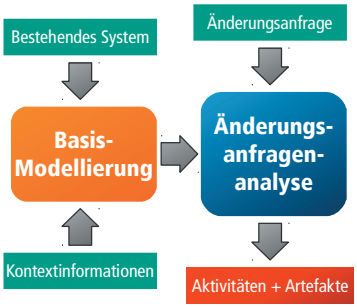
The Karlsruhe Series on
Software Design
and Quality
19

Architekturbasierte Bewertung und Planung von Änderungsanfragen

Johannes Josef Stammel



KIT Scientific Publishing

Johannes Josef Stammel

Architekturbasierte Bewertung und Planung von Änderungsanfragen

The Karlsruhe Series on Software Design and Quality
Volume 19

Chair Software Design and Quality
Faculty of Computer Science
Karlsruhe Institute of Technology

and

Software Engineering Division
Research Center for Information Technology (FZI), Karlsruhe

Editor: Prof. Dr. Ralf Reussner

Architekturbasierte Bewertung und Planung von Änderungsanfragen

von
Johannes Josef Stammel

KIT Scientific Publishing

Dissertation, Karlsruher Institut für Technologie (KIT)
Fakultät für Informatik
Tag der mündlichen Prüfung: 3. Juni 2015
Erster Gutachter: Prof. Dr. Ralf H. Reussner
Zweiter Gutachter: Prof. Dr. Andreas Oberweis

Impressum

Karlsruher Institut für Technologie (KIT)
KIT Scientific Publishing
Straße am Forum 2
D-76131 Karlsruhe

KIT Scientific Publishing is a registered trademark of Karlsruhe Institute of Technology. Reprint using the book cover is not allowed.

www.ksp.kit.edu

This document – excluding the cover, pictures and graphs – is licensed under the Creative Commons Attribution-Share Alike 3.0 DE License (CC BY-SA 3.0 DE): http://creativecommons.org/licenses/by-sa/3.0/de/

The cover page is licensed under the Creative Commons Attribution-No Derivatives 3.0 DE License (CC BY-ND 3.0 DE): http://creativecommons.org/licenses/by-nd/3.0/de/

Print on Demand 2017 – Gedruckt auf FSC-zertifiziertem Papier

ISSN 1867-0067
ISBN 978-3-7315-0524-2
DOI: 10.5445/KSP/1000054452

Zusammenfassung

Die Software-Architektur umfasst die technische Organisation eines Software-Systems und die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen. Die Projektplanung trifft die organisatorischen Entscheidungen, wie die Aufgabenstrukturierung in Arbeitspakete, die Zeit- und Aufwandsplanung und die Personalzuordnung. Da es zwischen den technischen und organisatorischen Entscheidungen Wechselwirkungen und Abhängigkeiten gibt, sollten beide Seiten eng miteinander verzahnt sein. In der Realität zeichnet sich der Informationsaustausch zwischen Architekten und Projektverwaltern als komplex und von Projekt zu Projekt unterschiedlich ab. Die ganzheitliche Betrachtung von technischen und organisatorischen Informationen ist wichtig, um fundierte Entscheidungen treffen zu können. Hierbei verlässt man sich heutzutage hauptsächlich auf die Erfahrung und Kommunikationsfähigkeiten der Entscheidungsträger. Eine systematische Unterstützung durch Methoden und Werkzeuge ist wünschenswert.

Eine konkrete Problemstellung stellt sich im Rahmen der Software-Evolution, wenn das System an neue Rahmenbedingungen, Anforderungen und technische Weiterentwicklungen angepasst werden muss. In dieser Phase werden die Projektverantwortlichen mit Änderungsanfragen konfrontiert und haben die Aufgabe zu entscheiden, wie diese im System umgesetzt werden. Das kann bedeuten, dass die Umsetzung mehrerer Änderungsanfragen koordiniert werden muss oder auch, dass verschiedene alternative Umsetzungsmöglichkeiten einer einzigen Änderungsanfrage gegeneinander abgewogen werden müssen. Gewählte Umsetzungswege wirken sich unterschiedlich stark auf Software-Artefakte, Tätigkeitsbereiche und Lebenszyklusphasen des Software-Systems aus. Die Kontextinformationen, welche zur Abschätzung dieser Auswirkungen benötigt werden sind in der Regel verstreut und müssen mühsam zusammen getragen werden.

Wir sehen die Lösung zur Entscheidungsunterstützung im Bereich der Architekturmodellierung, der zentralen Zusammenführung von entscheidungsre-

levanten Informationen im Architekturmodell und einer darauf aufbauenden szenarienbasierten Architekturanalyse. Deshalb stellen wir in dieser Arbeit ein Analyseverfahren zur Bewertung und Planung von Änderungsanfragen mit Architekturmodellen vor. Ein wesentlicher Beitrag hierbei ist, dass Änderungsschritte im Architekturmodell geplant und direkt in entsprechende Aufgaben für die operative Projektplanung überführt werden können. Dadurch werden Architekturmodellierung und Projektplanung miteinander verbunden. Dabei werden verschiedene Arten von Tätigkeiten, bei denen bei der Umsetzung der Änderungsanfrage Aufwände anfallen, explizit bei der Arbeitsplanableitung berücksichtigt. Hierzu zählen beispielsweise das Programmieren im engeren Sinne, die Konfiguration von eingebundenen Komponenten in Metadaten, das Bauen, das Testen, die Bereitstellung und die Inbetriebnahme. Tätigkeiten in diesen Bereichen werden aufgrund von Folgerungsregeln aus Architekturmodellmodifikationen und Modellannotationen automatisiert abgeleitet. Das Ergebnis des Verfahrens sind Aktivitätslisten mit den jeweils betroffenen Artefakten, welche zur Umsetzung bearbeitet werden müssen.

Der Beitrag dieser Arbeit ist ein Verfahren zur Änderungsanfragenanalyse im Architekturmodell. Dabei erfolgt die **Modellierung der Ausgangs- und Zielarchitektur** mittels eines werkzeug-gestützten, meta-modellierten Architekturmodells. Im Zielarchitekturmodell werden durch den Architekten die für die Änderungsanfrage notwendigen **strukturellen Architekturmodifikationen** vorgenommen und mittels Annotationen der Bedarf **interner Änderungen** an den jeweiligen Architekturartefakten kenntlich gemacht. In das Verfahren ist eine architektur-basierte **Änderungsausbreitungsanalyse** integriert, mit deren Hilfe der Architekt die Änderungsausbreitung zwischen verschiedenen Architekturartefakten, d.h. Komponenten, Datentypen oder Schnittstellen explizit berücksichtigen kann. Zur Bestimmung des resultierenden Arbeitsplans wird die **Differenz zwischen Ausgangs- und Zielarchitekturmodell berechnet** und die Unterschiede werden mit Hilfe von **Interpretations- und Folgerungsregeln** in Arbeitsplan-Aktivitäten überführt. Die Ableitung von **Tätigkeiten in nachgelagerten Tätigkeitsfeldern** (Testen, Inbetriebnahme, etc.) ist ein zentraler Beitrag, der das Verfahren von verwandten Analyseverfahren abgrenzt. Durch die Betrachtung unterschiedlicher Tätigkeitsfelder ist es darüber hinaus möglich, unterschiedliche Lebenszyklusphasen zu analysieren.

Das in dieser Arbeit vorgestellte Verfahren wurde erfolgreich in einer empirischen Studie validiert. Dazu wurde das Verfahren als Erweiterung eines

bestehenden Architekturmodellierungswerkzeugs implementiert. Die automatisierte Ableitung von Aktivitätslisten verbessert die Skalierbarkeit der Analyse. Wie die Studie zeigt, weisen die resultierenden Aktivitätslisten im Vergleich zur manuellen Herleitung eine gleiche und bessere Qualität auf. Der Studie zur Folge liefert das automatisierte Verfahren in Bezug auf Vollständigkeit und Präzision mindestens gleich gute Aktivitätslisten wie die manuelle Aktivitätsherleitung. Außerdem sind resultierende Aktivitätslisten aus dem automatischen Verfahren (von verschiedenen Anwendern) für eine Änderungsanfrage ähnlicher, d.h. das Ergebnis ist unabhängiger vom Anwender des Verfahrens. Bei den manuell bestimmten Aktivitätslisten wurden oftmals Informationen nicht angegeben. Zudem variierten die manuellen Lösungen sehr von Anwender zu Anwender.

Danksagung

Viele Menschen haben mich in den letzten Jahren auf dem Weg zur Dissertation begleitet, unterstützt und ermutigt. Ihnen möchte ich an dieser Stelle danken.

Zu aller erst danke ich meiner lieben Frau Barbara. Sie hat mit mir viel mitgemacht und mich bis zum Ende angetrieben und unterstützt. Durch ihre Entlastung konnte ich die Dissertation neben meiner beruflichen Tätigkeit zu einem erfolgreichen Abschluss führen.

Mein ganz besonderer Dank gilt meinem Doktorvater Prof. Dr. Ralf Reussner, der mir diese Arbeit ermöglicht hat und mich bis zum Ende immer wieder unterstützt und mir wertvolle Ratschläge gegeben hat. Durch ihn habe ich sehr viel gelernt und die Arbeit in seiner Forschungsgruppe ist einfach einzigartig und von einer sehr liebenswerten Atmosphäre geprägt.

Ich danke Prof. Dr. Andreas Oberweis dafür, dass er das Korreferat übernommen hat. Ich danke Mircea Trifu, der eine Zeit lang mein inhaltlicher Betreuer war und mir viele inhaltliche und methodische Anregungen gegeben hat.

Weiterhin danke ich dem Forschungsbereich SE (Software Engineering) am FZI Forschungszentrum Informatik, Karlsruhe und allen Kolleginnen und Kollegen, darunter Christian Bartsch, Steffen Becker, Franz Brosch, Zoya Durdik, Thomas Goldschmidt, Henning Groenda, Jens Happe, Michael Hauck, Jan Kofron, Klaus Krogmann, Volker Kuttruff, Martin Küster, Marco Mevius, Christof Momm, Pierre Parrend, Christoph Rathfelder, Thomas Schuster, Gabor Szeder, Peter Szulmann, Adrian Trifu, Mircea Trifu und Jan Wiesenberger.

Darüber hinaus danke ich dem Lehrstuhl Software Design and Quality am KIT und allen Kolleginnen und Kollegen, darunter Steffen Becker, Fabian Brosig, Erik Burger, Viktoria Firus, Jens Happe, Jörg Henß, Matthias Huber, Nikolaus Huber, Elena Kienhöfer, Samuel Kounev, Heiko Koziolek, Rouven Krebs,

Klaus Krogmann, Steffen Kruse, Michael Kuperberg, Anne Martens/Koziolek, Lucia Kapová/Happe, Philipp Merkle, Qais Noorshams, Fouad Omri, Andreas Rentschler, Tatiana Rhode, Vanessa Martin Rodríguez, Dennis Westermann.

Ich danke den Studentinnen und Studenten, die mich bei der Konzeption und der Implementierung meines Dissertationsansatzes unterstützt haben, darunter Nicole Nöldner, Thomas Heilbronner, Oliver Burkhardt und ganz besonders Thomas Knapp.

Ich danke den vielen Personen, die mir Ratschläge und inhaltliche Anstöße gegeben haben, unter anderem Dr. Matthias Naab und Dr. Ralf Carbon (Fraunhofer IESE, Kaiserslautern), Dr. Roland Weiß (ABB), Astrid Kreissig (IBM), Dr. Wolfgang Weck, Dr. Carola Lilienthal und Prof. Dr. Walter Tichy.

Ich danke meinem derzeitigen Arbeitgeber andrena objects ag, Karlsruhe bei dem ich nach der Forschungsarbeit eine sehr gute berufliche Heimat in der Software-Entwicklung gefunden habe.

Und natürlich danke ich meinen lieben Eltern Margarete und Richard Stammel. Sie erst haben mein Leben so ermöglicht, wie es sich entwickelt hat. Vielen herzlichen Dank!

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Diese Dissertation stellt ein Verfahren vor, mit dem Projektverwalter und Software-Architekten im Rahmen der Software-Entwicklung auf der Basis von angereicherten Architekturmodellen, Änderungsanfragen bewerten und planen können. Dieses Kapitel vermittelt die Motive für die Entwicklung dieses Verfahrens und führt den Ansatz und die damit verbundenen wissenschaftlichen Beiträge ein. Das Kapitel beginnt damit, das Verhältnis von Software-Architektur und Projektverwaltung zu beleuchten und den Unterstützungsbedarf für Software-Architekten und Projektverwaltern zu motivieren (Abschnitt 1.1). Abschnitt 1.2 stellt die Problemstellung dieser Dissertation vor und formuliert daraus die wissenschaftliche Fragestellung. Die Anforderungen an eine Lösung beschreibt Abschnitt 1.3. Es folgt eine kurze Betrachtung existierender Lösungen (in Abschnitt 1.4) im Vergleich zu gewünschten Eigenschaften, sowie ein Überblick über das vorgeschlagene Lösungsverfahren und die wissenschaftlichen Beiträge (in Abschnitt 1.5). Darüber hinaus gehen wir in Abschnitt 1.6 auf mögliche Anwendungsszenarien für das Lösungsverfahren ein und weisen in Abschnitt 1.7 auf die Validierung hin.

1.1. Kontext und Motivation

Die Software-Architektur hat in den vergangenen Jahren für die Software-Entwicklung an Bedeutung gewonnen. Sie hat sich zu einem wichtigen Instrument für die Entscheidungsfindung von Architekten und Projektverantwortlichen entwickelt und ergänzt zunehmend die klassische Projektplanung. Wir stützen uns bezüglich der Software-Architektur auf folgende Definition aus [KRH08].

Software-Architektur Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren

> Beziehungen zueinander und zur Umgebung sowie die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen.

