
Enterprise Architect [59]		X	X	
Imaging [13]		X		
Imagix 4D [38]		X	X	
MICROLYZE [47]				X
Lattix Architect [48]			X	
Rational Software Architect Designer [37]		X		
MoDisco [10]	X	X		
Buschmais SAR-Framework [12]	X		X	
jQAssistant [11]	X		X	
Teamscale [17]		X		
Structure101 Studio [33]			X	

Neben der Architektur wurde die genutzte Programmiersprache der Fallstudien in den Auswahlprozess miteinbezogen. Dieses Vorgehen war notwendig, da zur Gewährleistung der Vergleichbarkeit der Ergebnisse der Ansätze eine Programmiersprache gewählt werden musste, die mit möglichst allen Ansätzen kompatibel ist. Als eine solche Programmiersprache wurde Java identifiziert. Eine nähere Betrachtung der Kompatibilität der Programmiersprache Java zu den gewählten Ansätzen ist in Abschnitt 4.2 zu finden. Die mit der Hochzahl Eins markierten Fallstudien weisen neben Projektteilen in Java auch andere Programmiersprachen auf.

Die ebenfalls in Tabelle 4.2 vorzufindende Verfügbarkeit des Java-Bytecodes der Fallstudien stellte sich im Verlauf der Evaluation als Einschränkung der Kompatibilität von Fallstudien zu Ansätzen heraus. So konnten bestimmte Ansätze zwar auf Java-Klassen, nicht jedoch auf Java-Quelltextdateien angewendet werden. Ansätze dieser Art konnten auf sechs der 22 Fallstudien nicht angewandt werden. Eine Beschreibung der betroffenen Ansätze ist in Kapitel 5 zu finden. Die Fallstudien BigBlueButton, Hadoop und Ghidra setzten für das Kompilieren des Projektes zusätzliche Software sowie Konfiguration voraus. Eine Installation dieser benötigten Software und eine Konfiguration gemäß den Angaben der Ersteller der Fallstudien wurde vorgenommen, führte jedoch ebenfalls zu Fehlern beim Kompilieren. Die Fallstudie Hadoop konnte in Form bereits gepackter Jar-Archivdateien aus dem Internet heruntergeladen werden. Eine Extraktion der Fallstudien-eigenen Java-Klassendateien war aufgrund der Anzahl sowie des Umfanges der Jar-Archivdateien jedoch nicht möglich. Die Fallstudien Acme Air sowie MediaStore konnten aufgrund fehlender Abhängigkeiten nicht kompiliert werden. Die Fallstudie Sock Shop umfasst einzelne Dienste, die in den Programmiersprachen Java, Go und mittels .Net Core und NodeJS realisiert wurden. Das Haupt-Repository der Fallstudie Sock Shop bietet Funktionalität zur Containerisierung der Dienste. Darüber hinaus werden keine Funktionalitäten angeboten und insbesondere das Kompilieren und Packen der Dienste ist im Haupt-Repository nicht möglich.

Für die oben genannten Kompiliervorgänge sowie das Packen von Fallstudien in Jar-, War- oder Ear-Dateien wurden die durch die Fallstudien vorgegebenen und vorkonfigurierten Build-Management-Werkzeuge genutzt. Bei den Build-Management-Werkzeugen der Fallstudien handelt es sich um Apache Maven [63] und Gradle [27]. Insgesamt wird Gradle von neun und Apache Maven von elf der 22 Fallstudien genutzt. In den Diensten der Fallstudie Sock Shop kommen neben Apache Maven weitere Build-Management-Werkzeuge anderer Programmiersprachen zum Einsatz. Analog nutzt BigBlueButton sowohl Apache Maven als auch Gradle sowie weitere Build-Management-Werkzeuge anderer Programmiersprachen. Weder das Haupt-Repository von BigBlueButton noch von Sock Shop nutzen Apache Maven oder Gradle. Eine Einteilung dieser Fallstudien zu einem der beiden Build-Management-Werkzeuge wurde daher nicht vorgenommen.

Tabelle 4.2.: Eigenschaften der Projekte und der Software-Architekturen der genutzten Fallstudien

Fallstudie	Java-Quelltext	Java-Bytecode	Gradle-Projekt	Maven-Projekt	Komponenten	Microservices
TeaStore [46]	X	X		X	X	X
Piggy Metrics [49]	X	X		X	X	X
Spring PetClinic [66]	X	X		X	X	X
Spring Cloud Event Sourcing Example [6]	X	X		X	X	X
Energy State Data Analysis [43]	X	X		X	X	
Teammates [60]	X	X	X		X	
OFBiz [64]	X[1]	X	X		X	
Hadoop [62]	X[1]	X		X	X	
Commons Lang [61]	X	X		X		
JabRef [39]	X	X	X			
Ghidra [51]	X[1]		X		X	
Acme Air [36]	X		X		X	
MediaStore [44]	X			X	X	
Sagan [53]	X	X	X		X	
Sock Shop [67]	X[1]				X	X
BigBlueButton [8]	X[1]				X	
Microservice Sample [69]	X	X		X	X	X
TimeSheetGenerator [57]	X	X		X	X	
Meet & Eat Server [31]	X	X	X			
Meet & Eat Data [30]	X	X	X			
Artemis [14]	X[1]	X	X		X	
CloudStore [16]	X	X		X	X	

5. Evaluation bestehender Ansätze

Dieses Kapitel dient der Evaluation ausgewählter Reverse-Engineering-Ansätze. Die Beschreibung des Auswahlverfahrens der evaluierten Reverse-Engineering-Ansätze ist Abschnitt 4.2 zu entnehmen. In Abschnitt 5.1 sind die Evaluationen der Reverse-Engineering-Ansätze zu finden, deren Aufgabe die Gewinnung von Rohdaten einer Software-Architektur ist. Abschnitt 5.2 umfasst die Evaluationen der Reverse-Engineering-Ansätze, die der Rückgewinnung sowie Analyse von Software-Architekturen dienen.

5.1. Ansätze zur Gewinnung von Rohdaten der Architektur

Dieser Abschnitt dient der Evaluation von Reverse-Engineering-Ansätzen zur Gewinnung von Rohdaten einer Software-Architektur. Im Folgenden sind die Evaluationen zweier solcher Reverse-Engineering-Ansätze zu finden. In Unterabschnitt 5.1.1 ist die Evaluation des modellgetriebenen Reverse-Engineering-Rahmenwerkes MoDisco [10] vorzufinden. Unterabschnitt 5.1.2 dient der Evaluation des Graphen-basierten Reverse-Engineering-Ansatzes jQAssistant [11].

5.1.1. MoDisco

Dieser Abschnitt dient der Evaluation des Rahmenwerkes MoDisco [20]. MoDisco wird in vorliegender Arbeit gemäß Unterabschnitt 2.2.1 als Ansatz zur Gewinnung von Rohdaten der Architektur mittels statischer Analyse angesehen.

5.1.1.1. Einrichtung und Inbetriebnahme

Das Rahmenwerk MoDisco ist eine Erweiterung des Eclipse-SDK. Die Einrichtung und Inbetriebnahme von MoDisco wurde gemäß des Eclipse-Wiki [19] vorgenommen. Als eine Erweiterung von Eclipse erfolgt die Einrichtung innerhalb einer zuvor installierten Eclipse-Distribution. Hierfür wurde die Eclipse-Distribution ModelingPackage der Version 4.18 genutzt. Der Prozess des Herunterladens und der Einrichtung der Erweiterung geschah innerhalb der Distribution und lief vollautomatisch ab. Ein Neustart der Distribution schloss den Einrichtungsprozess ab.

5.1.1.2. Ablauf der Anwendung

Die Anwendung des Rahmenwerks MoDisco erfolgt innerhalb der Eclipse-Distribution. Zur Analyse eines bestehenden Projektes, muss dieses zu Beginn in der Distribution geöffnet werden.

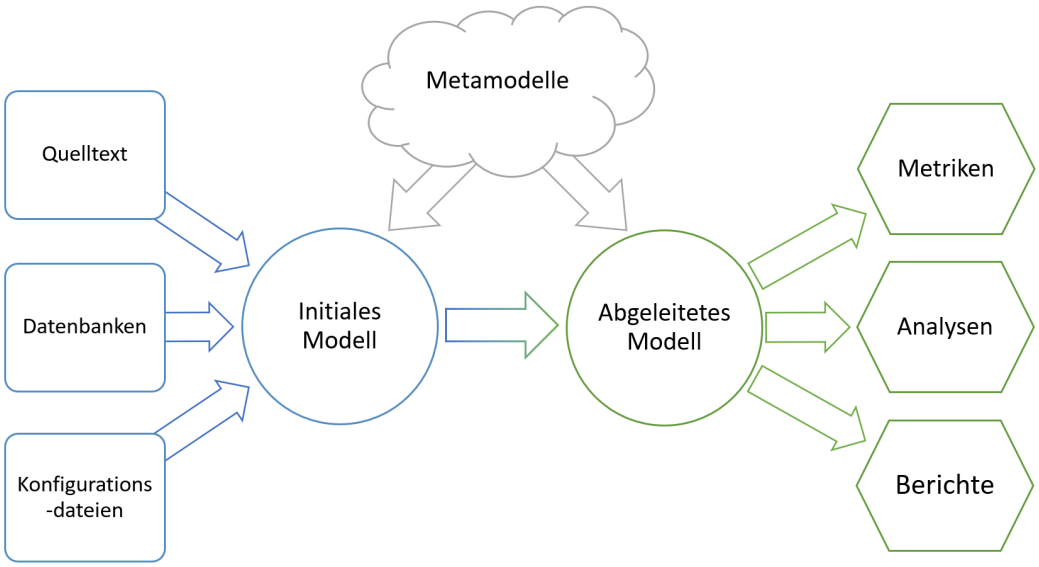


Abbildung 5.1.: Schematischer Ablauf einer Anwendung des Rahmenwerkes MoDisco gemäß Brunelière u. a. [10]

Innerhalb des geöffneten Projektes können Untersuchungsinstanzen, sogenannte Discoverer, ausgeführt werden. Ein Discoverer überführt Elemente des Projektes in ein Modell. Nach Brunelière u. a. [10] handelt es sich bei diesen Elementen um Quelltext, Datenbanken, Konfigurationsdateien und weitere Quellen. Ein Modell bezeichnet eine Repräsentation des Projektes auf einer höheren Ebene der Abstraktion [56].

Sobald der Vorgang der gewählten Untersuchungsinstanz abgeschlossen ist, speichert MoDisco das Modell in Form einer Datei im Projekt ab. Dieses Modell kann innerhalb der Eclipse-Distribution mithilfe eines Modell-Browsers exploriert werden.

Es besteht zudem die Möglichkeit der Überführung des gespeicherten Modelles in ein anderes Modell. Genutzt wird hierzu die Konformität der Modelle zu ihren Metamodellen. Zudem ist eine Ableitung von Information und Artefakten aus dem Modell möglich. Nach Brunelière u. a. [10] zählen hierzu zum Beispiel Metriken, Berichte, Dokumentationen oder Folgenabschätzungen.

In Abbildung 5.1 ist der schematische Ablauf einer möglichen Anwendung des Rahmenwerkes MoDisco zu sehen. Ein Discoverer überführt hierbei Quelltext, Datenbanken und Konfigurationsdateien in ein initiales Modell. Dieses initiale Modell ist konform zu einem Metamodell. Die Konformität zu einem Metamodell erlaubt die Transformation des initialen in ein abgeleitetes Modell. Ausgehend vom abgeleiteten Modell ist die Generierung gewünschter Metriken, Analysen und Berichte möglich.

5.1.1.3. Beschreibung generierter Ergebnisse

Es handelt sich bei MoDisco um ein Rahmenwerk zur Entwicklung von modellgetriebenen Werkzeugen. MoDisco bietet die Möglichkeit der Überführung von Quelldateien eines Projektes in Modelle. Es besteht zudem die Möglichkeit der Transformation eines Model-

les in ein weiteres Modell sowie die Möglichkeit der Ableitung von Informationen aus Modellen. Hierbei ist die Nutzung unterschiedlicher Modelle möglich. Die Beschreibung eines spezifischen Ergebnisses oder Modelles ist daher nicht möglich.

5.1.1.4. Einschränkungen und Probleme

Die Identifikation von Problemen in der Anwendung des Rahmenwerk MoDisco konnte nicht vorgenommen werden. Der Grund hierfür ist die Abhängigkeit von MoDisco zum Eclipse-SDK, die die Herleitung des Ursprunges eines Problems nicht zulässt. Exemplarisch hierfür war das Problem, dass Fallstudien nicht fehlerfrei in der Eclipse-Distribution als Projekte eingelesen werden konnten. Das betraf unter anderem die Microservice-basierten Fallstudien CloudStore [16], Spring PetClinic [66] und Piggy Metrics [49] sowie Gradle-basierten Fallstudien. Bei den Fallstudien TimeSheetGenerator [57] sowie Commons Lang [61] wurden durch die Eclipse-Distribution zudem Fehler in den Pom-Dateien festgestellt. Folglich lässt sich die Einschränkung formulieren, dass MoDisco eine Verflechtung mit dem Eclipse-SDK aufweist.

5.1.2. jQAssistant

Dieser Abschnitt dient der Evaluation des Qualitätssicherungswerkzeuges jQAssistant [11]. In der vorliegenden Arbeit wird jQAssistant, gemäß Unterabschnitt 2.2.1, als Ansatz zur Gewinnung von Rohdaten der Architektur mittels statischer Analyse klassifiziert.

5.1.2.1. Einrichtung und Inbetriebnahme

Die Einrichtung des Programmes jQAssistant kann auf zwei Arten vorgenommen werden. Einerseits ist die Nutzung von jQAssistant in Form eines eigenständigen Kommandozeilenprogrammes möglich. Andererseits kann jQAssistant als Maven Plugin in ein Projekt eingebunden werden. Aufgrund der Heterogenität der verwendeten Build-Management-Werkzeuge der Fallstudien wurde die Nutzung mittels eigenständigem Kommandozeilenprogramm für die Evaluation ausgewählt. Das Kommandozeilenprogramm kann in Form einer Zip-Datei heruntergeladen werden und ist sofort betriebsbereit.

5.1.2.2. Ablauf der Anwendung

Die Anwendung des Programmes jQAssistant findet in mehreren Schritten statt. Zu Beginn muss die zu untersuchende Fallstudie als Jar-Datei gepackt werden. Durch Aufrufen des scan-Befehls wird strukturelle Information des Projektes aus der Jar-Datei in eine lokale Instanz der Graphen-basierten Datenbank Neo4j gespeichert. Mithilfe des server-Befehls wird ein Web-Server gestartet, durch den ein Web-basierter Zugriff auf diese Datenbank möglich ist. Dieser Web-Server erlaubt Abfragen der lokalen Datenbank, die in der Abfragesprache Cypher geschrieben sind. Das Anwendungsspektrum von jQAssistant ist nicht auf die reine Gewinnung von Rohdaten der Architektur sowie Abfragen auf diesen Daten beschränkt. Auch das Erstellen von Regeln in Form von Abfragen ist möglich. Erstellte

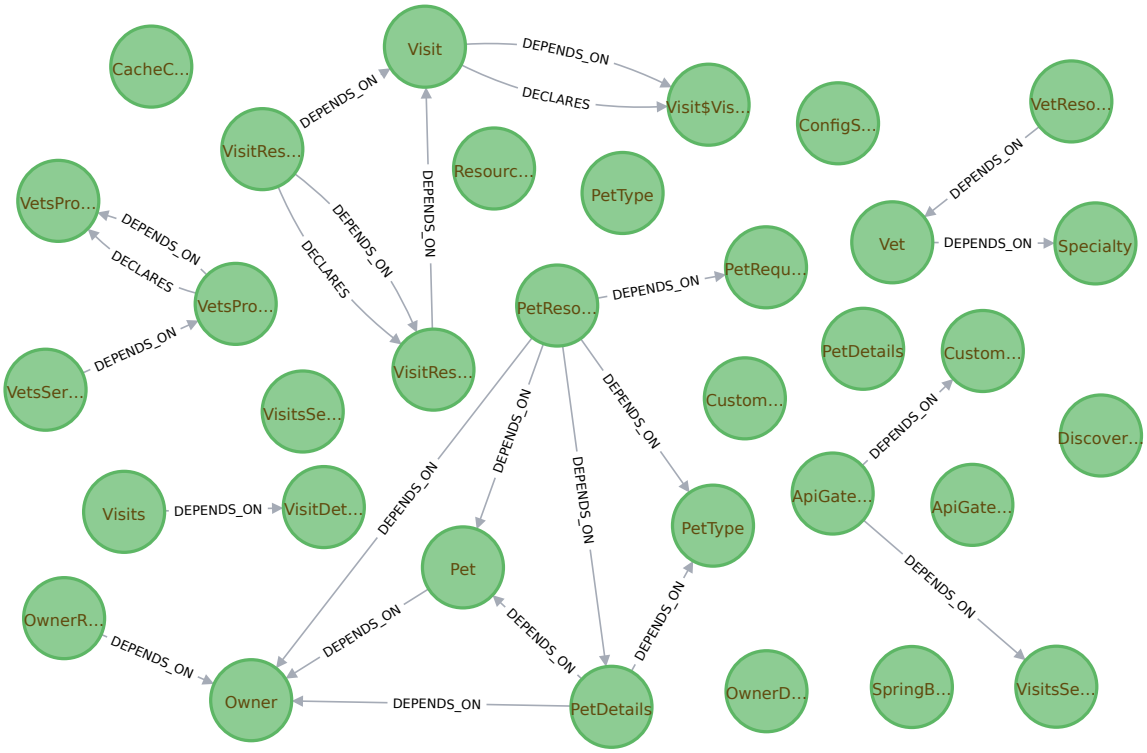


Abbildung 5.2.: jQAssistant [11] Ergebnisgraph der Klassen der Fallstudie Spring PetClinic [66]

```
1 MATCH (cls:Class)
2 WHERE cls.fqn STARTS WITH "org.springframework.samples.petclinic"
3 RETURN cls
```

Abbildung 5.3.: Cypher-Abfrage des in Abbildung 5.2 dargestellten Ergebnisgraphen

Regeln erlauben die wiederholte Überprüfung von Konformität der Strukturinformationen der Architektur mit definierten Soll-Bedingungen. Dies dient der Qualitätssicherung und der Vorbeugung der in Unterabschnitt 2.1.1 beschriebenen Architekturerosion.

5.1.2.3. Beschreibung generierter Ergebnisse

Das Ergebnis einer Anwendung von jQAssistant sind die strukturellen Informationen der Architektur, die durch Aufrufen des scan-Befehls abgeleitet und in der lokalen Datenbank gespeichert werden. Die Möglichkeit Abfragen in der Abfragesprache Cypher zu schreiben diversifiziert die Form sowie den Inhalt der Ergebnisse, die aus den gespeicherten Daten abgeleitet werden können.

Exemplarisch ist in Abbildung 5.2 das Ergebnis einer Abfrage auf den Strukturinformationen der Architektur der Fallstudie Spring PetClinic [66] zu sehen. Die hierfür verwendete Cypher-Abfrage ist in Abbildung 5.3 dargestellt. Diese Abfrage gibt Elemente der Datenbank zurück, die als Klassen gekennzeichnet sind. Eine Einschränkung der Abfrage

lautet, dass deren vollständig qualifizierter Name mit der in Abbildung 5.3 dargestellten Zeichenkette beginnen muss. Hierdurch ist die Rückgabe ausschließlich eigener Klassen der Microservices der Fallstudie Spring PetClinic möglich. Klassen von Abhängigkeiten der Dienste werden nicht zurückgegeben.

Durch Jürgen Dufner [42] werden Abfragen für jQAssistant vorgestellt, die der Ableitung von Metriken dienen. Unter anderem werden Metriken des objektorientierten Entwurfes und der Sichtbarkeit beschrieben und in Form von Cypher-Abfrage präsentiert.

5.1.2.4. Einschränkungen und Probleme

jQAssistant ermöglicht die Durchführung von Abfragen auf den Strukturinformationen eines Projektes. Dadurch entsteht die Möglichkeit der Ableitung gesuchter Informationen, sofern die dafür benötigte Abfrage in Cypher formuliert werden kann. Vorkenntnisse in der Sprache Cypher sind somit notwendig, im Besonderen bei der Konstruktion nichttrivialer Abfragen.

Eine Einschränkung des Ansatzes jQAssistant ist, dass das zu analysierende Projekt stets in Form einer Jar-, War- oder Ear-Datei vorliegen muss. Ein vorangegangener Bausowie Packprozess zur Erstellung einer der genannten Dateien ist für die Durchführung einer Analyse notwendig. Eine Analyse basierend auf vorliegenden Quelltextdateien ist nicht möglich.

Eine weitere Einschränkung stellt die Größe der generierten Datenbank dar. So umfasst die Fallstudie Spring PetClinic [66] beispielsweise sieben Jar-Dateien ihrer Microservices. Diese Dateien sind in Summe 380 Megabyte groß und umfassen Abhängigkeiten sowie die Microservices selbst. Die durch jQAssistant aus den Jar-Dateien generierte Datenbank ist 2,83 Gigabyte groß. Gerundet ergibt sich ein Zuwachs der Größe um den Faktor 7,45. Bei der Microservice-basierten Fallstudie Piggy Metrics [49] lag die Ausgangsgröße der neun Jar-Dateien bei 391 Megabyte. Es konnte gerundet ein Zuwachs der Größe um den Faktor 7,8 auf eine Datenbankgröße von 3,05 Gigabyte beobachtet werden. Einen noch größerer Zuwachs konnte bei der monolithischen Software JabRef [39] beobachtet werden. Die Jar-Datei von JabRef hat eine Größe von 8,94 Megabyte. Die jQAssistant Datenbank zu JabRef erreichte eine Größe von 88,7 Megabyte. Dies entspricht gerundet einem Zuwachs um den Faktor 9,92.

Ein Problem bei der Anwendung von jQAssistant auf die Fallstudien stellte die Ressourcennutzung dar. Bei der Anwendung auf die Fallstudie Artemis [14] kam es nach 15 Sekunden zu einer OutOfMemoryException, die sich auch durch Anpassung der Speicher Parameter der JVM nicht beheben ließ. Die Auslastung pro Prozessorkern lag im arithmetischen Mittel über die Zeit bei 30,4% und im Maximum bei 51%. Die maximale Auslastung des Arbeitsspeichers durch jQAssistant erreichte 7,6 Gigabyte bei den betrachteten Fallstudien.

Eine weitere Einschränkung ist die notwendige Zeit für die Durchführung des scan-Befehls. Die Zeit, die benötigt wurde um die Fallstudie Spring PetClinic [66] zu untersuchen, lag bei 14 Minuten und 47 Sekunden. Für die Untersuchung der Fallstudie Piggy Metrics [49] waren 15 Minuten und 10 Sekunden notwendig. Für die Untersuchung der Fallstudie JabRef [39] wurden 15 Sekunden benötigt.

5.2. Ansätze zur Architekturrückgewinnung und -analyse

Dieser Abschnitt dient der Evaluation von Reverse-Engineering-Ansätzen, die Funktionalitäten für die Rückgewinnung oder Analyse bestehender Software-Architekturen bieten. Im Folgenden sind die Evaluationen der sechs Reverse-Engineering-Ansätze Buschmais SAR-Framework [12], Sonargraph Explorer [35], Teamscale [17], Structure101 Studio [33], Rational Software Architect Designer [37] sowie Enterprise Architect [59] zu finden.

5.2.1. Buschmais SAR-Framework

Dieser Abschnitt dient der Evaluation des Buschmais Rahmenwerkes zur Rückgewinnung von Software-Architekturen [12]. Das Rahmenwerk dient nach Angabe von Buschmais der Identifikation von Komponenten durch Analyse von Kohäsions- sowie Kopplungseigenschaften. Darüber hinaus dient es der Unterstützung des Verständnis- und Restrukturierungsprozesses bestehender Java-Software-Systeme. Es verfügt nicht über ein eigenes Rohdatengewinnungsverfahren, sondern nutzt bestehende Datenbanken von jQAssistant [11].

5.2.1.1. Einrichtung und Inbetriebnahme

Die Einrichtung ist durch Herunterladen eines gepackten Jar-Projektes von der GitHub-Seite des SAR-Frameworks [12] möglich. Zu beachten ist, dass das SAR-Framework JavaFX nutzt. Ein Ausführen ohne zusätzliche Installationen ist somit nur bis Java SE 10 möglich, da spätere Versionen JavaFX nicht mehr enthalten [18]. Da das SAR-Framework bereits erstellte Datenbanken von jQAssistant zur Analyse nutzt, ist die Einrichtung von jQAssistant gemäß Unterunterabschnitt 5.1.2.1 notwendig. Zu beachten ist die eingeschränkte Kompatibilität des SAR-Framework mit bestimmten jQAssistant-Versionen. Eine Beschreibung getesteter Versionen und aufgetretener Inkompatibilitäten ist in Unterunterabschnitt 5.2.1.4 zu finden.

5.2.1.2. Ablauf der Anwendung

In Abbildung 5.4 ist die grafische Benutzeroberfläche nach dem Ausführen der Jar-Datei zu sehen. Es besteht darüber hinaus die Möglichkeit der Verwendung einer Kommandozeilen-Variante des Rahmenwerkes. Diese ist ebenfalls im GitHub-Repository zu finden. Eine Durchsicht des Quelltextes ergab, dass die nutzbaren Parameter der Kommandozeilen-Variante des Rahmenwerkes, von denen der grafischen Benutzeroberfläche abweichen. Es wurde daher die Evaluation der grafischen Benutzeroberfläche gewählt, da ihre Parameter in der Readme-Datei des GitHub-Repository beschrieben sind.

Der erste Parameter dient hierbei der Festlegung des Pfades zu einer Datenbank, die durch jQAssistant gemäß Unterunterabschnitt 5.1.2.2 erzeugt wurde. Anzugeben ist der Pfad des Store-Verzeichnisses innerhalb des Ergebnisverzeichnisses von jQAssistant.

Die nächsten drei Parameter dienen der Spezifikation der zu analysierenden Klassen. Der Artefakt-Parameter grenzt die zu analysierenden Klassen ein, indem er den Namen der Artefakte auf eine Übereinstimmung mit dem gegebenen regulären Ausdruck prüft.

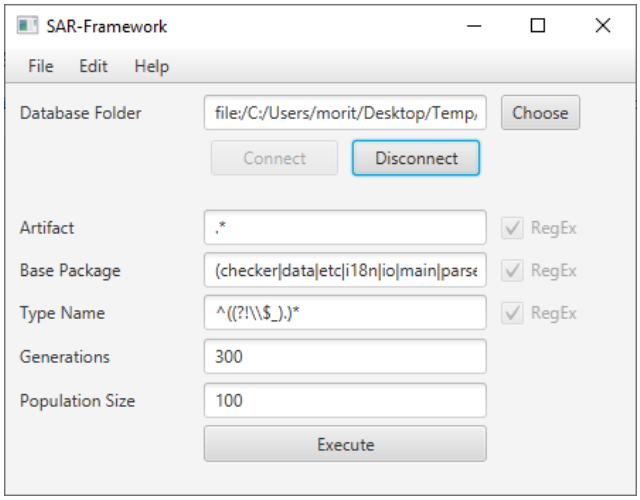


Abbildung 5.4.: Grafische Benutzeroberfläche des Buschmais SAR-Framework [12], konfiguriert für die Anwendung auf die Fallstudie TimeSheetGenerator [57]

Ist eine Klasse kein Teil der übereinstimmenden Artefakte, so wird sie im Analyseprozess nicht berücksichtigt. Analog wird für den Basispaket-Parameter die Zugehörigkeit einer Klasse zu einem Paket geprüft, dessen Bezeichner per regulärem Ausdruck angegeben werden muss. Der Typname-Parameter prüft die Übereinstimmung eines Klassennamens mit einem gegebenen regulären Ausdruck und schließt nicht-übereinstimmende Klassen aus der Analyse aus. Eine Klasse muss durch alle drei Parameter zugelassen sein, um Teil der Analyse zu sein. In den Anwendungen des SAR-Framework auf die Fallstudien wurde insbesondere der Basispaket-Parameter genutzt. Hierdurch konnten Klassen der Abhängigkeiten ausgeschlossen werden. Der Typname-Parameter wurde gemäß der Readme-Datei des GitHub-Repository konfiguriert, um generierte Klassen mit dem Präfix $_ auszuschließen.

Der Generationen-Parameter sowie der Populationsgröße-Parameter dienen der Konfiguration des genutzten genetischen Algorithmus. Eine Einflussnahme auf die Parameter wirkt sich auf die Ausführungszeit und Qualität aus [12]. Für die Evaluation des Ansatzes wurden die Parameter gemäß den Vorgaben der Ersteller des Ansatzes fest gewählt.

Exemplarisch ist in Abbildung 5.4 die Konfiguration des Ansatzes auf die Fallstudie TimeSheetGenerator [57] zu sehen. Typname-, Generationen- sowie Populationsgröße-Parameter wurden gemäß den Vorgaben der Ersteller des Ansatzes gewählt. Der Artefakt-Parameter akzeptiert alle Zeichenketten. Der Basispaket-Parameter wurde hinsichtlich einer Übereinstimmung mit allen Paketen der Fallstudie TimeSheetGenerator gewählt. Analog wurde bei der Anwendung des SAR-Framework auf weitere Fallstudien vorgegangen.

5.2.1.3. Beschreibung generierter Ergebnisse

Das Ergebnis der Anwendung des SAR-Framework ist eine Zip-Datei. Beobachtete Größen von Ergebnisdateien nach Anwendung des SAR-Framework auf Fallstudien sind in Tabelle 5.1 aufgelistet. Innerhalb dieser Datei befinden sich die generierten Ergebnisse

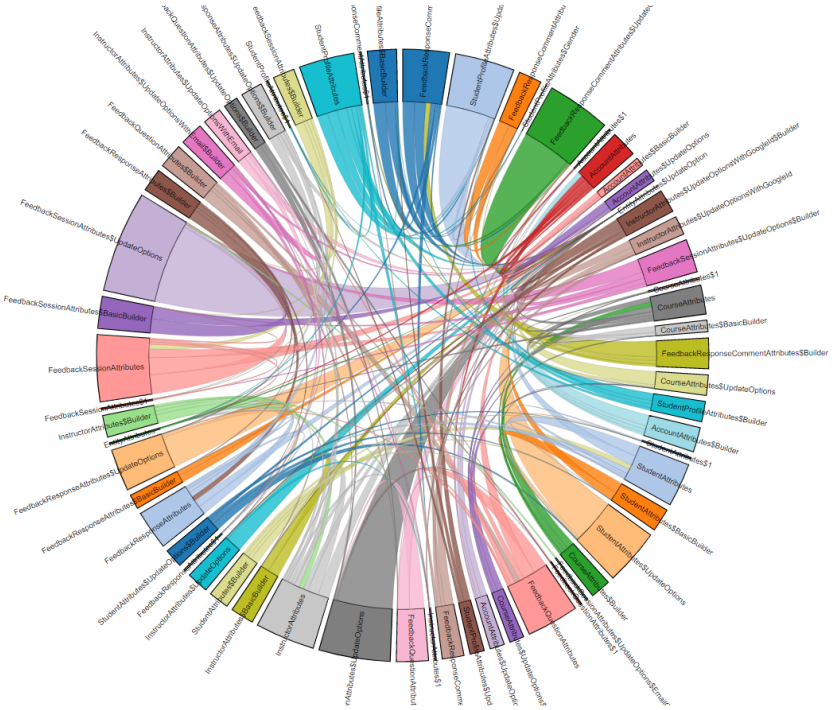


Abbildung 5.5.: Teilergebnis der Anwendung des SAR-Framework [12] auf die Fallstudie Teammates [60] in Form eines Sehnendiagrammes

des SAR-Framework in Form von Json-Dateien. Eine Exploration der Ergebnisse ist durch Html- sowie JavaScript-Dateien möglich, die sich ebenfalls in der Zip-Datei befinden. Diese Dateien ermöglichen die Visualisierung der Ergebnisdaten innerhalb eines Browsers. Es besteht die Möglichkeit der Visualisierung in Form eines Sehnendiagrammes sowie einer Kreispackung der ermittelten Komponenten. Beide Typen von Visualisierung sind interaktiv und erlauben eine Exploration der Daten auf unterschiedlich detaillierten Ebenen.

Exemplarisch ist in Abbildung 5.5 eine Ebene des Sehnendiagrammes der Fallstudie Teammates [60] zu sehen. Innerhalb eines Browsers ist das Ändern der Ebene des Sehnendiagrammes durch Selektieren eines Ringsegmentes möglich. Durch Schweben über dem Segment mithilfe des Cursors lassen sich zudem Relationen ausblenden, die in keiner Beziehung zu dem Segment stehen.