Der Definition folgend sollte man die Architektur sowohl beim Entwurf von Neusystemen als auch bei der Evolution bestehender Systeme berücksichtigen. Die explizite Betrachtung der Architektur ist wichtig, weil diese langfristige Entscheidungen umfasst, die schwierig zu ändern sind. Darüber hinaus enthält eine Architekturdokumentation, falls sie existiert, in der Regel wichtige Zusatzinformationen, wie zum Beispiel Dienstgütevereinbarungen oder grobe Strukturierungsmerkmale, auch unter den Bezeichnungen „Stile“ oder „Muster“ bekannt. Diese sind, wenn überhaupt, in den anderen Systemartefakten oder in der Entwicklungsinfrastruktur nur implizit vorhanden.

Zusammengefasst umfasst die Software-Architektur die technische Organisation eines Software-Systems und die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen und ist damit für alle Lebensphasen des Software-Systems relevant, wie man an zahlreichen Ansätzen erkennen kann, welche die Modellierung und Analyse der Software-Architektur im Rahmen von Software-Entwurf, -Entwicklung und -Evolution unterstützen (u.a. [Reu+11], [BZJ04], [DN02], [Pau01], [CKK05]).

Neben der technischen Organisation durch die Software-Architektur auf der einen Seite gibt es auf der anderen Seite die Projektplanung, welche die organisatorischen Entscheidungen trifft, wie zum Beispiel die Aufgabenstrukturierung in Arbeitspakete und die Ressourcenplanung zu welcher beispielsweise die Aufwandsschätzung, die Zeitplanung und die Personalzuordnung zählen.

Allerdings sollte man bedenken, dass technische und organisatorische Belange eines Systems nicht unabhängig voneinander sind. Vielmehr bestehen zwischen den jeweiligen Entscheidungen Wechselwirkungen und Abhängigkeiten, so dass diese ohne eine ganzheitliche Betrachtungsweise nicht einfach getroffen werden sollten. Demnach sollten Software-Architektur-Entwurf und Projektplanung eng miteinander verzahnt werden.

In verschiedenen Dialogen mit Vertretern der Praxis haben wir gelernt, dass der Informationsaustausch zwischen Architekten und Projektverwaltern oftmals komplex und heterogen ist, das heißt je nach Projekt und Rolleninhaber unterschiedlich abläuft. Das Zusammenbringen von technischen und organisatorischen Entscheidungen bei täglichen Problemsituationen ist wichtig, wird

jedoch heutzutage hauptsächlich durch die Erfahrung und Kommunikationsfähigkeiten von Architekten und Projektverwaltern überwunden. Folglich bleibt eine systematische Unterstützung mit Verfahren und Werkzeugen wünschenswert.

In welchem Verhältnis stehen Architekturmodellierung und Projektplanung? Die Rolle der Architekturmodellierung im Zusammenhang mit der Projektplanung ist nicht eindeutig zu bestimmen. Ebenso sieht es mit dem Zusammenspiel von Projektverwaltern und Software-Architekten aus. Die Verteilung der Aufgaben und Verantwortlichkeiten ist nicht eindeutig geklärt und wird von Fall zu Fall unterschiedlich geregelt. Für die nachfolgend vorgestellte Problemstellung ist eine exakte Rollendefinition jedoch nicht notwendig, daher reicht uns hier ein intuitives Bild. Intuitiv ist der Software-Architekt dafür zuständig, die Grundlage für langfristige technische Entscheidungen zu schaffen und auf dieser Basis begründete Entscheidungen abzuleiten. Die Umsetzung dieser Entscheidungen erfolgt dann in Abstimmung mit dem Projektverwalter, der die zeitlichen, ökonomischen und arbeitsorganisatorischen Rahmenbedingungen einbeziehen muss.

1.2. Problemstellung

Im Spannungsfeld von Software-Architektur und Projektverwaltung betrachtet diese Dissertation einen konkreten Problemaspekt aus dem Umfeld der Software-Evolution. Der Begriff Software-Evolution beschreibt die Erkenntnis, dass Software-Systeme ständig an neue Rahmenbedingungen, Anforderungen und technische Weiterentwicklungen angepasst werden müssen, um einsatztauglich zu bleiben. Konkret bedeutet dies, dass die Projektverantwortlichen regelmäßig mit Änderungsanfragen konfrontiert werden und entscheiden müssen, wie diese im System umgesetzt werden. Dabei müssen sie die Umsetzung mehrerer Änderungsanfragen koordinieren oder zwischen verschiedenen alternativen Umsetzungsmöglichkeiten einer einzigen Änderungsanfrage abwägen. Abbildung 1.1 zeigt diese Ausgangssituation schematisch.

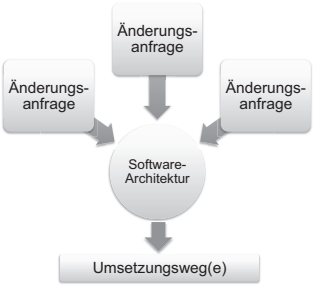


Abbildung 1.1.: Die Software-Architektur wird mit Änderungsanfragen konfrontiert. Die Projektverantwortlichen müssen jeweils Umsetzungswege bestimmen.

Eine Änderungsanfrage wird dann beispielsweise mit Hilfe eines Architekturmodells besprochen. Dabei müssen sie zwei Herausforderungen bewältigen. Erstens müssen sie ausgehend von der geplanten Änderung im Architekturmodell deren Auswirkungen auf das übrige System verstehen. Wir bezeichnen das als strukturelle Änderungsausbreitung. Zweitens müssen sie, sobald sie die Auswirkungen auf die Systemstruktur verstanden haben, die Auswirkungen auf die Arbeitsorganisation ermitteln und berücksichtigen.

In komplexen Entwicklungsprozessen und -organisationen gibt es viele Arbeitsbereiche, die direkt oder indirekt von einer Änderung betroffen sein können. Das sind beispielsweise Arbeitsbereiche wie die Programmierung, Metadatenbearbeitung, Testentwicklung, Testdurchführung, Baukonfiguration, Baudurchführung, Bereitstellung, Auslieferung, Inbetriebnahme oder Anwenderunterstützung (engl. Support).

Ein Teilproblem ist hierbei, dass die strukturelle Architekturbeschreibung so manche entscheidungsrelevante Information verbirgt. Nehmen wir als Beispiel die Situation in Abbildung 1.2 mit einer einfachen Applikation bestehend aus Client, Server und Datenbank.

Aufgrund langer Ladezeiten von der Datenbank entscheiden die Beteiligten, dass zwischen Server und Datenbank (engl. Database) ein Zwischenspeicher (engl. Cache) eingefügt werden soll. Was bedeutet diese Änderung? Welche Auswirkungen haben diese Änderungen? Das kann man nicht so einfach sagen, weil dazu entsprechende Kontextinformationen fehlen. Betrachten wir dagegen ein mit Kontextinformationen angereichertes Architekturmodell wie

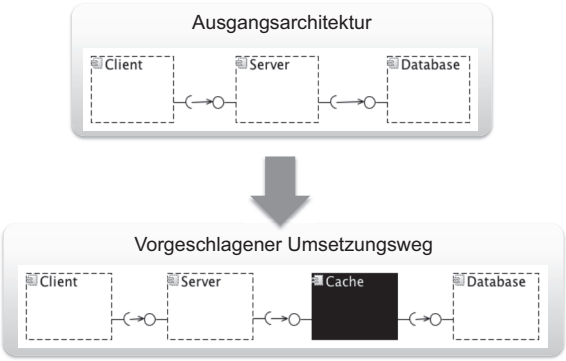


Abbildung 1.2.: Architekturmodelländerungen verbergen Folgeauswirkungen

in Abbildung 1.3, so wird schnell klar, welche Konsequenzen mit einer Modifikation des Systems verbunden sein können. Aus der Graphik geht hervor, dass mit den Komponenten zahlreiche Entwicklungsartefakte verbunden sind, an denen im Rahmen einer Änderung Hand angelegt werden müsste.

Das Problem, welches diese Dissertation löst, ist Software-Architekten und Projektverwaltern bei der Bewertung und Planung von Umsetzungswegen zu unterstützen, so dass sie die strukturelle Änderungsausbreitung und die Auswirkungen auf das Entwicklungs- und Betriebsumfeld des Software-Systems berücksichtigen können.

Wir sehen das Lösungspotenzial vor allem im Bereich der Automatisierung, d.h. werkzeug-gestützten Auswertung von Kontextinformationen im Architekturmodell. Das wird insbesondere verständlich, wenn man bedenkt, dass das System, welches in Abbildung 1.3 zu sehen ist, noch recht überschaubar ist. Bei größeren Systemen nimmt die Anzahl der relevanten Kontextinformationen noch erheblich zu, so dass eine Analyse ohne eine automatisierte Lösung sehr komplex und fehleranfällig wird.

Aus Problem und Lösungspotenzial ergibt sich die These für diese Dissertation in Tabelle 1.1.

Durch die Anreicherung des Architekturmodells mit technischen und organisatorischen Informationen *ist es möglich,* Umsetzungswege im Architekturmodell zu beschreiben und automatisiert in Aktivitäten zu überführen *und dadurch,* die Skalierbarkeit zu verbessern, gleichzeitig ein zur manuellen Herleitung vergleichbares oder sogar besseres Ergebnis in Bezug auf Vollständigkeit und Genauigkeit zu erzielen.

Tabelle 1.1.: These

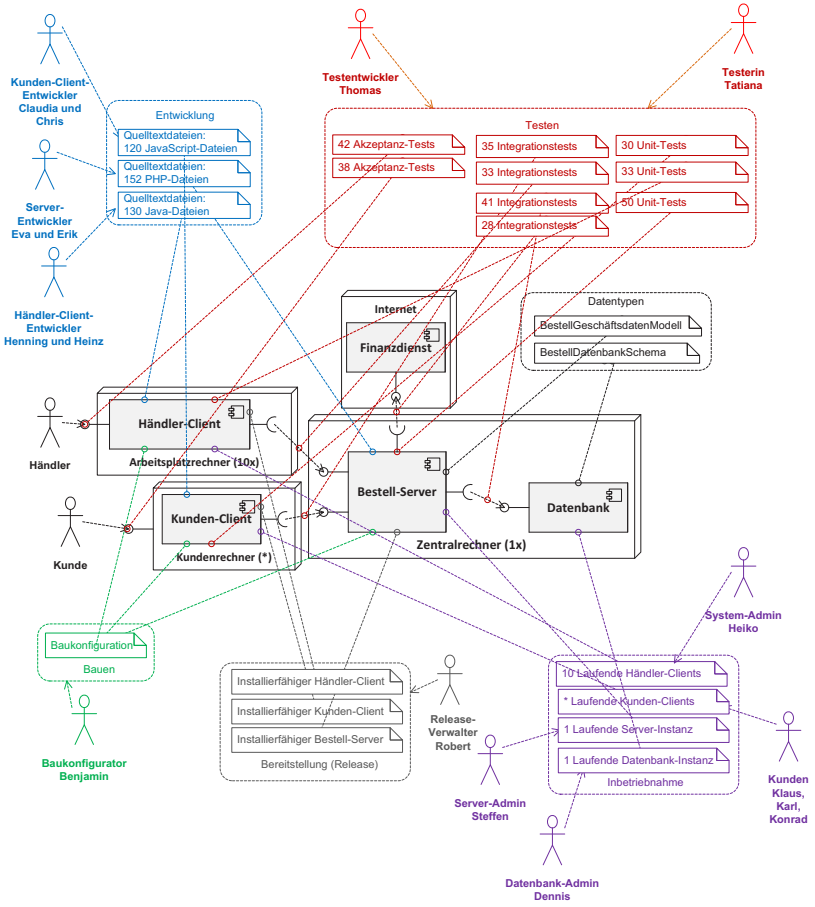


Abbildung 1.3.: Entwicklungsumfeld des Internet-Bestell-Systems

1.3. Anforderungen

Damit die Problemstellung zufriedenstellend behandelt werden kann sollte eine Lösung die folgenden Anforderungen erfüllen.

Verknüpfung von Architekturmodellierung und Projektverwaltung Das Verfahren sollte Architekturmodellierung und Projektverwaltung explizit miteinander verbinden.

Leichtgewichtigkeit Es ist ein leichtgewichtiger Ansatz gewünscht, der mittels Werkzeugunterstützung einfach in verschiedene Arten von Entwicklungsprozessen eingebunden werden kann und schnelle Rückkopplung ermöglicht.

Lebenszyklusabdeckung Es ist ein Verfahren gewünscht, welches den Lebenszyklus des Software-Systems ganzheitlich betrachtet und dabei in verschiedenen Phasen, d.h. Software-Entwurf, Software-Implementierung und Software-Evolution, eingesetzt werden kann.

Tätigkeitsfeldabdeckung Es ist ein Verfahren gefordert, welches die gängigen Tätigkeitsfelder der Software-Entwicklung berücksichtigt, welche in vielen Entwicklungsvorhaben vorkommen. Dazu zählen insbesondere die Tätigkeitsfelder Programmierung, Fremdkomponenten-Konfiguration, Bauen, Testen, Bereitstellung und Inbetriebnahme.

Szenarien-Basiertheit Das Verfahren sollte die Software-Evolution anhand konkreter Änderungsanfragen bewerten. Änderungsanfragen sind ein Spezialfall von Szenarien.

Berücksichtigung der strukturellen Änderungsausbreitung Das Verfahren sollte die strukturelle Änderungsausbreitung ausgehend von Modifikationen im System berücksichtigen.

Berücksichtigung der organisatorischen Auswirkungen Das Verfahren sollte die Auswirkungen von Modifikationen auf die Tätigkeitsfelder bestimmen können.

Skalierbarkeit durch Automatisierung Das Verfahren sollte die entscheidungsrelevanten Daten automatisiert verarbeiten können.

1.4. Existierende Lösungen

Es existieren bereits einige Lösungen, welche das genannte Problem teilweise lösen und einige gewünschte Eigenschaften erfüllen. Allerdings weisen diese Ansätze bestimmte Einschränkungen auf. Wir betrachten im Folgenden kurz bestehende Lösungen und deren Einschränkungen. Eine ausführliche Betrachtung findet sich in Kapitel 3.

Die Spiegel-Hypothese (Conways Gesetz) [Con68] motiviert die Zusammenarbeit von Software-Architekten und Projektverwaltern, steuert jedoch alleine keine aktive Lösung bei.