Die vollständige Kreispackung der Fallstudie Teammates ist in Abbildung 5.6 zu sehen. Analog zur Exploration des Sehnendiagrammes, ist in der Kreispackung ein Heranzoomen durch Selektion eines Kreises möglich. In Abbildung 5.7 ist die Selektion einer Gruppe von Klassen dargestellt. Eine solche Gruppe wird durch die Auswertung berechneter Kohäsions- und Kopplungswerte der Klassen erzeugt. Die Ebene der größten Tiefe entspricht der Selektion einer Klasse. Diese werden durch orangefarbene Kreise in der Kreispackung repräsentiert.

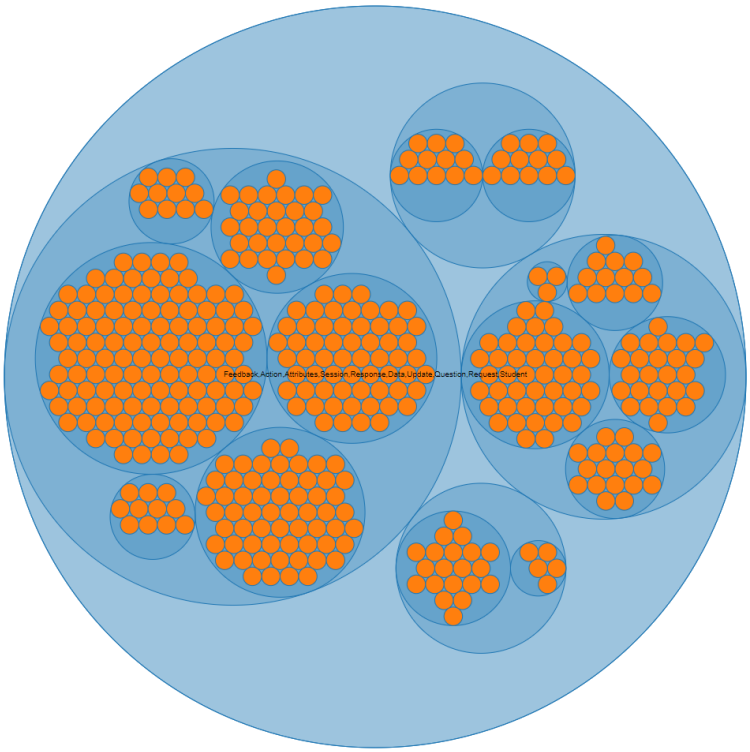


Abbildung 5.6.: Kreispackung der Anwendung des SAR-Framework [12] auf die Fallstudie Teammates [60]

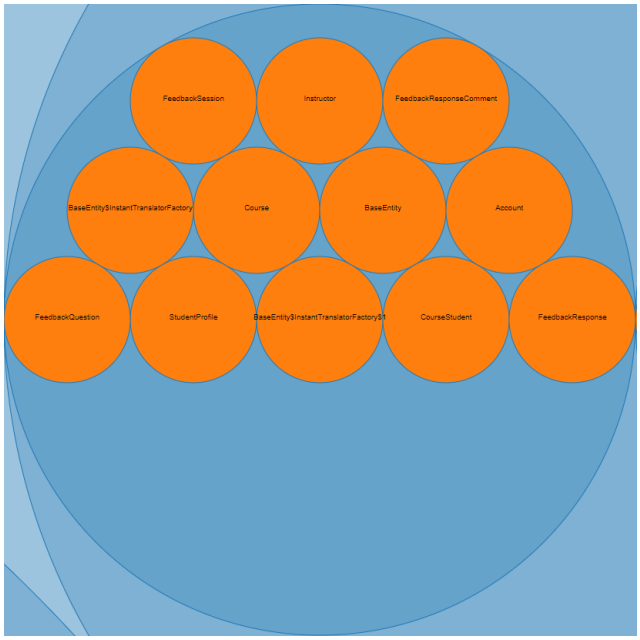


Abbildung 5.7.: Ausschnitt der Kreispackung der Anwendung des SAR-Framework [12] auf die Fallstudie Teammates [60]

Tabelle 5.1.: Eckdaten der Anwendung des SAR-Frameworks [12] auf Fallstudien

Fallstudie	Datenbankgröße [MB]	Ergebnisgröße [MB]	Benötigte Zeit [mm:ss]
TimeSheetGenerator [57]	44,9	0,265	00:34
CloudStore [16]	693,0	0,302	00:39
Spring Cloud Event Sourcing Example [6]	3540,0	0,329	01:50
Commons Lang [61]	12,2	0,522	05:11
Teammates [60]	15,8	0,548	07:45
OFBiz [64]	121,0	unbekannt	Exception
Hadoop [62]	988,0	unbekannt	Exception

5.2.1.4. Einschränkungen und Probleme

Da für die Ausführung des SAR-Framework eine jQAssistant-Datenbank notwendig ist, sind die Einschränkungen und Probleme gemäß Unterunterabschnitt 5.1.2.4 zu beachten.

Das SAR-Framework ist mit der Ergebnisdatenbank aktueller jQAssistant Versionen nicht kompatibel. Die in Unterabschnitt 5.1.2 untersuchte Version von jQAssistant, namentlich 1.8.0, ist inkompatibel mit dem SAR-Framework. Es wurde daher auf die Version 1.3.0 zurückgegriffen. Diese findet sich in Form einer Referenz innerhalb der Readme des GitHub-Repository des SAR-Framework. jQAssistant in der Version 1.3.0 konnte jedoch die Fallstudien mit einer Java-Version größer 8 nicht fehlerfrei untersuchen. Diese Fallstudien mussten daher von der Anwendung ausgeschlossen werden. Die Fallstudien, deren Untersuchung möglich war, sind in Tabelle 5.1 aufgeführt.

Eine Einschränkung des SAR-Framework ist die benötigte Zeit zur Berechnung eines Ergebnisses. Eine Herleitung der erwarteten Dauer und ihrer Einflussfaktoren ist aufgrund der Menge inkompatibler Fallstudien nicht möglich. Es ist zu beachten, dass die Dauer für die Generierung der jQAssistant-Datenbank nicht in die gemessene Zeit einzelner Fallstudien aus Tabelle 5.1 eingeflossen ist.

Eine weitere Einschränkung ist die Auslastung des Prozessors während einer laufenden Analyse. Ein Pendeln der Auslastung zwischen 30% und 50% der Prozessorkerne wurde bei den Analysen beobachtet. Bei der Analyse der Fallstudien OFBiz [64] und Hadoop [62] traten zudem Lastspitzen von 95% auf.

Bei der Analyse der genannten Fallstudien OFBiz und Hadoop traten OutOfMemory-Ausnahmen auf. Die Nachricht der Ausnahmen besagte, dass das Garbage-Collector-Overhead-Limit überschritten worden sei.

5.2.2. Sonargraph Explorer

Der folgenden Abschnitt dient der Evaluation der Anwendung Sonargraph Explorer [35]. Diese Anwendung ist Teil der Produktfamilie Sonargraph, die die Anwendungen Explorer, Architect und Developer umfasst. Bei der Anwendung Explorer handelt es sich um eine Vereinfachung der Anwendung Architect. Die Anwendung Developer bietet die Funktio-

nalität der Anwendung Architect in Form einer Entwicklungsumgebungserweiterung für IntelliJ und Eclipse. Der Ersteller der Produktfamilie Sonargraph wurde hinsichtlich einer temporären Lizenz für die Anwendung Architect angefragt. Es folgte eine Antwort mit einem Verweis auf die kostenfreie Anwendung Explorer. Eine Nachfrage nach Unterschieden der Anwendungen Explorer und Architect blieb unbeantwortet.

5.2.2.1. Einrichtung und Inbetriebnahme

Vor der Einrichtung der Anwendung Sonargraph Explorer ist das Erstellen eines Benutzerkontos sowie die Beantragung eines Lizenzschlüssels erforderlich. Die Beantragung des Lizenzschlüssels ist kostenfrei, die Nutzung der Lizenzdateien jedoch auf drei Anwendungen und ein Jahr Laufzeit beschränkt. Eine solche Lizenzdatei muss nach Beantwortung des Antrages heruntergeladen werden.

Das Herunterladen der Anwendung geschieht in Form einer Zip-Datei. Nach dem Entpacken muss eine Lizenzdatei im Programmordner abgelegt werden. Das Starten der Anwendung ist ohne weitere Einrichtungsschritte möglich.

5.2.2.2. Ablauf der Anwendung

Bei Sonargraph Explorer handelt es sich um eine Anwendung mit grafischer Benutzeroberfläche. Die Anwendung beginnt mit einer Aufbereitung der Eingabeprojekte. Ein solches Eingabeprojekt, bestehend aus Modulen, wird in Sonargraph Explorer als System bezeichnet. Es besteht die Möglichkeit der Erstellung eines neuen Systems sowie die Möglichkeit der Erstellung eines Systems unter der Nutzung eines bestehenden Eclipse-, IntelliJ-, Bazel- oder Maven-Projektes. Bestehende Projekte müssen dabei sowohl Quelltext- als auch deren Klassendateien umfassen. Für die Evaluation wurden die Fallstudien in IntelliJ Projekte umgewandelt und gemäß ihrer Spezifikation gebaut. Diese Projekte konnten dann als Systeme in Sonargraph Explorer eingelesen werden.

Wurde das Projekt als System eingelesen, beginnt Sonargraph Explorer mit der Berechnung von Architekturmetriken. Die berechneten Metriken dienen des Weiteren der automatischen Ermittlung von Architekturproblemen. Nach Abschluss der Berechnung der Architekturmetriken und der Ermittlung der Architekturprobleme stehen die Ergebnisse in der Anwendung zur Exploration bereit.

5.2.2.3. Beschreibung generierter Ergebnisse

Sobald die Architekturmetriken und Architekturprobleme durch Sonargraph Explorer ermittelt wurden, ist eine Exploration der Daten möglich. In Abbildung 5.8 ist die sogenannte Explorationsansicht von Sonargraph Explorer zu sehen. Diese visualisiert für den Benutzer die Modul-, Paket- und Klassenhierarchie des Systems. Aus Gründen der Übersichtlichkeit sind in Abbildung 5.8 ausschließlich Elemente der höchsten Paketebene dargestellt. Relationen dieser Elemente werden in Form von ungerichteten Pfaden dargestellt. Eine Konfiguration der zu visualisierenden Richtung der Pfade ist möglich. Eine Visualisierung oder Erkennung von HTTP/S-basierten Relationen zwischen Diensten Microservice-basierter Systeme ist nicht möglich.

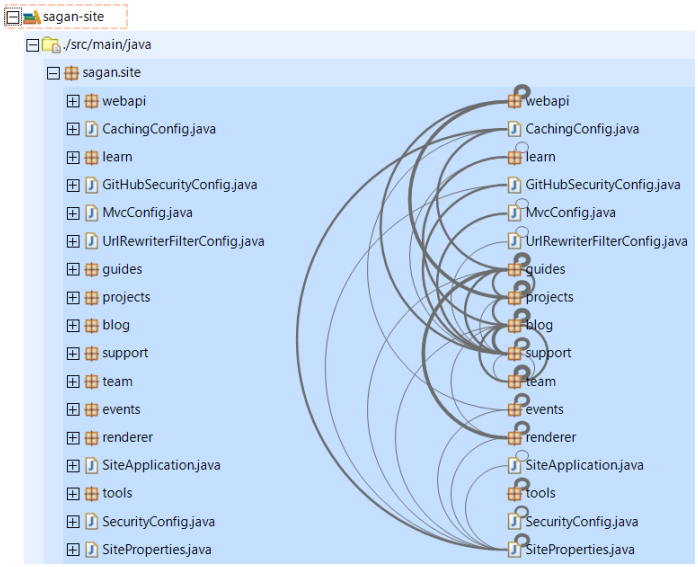


Abbildung 5.8.: Explorationsansicht der Fallstudie Sagan [53] erzeugt durch Sonargraph [35]

Des Weiteren bietet Sonargraph Explorer eine vordefinierte Menge von Architekturmetriken. Eine Definition eigener Metriken in Form von Skripten ist in Sonargraph Architect jedoch nicht in Sonargraph Explorer möglich. Die berechneten Architekturmetriken gliedern sich in Untersuchungsebenen. Die Untersuchungsebenen sowie die Anzahl der vordefinierten Architekturmetriken sind Tabelle 5.2 zu entnehmen. Es existieren Architekturmetriken höherer Schichten, die auf den Werten niedrigerer Schichten basieren. Exemplarisch hierfür ist die Wartbarkeit auf der Ebene des Systems, die sich aus der Wartbarkeit einzelner Module errechnet. Im Besonderen bei monolithischen Systemen ohne Gliederung in Module, wurden in den Fallstudien Redundanzen der Architekturmetriken auf Modul- und System-Ebene festgestellt. Eine Exploration berechneter Architekturmetriken kann mithilfe einer Liste pro Untersuchungsebene vorgenommen werden. Das Exportieren der errechneten Architekturmetriken in eine Excel-Arbeitsmappe ist zudem möglich. Darüber hinaus besteht die Möglichkeit der Festlegung von Grenzwerten für Architekturmetriken. Ein Über- oder Unterschreiten eines Grenzwertes wird als Architekturproblem angesehen. Nach Unterabschnitt 2.1.1 dient dies der Vermeidung von Architekturerosion.

Sonargraph Explorer identifiziert des Weiteren Architekturprobleme des analysierten Systems. Die Identifikation eines solchen Architekturproblems wird automatisch vorgenommen. Nach der Identifikation werden dem Benutzer die Probleme in Form einer Liste dargestellt. Elemente dieser Liste können interaktiv exploriert werden. Es besteht dabei die Möglichkeit zwischen Visualisierungstypen zu wechseln. Exemplarisch ist in Abbildung 5.9 eine zyklische Abhängigkeit von Klassen der Fallstudie JabRef [39] in Form eines Graphen zu sehen. Eine Visualisierung dieses Problems in hierarchischer Art und Weise stellt eine alternative Darstellung dar.

Tabelle 5.2.: Anzahl und Untersuchungsebenen vordefinierter Architekturmetriken der Anwendung Sonargraph Explorer [35]

Untersuchungsebene	Anzahl
System	42
Modul	39
Paket	15
Komponente	18
Quelldatei	9
Typ	7
Routine	7

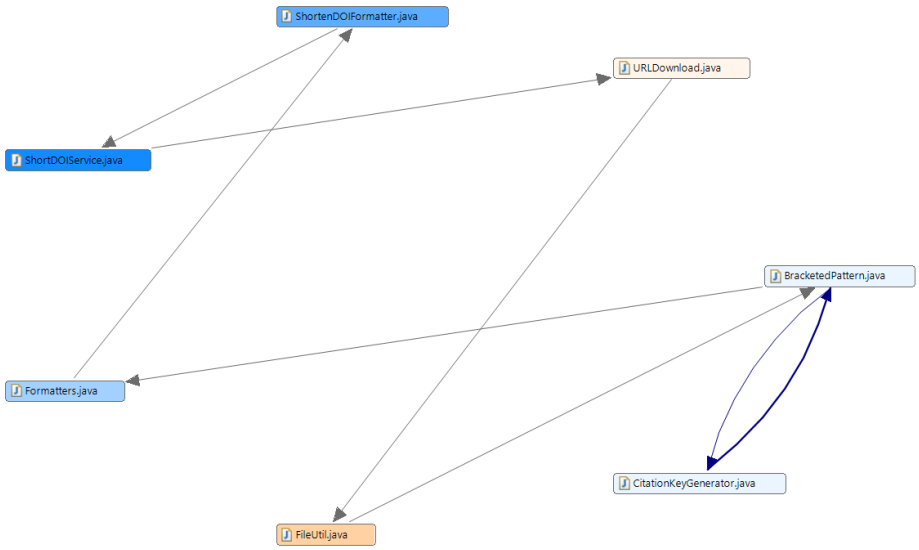


Abbildung 5.9.: Graphdarstellung einer zyklischen Abhängigkeit der Fallstudie JabRef [39] erzeugt durch Sonargraph [35]

5.2.2.4. Einschränkungen und Probleme

Eine Exploration der vorliegenden Architektur eines Systems auf unterschiedlichen Abstraktionsebenen für Klassen, Komponenten, Module und Dienste ist nicht möglich. Die Rückgewinnung bestehender Komponenten sowie Dienste Microservice-basierter Systeme kann durch Sonargraph Explorer unterstützt, jedoch nicht durchgeführt werden. Eine Unterstützung des Rückgewinnungsprozesses kann durch verbessertes Verständnis der Architektur mithilfe errechneter Architekturmetriken von Sonargraph Explorer geschehen. Des Weiteren ist eine Exploration von Klassen und Paketen sowie ihrer Relationen mithilfe der Explorationsansicht möglich. In Abbildung 5.8 ist eine solche Explorationsansicht am Beispiel der Fallstudie Sagan [53] zu sehen.

Die Anwendung Sonargraph Explorer konnte für die Evaluation nur auf bestimmte Fallstudien angewendet werden. Diese Einschränkung entsteht durch die Notwendigkeit der Quelltext- und Klassendateien. Obgleich die Quelltextdateien der Fallstudien vorhanden waren, gelang das Kompilieren für die Fallstudien Ghidra [51] und Hadoop [62] nicht. Eine Analyse dieser Fallstudien war nicht möglich. Die Analyse der Fallstudie Sock Shop [67] war aufgrund der Container-basierten Projektstruktur ebenfalls nicht möglich. Die kompatiblen Fallstudien wurden gemäß Unterunterabschnitt 5.2.2.2 vorbereitet, um das Einlesen in Sonargraph Explorer zu ermöglichen. Diese Vorbereitung umfasste je Wurzel-Modul einer Fallstudie das Konvertieren in ein IntelliJ Projekt sowie das Kompilieren der Fallstudie. Im Besonderen der Vorgang des Kompilierens erwies sich als zeitliche Einschränkung des Evaluationsprozesses. Die Fallstudie Sagan [53] konnte zudem nicht als einzelnes System eingelesen werden. Die verwendete Projektstruktur, bestehend aus drei Modulen, wurde durch Sonargraph Explorer nicht erkannt. Die Module wurden daher separat vorbereitet, in Sonargraph Explorer eingelesen und analysiert.

Eine weitere Einschränkung von Sonargraph Explorer stellen die kompatiblen Programmiersprachen dar. Die Explorer-Version erlaubt das Einlesen und Analysieren von Projekten in den Sprachen C# und Java. Die Architect-Version erlaubt darüber hinaus die Sprachen C und C++. Die Analyse von Projekten, die in anderen als den genannten Programmiersprachen geschrieben sind, ist nicht möglich. Das Modul Client der Fallstudie Sagan [53] konnte somit nicht analysiert werden, da es sich um ein JavaScript Projekt handelt.

5.2.3. Teamscale

Dieser Abschnitt dient der Evaluation der Anwendung Teamscale [17]. Teamscale bietet Funktionalitäten zur Unterstützung von Analyse, Überwachung und Optimierung der Quelltextqualität eines Projektes. Im Umgang mit Software-Architekturen umfassen die Funktionalitäten der Anwendung die Prüfung von Architekturkonformität, die Ableitung von Metriken sowie die Identifikation von Architekturproblemen. Die Funktionalitäten werden in Form einer Web-Anwendung erbracht. Diese Web-Anwendung dient der Visualisierung von Informationen sowie der Konfiguration zu analysierender Projekte.

5.2.3.1. Einrichtung und Inbetriebnahme

Die Einrichtung der Anwendung Teamscale geschieht durch das Herunterladen und Entpacken einer Zip-Datei, die auf der Web-Seite von Teamscale [17] zu finden ist. Bei Teamscale handelt es sich um eine kostenpflichtige Anwendung. Um die Anwendung ausführen zu können, muss eine Lizenzdatei im Konfigurationsverzeichnis des entpackten Verzeichnisses abgelegt werden. Eine solche Lizenzdatei kann beim Hersteller der Anwendung erworben werden. Für die Evaluation konnte eine kostenfreie Testlizenz für die Dauer von zwei Monaten beim Hersteller der Anwendung beantragt werden. Die Bestätigung dieser Testlizenz erfolgt per E-Mail und beinhaltete eine Lizenzdatei. Nach Ablegen der Datei im Konfigurationsverzeichnis kann die Anwendung gestartet werden. Die Anwendung wird in einem Kommandozeilenfenster ausgeführt. Der Benutzer wird textuell informiert, sobald der integrierte Web-Server gestartet und die Web-Anwendung erreichbar ist.

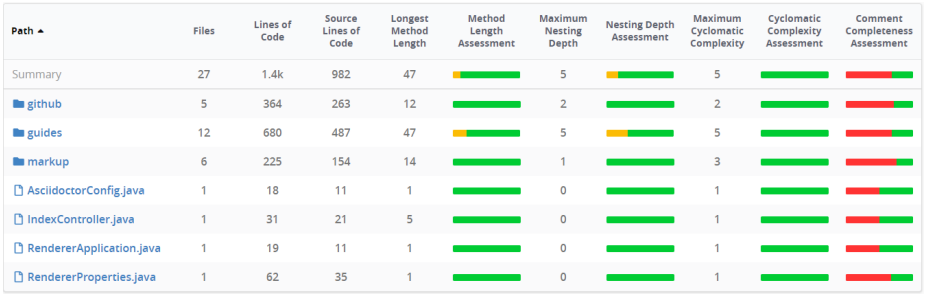
5.2.3.2. Ablauf der Anwendung

Zur Anwendung der Anwendung muss in einem Web-Browser die Web-Seite des gestarteten Servers aufgerufen werden. Diese Web-Seite umfasst die gesamte Funktionalität von Teamscale und erlaubt sowohl Konfiguration als auch Anwendung der Anwendung. Die vorliegende Arbeit beschränkt sich in der Evaluation auf Funktionalitäten, die sich auf die Software-Architektur eines Projektes beziehen.

Die Anwendung der Anwendung beginnt mit der Konfiguration eines Projektes. Hierbei wird die Eingabe eines Namens, die Auswahl eines Analyseprofiles sowie die Angabe des Projekt-Repository verlangt. Das Analyseprofil bestimmt die Programmiersprachen eines Projektes, die anzuwendenden statischen Bytecode und Quelltext Analysemethoden sowie die Verfahren zur Bestimmung der Überdeckung von Tests. Des Weiteren enthält des Analyseprofil die Qualitätsindikatoren, die auf Grundlage der Analyse eingesetzt werden. Es stehen nach der Installation bereits vorkonfigurierte Analyseprofile für die 28 unterstützten Programmiersprachen zur Verfügung. Die Erstellung weiterer Analyseprofile ist möglich. Die Angabe des Projekt-Repository erfolgt über die Eingabe von Anmeldeinformationen und URL zu einem Repository-Server. Die Nutzung lokaler Repositories ist ebenfalls möglich und geschieht durch Eingabe des Dateisystempfades zu dem Projekt. Neben der Erstkonfiguration eines neuen Projektes ist die Konfiguration von Metriken, Befunden und Software-Architekturrichtlinien möglich. Eine Beschreibung der Metriken, Befunde und Software-Architekturrichtlinien ist in Unterunterabschnitt 5.2.3.3 zu finden. Nach erfolgreicher Konfiguration wird das Projekt analysiert. Nachfolgende Analysen erfolgen, basierend auf durchgeführten Commits, inkrementell. Sobald die Analyse abgeschlossen ist, können die Ergebnisse der Analyse durch den Benutzer exploriert werden.

5.2.3.3. Beschreibung generierter Ergebnisse

Die Ergebnisse der Anwendung Teamscale stellen die errechneten Metriken, Befunde sowie die Software-Architekturkonformität dar. Bei Auslieferung der Anwendung sind bereits Metriken vorkonfiguriert. Diese Metriken umfassen unter anderem die Anzahl von Quelldateien, die Anzahl von Quelltextzeilen, Methodenlänge, Schachtelungstiefe sowie zyklische Komplexität. Ein Hinzufügen eigener Metriken ist zudem möglich. Exemplarisch



Path	Files	Lines of Code	Source Lines of Code	Longest Method Length	Method Length Assessment	Maximum Nesting Depth	Nesting Depth Assessment	Maximum Cyclomatic Complexity	Cyclomatic Complexity Assessment	Comment Completeness Assessment
Summary	27	1.4k	982	47		5		5		
github	5	364	263	12		2		2		
guides	12	680	487	47		5		5		
markup	6	225	154	14		1		3		
AsciidoctorConfig.java	1	18	11	1		0		1		
IndexController.java	1	31	21	5		0		1		
RendererApplication.java	1	19	11	1		0		1		
RendererProperties.java	1	62	35	1		0		1		

Abbildung 5.10.: Auswahl vorkonfigurierter Metriken des Moduls Sagan-Renderer der Fallstudie Sagan [53], entnommen aus der Web-Anwendung Teamscale [17]

ist in Abbildung 5.10 eine Auswahl vorkonfigurierter Dateimetriken anhand der Fallstudie Sagan [53] zu sehen. Bei den abgebildeten Dateien beziehungsweise Verzeichnissen handelt es sich um die Klassen sowie Pakete innerhalb des Wurzelpaketes des Moduls Sagan-Renderer. Neben den oben genannten numerischen Metriken umfasst die Abbildung Beurteilungen dieser Metriken, die auf Grundlage festgelegter Schwellwerte erstellt werden. Hierzu werden drei Schwellwertbereiche genutzt. Grüne Bereiche stellen optimale Werte dar, gelbe Bereiche stellen tolerierte Werte dar und rote Bereiche kennzeichnen kritische Werte. Eine Festlegung geeigneter Schwellwerte der Metriken kann somit genutzt werden, um Architekturprobleme zu identifizieren.

Ein solches Problem wird in Teamscale als Befund bezeichnet. Neben Über- und Unterschreitungen von Schwellwerten der Metriken, umfassen Befunde Fehler im Quelltext, Verletzungen von Stilrichtlinien sowie Abweichungen der Architektur von Soll-Bedingungen. Analog zu Metriken können vorhandene Befunde in Teamscale genutzt und neue Befunde erstellt werden. Die Exploration von Befunden geschieht in Listenform. Eine Detailansicht eines Befundes führt den Nutzer zudem zur fehlerhaften Quelltextstelle im Projekt.

Zur Vermeidung von Software-Erosion bietet Teamscale die Möglichkeit der Erstellung von Bedingungen für die Software-Architektur des Projektes. Eine solche Erstellung von Bedingungen für die Software-Architektur wird mithilfe des Architektur-Editors erstellt und exploriert. Ein Ausschnitt valider Bedingungen für die Software-Architektur der Fallstudie Meet & Eat Server ist in Abbildung 5.11 zu sehen. Bei der Erstellung einer solchen Soll-Architektur können drei Typen von Bedingungen in Form von Abhängigkeiten der Elemente festgelegt werden. Grüne Abhängigkeiten der Elemente sind erlaubt, gelbe Abhängigkeiten werden toleriert und rote Abhängigkeiten sind nicht erlaubt. Das Statussymbol der Abhängigkeiten zeigt dem Benutzer an, ob es in der Architektur zu Verstößen hinsichtlich der festgelegten Typen von Abhängigkeiten kommt. Ein Verstoß gegen eine Bedingung führt zur Erstellung eines Befundes. Ein solcher Verstoß ist in Abbildung 5.11 nicht dargestellt.

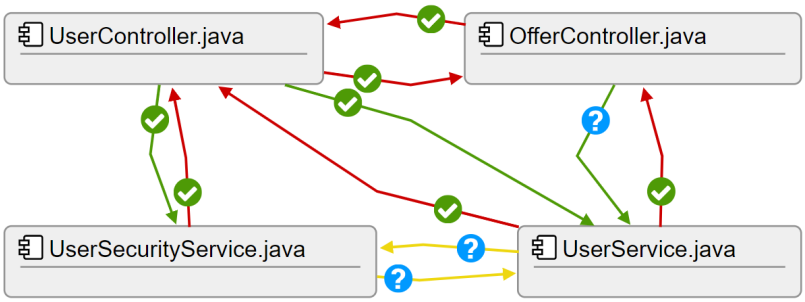


Abbildung 5.11.: Ausschnitt der Architektur der Fallstudie Meet & Eat Server [31], erstellt mithilfe des Architektur-Editors der Web-Anwendung Teamscale [17]

5.2.3.4. Einschränkungen und Probleme

Die Anwendung Teamscale bietet dem Benutzer die Möglichkeit die Soll-Architektur des Projektes manuell zu modellieren und auf Konformität mit dem Projekt zu überprüfen. Eine Einschränkung dahingehend stellt die fehlende Funktion zur automatischen Erstellung eines Modelles der aktuellen Architektur dar. Die initiale Festlegung von Bedingungen ließe sich beschleunigen, wenn auf den Aufwand der Erstellung eines Modelles der aktuellen Architektur verzichtet werden könnte. Eine Rückgewinnung der Architektur auf Klassenebene ist mithilfe von Teamscale zudem nicht möglich. Des Weiteren bietet die Anwendung Teamscale keine Möglichkeit zur Rückgewinnung von Komponenten- sowie Dienst-Ebene der Architektur eines Projektes.

Eine weitere Einschränkung stellt die Zentralisation der Anwendung dar. Da es sich um eine Server-Anwendung handelt, die ihre Daten innerhalb einer eigenen Datenbank sichert, ist eine Speicherung der Konfigurationen innerhalb eines Projektes nicht möglich. Für die Arbeit als Team ist ein Server somit notwendig.

5.2.4. Structure101 Studio

Dieser Abschnitt dient der Evaluation des Reverse-Engineering-Ansatzes Structure101 Studio [33]. Bei Structure101 handelt es sich um Anwendungen, die das Reverse-Engineering gemäß Abschnitt 2.2 mithilfe einer Menge von bereitgestellten Werkzeugen unterstützen. Structure101 Studio stellt für die Anwendung dieser Werkzeuge eine eigenständige grafische Anwendung zur Verfügung. Neben Studio bietet Structure101 die Varianten Build und Workspace an. Diese Varianten weisen ein eingeschränktes Funktionsspektrum auf, das an die Verwendung in speziellen Umgebungen angepasst ist. Bei Structure101 Build handelt es sich um eine Kommandozeilenanwendung zur Qualitätssicherung der Architektur. Hierzu kann Structure101 Build in den Ablauf der kontinuierlichen Integration eingebettet werden. Structure101 Workspace bietet explorative Werkzeuge zur Analyse einer Architektur in Form einer Erweiterung der integrierten Entwicklungsumgebungen Eclipse und IntelliJ IDEA. Aufgrund des gebotenen Funktionsspektrums wurde ausschließlich die Variante Studio für die Evaluation ausgewählt.

5.2.4.1. Einrichtung und Inbetriebnahme

Die Einrichtung und Inbetriebnahme wurden unter Verwendung einer akademischen Lizenz durchgeführt. Diese wurde durch den Hersteller von Structure101 Studio für die vorliegende Arbeit zur Verfügung gestellt. Eine Anpassung des Einrichtungsprozesses bei Verwendung einer kostenpflichtigen Lizenz kann nicht ausgeschlossen werden.

Die Einrichtung der eigenständigen grafischen Structure101 Variante Studio beginnt mit dem Herunterladen einer ausführbaren Datei. Nach Ausführung dieser Datei wird der Benutzer mittels einer Wizard-Oberfläche durch den Installationsprozess geleitet. Für die Evaluation wurde eine Installation von Structure101 Studio per Standard-Parameter der Wizard-Oberfläche vorgenommen. Nach Abschluss der Installation kann die Anwendung gestartet und verwendet werden.

5.2.4.2. Ablauf der Anwendung

Der Ablauf der Anwendung von Structure101 Studio beginnt mit der Erstellung eines neuen Structure101 Projektes. Der Erstellungsprozess des Structure101 Projektes hat zum Ziel, die Klassenpfade eines zu analysierenden Projektes zu spezifizieren. Eine solche Spezifikation der Klassenpfade beziehungsweise die Erstellung des Structure101 Projektes kann auf vier Arten vorgenommen werden. Die erste Art ist die Erstellung eines Projektes basierend auf einem bestehenden Structure101 Workspace Projekt. Die zweite Art der Erstellung geschieht mittels einer pom.xml-Datei eines Projektes des Build-Management-Werkzeuges Maven. Die dritte Art ist die Erstellung unter Verwendung einer .classpath-Datei eines bestehenden Projektes der integrierten Entwicklungsumgebung Eclipse. Die vierte Art erlaubt eine Erstellung durch statische Spezifikation der Klassenpfade eines Projektes. Letztere Art erlaubt die Spezifikation der Klassenpfade in Form von Verzeichnissen oder Jar-, War- und Ear-Archiven. Aufgrund der Heterogenität der Fallstudien hinsichtlich verwendeter Build-Management-Werkzeuge, wurde die vierte Art zur Evaluation der Fallstudien ausgewählt. Insbesondere wurden für die statische Spezifikation Verzeichnisse verwendet. Auf die Nutzung von Archiven wurde verzichtet. Der Grund hierfür ist in Unterunterabschnitt 5.2.4.4 beschrieben. Sobald die Spezifikation der Klassenpfade eines Projektes abgeschlossen ist, beginnt Structure101 Studio die Analyse der Klassendateien. Das Ergebnis dieser Analyse ist durch den Benutzer innerhalb der grafischen Benutzeroberfläche interaktiv explorierbar.