Die Ansätze, welche es im Bereich der architekturbasierten Projektplanung (Paulish [Pau01], Carbon [Car12]) gibt, betonen ausdrücklich die Notwendigkeit der Koordination von Software-Architektur und Projektplanung. Allerdings erfüllen die vorhandenen Verfahren nicht alle Anforderungen an eine zufrieden stellende Lösung oder besitzen einen anderen inhaltlichen Schwerpunkt.

Im Bereich der architekturbasierten Software-Evolution gibt es einige Verfahren, welche Unterstützung bieten, wie zum Beispiel von Garlan [Gar+09] und Naab [Naa12]. Allerdings berücksichtigen diese nicht die strukturellen und organisatorischen Auswirkungen, insbesondere auf konkrete Tätigkeitsfelder.

Das Feld der szenarienbasierten Architekturanalyseverfahren ist mittlerweile recht groß, u.a. SAAM, ATAM (beide [CKK05]), ALPSM [BB99], ALMA [Ben+04]. Jedoch konnte kein Verfahren identifiziert werden, welches eine automatisierte Herleitung von Aktivitäten aus dem Architekturmodell vornimmt.

Zusammenfassend können wir keinen Ansatz identifizieren, der die von uns gewünschten Eigenschaften aufweist und die von uns gestellte These untersucht.

1.5. Lösungsvorschlag und Wissenschaftliche Beiträge

Wir sehen die Lösung zur Entscheidungsunterstützung im Bereich der Architekturmodellierung, der szenarienbasierten Architekturanalyse und der zentralen Zusammenführung von entscheidungsrelevanten Informationen im Architekturmodell. Deshalb wird in dieser Arbeit ein Analyseverfahren zur Bewertung und Planung von Änderungsanfragen auf Architekturmodellen vorgestellt. Ein wesentlicher Beitrag ist hierbei, dass Änderungsschritte im Architekturmodell geplant und direkt in entsprechende Aufgaben im Projektplan überführt werden können. Dadurch werden Architekturmodellierung und Projektplanung miteinander verbunden. Weiterhin werden verschiedene Arten von Entwicklungstätigkeiten, bei denen bei der Umsetzung der Änderungsanfrage Aufwände anfallen, explizit bei der Aktivitätsermittlung berücksichtigt. Hierzu zählen beispielsweise Programmierung, Metadatenbearbeitung, Bauen, Testen, Bereitstellung und Inbetriebnahme. Tätigkeiten in diesen Bereichen werden aufgrund von Folgerungsregeln aus Architekturmodellmodifikationen und Modellannotationen automatisiert abgeleitet.

Der wissenschaftliche Beitrag ist ein Verfahren zur Änderungsanfragenanalyse im Architekturmodell. Dabei erfolgt die Modellierung der Ausgangs- und Zielarchitektur in einem werkzeug-gestützten, meta-modellierten Architekturmodell. Im Zielarchitekturmodell nimmt der Anwender strukturelle Architekturmodifikationen vor und kennzeichnet den Bedarf interner Änderungen an den jeweiligen Architekturartefakten. In das Verfahren ist eine architektur-basierte Änderungsausbreitungsanalyse integriert, mit deren Hilfe der Architekt die Änderungsausbreitung innerhalb und zwischen Komponenten berücksichtigen kann. Zur Bestimmung des resultierenden Arbeitsplans wird die Differenz zwischen Ausgangs- und Zielarchitekturmodell berechnet. Aus der Differenz werden mit Hilfe von Interpretations- und Folgerungsregeln Arbeitsplan-Aktivitäten abgeleitet. Die Ableitung von Tätigkeiten in nachgelagerten Tätigkeitsfeldern (Testen, Inbetriebnahme, etc.) ist ein zentraler Beitrag, der das Verfahren von verwandten Analyseverfahren abgrenzt.

Folgende Fragen kann der Anwender mit Hilfe des Verfahrens beantworten:

- Was sind die strukturellen Auswirkungen eines geplanten Umsetzungswegs?

- Was sind die organisatorischen Auswirkungen eines geplanten Umsetzungswegs?
- Welche Aktivitäten sind zur Realisierung eines Umsetzungswegs, insbesondere in nachgelagerten Tätigkeitsfeldern und Lebenszyklusphasen erforderlich?
- Welche Artefakte (Dateien, Konfigurationen, etc.) müssen für die Umsetzung bearbeitet werden?

1.6. Anwendungsszenarien

Das Verfahren liefert als Ergebnis Aktivitätslisten, welche die Umsetzungsschritte für Änderungsanfragen auflisten. Diese Aktivitätslisten können für verschiedene Zwecke verwendet werden. Unter anderem können die folgenden Anwendungsszenarien durch das Verfahren unterstützt werden.

Bewertung einzelner Umsetzungswege: Einzelne Umsetzungswege können unabhängig von anderen bewertet werden.

Abwägung alternativer Umsetzungswege: Es können zwei Aktivitätslisten miteinander verglichen werden, die alternative Lösungen darstellen.

Kombination ähnlicher Umsetzungswege: Ähnliche oder überlappende Umsetzungswege können kombiniert werden.

Erkennung von Umsetzungskonflikten: Bestimmte Arten von Konflikten zwischen Umsetzungswegen können durch den Vergleich von Aktivitätslisten erkannt werden.

Unterstützung der Aufwands- und Kostenschätzung: Aktivitätslisten können direkt zur Kostenschätzung, beispielsweise durch aufsteigende Schätzung (engl. bottom-up) [Jør04], verwendet werden oder als Eingabe in andere gängige Kostenschätzmethodiken (z.B. COCOMO II [Boe+00]) dienen.

Untersuchung von Auswirkungen der Evolution von Drittanbieter-Komponenten: Mit dem Verfahren kann man analysieren, welche Auswirkungen die Evolution von eingesetzten Drittkomponenten (Bibliotheken) auf die Architektur oder allgemein auf das Software-System hat.

Flexibilitätsanalyse und -ausnutzung: Mit dem Verfahren kann man analysieren, inwiefern bereitgestellte Flexibilität, beispielsweise in Form von architekturellen Strukturen, von Änderungsanfragen ausgenutzt werden kann und wo die Flexibilität unzureichend ist. Darüber hinaus kann das Verfahren bei der Planung der Flexibilitätsausnutzung behilflich sein (siehe [NS12])

1.7. Umsetzung und Validierung

Das Verfahren wurde implementiert und in ein existierendes Architekturmodellierungswerkzeug integriert.

Hauptvorteil des automatisierten Verfahrens ist die Skalierbarkeit. Demgegenüber müssen wir in der Validierung sicherstellen, dass die automatisierte Analyse eine ausreichende Ergebnisqualität liefert.

Im Rahmen der Validierung wurde mit Hilfe des Werkzeuges eine empirische Studie durchgeführt. Die Studie hatte das Ziel, die Ergebnisqualität des automatisierten Verfahrens mit der Ergebnisqualität von manuellen Analysen zu vergleichen. Es sollte untersucht werden, wie vollständig und präzise automatisiert ermittelte Aktivitätslisten im Vergleich zu manuell ermittelten Aktivitätslisten sind.

1.8. Überblick

Diese Arbeit ist wie folgt organisiert.

In Kapitel 2 präsentieren wir die Grundlagen. Wir geben eine Einführung in die komponenten-basierte Software-Entwicklung. Wir stellen die komponenten-basierte Software-Architektur vor, insb. das PCM-Architekturmetamodell,

welches wir für die Umsetzung des Verfahrens verwenden. Ein weiterer Abschnitt erläutert die Tätigkeitsfelder der Software-Entwicklung, für welche das Verfahren Aktivitäten ermittelt. Des weiteren werden die Grundlagen der Metamodellierung und Graphformalisierung, sowie Modelltransformationen mit Triple-Graph-Grammatiken kurz aufbereitet um die Formalisierung des Verfahrens besser verständlich zu machen.

Das Kapitel 3 reflektiert verwandte Arbeiten und grenzt diese gegenüber dem eigenen Ansatz ab. Die verwandten Arbeiten werden dabei in vier Themenblöcke untergliedert. Für jeden Themenblock gibt es einen eigenen Unterabschnitt. Der erste Abschnitt behandelt architekturbasierte Projektplanungsansätze. Im zweiten Block gehen wir auf Verfahren zur architekturbasierten Software-Evolution ein. Als dritten Themenblock betrachten wir szenarienbasierte Architekturbewertungsverfahren. Der letzte Teil stellt Verfahren zur Änderungsausbreitungsanalyse vor.

Die Kapitel 4 bis 6 sind die Kernkapitel dieser Arbeit und stellen die Hauptbeiträge inklusive Validierung vor. In Kapitel 4 stellen wir das Verfahren zu Änderungsanfragenanalyse generell und anhand eines durchgängigen Beispiels vor. In Kapitel 5 beschreiben wir das Verfahren formeller auf der Basis von Metamodellen und Graphen, sowie Modelltransformationen und Graphoperationen.

In Kapitel 6 evaluieren wir die Beiträge dieser Arbeit auf der Grundlage einer empirischen Studie.

Das Kapitel 7 fasst die Beiträge der Arbeit zusammen, reflektiert die Vorteile, Grenzen und Annahmen des Verfahrens und schließt mit einem Ausblick auf zukünftige Forschungsmöglichkeiten.

2. Grundlagen

In diesem Kapitel führen wir die Grundlagen ein und erläutern einige Begriffe, die in den weiteren Kapiteln verwendet werden. Zu den behandelten Themen zählen die komponentenbasierte Software-Entwicklung (in Abschnitt 2.1), die komponentenbasierte Architekturmodellierung mit besonderer Konzentration auf das Palladio Component Model (PCM) als verwendetes Metamodell für die Architekturmodellierung (in Abschnitt 2.2), die Tätigkeitsfelder in der Software-Entwicklung (in Abschnitt 2.3), sowie die Metamodellierung und Modelltransformationen für die formale Beschreibung des Verfahrens (in Abschnitt 2.4.1).

2.1. Komponentenbasierte Software-Entwicklung

Im Jahre 1968 wurde auf der NATO-Konferenz für Software-Ingenieurwesen der Begriff Software-Komponente (engl. software component) geprägt [McI68]. Damals kam der Gedanke auf, Software-Systeme aus wiederverwendbaren Einheiten aufzubauen, den Software-Komponenten. Das führte zu dem Software-Entwicklungs-Paradigma, das wir heutzutage unter der Bezeichnung komponentenbasiertes Software-Ingenieurwesen (engl. Component-Based
Software Engineering, [Szy02], [HC01]) kennen. Neben theoretischen Erkenntnissen resultierten aus diesem Paradigma zahlreiche technische Implementierungen und Basistechnologien, wie beispielsweise Microsoft COM [Mic15], Enterprise Java Beans [Ora15], OSGi [OSG15] oder das Corba Component Model der OMG [Obj06].

In der Literatur gibt es viele Definitionen für Software-Komponenten. Wir verwenden in dieser Arbeit die Definition von Clemens Szyperski [Szy02].

Software-Komponente Eine Software-Komponente ist eine Kompositionseinheit mit vertraglich spezifizierten Schnittstellen bei der alle Abhängigkeiten explizit sind. Eine Software-Komponente kann unabhängig in Betrieb genommen werden und kann durch Dritte mit anderen Komponenten verbunden werden.

Software-Komponenten werden insbesondere im Zusammenhang mit der komponenten-basierten Architekturmodellierung verwendet, welche wir im nächsten Abschnitt erklären.

2.2. Komponentenbasierte Architekturmodellierung

Ebenso wie für den Begriff „Software-Komponente“ gibt es auch für den Begriff „Software-Architektur“ zahlreiche Definitionen in der Literatur. Der ISO/IEC/IEEE-Standard 42010 [ISO11] definiert die Architektur eines Software-Systems als „Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution“. Wir halten uns in dieser Arbeit an die daraus abgeleitete deutsche Definition, welche im Handbuch der Software-Architektur [KRH08] zu finden ist.

Software-Architektur Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung sowie die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen.

Um die Architektur in den Entwicklungsprozess einzubeziehen bedient man sich im Allgemeinen der Architekturmodellierung und -dokumentation. Architekturmodelle beschreiben Software-Systeme aus mehreren Perspektiven. Die statische Perspektive eines komponentenbasierten Architekturmodells erfasst beispielsweise die strukturelle Aufteilung des Systems in Komponenten und zeigt, wie die Komponenten zum Gesamtsystem zusammengesetzt werden. Neben der strukturellen Perspektive betrachten Architekturmodellierungsansätze in der Regel weitere Perspektiven, etwa zur Inbetriebnahme (engl. Deployment) der Komponenten auf Laufzeitumgebungsknoten oder zur Nutzerinteraktion mit dem System.

Ein Architekturmodell ist die Grundlage für die konkrete Planung und Analyse der Software-Architektur. Das Architekturmodell ist in der Regel durch ein entsprechendes Metamodell spezifiziert.

Unser Ansatz verwendet ein Architekturmodell. Grundsätzlich ist unser Ansatz nicht auf die Verwendung eines spezifischen Architekturmodells beschränkt, allerdings sollten die folgenden Anforderungen erfüllt sein.

Verarbeitbarkeit durch Werkzeug: Da wir eine automatisierte Analyseunterstützung bereitstellen, sollten die Architekturmodelle mit Werkzeugen verwaltet werden können und analytisch verarbeitbar sein.

Komponenten-Basierung: Das Architekturmodell sollte eine komponentenbasierte Dekomposition der Systemstruktur vornehmen. Das bedeutet, das Modell sollte Komponenten beschreiben und wie diese zusammengesetzt werden um das Gesamtsystem zu bilden. Die Komponenten sollten mit angebotenen und nachgefragten Schnittstellen spezifiziert werden, die konform zu Schnittstellentypen sind. Ein Systemmodell sollte die Konnektoren zwischen den nachgefragten und angebotenen Schnittstellen assemblierter Komponenten spezifizieren und damit die statischen Abhängigkeiten zwischen den Komponenten ausdrücken.

Versions-Verwaltung: Technisch betrachtet erfordert unser Analyseverfahren die Fähigkeit, mehrere Versionen eines Architekturmodells zu halten und zu verarbeiten und Modifikationen im Architekturmodell zu erfassen und nachzuvollziehen.