5.2.4.3. Beschreibung generierter Ergebnisse

Structure101 Studio bietet dem Benutzer die Möglichkeit der Exploration der Analyse-Ergebnisse in Form von Ansichten. Diese Ansichten dienen der Darstellung eines bestimmten Aspektes der Analyse.

Die initiale Ansicht nach abgeschlossener Analyse ist die Strukturansicht des Projektes. Die Strukturansicht bietet die Möglichkeit der Exploration der Paket- und Klassenstruktur des Projektes. Diese Exploration wird durch die sogenannte Levelized-Structure-Map (LSM), eine Ebenen-basierte interaktive Darstellung der Pakete und Klassen, ermöglicht. Eine solche LSM-Darstellung der Fallstudie Artemis [14] ist in Abbildung 5.12 zu sehen. Eine Manipulation der LSM-Darstellung dient der Simulation des Projektes nach

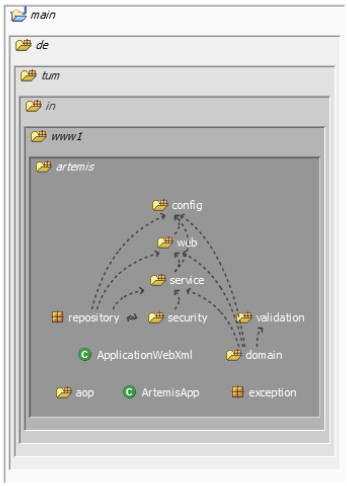


Abbildung 5.12.: Levelized-Structure-Map der Fallstudie Artemis [14] erzeugt durch Structure101 Studio [33]

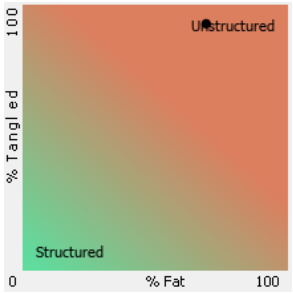
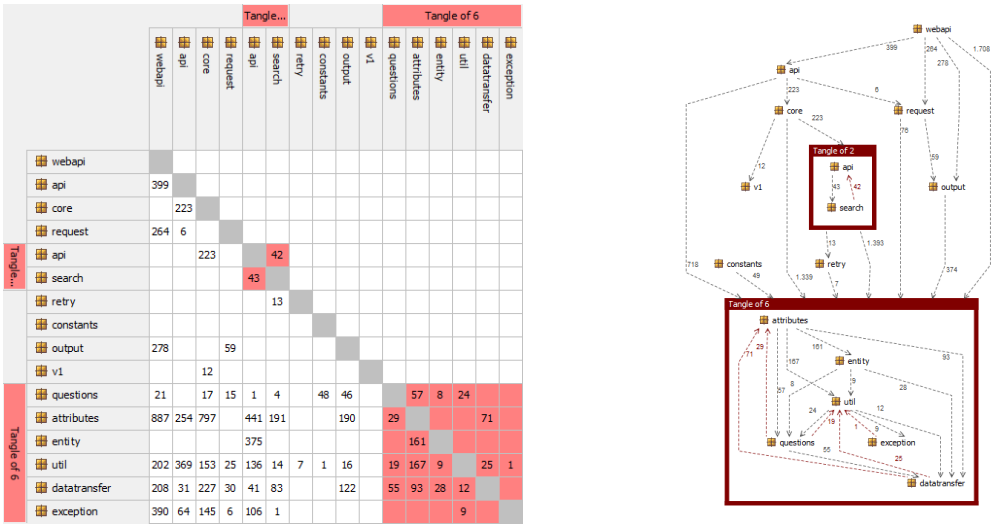


Abbildung 5.13.: Grad der Strukturiertheit der Fallstudie Artemis [14], gemäß einer Analyse der Anwendung Structure101 Studio [33]

Restrukturierungsprozessen. Dies erlaubt die Ableitung notwendiger Prozessschritte in chronologischer Reihenfolge sowie die Berechnung von Metriken auf Basis der Simulation.

Des Weiteren bietet die Strukturansicht eine Darstellung der Exzessiven-Strukturellen-Komplexitätsmetrik (XS-Metrik) des Projektes. Die XS-Metrik basiert dabei einerseits auf dem Grad der Abhängigkeiten beziehungsweise Verflechtungen von Strukturen wie Paketen, Klassen oder Methoden. Andererseits basiert die XS-Metrik auf dem Anteil sowie dem Ausmaß der Überfüllung ebendieser Strukturen. Exemplarisch ist in Abbildung 5.13 der Grad der Strukturiertheit der Fallstudie Artemis [14] auf Basis des Grades der Abhängigkeiten sowie der Überfüllung zu sehen. Die errechneten Metriken sowie weitere quantifizierbare Informationen eines Projektes sind in ihrer Gesamtheit in Form eines Berichtes exportierbar.

Neben der quantifizierten Erfassung von Architekturmetriken und Projektinformationen bietet Structure101 Studio die Möglichkeit der Visualisierung von Abhängigkeiten auf Ebene der Pakete, Klassen und Methoden. Im Besonderen ist auf Ebene der Pakete und



(a) Matrixdarstellung (b) Graphdarstellung

Abbildung 5.14.: Visualisierung der Abhängigkeiten der Pakete der Fallstudie Teammates [60], entnommen aus einer Analyse von Structure101 Studio [33]

Klassen dabei eine Erkennung und Visualisierung von Clustern möglich. Es werden durch die Anwendung auch Probleme der Architektur basierend auf Abhängigkeiten identifiziert und visualisiert. Ein solches Problem ist dabei im Besonderen eine zyklische Abhängigkeit zwischen Klassen oder Paketen. Ein solcher Zyklus ist in Structure101 Studio mithilfe einer Graph- oder Matrixdarstellung der Abhängigkeiten identifizierbar. Die Visualisierung der Abhängigkeiten der Fallstudie Teammates [60] ist in Abbildung 5.14 dargestellt. Aus Gründen der Übersichtlichkeit wurde die Ebene der Pakete zur Visualisierung gewählt. Analog ist innerhalb von Structure101 Studio auch die Visualisierung auf Ebene der Klassen oder Methoden möglich. Abbildung 5.14a stellt die Abhängigkeiten in Form einer Matrix dar. Die Einträge der Matrix stehen für die Anzahl der auftretenden gerichteten Abhängigkeiten zweier Pakete. Abbildung 5.14b stellt die Abhängigkeiten in Form eines Graphen dar. Analog zur Visualisierung in Form einer Matrix, entsprechen die Kantengewichte der Anzahl der auftretenden gerichteten Abhängigkeiten zweier Pakete. Ein Zyklus zweier Pakete, hier beispielhaft des Search- und Api-Paketes, wird durch Structure101 Studio als Architekturproblem identifiziert und visuell hervorgehoben.

5.2.4.4. Einschränkungen und Probleme

Structure101 Studio bietet Werkzeuge zur Rückgewinnung und Visualisierung von Klassen. Nach Unterunterabschnitt 5.2.4.3 ist zudem die Erkennung von Clustern möglich. Eine Spezifikation des dabei verwendeten Terminus Cluster wird innerhalb der Anwendung nicht gegeben. Ein festlegbarer Grenzwert zur Gruppierung von Klassen zu Clustern konnte in Structure101 Studio nicht aufgefunden werden. Eine Rückgewinnung von Komponenten in Form von Clustern kann ohne die Möglichkeit der Konfiguration der Parameter eines solchen Clusters nicht bestätigt werden. Des Weiteren ist Structure101 Studio nicht in der Lage Relationen zwischen Diensten Microservice-basierter Fallstudien zu erkennen. Eine

vollständige Rückgewinnung der Architektur von Komponenten- sowie Microservice-basierten Systemen konnte mithilfe der Anwendung Structure101 Studio nicht durchgeführt werden.

Eine Einschränkung von Structure101 Studio stellt die Notwendigkeit der Klassendateien eines Projektes dar. Dies erforderte vor der Anwendung der Anwendung auf die Fallstudien, dass die Fallstudien kompiliert werden müssen. Das Kompilieren der Quelltextdateien war für die Fallstudien Ghidra [51] und Hadoop [62], wegen auftretender Fehler beim Vorgang des Kompilierens, nicht möglich. Des Weiteren wird Structure101 Studio in der aktuellen fünften Version ausschließlich für die Programmiersprache Java angeboten. Die Analyse der Fallstudie Sock Shop [67] war aufgrund der Container-basierten Projektstruktur nicht möglich. Das Modul Client der Fallstudie Sagan [53] konnte aufgrund der inkompatiblen Programmiersprache JavaScript nicht analysiert werden.

Die Anwendung von Structure101 Studio auf die Fallstudien identifizierte ein Problem hinsichtlich des Erstellens eines Projektes basierend auf einer Jar-Datei. Im konkreten Fall handelte es sich dabei um die Fallstudie TimeSheetGenerator [57]. Beim Erstellen des Projektes auf Basis der Jar-Datei der Fallstudie konnte nicht korrekt zwischen Projekt-eigenen und Projekt-fremden Paketen unterschieden werden. Die Pakete org.apache sowie com.fasterxml wurden bei der Analyse ebenfalls berücksichtigt. Zur Lösung dieses Problems verfügt Structure101 Studio über die Möglichkeit Pakete von der Analyse auszuschließen. Um die Menge der notwendigen Einflussnahmen auf die Konfiguration zu minimieren, wurde für die Fallstudien die Erstellung per statischem Klassenpfad gewählt.

Eine weitere Einschränkung der Anwendung Structure101 Studio ist die Häufigkeit der Analysen. Konfigurationen, die innerhalb der Anwendung vorgenommen werden, sind in der Structure101 Projektdatei gespeichert. Eine Speicherung der vorangegangenen Analysen findet jedoch nicht statt. Dies erfordert bei jedem Start der Anwendung die erneute Analyse des Projektes, unabhängig davon, ob sich die Eingabedateien oder Konfiguration geändert haben.

5.2.5. Rational Software Architect Designer

Der folgende Abschnitt dient der Evaluation der Anwendung Rational Software Architect Designer (RSAD) [37]. Bei RSAD handelt es sich um eine Plattform, die den Umgang mit Software mittels Werkzeugen abstrakter gestaltet. Eine solche Abstraktion geschieht in RSAD durch Unterstützung der Modellbildung, Komplexitätsreduktion, Visualisierung und Identifikation von Auswirkungen sowie Problemen. Für den Umgang mit IBM WebSphere Software bietet der Hersteller IBM zudem eine alternative Variante der Anwendung an. Diese Variante wird in der Evaluation von RSAD nicht näher betrachtet.

5.2.5.1. Einrichtung und Inbetriebnahme

Da es sich bei RSAD um eine kostenpflichtige Anwendung handelt, wurde eine 30-tägige Testversion für die Evaluation genutzt. Anzumerken ist, dass eine Registrierung bei IBM zur Beantragung dieser Testversion notwendig ist. Die Einrichtung von RSAD erfolgt mithilfe eines Installationsassistenten. Dieser Installationsassistent muss in Form einer ausführbaren Datei heruntergeladen werden. Die Ausführung des Installationsassisten-

ten erfordert zwei Archive, die separat heruntergeladen werden müssen. Diese Archive beinhalten die zu installierenden Daten der Anwendung RSAD. Vor Beginn der Installation müssen Erweiterungen der zu installierenden RSAD Instanz ausgewählt werden. Für die Evaluation wurden hierbei die Architektur- sowie Modell-bezogenen Erweiterungen ausgewählt. Der Abschluss der Auswahl startet den Installationsprozess der Anwendung. Nach Abschluss der Installation ist die Anwendung bereit für die Verwendung.

5.2.5.2. Ablauf der Anwendung

Die Anwendung der Anwendung RSAD beginnt mit der Einrichtung eines neuen Projektes. Da RSAD die Eclipse Plattform nutzt, läuft die Einrichtung neuer Projekte gemäß der Vorgehensweise und den Möglichkeiten der Eclipse Plattform ab. Bei der Anwendung von RSAD auf die Fallstudien wurde eine Einrichtung auf Grundlage des genutzten Build-Management-Werkzeuges durchgeführt. Für Maven-Projekte wurde zur Einrichtung eine vorliegende Pom-Datei genutzt. Andere Projekte wurden in Form eines generischen Java-Projektes eingerichtet.

Wie oben genannt, dient RSAD der Abstraktion von Software, um die Arbeit mit ebendieser zu unterstützen. Eine Abstraktion von vorliegender Software wird hinsichtlich der Software-Architektur durch die Modellbildung erreicht. Diese geschieht durch die Möglichkeit eine Transformation des Quelltextes eines Projektes in ein Modell durchzuführen. Ein solches Modell kann durch den Benutzer interaktiv exploriert und manipuliert werden. Ein Modell kann in ein weiteres Modell überführt werden. RSAD bietet darüber hinaus die Möglichkeit Forward-Engineering durch Generierung von Quelltext aus einem Modell durchzuführen. In Kombination mit der Modellbildung bietet das Forward-Engineering die Möglichkeit des Round-Trip-Engineering in RSAD.

Neben Modellbildung und Forward-Engineering umfasst das Spektrum der Funktionalitäten von RSAD Software-Analysen. Im Anschluss an eine Software-Analyse ist eine Exploration ihrer Ergebnisse innerhalb der grafischen Benutzeroberfläche möglich. Eine Beschreibung der Software-Analysen ist in Unterunterabschnitt 5.2.5.3 zu finden.

5.2.5.3. Beschreibung generierter Ergebnisse

Hinsichtlich der Software-Architektur eines Projektes liefert RSAD zwei Ergebnisse. Die erste Art solcher Ergebnisse stellen erzeugte Modelle dar. Diese dienen der Abstraktion vom Quelltext und der Komplexitätsreduktion. RSAD bietet die Möglichkeit erstellte Modelle zu visualisieren. Eine solche Visualisierung von Modellen kann durch den Nutzer interaktiv exploriert werden. Auch Manipulationen an visualisierten Modellen sind möglich. Exemplarisch für die Transformation von Quelltext in ein Modell ist in Abbildung 5.15 das Paket Data der Fallstudie TimeSheetGenerator [57] zu sehen. Hierbei wurden die Java-Quelltextdateien der Fallstudie in ein Modell transformiert und mithilfe eines Klassendiagrammes visualisiert. Auf die Darstellung genutzter Klassen des Java-API wurde aus Gründen der Übersichtlichkeit der Abbildung verzichtet.

Die zweite Art von Ergebnissen der Anwendung RSAD werden durch Software-Analysen erbracht. Software-Analysen können zur Vermeidung der in Unterabschnitt 2.1.1 beschriebenen Architekturerosion und der Identifikation von Architekturproblemen einge-

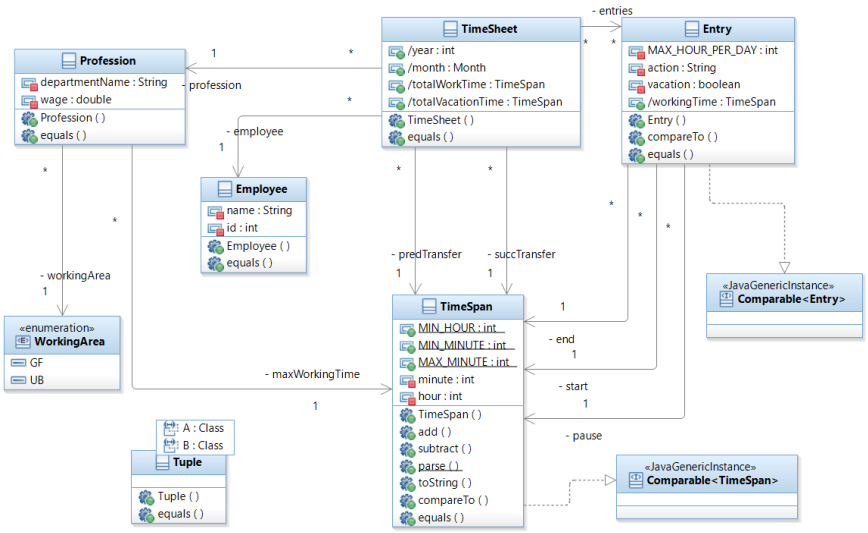


Abbildung 5.15.: Klassendiagramm des Paketes Data der Fallstudie TimeSheetGenerator [57], erstellt mit der Anwendung Rational Software Architect Designer [37]

setzt werden. Das Ergebnis einer solchen Software-Analyse ist in drei Gruppen unterteilt. Die erste Gruppe des Ergebnisses sind Software-Metriken. Diese decken die Bereiche Kohäsion, Abhängigkeit, Komplexität und Vererbung ab. Des Weiteren werden Basis-Metriken, wie beispielsweise die Anzahl der Quelltextzeilen, und Halstead-Metriken berechnet. Exemplarisch sind in Abbildung A.1 im Anhang die berechneten Software-Metriken der Fallstudie Energy State Data Analysis [43] zu sehen. Die zweite Gruppe des Ergebnisses einer Software-Analyse sind Quelltextüberprüfungen. Hierbei wird der Quelltext auf Einhaltung von Entwurfsrichtlinien, Konventionen, Sicherheit, Performanz und Best-Practise geprüft. Die letzte Gruppe ist die der Architekturentdeckung. Die Architekturentdeckung dient der Identifikation von Mustern im Projekt. Es werden durch RSAD Entwurfsmuster, Objektorientierungsmuster, Strukturmuster und Systemmuster unterschieden. Die Ergebnisse aller Gruppen können innerhalb der grafischen Benutzeroberfläche in Listenform exploriert oder in Form einer XML-Datei exportiert werden.

5.2.5.4. Einschränkungen und Probleme

Die Anwendung RSAD weist ein umfangreiches Spektrum an Funktionalitäten auf. Die Heterogenität der Funktionalitäten erschwert jedoch den Lern- sowie Evaluationsprozess der Anwendung. Im Besonderen die verfügbare Dokumentation legt den Fokus auf einzelne Funktionalitäten, anstatt zielorientierte Verfahrensabläufe zu präsentieren.

Hinsichtlich des Reverse-Engineerings bietet RSAD die Möglichkeit der Transformation des Quelltextes eines Projektes in ein Modell. Eine Einschränkung stellt hierbei die Menge der abgeleiteten Informationen dar. Es besteht beispielsweise die Möglichkeit der Berechnung von Kohäsions- und Kopplungsmetriken. Eine Ableitung möglicher Komponenten der Architektur aus diesen Informationen wird jedoch nicht angeboten. Eine Rückge-

winnung der Dienste Microservice-basierter Systeme ist zudem nicht möglich. Folglich beschränkt sich das Spektrum der Funktionalitäten hinsichtlich der Rückgewinnung von Software-Architekturen auf die Rückgewinnung der Klassenebene.

Ein weiteres Problem stellt die fehlende native Unterstützung des Build-Management-Werkzeuges Gradle dar. Zu Beginn der Evaluation der Anwendung RSAD wurde angestrebt, die Fallstudie anhand der Konfiguration ihres Build-Management-Werkzeuges als Projekt einzulesen. Obgleich dies für Maven-Projekte funktionierte, verursachten konfigurierte Abhängigkeiten der Gradle-basierter Fallstudien Fehler nach Erstellung der Projekte.

5.2.6. Enterprise Architect

Dieser Abschnitt dient der Evaluation der Anwendung Enterprise Architect [59]. Enterprise Architect wird in den Varianten Professional, Corporate, Unified und Ultimate angeboten. Für die Evaluation wurde die Variante Ultimate gewählt, da diese das gesamte Spektrum verfügbarer Funktionalitäten beinhaltet. Enterprise Architect bietet Funktionalitäten, die den Umgang mit Projekten unterstützen. Das Funktionsspektrum umfasst Funktionalitäten, die über die Arbeit an Software-Systemen hinausgehen. Die nachfolgenden Teilabschnitte der Evaluation und Untersuchung der Anwendung beziehen sich ausschließlich auf die Funktionalitäten für das Reverse-Engineering und die Arbeit mit Software-Architekturen.

5.2.6.1. Einrichtung und Inbetriebnahme

Die Einrichtung der Anwendung Enterprise Architect beginnt mit dem Herunterladen sowie dem Ausführen eines geführten Installationsprogrammes. Für die Evaluation wurden das Installationsprogramm mittels vorhandener Standardparameter konfiguriert und durchlaufen. Nach Abschluss der Installation ist Enterprise Architect ohne weitere Konfiguration ausführbar. Für die Evaluation wurde die Option einer 30-tägigen Testversion der Variante Ultimate in Anspruch genommen. Dies erforderte die Eingabe einer E-Mail-Adresse beim ersten Ausführen von Enterprise Architect. Nachfolgende Ausführungen der Anwendung erforderten keine Eingaben mehr, sondern wiesen ein erscheinendes Fenster mit der verbleibenden Testzeit auf. Nach Quittierung dieses eigenständigen Fensters öffnet sich die grafische Benutzeroberfläche der Anwendung.

5.2.6.2. Ablauf der Anwendung

Die Anwendung der Anwendung Enterprise Architect beginnt mit der Erstellung eines neuen Projektes. Die Erstellung erfordert die Eingabe eines Projektverzeichnisses. Ein bestehendes Software-System wird zum Zeitpunkt der Erstellung nicht benötigt. Für die Evaluation wurde das Wurzelverzeichnis der zu analysierenden Fallstudien als Pfad der Projektordner angegeben.

Für das Reverse-Engineering bestehender Software-Systeme stellt Enterprise Architect die Möglichkeit der Transformation des Quelltextes in ein Modell bereit. Um eine solche Transformation durchführen zu können, wird eine leere Ansicht benötigt. Diese kann innerhalb des Kontextmenüs der Wurzel des Enterprise Architect Projektes erstellt werden. Nach Auswahl dieser Ansicht gewährt der Reiter Quelltext die Möglichkeit Dateien in die

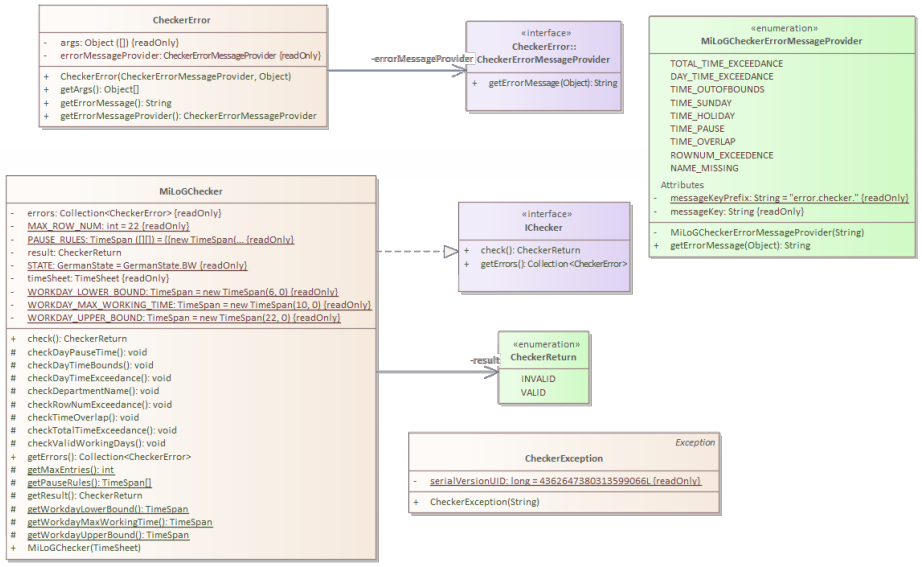


Abbildung 5.16.: Klassendiagramm des Paketes Checker der Fallstudie TimeSheetGenerator [57], erstellt durch Enterprise Architect [59]

Ansicht einzulesen. Hierfür können Quelltextdateien einzeln oder in Form eines Quellverzeichnisses angegeben werden. Für die Programmiersprache Java ist darüber hinaus die Angabe von Klassendateien an Stelle von Quelltextdateien möglich, jedoch nicht verpflichtend. Die Angabe von Quelltextdateien ist in 16 Sprachen möglich. Pro Einleseprozess ist die Angabe lediglich einer Sprache möglich. Module oder Dienste der gleichen Sprache eines Projektes können gemeinsam eingelesen werden, da Enterprise Architect das angegebene Verzeichnis rekursiv nach Quelltextdateien durchsucht. Der Einleseprozess resultiert in der Standardeinstellung in einem Modell pro eingelesener Paketebene. Eine manuelle Erstellung und Manipulation von Modellen ist in Enterprise Architect möglich. Exemplarisch ist in Abbildung 5.16 das UML-Klassendiagramm des Paketes Checker der Fallstudie TimeSheetGenerator [57] zu sehen. Dieses Diagramm wurde durch Enterprise Architect automatisch nach Abschluss des Einleseprozesses des Quellverzeichnisses der Fallstudie generiert. Eine Beschreibung der Abbildung ist in Unterunterabschnitt 5.2.6.3 zu finden.

Enterprise Architect bietet darüber hinaus Funktionalität zur Transformation eines Modelles in Quelltext. Es besteht hierbei die Möglichkeit, auf Grundlage der Informationen des Modelles, vorhandenen Modellelemente in Quelltextdateien zu transformieren. Die Generierung von Quelltext aus dem Modell einer Klasse der Fallstudie TimeSheetGenerator [57] ist in Abbildung A.2 im Anhang zu sehen. In Kombination mit der Transformation des Quelltextes in ein Modell ist in Enterprise Architect Round-Trip-Engineering möglich. Unter Round-Trip-Engineering wird eine Synchronisation von Modellen und Quelltextdateien verstanden. Somit werden Änderungen in Quelltextdateien in die Modelle übernommen und Änderungen in den Modellen führen zu geeigneten Änderungen der Quelltextdateien.

5.2.6.3. Beschreibung generierter Ergebnisse

Die Ergebnisse der Anwendung Enterprise Architect sind die Modelle nach vorangegangener Transformation von Quelltext. Ein solches Modell ist in visualisierter Form beispielhaft in Abbildung 5.16 zu sehen. Es handelt sich bei der Abbildung um ein UML-Klassendiagramm. Dieses stellt die Modellelemente, abgeleitet aus dem Quelltext der Elemente des Paketes Checker der Fallstudie TimeSheetGenerator [57], dar. Die Modellelemente enthalten neben Verhalten und Zustand der Schnittstellen auch privates Verhalten und Zustand. Neben der Visualisierung in Form eines UML-Klassendiagrammes, bietet Enterprise Architect die Möglichkeit der Visualisierung als Liste oder Abhängigkeitsmatrix. Außerdem können die Modellelemente als Teil von Ansichten zur Konstruktion und Spezifikation sowie Gantt-Diagrammen genutzt und dargestellt werden.

Die Ergebnisse von Enterprise Architect umfassen auch Quelltext, der basierend auf Modellen generiert wurde. Exemplarisch ist in Abbildung A.2 im Anhang eine generierte Klasse zu sehen. Diese generierte Klasse basiert auf den Informationen der Klasse Employee des Paketes Data der Fallstudie TimeSheetGenerator [57]. Die Quelltextdateien der Fallstudie wurde durch Enterprise Architect eingelesen und in ein Modell transformiert. Aus diesem Modell wurde die in Abbildung A.2 dargestellt Klasse generiert. Ein Vergleich der generierten Klasse mit der originalen Klasse dient im Folgenden der Einschätzung der Qualität des Ergebnisses. Der Vergleich zeigt, dass die Attribute der originalen Klasse korrekt in die generierte Klasse übergangen sind. Abweichungen sind hinsichtlich der Methoden zu erkennen. Es wurde eine zuvor nicht vorhandene finalize-Methode eingefügt. Diese steht aufgrund der fehlenden Override-Annotation in Konflikt mit der gleichnamigen Methode der Klasse Object, die seit Java SE 11 als veraltet markiert ist [52]. Des Weiteren wurde ein neuer Konstruktor ohne Parameter und Implementierung hinzugefügt, der zuvor nicht Teil der Klasse war. Die Implementierungen der Methoden und Konstruktoren wurde nicht von der originalen in die generierte Klasse übernommen, sondern durch Platzhalter ersetzt.

5.2.6.4. Einschränkungen und Probleme

Die Anwendung Enterprise Architect umfasst ein Spektrum von Funktionalitäten, deren Fokus nicht ausschließlich auf dem Reverse-Engineering und der Rückgewinnung von Architekturen liegt. Dies erschwert die Erlernbarkeit und Verständlichkeit der Anwendung. Zur Verwendung der Anwendung als Reverse-Engineering-Werkzeug wird dem Benutzer die Möglichkeit der Transformation von bestehendem Quelltext in ein Modell geboten. Dieses Modell kann durch den Benutzer exploriert werden. Eine Rückgewinnung der Klassen- und Paketebene der Architektur eines Projektes kann durch eine solche Exploration vorgenommen werden. Die Rückgewinnung von Dienst- oder Komponenten-Ebene eines Projektes aus dem Quelltext ist nicht möglich.

Obgleich Enterprise Architect 16 Sprachen beim Einlesen von Quelltextdateien unterstützt, war die Analyse des Moduls Client der Fallstudie Sagan [53] aufgrund der inkompatiblen Programmiersprache JavaScript nicht möglich. Die Analyse der Fallstudie Sock Shop [67] war aufgrund der Container-basierten Projektstruktur nicht möglich.

Eine weitere Einschränkung ist die Zeit, die zur Transformation der Quelltextdateien in ein Modell eines Projektes benötigt wird. So waren beispielsweise 240:50 Minuten nötig, um die Fallstudie Hadoop [62] in ein Modell zu transformieren. Die Größe der Projektdatei von Enterprise Architect wuchs dabei auf eine Größe von 212,39 Megabyte an. Für die Transformation der Fallstudie Artemis [14] waren 04:40 Minuten notwendig und die entstandene Projektdatei wies eine Größe von 13,9 Megabyte auf. Ein Anstieg der Prozessor- sowie Arbeitsspeicherlast konnte während der Transformation nicht beobachtet werden.

5.3. Diskussion

Dieser Abschnitt dient der Diskussion der vorangegangenen Evaluationen der Reverse-Engineering-Ansätze. Die Diskussion orientiert sich strukturell an den Stufen der Evaluation gemäß Abschnitt 4.1.

5.3.1. Einrichtung und Inbetriebnahme

Eine erfolgreiche Einrichtung und Inbetriebnahme stellte gemäß Abschnitt 4.2 eine Voraussetzung für die Auswahl eines Reverse-Engineering-Ansatzes dar. Die Abläufe der Einrichtung und Inbetriebnahme evaluierter Reverse-Engineering-Ansätze konnten in zwei Gruppen aufgeteilt werden. Die lizenzfreien Reverse-Engineering-Ansätze bildeten die erste Gruppe. Hierzu zählen die Ansätze MoDisco, jQAssistant und das Buschmais SAR-Framework. Für die Einrichtung ebendieser Ansätze genügte das Herunterladen und die Installation. Im Falle von MoDisco geschah dies innerhalb einer Eclipse Distribution. Der Ansatz jQAssistant sowie das Buschmais SAR-Framework konnten in ausführbarer Form heruntergeladen und sofort verwendet werden.

Die lizenzierten Reverse-Engineering-Ansätze bildeten die zweite Gruppe. Für die Einrichtung und Inbetriebnahme der Ansätze Sonargraph Explorer, Teamscale, Structure101 Studio, RSAD sowie Enterprise Architect war die Beantragung einer Lizenz notwendig. Die Beantragung dieser Lizenzen geschah durch Angabe personenbezogener Daten auf den Web-Seiten der Hersteller der Ansätze. Nach Angabe dieser Daten konnten die Ansätze heruntergeladen und installiert werden. Vor der ersten Inbetriebnahme mussten die beantragten Lizenzen in die Ansätze eingepflegt werden. Bei den Ansätzen Sonargraph Explorer, Teamscale und Structure101 Studio geschah dies durch eine herunterladbare Lizenzdatei. Die Ansätze RSAD und Enterprise Architect pflegten die Lizenzen selbstständig ein. Hierzu war die Einrichtung der Ansätze RSAD und Enterprise Architect mithilfe der E-Mail-Adresse notwendig, die für die Beantragung der Lizenz verwendet wurde. Nach dem Einpflegen der Lizenzen konnten die Ansätze verwendet werden.