Die weiteren Kapitel verwenden als konkretes Architekturmetamodell das Palladio Component Model (PCM) ([Reu+11], [BKR09]). Die nachfolgenden Abschnitte beschreiben die Teile des PCMs, welche für das Verfahren von Relevanz sind. Wir haben die relevanten Teile auf Deutsch übersetzt. Eine ausführliche Darstellung des PCM findet sich in [Reu+11].

Die Entwicklung des PCM begann im Jahr 2003 an der Universität Oldenburg. Seit 2006 wurde es am Karlsruher Institut für Technologie (KIT) und am Forschungszentrum Informatik (FZI) weiterentwickelt. Das PCM umfasst neben dem eigentlichen Metamodell eine integrierte Modellierungs- und Vorhersageumgebung, die PCM-Bench, welche auf Basis der Eclipse-Technologie unter Verwendung von EMF (Eclipse Modeling Framework) und GMF (Graphical Modeling Framework) entwickelt wurde. Zur Erstellung von Architekturmodellen stellt die PCM-Bench graphische Editoren bereit.

Die konkrete Syntax der graphischen Editoren des PCM lehnt sich an die UML-Syntax für Komponentenmodelle an.

Im folgenden betrachten wir die relevanten Bestandteile des PCM-Metamodells.

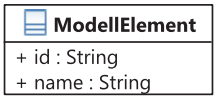


Abbildung 2.1.: Klassendiagramm mit der zentralen Oberklasse „ModellElement“

Alle Metaklassen des Architekturmetamodells sind von der Metaklasse `ModellElement` abgeleitet. Die Klasse bildet damit eine Wurzel in der Vererbungshierarchie. Wie das Klassendiagramm in Abbildung 2.1 zeigt, besitzt die Klasse ein Identifikator-Attribut (`id`) und ein Namen-Attribut (`name`). Bei der Erzeugung eines Modellelements wird sichergestellt, dass der Wert dessen Identifikator-Attributs (`id`) universell eindeutig ist. Jedes Modellelement besitzt demnach einen universell und global eindeutigen Identifikator. Zur technischen Realisierung dieses Identifikators wird das Konzept des UUID-Standards (engl. Universally Unique Identifier) [LMS05] verwendet.

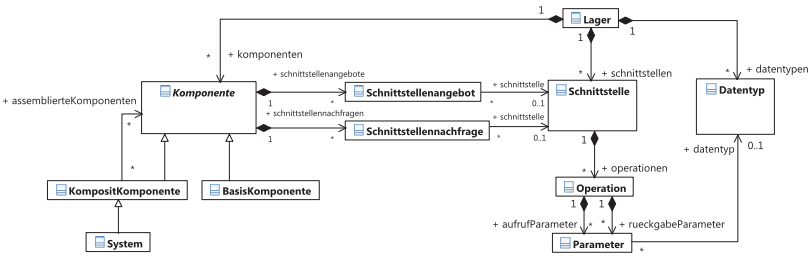


Abbildung 2.2.: Klassendiagramm zum Architekturmetamodell-Kern

Das Klassendiagramm in Abbildung 2.2 zeigt den Kern des Architektur-Metamodells. Wie man sieht, stellt das `Lager` (Metaklasse: `Lager`) die Wurzel des Architekturmodells dar. Im `Lager` werden Komponenten (Metaklasse: `Komponente`), Schnittstellen (Metaklasse: `Schnittstelle`) und Datentypen

(Metaklasse: `Datentyp`) spezifiziert. Eine Komponente bietet Schnittstellen an (Metaklasse: `Schnittstellenangebot`) oder frägt diese nach (Metaklasse: `Schnittstellennachfrage`). Für Schnittstellen (Metaklasse: `Schnittstelle`) werden Operationen (Metaklasse: `Operation`) definiert. Für Operationen können Aufrufparameter oder Rückgabeparameter (Metaklasse: `Parameter`) definiert werden.

Die Metaklasse `Komponente` ist abstrakt modelliert. Im Metamodell werden drei konkrete Typen von Komponenten unterschieden. Basiskomponenten (Metaklasse: `BasisKomponente`), die im Modell nicht weiter untergliedert werden, Komposit-Komponenten (Metaklasse: `KompositKomponente`), die aus anderen Komponenten (Basiskomponenten oder Komposit-Komponenten) zusammengesetzt werden und das System (Metaklasse: `System`), welches das Gesamtsystem repräsentiert und im Modell als eine spezielle Komposit-Komponente angesehen wird.

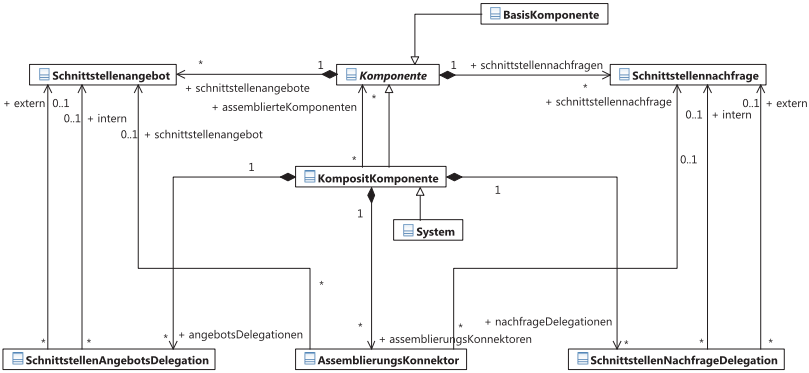


Abbildung 2.3.: Klassendiagramm zur Metamodellierung von horizontaler und vertikaler Komposition

Das Klassendiagramm in Abbildung 2.3 zeigt die Modellierung der horizontalen und vertikalen Komposition von Komponenten. Die *horizontale Komposition* zweier Komponenten erfolgt durch einen Assemblierungskonnektor (Metaklasse: `AssemblierungsKonnektor`), der zwischen der Schnittstellennachfrage der einen Komponente und dem Schnittstellenangebot der anderen Komponente spezifiziert wird. Die *vertikale Komposition* (d.h. Verschachtelung oder Hierarchiebildung) wird dagegen durch Komposit-Komponenten

modelliert. Dabei stellen Delegationskonnektoren die Verbindung von internen und externen Schnittstellenangeboten bzw. -nachfragen her. Eine Schnittstellenangebots-Delegation (Metaklasse: `SchnittstellenAngebotsDelegation`) beschreibt die Beziehung zwischen dem Schnittstellenangebot einer Komposit-Komponente (extern) und dem entsprechenden Schnittstellenangebot einer darin enthaltenen Komponente (intern). Analog beschreibt eine Schnittstellennachfrage-Delegation (Metaklasse: `SchnittstellenNachfrageDelegation`) die Beziehung zwischen sich entsprechenden Schnittstellennachfragen einer Komposit-Komponente und einer darin enthaltenen Komponente.

Im Lager werden neben Komponenten und Schnittstellen auch Datentypen spezifiziert. Das Klassendiagramm in Abbildung 2.4 zeigt die Metaklassen für Datentypen. Das Modell unterscheidet primitive Datentypen (Metaklasse: `PrimitivDatentyp`), Kollektions-Datentypen (Metaklasse: `KollektionsDatentyp`) und Komposit-Datentypen (Metaklasse: `KompositDatentyp`). Primitive Datentypen werden durch eine fest vorgegebene Liste definiert, welche unter anderem Typen wie „int“, „string“ oder „bool“ enthält. Kollektions-Datentypen bestehen aus einer Menge von Datenelementen, welche jeweils konform zu einem (gemeinsamen) Datentyp sind. Komposit-Datentypen enthalten eine Liste mit Datentyp-Parametern (Metaklasse: `DatentypParameter`), welche jeweils zu einem eigenen Datentyp konform sind. Eine Instanz eines Komposit-Datentyps entspricht einem Tupel von Datenwerten, deren Bezeichnungen und Datentypen durch die Datentyp-Parameter spezifiziert werden.

Um einen einheitlichen deutschen Sprachstil zu erreichen, haben wir die PCM-Metaklassen mit deutschen Bezeichnungen versehen. Außerdem haben wir, um die Modelle übersichtlich darzustellen, einige PCM-Metaklassen zu einer einzigen Metaklasse zusammengefasst. Die Tabellen 2.1 und 2.2 beschreiben die Abbildung zwischen den englischen und deutschen Repräsentationen der Metaklassen und stellt Zusammenfassungen dar.

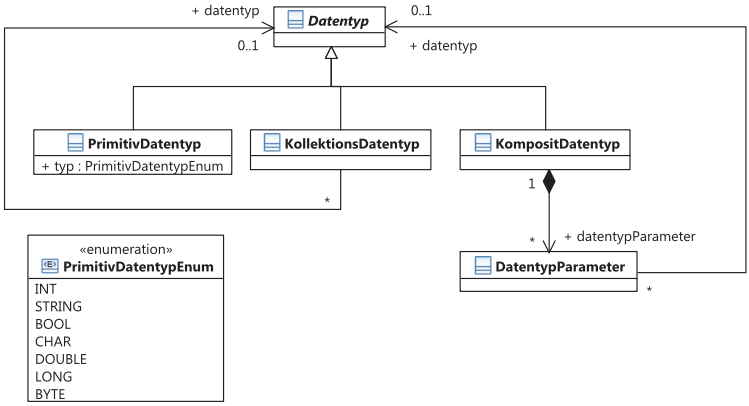


Abbildung 2.4.: Metamodellierung von Datentypen

PCM-Metaklasse(n)	**Entsprechung im deutschen Modell**
Identifier, NamedEntity, Entity	Zusammengefasst in Metaklasse „ModellElement“
Repository	Metaklasse „Lager“
InterfaceProvidingEntity, InterfaceRequiringEntity, InterfaceProvidingRequiring-Entity, RepositoryComponent	Zusammengefasst in Metaklasse „Komponente“
BasicComponent	Metaklasse „BasisKomponente“
CompositeComponent, AssemblyContext, ComposedStructure, ComposedProvidingRequiring-Entity	Zusammengefasst in Metaklasse „KompositKomponente“
System	Metaklasse „System“
ProvidedRole, OperationProvidedRole	Zusammengefasst in Metaklasse „Schnittstellenangebot“
RequiredRole, OperationRequiredRole	Zusammengefasst in Metaklasse „Schnittstellennachfrage“

Tabelle 2.1.: Abbildungstabelle zwischen englischen und deutschen Repräsentationen der PCM-Metaklassen – Teil 1

PCM-Metaklasse(n)	**Entsprechung im deutschen Modell**
AssemblyConnector	Metaklasse „AssemblierungsKonnektor“
ProvidedDelegationConnector	Metaklasse „SchnittstellenAngebotsDelegation“
RequiredDelegationConnector	Metaklasse „SchnittstellenNachfrageDelegation“
Interface, OperationInterface	Zusammengefasst in Metaklasse „Schnittstelle“
Signature	Metaklasse „Operation“
Parameter	Metaklasse „Parameter“ (bei Operationen)
DataType	Metaklasse „Datentyp“
PrimitiveDataType	Metaklasse „PrimitivDatentyp“
CompositeDataType	Metaklasse „KompositDatentyp“
CollectionDataType	Metaklasse „KollektionsDatentyp“
InnerDeclaration	Metaklasse „DatentypParameter“ (bei Kollektions-Datentypen)

Tabelle 2.2.: Abbildungstabelle zwischen englischen und deutschen Repräsentationen der PCM-Metaklassen – Teil 2

2.3. Tätigkeitsfelder in der Software-Entwicklung

Mit Hilfe des Verfahrens, das wir in dieser Arbeit vorstellen kann der Anwender Aktivitäten in verschiedenen Tätigkeitsfeldern der Software-Entwicklung ermitteln. Dieser Abschnitt stellt die berücksichtigten Tätigkeitsfelder vor und zeigt wie diese mit den Entwicklungs- und Planungsprozessen in Zusammenhang stehen.

Bei der Software-Entwicklung werden Prozesse verwendet. Die Prozesse werden in der Wissenschaft mit Prozessmodellen beschrieben. Ein Prozessmodell beschreibt die Phasen und die dazugehörigen Tätigkeiten.

Ein Prozess besteht aus Phasen, die sequenziell oder iterativ durchlaufen werden. Eine iterative Abarbeitung bedeutet, dass bestimmte Phasensequenzen mehrfach durchschritten werden.

Eine Phase lässt sich mit verschiedenen Merkmalen beschreiben. Zunächst sind mit einer Phase verschiedene konkrete Tätigkeiten verbunden. Die Tätigkeiten werden auf bestimmten Artefakten ausgeübt. Dies geschieht durch Personen, die phasenspezifische Rollen ausüben und dabei durch eine entsprechende technische Infrastruktur unterstützt werden.

Bekannte Prozessmodelle besitzen unterschiedliche inhaltliche Schwerpunkte. Viele Prozessmodelle orientieren sich an den Lebensphasen eines Software-Systems und decken diese ganz oder teilweise ab.

Entwicklungsprozesse beschreiben den Erzeugungsprozess eines neuen Software-Systems. Ein Beispiel für einen Entwicklungsprozess ist das *Wasserfallmodell* ([Som12], ab Seite 57). Im Wasserfallmodell gibt es die Phasen „Anforderungsanalyse und -definition“, „System- und Software-Entwurf“ „Implementierung und Unit-Tests“, „Integration und Systemtests“, „Betrieb und Wartung“. Die Phasen werden sequenziell bearbeitet, allerdings ist der Rücksprung in eine bereits abgeschlossene Phase erlaubt.

In der Literatur sind andere Prozess-Modelle bekannt, die unterschiedlich streng und formal definiert sind. Einige sind eher schwergewichtig und dokumentationslastig, andere versuchen Dokumentation zu vermeiden und möglichst ohne viel Vorabplanung und Entwurf an die Entwicklung zu gehen.

Im Umfeld der komponenten-basierten Software-Entwicklung werden spezielle Prozessmodelle vorgeschlagen, beispielsweise von Koziolek und Happe [KH06] oder von Daniels und Cheesman [CD01]. Darüber hinaus bestehen allgemeine architekturbasierte Entwicklungsprozesse wie zum Beispiel von J. Paulish [Pau01].