Die Evaluationen der Reverse-Engineering-Ansätze haben gezeigt, dass die Einrichtungen und Inbetriebnahmen ohne Schwierigkeiten durchführbar waren. Insbesondere wurden die Einrichtungen und Inbetriebnahmen der Ansätze durch geführte Installationen und angebotene Dokumentation unterstützt und vereinfacht.

5.3.2. Ablauf der Anwendung

Die Evaluation des Ablaufes der Anwendung der Reverse-Engineering-Ansätze auf die Fallstudien lies eine Klassifikation der evaluierten Ansätze zu. Die erste Klasse repräsentiert zustandsbehaftete Ansätze. Diese Ansätze erstellten je Fallstudie ein Projekt. Erst nach Abschluss einer Projekterstellung war eine Durchführung des Reverse-Engineerings einer Fallstudie möglich. Die Nutzung von Projekten lies insbesondere eine Speicherung durchgeführter Konfigurationen zu. Dies erlaubte das Fortsetzen der Anwendung eines Reverse-Engineering-Ansatzes, ohne eine erneute Durchführung etwaiger Konfigurationen. Die Projekte wurden dabei entweder durch Projektdateien in den Verzeichnissen der Fallstudien oder durch zentralisierte Speicherung der Projektinformationen realisiert. Die Reverse-Engineering-Ansätze MoDisco, Sonargraph Explorer, Structure101 Studio, RSAD und Enterprise Architect nutzten Projektdateien in den Verzeichnissen der Fallstudien. Die Reverse-Engineering-Ansätze Teamscale und jQAssistant verwendeten eine zentrale Datenbank zur Speicherung der Projektinformationen.

Die zweite Klasse repräsentiert zustandslose Ansätze. In der vorliegenden Arbeit konnte der Ansatz Buschmais SAR-Framework als ein solcher zustandsloser Ansatz identifiziert werden. Dieser Ansatz speicherte durchgeführte Konfigurationen nicht.

Als ein Einflussfaktor des Ablaufes der Anwendung konnte zudem die genutzte Benutzerschnittstelle identifiziert werden. Die Ansätze MoDisco, Sonargraph Explorer, Structure101 Studio, RSAD, Enterprise Architect und Teamscale boten grafische Benutzerschnittstellen. Bei den Ansätzen MoDisco, Sonargraph Explorer sowie RSAD basierte die Benutzerschnittstelle auf der Eclipse-Plattform. Die Exploration generierter Ergebnisse konnte bei jQAssistant sowie dem Buschmais SAR-Framework ebenfalls grafisch durchgeführt werden, obgleich die Ergebnisse per Kommandozeileninteraktion generiert werden mussten.

Die Benutzerschnittstellen der Ansätze boten zudem eine Vielzahl von Funktionalitäten. Lediglich das Buschmais SAR-Framework beschränkte die gebotene Funktionalität auf die Rückgewinnung von Software-Architekturen. Eine fehlende Festlegung auf Funktionalitäten des Reverse-Engineerings erschwerte das Verständnis und erhöhte die notwendige Zeit zum Erlernen der Bedienung eines Ansatzes. Insbesondere die Abläufe in den Benutzerschnittstellen der Ansätze RSAD und Enterprise Architect waren aufgrund einer hohen Anzahl gebotener Funktionalitäten schwer erlernbar.

5.3.3. Generierte Ergebnisse

Die untersuchten Ergebnisse wurden durch Anwendung der Reverse-Engineering-Ansätze auf die Fallstudien generiert. Die Ansätze MoDisco und jQAssistant dienten dabei der Generierung von Ergebnissen in Form von Rohdaten einer Software-Architektur. Diese Rohdaten umfassten insbesondere Abhängigkeiten zwischen Elementen der Software-Architekturen der Fallstudien. MoDisco bildete aus den gewonnenen Rohdaten einer Software-Architektur ein Modell. jQAssistant speicherte die Rohdaten in Form von Graphen in einer Datenbank. Sowohl MoDisco als auch jQAssistant erlaubten dabei eine Exploration gewonnener Rohdaten. Die Rückgewinnung von Komponenten- sowie Microservice-basierten Software-Architekturen war mithilfe verfügbarer Anwendungsabläufe dieser Ansätze nicht möglich.

Die Reverse-Engineering-Ansätze Buschmais SAR-Framework, Sonargraph Explorer, Teamscale, Structure101 Studio, RSAD sowie Enterprise Architect dienten der Software-Architekturanalyse sowie -rückgewinnung. Das Buschmais SAR-Framework war hierzu auf eine vorangegangene Gewinnung der Rohdaten mithilfe von jQAssistant angewiesen. Die restlichen Ansätze führten die notwendige Gewinnung von Informationen der Software-Architektur selbstständig durch. Die generierten Ergebnisse der Ansätze umfassten im Besonderen zwei Schwerpunkte. Die Ansätze Sonargraph Explorer, Teamscale, Structure101 Studio und RSAD erlaubten eine Analyse der Software-Architektur durch berechnete Metriken sowie die Erkennung von Software-Architekturproblemen. Sowohl die berechneten Metriken als auch die erkannten Software-Architekturprobleme wiesen eine hohe Diversität auf. Das Fehlen von Dokumentation zur Herleitung berechneter Metriken und erkannter Software-Architekturprobleme erschwerte die Nachvollziehbarkeit durchgeführter Software-Architekturanalysen.

Eine Rückgewinnung von Software-Architekturen der Fallstudien war durch Anwendung der Reverse-Engineering-Ansätze Buschmais SAR-Framework, Sonargraph Explorer, Structure101 Studio, RSAD sowie Enterprise Architect eingeschränkt möglich. Die Ansätze RSAD und Enterprise Architect vollzogen diese Rückgewinnung in Form eines generierten Modelles. Die Ansätze Sonargraph Explorer und Structure101 Studio erzeugten Graphen zur Visualisierung und Exploration einer Software-Architektur. Der Ansatz Buschmais SAR-Framework erlaubte eine solche Visualisierung und Exploration mittels eines Sehnendiagrammes und einer Kreispackung der Software-Architekturelemente. Eine Rückgewinnung von Klassen und Paketen konnte mithilfe der Ergebnisse der Ansätze durchgeführt werden. Eine Rückgewinnung von Komponenten- sowie Microservice-basierten Software-Architekturen konnte auf Grundlage der generierten Ergebnisse nicht durchgeführt werden. Ein Export zugrundeliegender Software-Architekturinformationen war zudem nicht mithilfe jedes Ansatzes möglich. Exportierbare Rohdaten wiesen darüber hinaus Ansatzspezifische Form und Struktur auf.

5.3.4. Einschränkungen und Probleme

Die Anwendung der Reverse-Engineering-Ansätze auf die Fallstudien erlaubte eine Identifikation von Einschränkungen und Problemen. Dieser Abschnitt dient im Besonderen der Nennung von Einschränkungen und Problemen, die im Rahmen mehrerer Ansatzevaluationen beobachtet werden konnten.

Die Notwendigkeit kompilierter Quelltextdateien stellte eine dieser Einschränkungen dar. Der Vorgang des Kompilierens als Voraussetzung des darauffolgenden Reverse-Engineerings stellte sich als Einschränkung hinsichtlich der benötigten Zeit sowie Kompatibilität heraus. Diese Einschränkung konnte bei den Reverse-Engineering-Ansätzen Structure101 Studio, Sonargraph Explorer, Buschmais SAR-Framework und jQAssistant festgestellt werden. Eine Anwendung dieser Ansätze konnte auf bestimmte Fallstudien nicht vorgenommen werden. Dies galt für Fallstudien, die im Zuge dieser Arbeit nicht kompiliert werden konnten.

Die notwendige Zeit zur Generierung eines Ergebnisses stellte eine weitere beobachtete Einschränkung der Ansätze dar. Diese Einschränkung konnte insbesondere bei den Ansätzen Buschmais SAR-Framework, jQAssistant und Enterprise Architect festgestellt werden.

Das Maximum der benötigten Zeit bildete der Transformationsprozess des Quelltextes der Fallstudie Hadoop in ein Modell mithilfe des Ansatzes Enterprise Architect. Hierfür waren insgesamt 240:50 Minuten notwendig.

Neben der Zeit konnte der notwendige Speicherplatz für gewonnene Rohdaten als Einschränkung identifiziert werden. Diese Einschränkung wurde bei den Ansätzen Buschmais SAR-Framework sowie jQAssistant festgestellt.

Eine weitere Einschränkung der Ansätze stellten die kompatiblen Programmiersprachen dar. Die Kompatibilität der Ansätze beschränkte sich dabei meist auf die Programmiersprachen Java oder C/C++. Die Ansätze Teamscale und Enterprise Architect bildeten dabei eine Ausnahme mit 29 beziehungsweise 16 unterstützten Programmiersprachen.

Eine Einschränkung hoher Relevanz in vorliegender Arbeit stellte die fehlende Funktionalität zur Rückgewinnung Komponenten- sowie Microservice-basierter Software-Architekturen dar. Obgleich eine Rückgewinnung von Klassen und Paketen sowie deren Abhängigkeiten mithilfe der Ansätze möglich war, ermöglichte keiner der evaluierten Ansätze im Zuge dieser Arbeit eine Rückgewinnung der Komponenten und Microservices der Software-Architektur einer Fallstudie.

6. Fazit

Dieses Kapitel bildet den Abschluss der vorliegenden Arbeit. In Abschnitt 6.1 ist eine Schlussfolgerung der Arbeit zu finden. Diese dient insbesondere der Beantwortung der Forschungsfragen, die in Abschnitt 1.2 genannt werden. Des Weiteren ist in Abschnitt 6.2 eine Erläuterung identifizierter Einschränkungen und deren Auswirkung auf die Validität des Ergebnisses zu finden. Das Ende dieses Kapitels bildet Abschnitt 6.3 und gewährt einen Ausblick sowie einen Vorschlag für zukünftige Arbeit.

6.1. Schlussfolgerung

Die in Kapitel 5 durchgeführten Evaluationen der Ansätze zeigen, dass die Ergebnisse der Anwendung der Ansätze auf die Fallstudien diverse Formen aufweisen. Eine Überführung der besprochenen Ergebnisse in eine einheitliche Form ist aufgrund der Heterogenität der enthaltenen Informationen nicht möglich. Im Besonderen ein Vergleich der Ansätze mithilfe einer Metrik, ist ohne die Möglichkeit der Vereinheitlichung der Ergebnisse nicht möglich. Auch eine globale Diskussion der erhaltenen Informationen ist auf Grundlage der Diversität der Ergebnisse nur eingeschränkt durchführbar.

Die Untersuchung der Ergebnisse ermöglicht das Erkennen zweier Gruppen von Ergebnissen. Die erste Gruppe besteht aus Metriken, die auf Basis des Quelltextes oder kompilierter Quelltextdateien berechnet werden. Ein Export berechneter Metriken ist in den Ansätzen möglich. Es können zudem Gemeinsamkeiten in den berechneten Metriken festgestellt werden. Die Seltenheit dieser Gemeinsamkeiten sowie fehlende Dokumentation über die Berechnungsgrundlage der Metriken lässt einen Vergleich errechneter Werte nur eingeschränkt zu. Die zweite Gruppe umfasst Modelle, die durch eine Überführung des Quelltextes oder kompilierter Quelltextdateien erzeugt werden. Eine Exploration der erzeugten Modelle war in den Anwendungen der Ansätze möglich. Ein Export der Modelle war nicht mithilfe jedes Ansatzes möglich. Folglich konnte kein Vergleich basierend auf exportierten Modellen vorgenommen werden.

Eine Rückgewinnung von Software-Architekturen mithilfe der Ergebnisse der Ansätze war nur auf Ebene der Klassen und Pakete möglich. Jedoch konnte ein Gruppieren von Klassen auf Grundlage errechneter Kopplungs- und Kohäsionsmetriken sowie bestimmter Abhängigkeiten beobachtet werden. Eine eindeutige Identifikation von Komponenten aus genannten Gruppierungen unterstützten die Ansätze nicht. Die Rückgewinnung einer komponentenbasierten Software-Architektur aus erzeugten Modellen konnte somit nicht durchgeführt werden. Eine Identifikation von Abhängigkeiten enthaltener Microservices der Fallstudien war mit keinem der Ansätze möglich. Folglich konnten die Microservice-basierten Software-Architekturen der Fallstudien nicht zurückgewonnen werden.

Zusammenfassend lässt sich feststellen, dass eine Unterstützung des Reverse-Engineering-Prozesses durch die untersuchten Ansätze vorgenommen werden kann. Eine vollständig automatisierte Rückgewinnung der Software-Architektur konnte mithilfe der Ansätze jedoch nur auf Ebene der Klassen und Pakete durchgeführt werden. Insbesondere Komponenten- sowie Microservice-basierte Software-Architekturen konnten nicht zurückgewonnen werden.

6.2. Einschränkungen der Validität

Eine Einschränkung der vorliegenden Arbeit stellt die Verwendung eines einzigen Computers für die Generierung der Ergebnisse dar. Die evaluierten Reverse-Engineering-Ansätze wurden auf diesem Computer eingerichtet und auf die Fallstudien angewandt. Eine Abhängigkeit der Ergebnisse vom verwendeten Computer kann nicht ausgeschlossen werden. Insbesondere nicht-funktionale Eigenschaften der Reverse-Engineering-Ansätze könnten sich durch einen Wechsel des verwendeten Computers ändern. Eine Einflussnahme des verwendeten Prozessors und Arbeitsspeichers auf die benötigte Zeit zur Anwendung eines Reverse-Engineering-Ansatzes auf eine Fallstudie gilt als möglich. Darüber hinaus konnte der Aufgabenplaner des Betriebssystems Windows 10 als möglicher Einflussfaktor auf die benötigte Zeit identifiziert werden. Eine Beeinflussung der Anwendung eines Ansatzes durch laufende Prozesse auf dem Computer kann ebenfalls nicht ausgeschlossen werden. Für die Dauer einer Anwendung eines Reverse-Engineering-Ansatzes wurden stets alle anderen Benutzerprogramme beendet, um die Beeinflussung durch Vordergrundprozesse zu minimieren.

Eine weitere Einschränkung dieser Arbeit stellt die Verfügbarkeit der Reverse-Engineering-Ansätze dar. Insgesamt wurden für diese Arbeit elf Hersteller kostenpflichtiger oder nicht frei verfügbarer Reverse-Engineering-Ansätze kontaktiert und um ihre Mithilfe bei der Realisierung dieser Arbeit gebeten. Drei der angefragten Hersteller lehnten diese Anfrage ab, sieben Hersteller beantworteten die Anfrage nicht und ein Hersteller stimmte der Anfrage zu. Durch die Nutzung kostenfreier alternativer Varianten sowie Testversionen konnte die Zahl der nicht verfügbaren Reverse-Engineering-Ansätze minimiert werden.

Eine weitere Einschränkung dieser Arbeit stellt eine mögliche Fehlbedienung der untersuchten Reverse-Engineering-Ansätze dar. Hierzu zählt insbesondere das Nichtauffinden von Funktionalitäten zur Rückgewinnung von Software-Architekturen. Auch eine fehlerhafte Konfiguration der Reverse-Engineering-Ansätze kann nicht ausgeschlossen werden. Darüber hinaus wurde eine fehlerhafte Interpretation generierter Ergebnisse nach Anwendung der Reverse-Engineering-Ansätze auf die Fallstudien als mögliche Fehlerquelle identifiziert. Zur Minimierung dieser Einflussfaktoren wurde auf verfügbare Dokumentationen der Hersteller der untersuchten Reverse-Engineering-Ansätze zurückgegriffen. Aufgrund des Umfanges genutzter Dokumentationen, mangelnder Verfügbarkeit anderer Dokumentationen sowie der Komplexität untersuchter Reverse-Engineering-Ansätze kann eine Fehlbedienung dennoch nicht ausgeschlossen werden.

6.3. Ausblick und zukünftige Arbeit

Diese Arbeit zeigt auf, dass Ansätze zur Rückgewinnung moderner Software-Architekturen nicht dem Standard angehören. Nach Francesco, Lago und Malavolta [24] gaben 14 von 18 befragten IT-Fachleuten an, dass sie Informationen bestehender Systeme für die Erstellung von Architekturen nachfolgender Microservice-basierter Systeme nutzen. Nur einer der 18 Befragten nutzte hierfür Informationen eines Werkzeugs zur Rückgewinnung von Architekturen.