Das hier vorgestellte Verfahren ist unabhängig vom verwendeten Prozessmodell. Wichtig ist jedoch, dass im Rahmen der Software-Entwicklung, -Wartung oder -Evolution verschiedene Arten von Tätigkeiten anfallen. Diese Tätigkeiten können nach Tätigkeitsfeldern gegliedert werden. Wir konzentrieren aus auf übliche Tätigkeitsfelder, die im Rahmen der Entwicklung und Evolution anfallen.

In Abbildung 2.5 geben wir eine Übersicht über die betrachteten Tätigkeitsfelder. Demnach konzentrieren wir uns in der Dissertation auf die Phasen „Programmierung“, „Bauen“, „Testen“, „Bereitstellung“ und „Inbetriebnahme“. Die Tätigkeitsfelder wurden so gewählt, dass sie viele Phasen in bekannten Entwicklungsprozessmodellen repräsentieren.

2.3.1. Programmierung

Im Bereich Programmierung unterscheiden wir die Tätigkeiten „Quelltextbearbeitung“ und „Metadatenbearbeitung“.

Die Tätigkeit „Quelltextbearbeitung“ umfasst die Programmierung im engeren Sinne. Dazu gehören zum Beispiel das Schreiben von Klassen, Methoden und die Entwicklung objekt-orientierter Datenstrukturen und die Arbeit mit unterschiedlichsten Programmiersprachen und Entwicklungstechnologien.

Die Tätigkeit „Metadatenbearbeitung“ umfasst das Editieren von Software-Artefakten außerhalb des Quelltextes. Das ist beispielsweise die Modifikation von Konfigurationsdateien eingebundener Drittanbieter-Komponenten. Dazu gehören unter anderem Datenbank-Schema-Spezifikationen, die Konfigurationsdateien von Objektrelationalen Abbildungskomponenten, Server-Einstellungen oder Plugin-Manifeste, wie sie beispielsweise bei Eclipse-Plugins vorkommen. Viele eingesetzte Technologien erfordern, dass man Anpassungen an den Einstellungen vornimmt und sie entsprechend konfiguriert. Das umfasst die Bearbeitung von Dateien in unterschiedlichen Formaten, wie z.B. XML, JSON oder proprietäre Formate.

Abbildung 2.5.: Übersicht der Tätigkeitsfelder mit dazugehörigen Teilaktivitäten, Personenrollen, Infrastrukturen und Resultate

2.3.2. Bauen

Software-Systeme durchlaufen nach der Programmierung eine Bauphase, in der Quelltextdateien in ausführbare Artefakte übersetzt werden. Außerdem werden in dieser Phase üblicherweise Artefakte gebündelt. Wir unterscheiden zwischen den Tätigkeiten „Baukonfiguration“ und „Baudurchführung“.

In einigen Software-Projekten werden automatisierte Methodiken und Werkzeuge für den Bau eingesetzt. Das automatisierte Bauen ermöglicht eine frühzeitige und kontinuierliche Übersetzung und Integration von Entwicklungsartefakten, was auch unter der Bezeichnung „Kontinuierliche Integration“ (engl. continuous integration) bekannt ist.

Zu den konkreten Artefakten, welche im Bereich „Bauen“ bearbeitet werden gehören beispielsweise Ant-Buildskripte, Makefiles (C/C++), Konfigurationseinstellungen von Bauautomatisierungslösungen wie Cruise Control [Cru15],

Jenkins [Jen15] oder TeamCity [Jet15]. Aus der Phase resultieren die gebauten Artefakte.

Obwohl die Baudurchführung oft automatisiert stattfindet, muss die Baukonfiguration doch in den meisten Fällen manuell vorgenommen werden. Die Baukonfiguration muss oft angepasst werden um Abhängigkeiten zu Bibliotheken, anderen Komponenten, oder externen Diensten zu spezifizieren.

Gängige Lösungen zur Steuerung des Baus, zur automatisierten Testausführung und zur Abhängigkeitsverwaltung wie Maven [The15] oder Gradle [Gra15] sind ebenfalls mit manuellen Editieraufgaben verbunden.

2.3.3. Testen

Ein wichtiges Tätigkeitsfeld zur Qualitätssicherung der Software ist das Testen. In diesem Tätigkeitsfeld unterscheiden wir die Tätigkeiten „Testentwicklung“, „Testaktualisierung“ und „Testdurchführung“. Die „Testentwicklung“ umfasst die Konzeption und das Schreiben neuer Testfälle. Die „Testaktualisierung“ beinhaltet die Überprüfung und Anpassung bestehender Testfälle. Bei der „Testdurchführung“ werden die Tests ausgeführt und die Testergebnisse bestimmt.

In der Software-Entwicklung werden verschiedene Arten von Tests eingesetzt. Wir unterscheiden in dieser Arbeit zwischen *Unit-Tests*, mit denen einzelne Einheiten isoliert getestet werden, *Integrationstests*, mit den das Zusammenspiel mehrerer Einheiten getestet wird und *Akzeptanztests*, mit denen Abnahmekriterien der vereinbarten Anforderungen an der Benutzerschnittstelle des Systems getestet werden. Dabei kann es sein, dass auf die Benutzerschnittstellen
menschliche Nutzer zugreifen oder aber, dass technische Nutzer, d.h. andere Systeme, darauf zugreifen.

Die Testentwicklung und Testaktualisierung sind in der Regel manuelle Tätigkeiten, während für die Testdurchführung insbesondere von Unit-Tests und Integrationstests automatisierte Verfahren üblich sind, wie beispielsweise mit JUnit [JUn15] im Java-Umfeld oder NUnit [NUn15] im .NET-Umfeld. Dennoch ist das Testen bei einer großen Anzahl von Testfällen oder sehr umfangreichen Testfällen mit zeitlichen Aufwand verbunden.

2.3.4. Bereitstellung

Ein weiterer Tätigkeitsbereich ist die „Bereitstellung“ (engl. Release). Das beinhaltet die Bündelung der Artefakte eines Software-Systems zu einem auslieferbaren und installationsfähigen Paket. Ähnlich wie beim Bauen unterscheiden wir hier zwischen „Bereitstellungskonfiguration“ und „Bereitstellungsdurchführung“.

Die technische Ausgestaltung der Bereitstellung ist je nach Technologie und Art der Anwendung unterschiedlich. Beispiele für die Bereitstellung sind die Bündelung von Eclipse-Plugins in Form von Update-Sites, die Bereitstellung von gebauten Artefakten und Bibliotheken in Maven [The15] oder Ivy-Repositories, oder das Einstellen von Anwendungen für Mobilgeräte in den Marktplatz.

2.3.5. Inbetriebnahme

Die bereitgestellte, installationsfähige Software wird schließlich installiert und in Betrieb genommen. Daher ist die „Inbetriebnahme“ ein weiterer Tätigkeitsbereich.

Wir unterscheiden die Aktivitätstypen „Inbetriebnahmekonfiguration“ und „Inbetriebnahmedurchführung“.

Zur Inbetriebnahmekonfiguration gehören die Installation der Software auf einer oder mehreren Laufzeitumgebungen und die Aktualisierung von Daten, welche zur Laufzeit benötigt werden, wie zum Beispiel Datenbank-Inhalte.

Zur Inbetriebnahme gehören auch Bestandteile, die das System-Setup unterstützen. Beispielsweise werden im Microsoft-Umfeld sogenannte MSI-Installer verwendet. Die Konfiguration solcher Module ist teilweise mit grossem manuellen Aufwand verbunden.

2.4. Formalisierung

2.4.1. Metamodellierung und Modelltransformationen

Zur Beschreibung und Implementierung von Analyse-Verfahren in der ingenieurmäßigen Software-Entwicklung kommen zunehmend modell-getriebene Technologien zum Einsatz. Es ist mittlerweile üblich, Daten mit denen die Verfahren arbeiten als Metamodelle zu beschreiben.

Ein OMG-Standard zur Erstellung einfacher Metamodelle ist Essential MOF (EMOF), einer abgespeckten Version der Meta Object Facility 2.0 (MOF 2.0). Eine zu EMOF kompatible Implementierung ist EMF Ecore.

Bei EMOF werden Entitäten der Anwendungsdomäne als Typ-Klassen in Metamodellen beschrieben und deren Eigenschaften in Form von Attributen definiert. Darüber hinaus können Abhängigkeiten zwischen den Entitäten modelliert werden.

Technologien für die Metamodellierung erlauben den Einsatz von Transformationen auf Modellen. Diese Transformationen können in Programmiersprachen implementiert werden.

2.4.2. Grundlagen zu Graphen

Graphen bestehen aus Knoten und Kanten. Eine übliche Notation für Graphen sieht wie folgt aus.

$G = (V, E)$ ist ein Graph mit der Knotenmenge V und der Kantenmenge E.

Für die Kantenmenge E gilt $E = \{e = (v_l, v_r) | v_l \in V \wedge v_r \in V\}$.

Sowohl Knoten als auch Kanten besitzen Eigenschaften, die über Attributfelder zugewiesen werden können. Wir betrachten in dieser Arbeit insbesondere typisierte Graphen. Beispielsweise lässt sich eine Eigenschaft „typ“ für Knoten definieren: Sei $v \in V$ ein Knoten mit dem Typ α dann schreibt man: $v.typ = \alpha$. Analog kann man die Eigenschaft „typ“ auch für Kanten definieren: Sei $e \in E$ eine Kante vom Typ β dann schreibt man: $e.typ = \beta$.

In dieser Arbeit verwenden wir Graphen zur Formalisierung der Modelle des Verfahrens und beschreiben automatisierte Vorgänge des Verfahrens mit Hilfe von Graph-Algorithmen und Mengen-Operationen auf Graphen. Man kann Metamodelle als spezielle Form typisierter Graphen auffassen.

2.4.3. Modelltransformationen mit Triple-Graph-Grammatiken

Die hier vorgestellte Notation von Triple-Graph-Grammatiken und die Erklärungen basieren auf [KW07]. TGGs lassen sich für unterschiedliche Anwendungsfälle einsetzen, zum Beispiel zur Modell-Transformation, zur Modell-Integration und zur Modell-Synchronisation. In dieser Arbeit verwenden wir Triple-Graph-Grammatiken zur formellen Beschreibung der Ableitungsregeln. Mit Hilfe von Triple-Graph-Grammatiken lassen sich Transformationen zwischen graphbasierten Modellen deklarativ beschreiben.

Triple-Graph-Grammatiken (TGG) stellen Transformationen als Zusammenspiel dreier Graphen dar. Sie drücken deklarativ die Transformationsbeziehungen zwischen zwei Graphen aus. Als Verbindungselement zwischen den zwei Modellen dient ein drittes Modell dazwischen.

Mit der Grammatik werden die Transformationen in Form von Regeln beschrieben. Sie werden auf Metamodell-Ebene spezifiziert. Die Transformationen selbst spielen sich zwischen Modellinstanzen ab. Die Modellinstanzen können zu unterschiedlichen Metamodellen konform sein. Dementsprechend können Transformationen sowohl innerhalb eines Metamodells beschrieben werden als auch zwischen unterschiedlichen Metamodellen.

Durch die Anwendung der Transformationsregeln wird ein Ausgangsmodell durch die Transformation in ein Zielmodell überführt. Bei Triple-Graph-Grammatiken wird das Ausgangsmodell und das Zielmodell jeweils durch einen Graphen repräsentiert. Die Verbindung zwischen den beiden Graphen wird über einen dritten Graphen spezifiziert.

TGG-Regeln definieren 1) welche Modellelemente und Beziehungen der drei Graphen jeweils als Vorbedingung vorhanden sein müssen und 2) welche Elemente und Beziehungen durch Anwendung der Regel hinzukommen. Zwischen der Definition der TGG-Regeln und der Anwendung der TGG-Regeln

ist zu unterscheiden. Ob und wann eine TGG-Regel angewandt wird hängt vom jeweiligen Anwendungsfall ab. Es kann sowohl eine manuelle Regelanwendung geben als auch eine automatisierte Regelanwendung.

2.5. Terminologie

In diesem Abschnitt erläutern wir einige Begriffe, die in der Arbeit verwendet werden und unterlegen sie mit Beispielen.

Änderungsanfrage Eine Änderungsanfrage stellt den Bedarf einer Änderung fest. Die Änderungsanfrage konzentriert sich dabei auf das „Was“ der Änderung, d.h. sie beschreibt „was“ geändert werden soll. In manchen konkreten Anfrageformulierungen ist eine klare Trennung vom „Was“ und „Wie“ nicht gegeben. Bei der Formulierung sollte auf die Trennung geachtet werden. Die Änderungsanfrage kann von unterschiedlichen Seiten aus geäußert werden und aus unterschiedlichen Motiven an die Projektverantwortlichen herangetragen werden. Beispiele sind der Bedarf neuer Funktionalität, die Änderung bestehender Funktionalität, geforderte Änderungen an Nutzerschnittstellen, die Anpassung des Systems an evolvierende genutzte Drittanbieterkomponenten oder die geforderte Verbesserung von nicht-funktionalen Anforderungen wie z.B. Performanz, Speicherausnutzung, Zuverlässigkeit oder Sicherheitsaspekte.

Umsetzungsweg Ein Umsetzungsweg beschreibt die Schritte, die zur Realisierung einer Änderungsanfrage notwendig sind. Ein Umsetzungsweg bezieht sich dabei explizit auf das „Wie“ der Änderung. Zu einer Änderungsanfrage kann es mehrere Umsetzungswege geben. Es ist eine Herausforderung, einen Umsetzungsweg für eine Änderungsanfrage zu identifizieren und ihn auf die Architektur abzubilden. Beispiele sind die Bearbeitung von Quelltext oder die Konfiguration von Systemen zur Kontinuierlichen Integration.

Aktivität Unter einer Aktivität verstehen wir eine konkrete Handlung einer Person, im Falle einer manuellen Tätigkeit, oder eines Software-Systems, im Falle einer automatisierten Aktivität. Die Aktivität wird auf

Artefakten ausgeführt und liefert in der Regel ein bestimmtes Ergebnis. Die Aktivität kann einem Tätigkeitsfeld zugeordnet werden.

Tätigkeitsfeld Unter Tätigkeitsfeld verstehen wir Arbeitsgebiete oder Entwicklungsbereiche, nach denen Aktivitäten differenziert werden können. Die Tätigkeitsfelder können auch mit Disziplinen verglichen werden. Beispielhafte Tätigkeitsfelder sind die Programmierung, das Testen, die Qualitätssicherung, das Bauen, das Bereitstellen und das Inbetriebnehmen von Software-Systemen. Je nach Prozessmodell werden Tätigkeitsfelder auch mit unterschiedlichen Lebenszyklusphasen der Software in Bezug gesetzt.

Software-Evolution Der Begriff Software-Evolution erfasst die Beobachtung, dass Softwaresysteme im Laufe ihres Lebens immer wieder angepasst werden müssen um einsatztauglich zu bleiben. Sie haben sich an neue Anforderungen aus der Umgebung heraus anzupassen. In der Software-Entwicklung ist die Evolution meist etwas, was aktiv betrieben werden muss und was mit Aufwand verbunden ist.

Rollen vs. Fähigkeiten und Fertigkeiten In dieser Arbeit wird Architekturmodellierung und Projektplanung miteinander in Bezug gesetzt. Dementsprechend unterscheiden wir bei den Personenrollen den Architekten, der vorwiegend die technischen Entscheidungen trifft und den Projektverwalter, der die organisatorischen Rahmenbedingungen plant, die Arbeitspakete definiert, die Aufwands- und Zeitplanung vornimmt und für die Personalzuordnung zuständig ist. In Hinblick auf Prozessmodelle, welche den Software-Architekten als explizite Rolle kritisch betrachten sei erwähnt, dass der Architekt in dieser Arbeit auch als „Sammelung von Fähigkeiten und Fertigkeiten“ verstanden werden kann, welche in der Praxis auch durch das Entwicklungsteam selbstorganisiert ausgeübt werden können.

3. Verwandte Arbeiten

In diesem Kapitel betrachten wir existierende Lösungen für Teile unserer Problemstellung. Wir zeigen die Gemeinsamkeiten auf und diskutieren Einschränkungen der existierenden Lösungen, welchen wir durch unsere eigene Lösung begegnen wollen.

Wir können die verwandten Arbeiten in Gruppen einteilen. Jeder dieser Gruppen ist ein Unterabschnitt dieses Kapitels gewidmet. Die erste Gruppe umfasst Ansätze zur architekturbasierten Projektplanung in Abschnitt 3.1. Diese Verfahren kombinieren die Software-Architektur explizit mit der Projektverwaltung. Die zweite Gruppe verwandter Arbeiten, in Abschnitt 3.2, beschreibt Ansätze, welche die Architektur zur Begleitung der Software-Evolution verwenden. In Abschnitt 3.3 besprechen wir Ansätze, welche Qualitätseigenschaften der Software auf Basis der Software-Architektur analysieren und dazu sogenannte Szenarien verwenden. Diese Verfahren betrachten unterschiedliche Qualitätseigenschaften, wie zum Beispiel, Laufzeitverhalten oder Wartbarkeit.

Wie bereits im ersten Kapitel beschrieben, besteht unsere Zielsetzung darin, ein Verfahren zu entwickeln, welches die Software-Architektur mit der Projektverwaltung verknüpft und Entscheidungsträger bei der Bewertung von Änderungsanfragen im Rahmen der Software-Evolution unterstützt.

Die zentralen Herausforderungen hierbei sind, die explizite Betrachtung von Änderungsausbreitungen innerhalb der strukturellen Software-Architektur zu berücksichtigen und die Auswirkungen auf die Tätigkeitsfelder zu bestimmen. Damit verbunden ist, dass entscheidungsrelevante Informationen an das Architekturmodell annotiert werden sollten und mit automatisierten Verfahren die Auswertung dieser Informationen für konkrete Änderungsanfragen möglichst ohne große Interaktion mit dem Anwender möglich ist.

Insgesamt lassen sich die gewünschten Eigenschaften des Verfahrens in folgenden Kriterien zusammenfassen.

Verknüpfung von Architekturmodellierung und Projektverwaltung Das Verfahren sollte Architekturmodellierung und Projektverwaltung explizit miteinander verbinden.

Leichtgewichtigkeit Es ist ein leichtgewichtiger Ansatz gewünscht, der mittels Werkzeugunterstützung einfach in verschiedene Arten von Entwicklungsprozessen eingebunden werden kann und schnelle Rückkopplung ermöglicht.

Lebenszyklusabdeckung Es ist ein Verfahren gewünscht, welches den Lebenszyklus des Software-Systems ganzheitlich betrachtet und dabei in verschiedenen Phasen, d. h. Software-Entwurf, Software-Implementierung und Software-Evolution, eingesetzt werden kann.

Tätigkeitsfeldabdeckung Es ist ein Verfahren gefordert, welches die gängigen Tätigkeitsfelder der Software-Entwicklung berücksichtigt, welche in vielen Entwicklungsvorhaben vorkommen. Dazu zählen insbesondere die Tätigkeitsfelder Programmierung, Fremdkomponenten-Konfiguration, Bauen, Testen, Bereitstellung und Inbetriebnahme.

Szenarien-Basiertheit Das Verfahren sollte die Software-Evolution anhand konkreter Änderungsanfragen bewerten. Änderungsanfragen sind ein Spezialfall von Szenarien.

Berücksichtigung der strukturellen Änderungsausbreitung Das Verfahren sollte die strukturelle Änderungsausbreitung ausgehend Modifikationen im System berücksichtigen.

Berücksichtigung der organisatorischen Auswirkungen Das Verfahren sollte die Auswirkungen von Modifikationen auf die Tätigkeitsfelder bestimmen können.

Skalierbarkeit durch Automatisierung Das Verfahren sollte die entscheidungsrelevanten Daten automatisiert verarbeiten können.

Diese Kriterien werden in den nachfolgenden Abschnitten dazu verwendet, um die Gemeinsamkeiten und Einschränkungen der verwandten Arbeiten zu erörtern.

3.1. Architekturbasierte Projektplanung

Dieser Abschnitt diskutiert Ansätze, welche die Software-Architektur mit der Projektplanung oder -organisation in Verbindung bringen. Wir werfen zunächst einen Blick auf die sogenannte Spiegel-Hypothese (auch bekannt als Conways Gesetz). Anschließend stellen wir den architektur-zentrierten Projektplanungsansatz von D. J. Paulish vor. Das letzte Verfahren in dieser Kategorie, von R. Carbon, behandelt die Produzierbarkeit (engl. producibility) von Software-Systemen.

3.1.1. Spiegel-Hypothese (Conways Gesetz)

Melvin Conway [Con68] vertrat 1968 den Standpunkt, Entwicklungsorganisationen seien gezwungen, Entwürfe hervorzubringen, die Kopien der Kommunikationsstrukturen dieser Organisationen repräsentieren. Dieses Konzept wird unter den Begriff der Spiegel-Hypothese oder auch Conways Gesetz gefasst.

Diese Hypothese wird auch heute noch in der Literatur behandelt. In [KCD12] diskutieren die Autoren die Angemessenheit und Gültigkeit von Conways Gesetz. Sie identifizieren andere Wissenschaftler ([Par72] [HG99] [BC00] [CB10]), die Conways Gesetz seither verfeinert haben und außerdem gezeigt haben, dass die Angleichung technischer und organisatorischer Strukturen die Produktivität und Produktqualität bei der Entwicklung materieller (engl. physical) Systeme verbessert. Sie sehen jedoch Einschränkungen bei der Übertragbarkeit auf Software-Systeme und weisen auf die Unterschiede zwischen Software-Systemen und materiellen Systemen hin.

Die Frage, ob Conways Gesetz in der Software-Entwicklung nach wie vor eine Rolle spielt beantwortet [KCD12] mit „Ja“. Allerdings sei die Betrachtungsweise maßgeblich. Software-Praktiker neigten dazu, die Produktstruktur als Software-Architektur und die Organisationsstruktur als Organigramm zu konzeptionalisieren. Das sei allerdings nicht ausreichend für den Projekterfolg.

Ein Problem mit der Vogelperspektive auf Organisation und Software-Architektur sei, dass Entwickler dort ihre Arbeit nicht wiederfinden. Sie sehen ihre Arbeit üblicherweise als eine Menge von Zielen, die das System erfüllen

müsse. Folglich bestehe ihre Aufgabe in der Modifikation des Systems und der Koordination dieser Anpassungen mit anderen, um die Ziele zu erfüllen [Nor03].

Insgesamt ist es demnach ein Problem, dass bei der Architekturbetrachtung die Feinheiten der Entwicklungsarbeit fehlten. In einigen Fällen könne die Architektur als Koordinationsmechanismus [SR11] dienen, allerdings fehle der Nachweis eines projektweiten Vorteils durch die Angleichung, wie Produktivitäts- oder Qualitätsverbesserung. Im Gegenteil seien ihnen zwei Studien [ORM03] [Bas+07] bekannt, in denen berichtet wird, dass eine Architektur-Angleichung nicht ausreichend war, um die benötigte Koordination zu ermöglichen.

Die Spiegel-Hypothese und die damit verbundenen Diskussionen zeigen eindeutig, dass es wichtig ist, Architektur und Organisation gemeinsam zu betrachten. Jedoch ist die Hypothese allein nicht ausreichend, um aktiv eine Lösung beizusteuern. Diese Dissertation möchte hierzu einen kleinen Beitrag leisten.

3.1.2. Architekturzentrierte Projektplanung (ACSPP) nach D. J. Paulish

D. J. Paulish präsentiert in [Pau01] die sogenannte architektur-zentrierte Software-Projektverwaltung. In diesem Buch fasst er unter dieser Bezeichnung mehrere Verfahren zusammen. Ein Verfahren aus diesem Buch, welches wir hier besonders hervorheben möchten, ist die architektur-zentrierte Software-Projektplanung (engl. Architecture-Centered Software Project Planning, ACSPP). Bei dem Verfahren geht es darum, die Software-Architektur als Artefakt in den verschiedenen Planungsschritten zu berücksichtigen. Ein Ziel ist es, mit Hilfe der Kombination von absteigender (engl. top-down) und aufsteigender (engl. bottom-up) Aufwandsschätzung realistische Zeitpläne zu ermitteln. ACSPP greift dabei die Idee auf, die Arbeit ausgehend von der Grobstruktur der Architektur auf Teilaktivitäten herunterzubrechen. Ein spezielles Schätzformular dient als Instrument zur aufsteigenden Schätzung.

Der ACSPP-Ansatz wurde parallel und in Verbindung mit dem Architekturentwurfsansatz von Hofmeister, Nord und Soni [HNS00] entwickelt und im Rahmen von Projekten bei Siemens erprobt und besitzt damit praktische

Relevanz. Konkrete praktische Erfahrungen mit seinem Ansatz beschreibt der Autor in [PNS96] und [PC94].

Der Autor sieht den Schlüssel für die Effektivität der vorgeschlagenen Methoden im Arbeitsverhältnis zwischen Projektverwalter und Software-Architekt. Demnach sollen sie als „Entscheidungsfindungsteam“ zusammenarbeiten. Das entspricht voll und ganz unserer Erwartungshaltung, derzufolge die Architektur explizit bei der Projektplanung miteinbezogen werden soll.

Ab einer bestimmten Gruppengröße empfiehlt D. Paulish eine personelle Trennung von Projektverwalter und Architekt. Diese Rollentrennung kommt übrigens in modernen Entwicklungsmethodiken, wie beispielsweise Scrum, ebenfalls zum Tragen, da technische und organisatorische Belange sich oftmals gegenseitig ausschließen und es ohne eine Trennung leicht zu inneren Konflikten kommt. Wer sich jetzt wundert, was Scrum mit Architektur zu tun hat, in Scrum entscheidet das Entwicklerteam selbständig über die Architektur (vgl. [PR11], Kapitel 2).

D. Paulish nennt den Standish Group CHAOS-Bericht (1995, 1998), in dem Gründe für den Erfolg und Misserfolg von Software-Entwicklungs-Projekten untersucht werden. Dieser kann ebenfalls als Argumentationsquelle für die notwendige Verknüpfung von Software-Architektur und Projekt-Management herangezogen werden. Allerdings gibt es mittlerweile auch kritische Stimmen, wie zum Beispiel in einem Netzwoche-Artikel von D. Liebhart [Lie09], in dem dieser den CHAOS-Bericht als nicht mehr ganz zeitgemäß oder sogar unseriös bezeichnet oder auch [Gla06].

Wir übernehmen für unser Verfahren die grundsätzliche Idee, die Software-Architektur und die Projektverwaltung zu kombinieren, sowie die aufsteigende Kostenschätzung auf der Basis von Architekturelementen (Komponenten) zu ermöglichen. Auf der anderen Seite gibt es Einschränkungen bezüglich unserer Erwartungen. Zum Einen konzentriert sich das Verfahren hauptsächlich auf die Neuentwicklung von Systemen und betrachtet weniger den Aspekt der Software-Evolution. Zum Anderen scheint das Verfahren nicht szenarienbasiert zu sein. Vor allem aber wollen wir die Entscheidungsträger durch Automatisierung entlasten. Für ACSPP sind uns jedoch weder eine Werkzeugunterstützung noch andere Arten der Automatisierung bekannt.

3.1.3. Produzierbarkeitsanalyse nach R. Carbon

R. Carbon [Car12] greift in seiner Dissertation das oft wenig gehegte Verhältnis zwischen der Software-Architektur und Software-Projekt-Planung auf. Er identifiziert als Hauptursache für verschiedene Probleme die Abhängigkeiten zwischen Architekturelementen, welche wiederum zu Abhängigkeiten zwischen Ressourcen führen. In der Folge komme es dazu, dass Aktivitäten, die sich auf das selbe Artefakt beziehen, mit Verzögerungen und Konflikten behaftet sind. Darüber hinaus werden Architekturelemente oftmals mehrfach modifiziert und führten so zu Mehraufwand.