Auf Grundlage der gewonnenen Informationen schlägt die vorliegende Arbeit die Anpassung bestehender Reverse-Engineering-Ansätze an Software-Architekturen mit hoher Aktualität vor. Insbesondere muss eine Identifikation von funktionalen und nicht-funktionalen Anforderungen vorgenommen werden, die die potenziellen Benutzer der Reverse-Engineering-Ansätze miteinbezieht. So soll sichergestellt werden, dass die Ergebnisse eines Ansatzes zukünftige Arbeit mit Software-Architekturen optimal unterstützen. Das vorgeschlagene Vorgehen sieht zu Beginn eine Umfrage unter IT-Fachleuten vor. Diese Umfrage dient dem Zweck der Identifikation relevanter und essenzieller Anforderungen an einen Reverse-Engineering-Ansatz. Anhand der Anforderungen soll ein Ansatz entworfen, implementiert und mithilfe von Fallstudien evaluiert werden. Zur Überprüfung der Gebrauchstauglichkeit des Ansatzes empfiehlt die vorliegende Arbeit die Durchführung des Evaluationsverfahrens Cognitive-Walkthrough.

Literatur

[1] F. Arcelli u. a. „A Comparison of Reverse Engineering Tools Based on Design Pattern Decomposition“. In: *2005 Australian Software Engineering Conference*. IEEE, 2005. DOI: 10.1109/aswec.2005.5.

[2] Archium Technology Pty Ltd. *Archium*. URL: https://www.archium.io/ (besucht am 29. 04. 2021).

[3] Auburn University. *jGRASP*. URL: https://www.jgrasp.org/index.html (besucht am 29. 04. 2021).

[4] Axivion GmbH. *Axivion Suite*. URL: https://www.axivion.com/produkte/axivion-suite/ (besucht am 29. 04. 2021).

[5] Len Bass, Paul Clements und Rick Kazman. *Software Architecture in Practice*. Pearson ITP, 25. Sep. 2012. URL: https://www.ebook.de/de/product/21359109/len_bass_paul_clements_rick_kazman_software_architecture_in_practice.html.

[6] Kenny Bastani. *Spring Cloud Event Sourcing Example*. URL: https://github.com/kbastani/spring-cloud-event-sourcing-example (besucht am 29. 04. 2021).

[7] Berndt Bellay und Harald Gall. „A comparison of four reverse engineering tools“. In: *Proceedings of the Fourth Working Conference on Reverse Engineering*. IEEE Comput. Soc, 1997. DOI: 10.1109/wcre.1997.624571.

[8] BigBlueButton Inc. *BigBlueButton*. URL: https://github.com/bigbluebutton/bigbluebutton (besucht am 29. 04. 2021).

[9] Jan Bosch, Clemens Szyperski und Wolfgang Weck. „Component-Oriented Programming“. In: *Object-Oriented Technology ECOOP 2002 Workshop Reader*. Springer Berlin Heidelberg, 2002, S. 70–78. DOI: 10.1007/3-540-36208-8_6.

[10] Hugo Brunelière u. a. „MoDisco: A model driven reverse engineering framework“. In: *Information and Software Technology* 56.8 (Aug. 2014), S. 1012–1032. DOI: 10.1016/j.infsof.2014.04.007.

[11] Buschmais GbR. *jQAssistant Developer Blog*. URL: https://jqassistant.org/ (besucht am 29. 04. 2021).

[12] Buschmais GbR. *SAR-Framework*. URL: https://github.com/buschmais/sar-framework (besucht am 29. 04. 2021).

[13] CAST. *Imaging*. URL: https://www.castsoftware.com/products/imaging (besucht am 29. 04. 2021).

[14] Chair for Applied Software Engineering, Technical University of Munich. *Artemis*. URL: https://github.com/ls1intum/Artemis (besucht am 29. 04. 2021).

[15] E.J. Chikofsky und J.H. Cross. „Reverse engineering and design recovery: a taxonomy“. In: *IEEE Software* 7.1 (Jan. 1990), S. 13–17. DOI: 10.1109/52.43044.

[16] CloudScale Consortium. *CloudStore*. URL: https://github.com/CloudScale-Project/CloudStore (besucht am 29. 04. 2021).

[17] CQSE GmbH. *Teamscale*. URL: https://www.cqse.eu/de/teamscale/overview/ (besucht am 29. 04. 2021).

[18] Donald Smith. *The Future of JavaFX and other Java Client Roadmap Updates*. 2018. URL: https://blogs.oracle.com/java-platform-group/the-future-of-javafx-and-other-java-client-roadmap-updates (besucht am 29. 04. 2021).

[19] Eclipse Foundation. *Eclipse Wiki: MoDisco*. URL: https://wiki.eclipse.org/MoDisco (besucht am 29. 04. 2021).

[20] Eclipse Foundation. *MoDisco*. URL: https://projects.eclipse.org/projects/modeling.modisco (besucht am 29. 04. 2021).

[21] Eclipse Foundation. *Papyrus Software Designer*. URL: https://wiki.eclipse.org/Papyrus_Software_Designer (besucht am 29. 04. 2021).

[22] Thomas Eisenbarth, Rainer Koschke und Gunther Vogel. „Static object trace extraction for programs with pointers“. In: *Journal of Systems and Software* 77.3 (Sep. 2005), S. 263–284. DOI: 10.1016/j.jss.2004.04.028.

[23] Martin Fowler und James Lewis. *Microservices: a definition of this new architectural term*. 2014. URL: https://martinfowler.com/articles/microservices.html (besucht am 29. 04. 2021).

[24] Paolo Di Francesco, Patricia Lago und Ivano Malavolta. „Migrating Towards Microservice Architectures: An Industrial Survey“. In: *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, Apr. 2018. DOI: 10.1109/icsa.2018.00012.

[25] David Garlan und Mary Shaw. „An introduction to software architecture“. In: *Advances in Software Engineering and Knowledge Engineering*. World Scientific, Dez. 1993, S. 1–39. DOI: 10.1142/9789812798039_0001.

[26] I. Gorton und Liming Zhu. „Tool support for just-in-time architecture reconstruction and evaluation: an experience report“. In: *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* IEEe, 2005. DOI: 10.1109/icse.2005.1553597.

[27] Gradle Inc. *Gradle Build Tool*. URL: https://gradle.org/ (besucht am 29. 04. 2021).

[28] Giona Granchelli u. a. „MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems“. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, Apr. 2017. DOI: 10.1109/icsaw.2017.9.

[29] Giona Granchelli u. a. „Towards Recovering the Software Architecture of Microservice-Based Systems“. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, Apr. 2017. DOI: 10.1109/icsaw.2017.48.

[30] Moritz Gstuer u. a. *Meet-Eat-Data*. 2020. URL: https://github.com/meet-eat/meet-eat-data (besucht am 29. 04. 2021).

[31] Moritz Gstuer u. a. *Meet-Eat-Server*. 2020. URL: https://github.com/meet-eat/meet-eat-server (besucht am 29. 04. 2021).

[32] Y.-G. Gueheneuc, K. Mens und R. Wuyts. „A comparative framework for design recovery tools“. In: *Conference on Software Maintenance and Reengineering (CSMR'06)*. IEEE, 2006. DOI: 10.1109/csmr.2006.1.

[33] Headway Software Technologies Ltd. *Structure101*. URL: https://structure101.com (besucht am 29. 04. 2021).

[34] hello2morrow GmbH. *Kontrolle über Architekturerosion durch Softwarearchitektur-Management*. 2009. URL: https://files.ifi.uzh.ch/rerg/arvo/si-se/sise2009/slides/20090130siseSchoen.pdf (besucht am 29. 04. 2021).

[35] hello2morrow GmbH. *Sonargraph Product Family*. URL: https://www.hello2morrow.com/products/sonargraph (besucht am 29. 04. 2021).

[36] IBM Corp. *Acme Air*. 2013. URL: https://github.com/acmeair/acmeair (besucht am 29. 04. 2021).

[37] IBM Corp. *Rational Software Architect Designer*. URL: https://www.ibm.com/products/rational-software-architect-designer (besucht am 29. 04. 2021).

[38] Imagix Corp. *Imagix 4D*. URL: https://www.imagix.com/ (besucht am 29. 04. 2021).

[39] *JabRef*. 2020. URL: https://github.com/JabRef/jabref (besucht am 29. 04. 2021).

[40] JetBrains s.r.o. *IntelliJ IDEA Ultimate*. URL: https://www.jetbrains.com/idea/ (besucht am 29. 04. 2021).

[41] Jonatan Kaźmierczak. *Class Visualizer*. URL: http://www.class-visualizer.net/ (besucht am 29. 04. 2021).

[42] Jürgen Dufner. *Calculate metrics*. 2019. URL: https://101.jqassistant.org/calculate-metrics/index.html (besucht am 29. 04. 2021).

[43] Karlsruher Institut für Technologie. *Energy State Data Analysis*. URL: https://github.com/kit-sdq/esda (besucht am 29. 04. 2021).

[44] Karlsruher Institut für Technologie. *MediaStore*. URL: https://svnserver.informatik.kit.edu/i43/svn/code/CaseStudies/MediaStore3/trunk/Implementation/ (besucht am 29. 04. 2021).

[45] Jörg Kienzle u. a. „Towards Concern-Oriented Design of Component-Based Systems“. In: *Proceedings of the 3rd International Workshop on Interplay of Model-Driven and Component-Based Software Engineering co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2016), Saint-Malo, France, October 2nd, 2016*. Hrsg. von Federico Ciccozzi und Ivano Malavolta. Bd. 1723. CEUR Workshop Proceedings. CEUR-WS.org, 2016, S. 31–36. URL: http://ceur-ws.org/Vol-1723/5.pdf.

[46] Joakim von Kistowski u. a. „TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research“. In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, Sep. 2018. DOI: 10.1109/mascots.2018.00030.

[47] Martin Kleehaus u. a. „MICROLYZE: A Framework for Recovering the Software Architecture in Microservice-Based Environments“. In: *Information Systems in the Big Data Era*. Hrsg. von Jan Mendling und Haralambos Mouratidis. Cham: Springer International Publishing, 2018, S. 148–162. ISBN: 978-3-319-92901-9. DOI: 10.1007/978-3-319-92901-9_14.

[48] Lattix. *Lattix Architect*. URL: https://www.lattix.com/products-architecture-issues/#architect (besucht am 29. 04. 2021).

[49] Alexander Lukyanchikov. *Piggy Metrics*. URL: https://github.com/sqshq/piggymetrics (besucht am 29. 04. 2021).

[50] Marc Miltenberger u. a. *CodeInspect*. URL: https://blogs.uni-paderborn.de/sse/tools/codeinspect/ (besucht am 29. 04. 2021).

[51] National Security Agency. *Ghidra*. 2020. URL: https://github.com/NationalSecurityAgency/ghidra (besucht am 29. 04. 2021).

[52] Oracle. *Class Object*. URL: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Object.html (besucht am 29. 04. 2021).

[53] Pivotal Software, Inc. *Sagan*. 2014. URL: https://github.com/spring-io/sagan (besucht am 29. 04. 2021).

[54] Marie Christin Platenius, Markus von Detten und Steffen Becker. „Archimetrix: Improved Software Architecture Recovery in the Presence of Design Deficiencies“. In: *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, März 2012. DOI: 10.1109/csmr.2012.33.

[55] Aoun Raza, Gunther Vogel und Erhard Plödereder. „Bauhaus – A Tool Suite for Program Analysis and Reverse Engineering“. In: *Reliable Software Technologies – Ada-Europe 2006*. Hrsg. von Luís Miguel Pinho und Michael González Harbour. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 71–82. ISBN: 978-3-540-34664-7. DOI: 10.1007/11767077_6.

[56] S. Rugaber und K. Stirewalt. „Model-driven reverse engineering“. In: *IEEE Software* 21.4 (Juli 2004), S. 45–53. DOI: 10.1109/ms.2004.23.

[57] Yves Schneider. *TimeSheetGenerator*. 2019. URL: https://github.com/kit-sdq/TimeSheetGenerator (besucht am 29. 04. 2021).

[58] Soyatec. *eUML2*. URL: http://www.soyatec.com/euml2/ (besucht am 29. 04. 2021).

[59] Sparx Systems Pty Ltd. *Enterprise Architect*. URL: https://www.sparxsystems.com/products/ea/index.html (besucht am 29. 04. 2021).

[60] *TEAMMATES*. URL: https://github.com/TEAMMATES/teammates/ (besucht am 29. 04. 2021).

[61] The Apache Software Foundation. *Apache Commons Lang*. 2020. URL: https://github.com/apache/commons-lang (besucht am 29. 04. 2021).

[62] The Apache Software Foundation. *Apache Hadoop*. URL: https://github.com/apache/hadoop (besucht am 29. 04. 2021).

[63] The Apache Software Foundation. *Apache Maven*. URL: https://maven.apache.org/ (besucht am 29. 04. 2021).

[64] The Apache Software Foundation. *OFBiz*. URL: https://github.com/apache/ofbiz-framework (besucht am 29. 04. 2021).

[65] Visual Paradigm. *Visual Paradigm*. URL: https://www.visual-paradigm.com/ (besucht am 29. 04. 2021).

[66] VMware, Inc. *Spring PetClinic*. URL: https://github.com/spring-petclinic/spring-petclinic-microservices (besucht am 29. 04. 2021).

[67] Weaveworks. *Sock Shop*. 2017. URL: https://microservices-demo.github.io/ (besucht am 29. 04. 2021).

[68] Weaveworks. *Weave Scope: Topology Mapping*. URL: https://www.weave.works/docs/scope/latest/features/#topology-mapping (besucht am 29. 04. 2021).

[69] Eberhard Wolff. *Microservice Sample*. URL: https://github.com/ewolff/microservice (besucht am 29. 04. 2021).

[70] Yatta Solutions GmbH. *Round-Trip-Engineering NG*. URL: https://www.uml-lab.com/de/uml-lab/features/roundtrip/ (besucht am 29. 04. 2021).

[71] Yatta Solutions GmbH. *UML Lab*. URL: https://www.uml-lab.com/de/uml-lab/ (besucht am 29. 04. 2021).

A. Anhang

Rule	Metric
Basic Metrics	
Average lines of code per method	0,57
Average number of attributes per class	0,00
Average number of comments	0,00
Average number of constructors per class	1,00
Average number of methods	6,00
Average number of parameters	0,67
Comment/Code Ratio	0,00 %
Lines of code	46,00
Number of attributes	0,00
Number of comments	0,00
Number of constructors	1,00
Number of import statements	2,00
Number of interfaces	0,00
Number of lines	59,00
Number of methods	6,00
Number of parameters	4,00
Number of types per package	1,00
Cohesion Metrics	
Lack of cohesion 1	0,00
Lack of cohesion 2	0,43
Lack of cohesion 3	0,50
Complexity Metrics	
Average block depth	1,14
Cyclomatic complexity	1,57
Maintainability index	243,09
Weighted methods per class	11,00
Dependency Metrics	
Abstractness	0,00
Afferent coupling	0,00
Efferent coupling	1,00
Instability	1,00
Normalized Distance	0,00
Halstead Metrics	
Difficulty level	24,72
Effort to implement	25728,84
Number of delivered bugs	0,29
Number of operands	89,00
Number of operators	104,00
Number of unique operands	27,00
Number of unique operators	15,00
Program length	193,00
Program level	0,04
Program vocabulary size	42,00
Program volume	1040,72
Time to implement	0,40 hours
Inheritance Metrics	
Depth of Inheritance	1,00

Abbildung A.1.: Ergebnismetriken einer Software-Analyse der Fallstudie Energy State Data Analysis [43], durchgeführt mit Rational Software Architect Designer [37]

```java
package data;


/**
 * Represents an employee.
 * @author morit
 * @version 1.0
 * @created 03-Apr-2021 14:55:16
 */
public class Employee {

        private final int id;
        private final String name;

        public Employee(){

        }

        public void finalize() throws Throwable {

        }
        /**
         * Constructs a new employee instance.
         *
         * @param name    - The full name of the employee
         * @param id    - The employee id
         */
        public Employee(String name, int id){

        }

        /**
         *
         * @param other
         */
        @Override
        public boolean equals(Object other){
                return false;
        }

        /**
         * Gets the id of an employee.
         * @return The id of the employee.
         */
        public int getId(){
                return 0;
        }

        /**
         * Gets the name of an employee.
         * @return The name of the employee.
         */
        public String getName(){
                return "";
        }
}//end Employee
```

Abbildung A.2.: Quelltext der Klasse Employee der Fallstudie TimeSheetGenerator [57], abgeleitet aus einem UML-Modell durch Enterprise Architect [59]