Seine Arbeit wird inspiriert durch einen Blick auf eine andere Disziplin, den Maschinenbau. Im Maschinenbau werde mehr darauf geachtet, Produktentwurf und Produktionsplanung aufeinander abzustimmen. Aus dieser Erkenntnis leitet R. Carbon eine Qualitätseigenschaft ab, welche er unter die Bezeichnung „Produzierbarkeit“ (engl. Producibility) fasst. Diese Qualitätseigenschaft erfasst die Abgleichung zwischen Architektur und Projektplan. Die Arbeit bietet neben der Bildung des Qualitätsbegriffs ein Verfahren zur Analyse der Produzierbarkeit. Der Autor unterstützt mit seinem Ansatz schwerpunktmäßig folgende Anwendungsfälle: Erstens, die frühzeitige Erkennung von potentiellen Produktionsproblemen und zweitens, die Überprüfung der Architektur auf Vollständigkeit.

Das Verfahren kümmert sich, was das Verhältnis von Software-Architektur und Produktionsplanung anbelangt, um ähnliche Problemstellungen. Allerdings handelt es sich nicht um ein szenarien-basiertes Verfahren, das es ermöglicht, konkrete Änderungsanfragen auf der Architektur zu untersuchen. Darüber hinaus sieht das Verfahren keine automatisierte Änderungsausbreitungsanalyse und Aktivitätsherleitung vor, sondern fokussiert die Bestimmung der Abgleichung von Software-Architektur und Produktionsplan als Qualitätsmaß.

3.2. Architekturbasierte Softwareevolution

In diesem Abschnitt betrachten wir Ansätze, welche die Software-Evolution auf der Basis der Software-Architektur analysieren und unterstützen. Wir

stellen einen Ansatz mit Evolutionspfaden und -stilen von D. Garlan vor, sowie einen Ansatz zur Flexibilitätskonstruktion und -analyse von M. Naab.

3.2.1. Evolutionspfade und Evolutionsstile nach D. Garlan

Laut David Garlan (und anderen) [Gar+09] stehen für Architekten nur wenige Werkzeuge und Methoden bereit, die bei der Evolution von Software-Architekturen helfen. Insbesondere existiere wenig Unterstützung bei der Planung von Evolutionspfaden und der damit verbundenen Abwägung verschiedener Aspekte unterschiedlicher Pfade. Sie schlagen deshalb ein Verfahren vor, das Architekten bei der Entwicklung und Bewertung (engl. reasoning) architektureller Evolutionspfade unterstützt.

Bei der Untersuchung ihres Ansatzes gelangen sie zu der Einsicht, dass architekturell betrachtet, viele Systeme bestimmten Mustern bei der Evolution folgen. Diese Muster nennen sie Evolutionsstile. Sie definieren Evolutionsstile und zeigen, wie diese zur automatisierten Unterstützung zur Beschreibung architektureller Evolution und für die Bewertung von „Korrektheit und Qualität“ der Evolutionspfade verwendet werden können.

Sie heben die Bedeutung der Architekturevolution hervor und bedauern die mangelnde Werkzeugunterstützung bei der Planung von Evolutionspfaden. Sie sehen die Betrachtung der Software-Evolution aus der Perspektive der Architektur als wichtige Ergänzung im Vergleich zur herkömmlichen Betrachtungsweise der Software-Evolution, da hiermit eine Planung und Restrukturierung auf einem höheren Abstraktionsniveau erlaubt sei, auf dem Abwägungen für Qualitäts- und Geschäftsziele analysiert und verstanden werden können.

Das vorgestellte Verfahren basiert auf dem Konzept der *Evolutionspfade*, die eigenständige Entitäten darstellen und deren Klassen domänen-spezifischer Evolutionspfade formal spezifiziert werden können um *Wiederverwendung, Korrektheitsprüfung und Qualitätsanalysen* zu unterstützen. Weiterhin können *Abwägungsanalysen* (engl. tradeoff analyses) zwischen alternativen Evolutionspfaden durchgeführt werden. Außerdem können *Werkzeuge* die Beschreibung, Analyse, Verfolgung und Modifikation der Architekturevolution unterstützen. Ein weiteres Konzept sind die *Evolutionsstile*, welche Familien

domänen-spezifischer Evolutionspfade definieren, die gemeinsame Eigenschaften besitzen und gemeinsamen Randbedingungen genügen. Die *Haupterkenntnis* auf der die Anwendungsmöglichkeiten für Evolutionsstile beruhen, ist, dass innerhalb einer Domäne die Evolutionspfade ähnlich sind und somit das Wissen aus der Vergangenheit innerhalb einer Domäne für spätere Evolutionsplanungen verwendet werden kann. In der Publikation beschreiben die Autoren ein Illustrationsbeispiel. Als *Formalismus* für die Beschreibung von Pfadbedingungen für Evolutionspfade verwenden sie Lineare Temoral-Logik (engl. linear temporal logic). Für Evolutionsstile werden individuelle *Operatoren* spezifiziert. Operatoren werden mit der Stitch-Sprache beschrieben, die Kontrollstrukturen und primitive Architekturoperatoren bereitstellt. Um die Auswahl zwischen alternativen Evolutionspfaden zu unterstützen werden *Bewertungsfunktionen* verwendet. Die bewerteten Attribute sind spezifisch für den Geschäftskontext. Das Verfahren wurde in einem Werkzeug umgesetzt. Eine ausführliche Validierung (abgesehen vom Illustrationsbeispiel) wird in [Gar+09] nicht erwähnt.

Das Verfahren ist in dem Sinne zu unserem Vorhaben verwandt, als dass wir ebenfalls Evolutionsschritte in Form von einer Folge von Änderungsanfragen auf dem Architekturmodell analysieren wollen. Demnach unterstützt der Ansatz wie wir das Bestreben, die Software-Evolution auf Basis der Software-Architektur zu analysieren. Allerdings nennt das Verfahren keine explizite Berücksichtigung verschiedener Tätigkeitsfelder und stellt keinen Bezug der Architekturmodifikation mit Auswirkungen auf die Projektplanung her. Darüber hinaus wird die strukturelle Ausbreitung von Änderungen im System nicht berücksichtigt.

3.2.2. Flexibilitätsmodellierung und -analyse nach M. Naab

M. Naab [Naa12] behandelt in seiner Dissertation die Flexibilität von Software-Systemen. Er stellt ein Verfahren vor, mit dem man zum Entwurfszeitpunkt die Flexibilität in der Software-Architektur explizit berücksichtigen kann. Sein Ansatz stellt als Instrument spezielle Sichten (engl. Change Impact Views) bereit, in welchen der Architekt die Auswirkungen von Änderungsszenarien auf Architekturelemente (Komponenten) während des Architekturentwurfs modellieren kann.

Er entwickelt keinen neuen Architekturentwurfsprozess, sondern erweitert vielmehr existierende Prozesse. Phasen, die beim Architekturentwurf durchlaufen werden sollen sind unter anderem 1) die funktionale Dekomposition und Abstraktion, 2) die Anwendung von Architekturmechanismen, 3) die Abbildung der Funktionalität (auf die Architektur), sowie 4) die Flexibilitätsbewertung.

Das bewusste Adressieren von Flexibilität beim Entwurf fasst der Autor unter den Begriff *Flexibilitätskonstruktion*. Neben der Konstruktion bietet das Verfahren auch einen analytischen Anteil, mit dem vorhandene Flexibilität gezielt ausgenutzt werden kann.

Die Analyse von Flexibilitätsszenarien ist sehr verwandt zur Untersuchung von Änderungsanfragen auf dem Architekturmodell. Darüber hinaus stellen die Auswirkungs-Sichten (Change Impact Views) eine ähnliche Art der Modellierung von Änderungen im Architekturmodell im Vergleich zu unserem Vorhaben dar. Jedoch ist das Verfahren mehr auf den Entwurfszeitpunkt fokussiert und zieht spätere Auswirkungen im Lebenszyklus nicht in die Betrachtung mit ein. Zudem werden die Auswirkungen auf einzelne Tätigkeitsfelder nicht ermittelt.

3.3. Szenarienbasierte Architekturbewertung

In diesem Abschnitt besprechen wir Architekturbewertungsansätze, welche zur Analyse von Qualitätseigenschaften Szenarien verwenden. Die folgenden Abschnitte behandeln die Verfahren SAAM, ATAM, ALPSM und ALMA.

3.3.1. SAAM

Die „Software Architecture Analysis Method“ (SAAM) [CKK05] wurde 1994 von Rick Kazman, Len Bass, Mike Webb und Gregory Abowd am SEI als eine der ersten Methoden zur Evaluierung der Software-Architektur bezüglich der Änderbarkeit (und weiterer Qualitätseigenschaften) entwickelt. Die Methode verwendet eine informell beschriebene Architektur - in erster Linie die strukturelle Sicht - und beginnt mit der Sammlung von Änderungsszenarien. In verschiedenen Schritten versucht dann das Verfahren, Szenarien zu finden,

die in Wechselbeziehungen stehen, bei denen zum Beispiel die Schnittmenge der betroffenen Komponenten nicht leer ist. Komponenten, die von mehreren Szenarien betroffen sind werden als kritisch angesehen und mit besonderer Aufmerksamkeit betrachtet. Für jedes Änderungsszenario werden die Kosten abgeschätzt. Das Ergebnis von SAAM sind klassifizierte Änderungsszenarien und eine überarbeitete Architektur mit weniger kritischen Komponenten.

Das Verfahren ist verwandt zu unserem Ansatz, da es wie wir Änderungsszenarien, bei uns als Änderungsanfragen bezeichnet, auf dem Architekturmodell analysiert. Allerdings ist das Verfahren nicht automatisiert und bietet zudem keine Auswirkungsanalyse für konkrete Tätigkeitsbereiche, wie zum Beispiel Inbetriebnahme (engl. Deployment).

3.3.2. ATAM

Die „Architecture Trade-Off Analysis Method“ (ATAM) [CKK05] wurde von einer ähnlichen Gruppe wie SAAM am SEI entwickelt und berücksichtigt Erfahrungen, welche mit SAAM gemacht wurden. Insbesondere wollte man die Einschränkung von SAAM überwinden, nur jeweils einzelne Qualitätsattribute zu betrachten. Vielmehr hat man festgestellt, dass viele Qualitätsattribute in Wechselwirkungen stehen und beispielsweise ein Qualitätsattribut andere Qualitätsattribute beeinflusst. Daher erlaubt es ATAM Abwägungen (engl. tradeoffs) zwischen Qualitätseigenschaften zu erörtern. Die Methode erweitert SAAM auch dahingehend, dass mehr Führung und Unterstützung bei der Ermittelung von Änderungsszenarien angeboten wird. Sobald Änderungsszenarien ermittelt sind, wird jedes Qualitätsattribut zunächst isoliert analysiert. Dann werden im Gegensatz zu SAAM Architekturentscheidungen identifiziert und die Auswirkung (Sensitivität) der Architekturentscheidungen auf jedes Qualitätsattribut abgeschätzt. Mit dieser Sensitivitätsanalyse werden abhängige Qualitätseigenschaften aufgespürt und Kompromisse explizit gemacht.

Wie SAAM ist auch ATAM in der Lage Änderungsanfragen auf dem Architekturmodell zu analysieren. Allerdings gibt es auch hier keine automatisierte Unterstützung für die Bestimmung von Auswirkungen. Es werden keine expliziten Auswirkungen auf die Tätigkeitsfelder und späteren Lebenszyklusphasen betrachtet.

3.3.3. ALPSM

„Architecture-Level Prediction of Software Maintenance“ (ALPSM) [BB99] ist eine Methode, die sich ganz auf die Vorhersage der Software-Wartbarkeit eines Software-Systems basierend auf dessen Architektur konzentriert. Die Methode beginnt mit der Definition einer repräsentativen Menge von Änderungsszenarien für unterschiedliche Wartungskategorien (z. B. Fehlerbehebung, Anpassung an veränderte Umgebung), welche im Anschluss bezüglich ihrer Eintrittswahrscheinlichkeit im Lebenszyklus gewichtet werden. Im nächsten Schritt - dem sogenannten „Szenario-Skripten“ - werden für jedes Szenario die Auswirkungen der Implementierung auf die Architektur bewertet, basierend auf Schätzungen der Komponentengrößen. Diese Informationen ermöglichen der Methode anschließend die Vorhersage des Gesamtwartungsaufwands durch die Berechnung des gewichteten Durchschnitts der Aufwände für die einzelnen Szenarien. Als Hauptvorteile gegenüber SAAM und ATAM betonen die Autoren, dass ALPSM weder eine abgeschlossene (fertige) Architektur benötigt noch alle Interessenseigner miteinbezieht. Folglich benötigt die Methode weniger Ressourcen und Zeit und kann von Software-Architekten (alleine) zur wiederholten Wartbarkeitsbewertung angewandt werden.

Die Methode hängt jedoch noch stark von der Erfahrung der Software-Architekten ab und bietet wenig Unterstützung durch Werkzeuge und Automatisierung. Die automatisierte Ableitung von Aktivitäten in den Tätigkeitsbereichen ist bei dem Verfahren nicht vorgesehen.

3.3.4. ALMA

Die „Architecture-Level Modifiability Analysis“ (ALMA) [Ben+04] ist eine szenarien-basierte Architekturanalysetechnik, die aus der Kombination verschiedener Konzepte aus dem SAAM- und ATAM-Umfeld entstanden ist. Ein wesentliches Merkmal von ALMA ist, dass man sich schwerpunktmäßig auf Modifizierbarkeit konzentriert, dabei aber drei konkrete Analyseziele begleitet. Die unterstützten Ziele sind: 1) die Vorhersage (Abschätzung) von Wartungskosten, 2) die Risikobewertung und 3) die Auswahl von alternativen Software-Architekturen.

Die Methodik von ALMA sieht fünf Schritte vor: 1) Ziel festlegen, 2) Software-Architektur beschreiben, 3) Szenarien ermitteln, 4) Szenarien evaluieren und

5) Ergebnisse interpretieren. Je nach gewählten Analyseziel in Schritt 1 variieren die nachgelagerten Analyseschritte. Bei der Beschreibung der Architektur erfassen die Anwender dabei die Dekomposition des Systems in Komponenten, die Beziehungen zwischen den Komponenten, sowie die Beziehungen zur System-Umgebung. Zur Ermittlung von Szenarien werden zwei prinzipielle Strategien vorgeschlagen, aufsteigend (engl. bottom-up) oder absteigend (engl. top-down). Die Evaluierung eines Szenarios geschieht in drei Schritten: 1) Betroffene Komponenten identifizieren, 2) Auswirkung auf die Komponenten bestimmen, 3) Änderungsausbreitungseffekte (engl. ripple effects) bestimmen.

Insgesamt kommt ALMA durch die Analyse von Änderungsszenarien und die systematische Vorgehensweise insbesondere durch die explizite Betrachtung von Änderungsausbreitungseffekten unserer Zielsetzung, Software-Architekten und Projektverwalter bei der Planung und Bewertung von Änderungsanfragen zu unterstützen, sehr nahe. Auf der anderen Seite vermissen wir die automatisierte Unterstützung durch Werkzeuge, sowie die Ermittlungen von Auswirkungen aus dem System heraus in die Tätigkeitsbereiche und damit für unterschiedliche Lebenszyklusphasen.

3.4. Änderungsausbreitungsanalyse

Neben den drei Kerngruppen, welche wir in den vorangehenden Abschnitten behandelt haben, gibt es noch einen Bereich, welchen wir nicht unerwähnt lassen möchten, welcher das Verfahrens inspiriert hat, der Bereich der Änderungsausbreitungsanalyse.

Stellvertretend für eine Vielzahl von Verfahren zur Änderungsausbreitungsanalyse betrachten wir einen Artikel von S. Black. S. Black stellt in [Bla01] eine Überarbeitung des Algorithmus von Yau und Collofello zur Berechnung von Änderungsauswirkungen (engl. ripple effects) vor. Sie wendet eine Implementierung ihres Algorithmus auf eine Sammlung von C-Programmen an und vergleicht ihren Algorithmus mit dem von Yau und Collofello. In der Einleitung weist sie auf bestehende Wartungsmethodiken hin, zum Einen das SADT-Modell von Pfleeger und Bohner aus dem Jahr 1990 und zum Anderen von Yau und Collofello aus dem Jahr 1980. Sie erwähnt die Wartungsklassifikationen von Swanson (1976) in korrektive, adaptive, perfektive Wartung

und die Erweiterung dieser Klassifikation durch den IEEE-Glossar (1990) um präventive Wartung. Weiterhin nennt sie die Wartungs-Ontologie von Kitchenham (1999), die Korrekturen (engl. corrections) von Verbesserungen (engl. enhancements) unterscheidet. Bei Kitchenham werden Verbesserungen weiter unterteilt in solche die 1) bestehende Anforderungen verändern, 2) die neue Anforderungen hinzufügen und 3) solche, die die Implementierung verändern, aber die Anforderungen unverändert lassen. Sie erwähnt eine Analyse und Klarstellung dieser Wartungsdefinitionen von Chapin et. al. (2001). Im nächsten Abschnitt führt die Autorin den Begriff der Änderungsauswirkung (engl. ripple effect) ein und erläutert seine Bedeutung für die Wartung. Sie zeigt wie sich die Änderungsauswirkungsanalyse zu einem Teil des Wartungsprozesses entwickelt hat. Ausgehend vom Wartungsprozess von Boehm (1987), der die Phasen 1) „Software verstehen“, 2) „Software verändern“ und 3) „Software überprüfen“ umfasst, betrachtet sie den Wartungsprozess von Yau (1980), der die Auswirkungsanalyse (engl. impact analysis) explizit in den Lebenszyklus aufnimmt. Der Begriff „ripple effect“ wurde laut der Autorin zum ersten Mal von Haney (1972) verwendet. Weitere Arbeiten zur Analyse von Änderungsauswirkungen werden genannt und verglichen. Das eigene Verfahren von S. Black basiert auf dem Algorithmus von Yau und Collofello (1984) und verwendet Matrizenarithmetik. Sie unterscheidet Intra-Modul-Änderungsausbreitung (engl. intramodule change propagation) und Inter-Modul-Ausbreitung (engl. intermodule change propagation).

Das Verfahren hat den in dieser Dissertation vorgestellten Ansatz dahingehend inspiriert, als dass die Wichtigkeit zur expliziten Betrachtung von Änderungsausbreitungen zum Ausdruck kommt. Weiterhin werden die Strategien, die Ausbreitung innerhalb von Modulen (Komponenten) und zwischen Modulen (Komponenten) zu ermitteln, in unserem Verfahren wieder aufgegriffen. Auf der anderen Seite erweitern wir den Ausbreitungsbegriff auf Auswirkungen in Tätigkeitsfelder und Lebenszyklusphasen.

4. Verfahren zur Bewertung und Planung von Änderungsanfragen

In diesem Kapitel beschreiben wir das Verfahren zur Änderungsanfragenanalyse im Architekturmodell. Wir beschreiben den Anwendungsrahmen, in welchen das Analyseverfahren eingebettet ist und zeigen ausgehend von einer Auswahl konkreter Änderungsanfragen in einem begleitenden Beispielsystem die Fragestellungen auf, mit denen ein Anwender an das Verfahren herantritt.

Die Anwender des Verfahrens sind die Projektverantwortlichen. Unter dieser Bezeichnung fassen wir in dieser Arbeit die Projektverwalter und Architekten, d.h. die organisatorischen und technischen Entscheidungsträger zusammen.

Im weiteren Verlauf identifizieren wir verschiedene Herausforderungen, welche wir bei der Lösung berücksichtigen müssen und erläutern, wie das Verfahren diese behandelt. Danach geben wir einen Überblick über die Bestandteile des Verfahrens. Wir gehen auf die Eingabedaten ein, die der Anwender in der Vorbereitungsphase bereitstellen muss und beschreiben die Schritte der Analysephase. Dabei betrachten wir die automatisierten Anteile des Verfahrens und zeigen die jeweiligen Interaktionsmöglichkeiten des Anwenders auf. Schließlich werfen wir einen Blick auf die Zwischen- und Endergebnisse der Analyse. Die einzelnen Bestandteile werden jeweils anhand des durchgängigen Beispiels illustriert.

Im nachfolgenden Kapitel beschreiben wir die konkreten Metamodelle, sowie die Formalisierung der Analyseschritte in Form von Transformationen auf den Metamodellen.

4.1. Ausgangssituation und Anwendungsszenarien

Während seines Lebens wird ein Software-System in der Regel mit Änderungsanfragen konfrontiert. Tritt eine Änderungsanfrage auf, so müssen die Projektverantwortlichen entscheiden, wie die Änderungsanfrage umgesetzt werden soll. Für jede Änderungsanfrage werden Umsetzungswege erörtert, bewertet und geplant. Abbildung 4.1 zeigt diese Ausgangssituation.

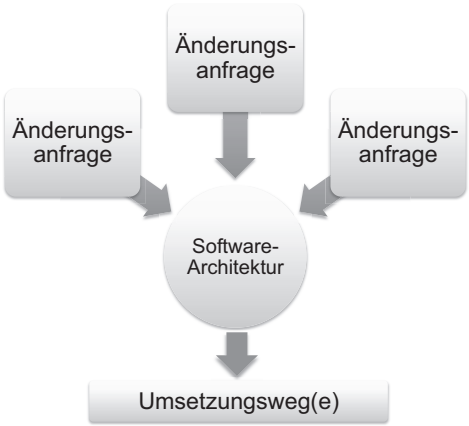


Abbildung 4.1.: Die Software-Architektur wird mit Änderungsanfragen konfrontiert. Die Projektverantwortlichen müssen jeweils Umsetzungswege bestimmen.

Die Identifikation von Umsetzungswegen ist in vielen Fällen ein kreativer Prozess, der vor allem davon abhängt, wie die Anforderungen im System realisiert sind und welches technische Lösungsrepertoire den Architekten zur Verfügung steht. Selbst mit einer genauen Anforderungsabbildung auf das Architekturmodell bedarf es bei der Bestimmung von Umsetzungswegen manueller Beteiligung, denn die Informationen zur Lösungsfindung können aufgrund der Vielfalt nicht universell bereitgestellt werden. Wir überlassen die Identifikation von Umsetzungswegen daher den Projektverantwortlichen. Allerdings unterstützt unser Ansatz die Projektverantwortlichen dabei, einen vorgeschlagenen Umsetzungsweg auf Modellebene zu verfeinern und auf

seine strukturellen und projekt-organisatorischen Auswirkungen hin zu untersuchen.

Denn eine Änderung am System hat in der Regel weitere Auswirkungen zur Folge. Änderungen an Komponenten können sich über die angebotenen Schnittstellen auf abhängige Komponenten ausbreiten. Natürlich sollten Schnittstellen nach Möglichkeit stabil bleiben, aber das ist nicht immer garantiert. Selbst wenn die Signatur der Schnittstelle unverändert bleibt, kann es zu „versteckten“ semantischen Änderungen kommen, die in den nutzenden Komponenten behandelt werden müssen. Die strukturelle Ausbreitung muss bei der Bewertung und Planung von Umsetzungswegen berücksichtigt werden.

Aus projekt-organisatorischer Perspektive bedeutet eine Ausbreitung von Änderungen in der strukturellen Architektur oftmals eine Ausbreitung auf unterschiedliche Arbeitspakete und somit innerhalb der Personalstruktur. Je komplexer und größer Änderungen sind, desto mehr Personen werden in die Umsetzung involviert. Demnach ist es für die Projektverwalter von Interesse, über die Änderungsausbreitung und die damit verbundenen personellen Konsequenzen Bescheid zu wissen.

Die Grundidee des Ansatzes ist es, Projektverantwortliche bei der Analyse der Auswirkungen von Änderungsanfragen auf das Software-System zu unterstützen. Die Analyse wird auf Architekturebene mit Hilfe von Architekturmodellen durchgeführt.

Der Vorteil der Architekturmodellierung ist, dass sich der Anwender auf abstrakter Ebene ein Bild über die Struktur des Systems, d.h. die Komponenten und deren Zusammenspiel, machen kann. Viele Details werden ausgeblendet, damit man sich auf das Wesentliche konzentriert und den Überblick über den Änderungsvorgang behält. Allerdings sollten die für die jeweilige Entscheidungssituation relevanten Informationen im Modell vorhanden sein. Deshalb beziehen wir entscheidungsrelevante Informationen abgebildet und assoziiert auf die Architekturelemente mit ein.

Für die technischen Entscheidungsträger, die Architekten, wäre es angenehm, wenn sie Umsetzungswege direkt in ihrer „gewohnten“ Arbeitsumgebung, d.h. mit Modellierungswerkzeugen auf Architekturmodellen entwickeln könnten. Jedoch ist eine Veränderung des Architekturmodells nicht zu verwechseln mit der eigentlichen Umsetzung der Änderungsanfrage im System. Kästchen und Linien sind in Modellen (oder auf Präsentationsfolien) schnell gezogen

und verändert. Die tatsächliche Komplexität, die hinter hinzugefügten, modifizierten und entfernten Architekturelementen steht bleibt zunächst verborgen. Beispielsweise zeigt Abbildung 4.2 eine modellierte strukturelle Architekturmodifikation, bei der eine Zwischenspeicher-Komponente (engl. cache) eingefügt wird. Diese Veränderung wirkt, rein auf dem Modell betrachtet, einfach. Die eigentlichen Aufwände entstehen bei der Umsetzung in konkreten Entwicklungsbereichen und Tätigkeitsfeldern. Quelltext muss modifiziert werden, Komponenten müssen neu gebaut werden, Testfälle müssen entwickelt oder angepasst und durchgeführt werden, eine installationsfähige Version des Systems muss bereitgestellt werden und das modifizierte System muss (erneut) in Betrieb genommen werden. Folglich gilt, um Umsetzungswege zuverlässig bewerten und gegeneinander abwägen zu können, ist es erforderlich, die konkreten Aktivitäten bei der Umsetzung zu betrachten.

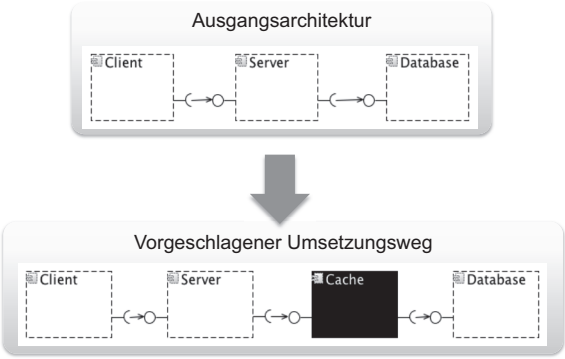


Abbildung 4.2.: Architekturmodelländerungen verbergen Folgeauswirkungen

Das Verfahren zur Bewertung und Planung von Änderungsanfragen kann in vielfältigen Anwendungsszenarien eingesetzt werden. Im Folgenden nennen wir drei prinzipielle Anwendungsszenarien.

1) Abwägung alternativer Umsetzungswege: In manchen Fällen kann eine Änderungsanfrage auf unterschiedliche Weisen umgesetzt werden, so dass verschiedene alternative Umsetzungswege existieren, zwischen denen abgewogen werden muss. Beispielsweise könnte ein Performanzproblem mit unterschiedlichen Lösungen behoben werden.

2) Kombination ähnlicher Umsetzungswege: Auf der anderen Seite könnten unterschiedliche Änderungsanfragen ähnliche Umsetzungswege haben oder in einem kombinierten Umsetzungsweg erledigt werden. Nehmen wir beispielsweise an, in mehreren Anfragen sei gefordert, dass zusätzliche Daten in einer Datenstruktur erfasst werden können. Dabei kann es sinnvoll sein, die Änderungen an der Datenstruktur in einem gemeinsamen Vorgang umzusetzen, um sich beispielsweise mehrfaches Testen oder mehrfache Inbetriebnahme zu sparen.

3) Erkennung von Umsetzungskonflikten: Umsetzungswege zu unterschiedlichen Änderungsanfragen können in einem Konfliktverhältnis zueinander stehen, so dass eine Abwägung oder ein Kompromiss gefunden werden muss. Beispielsweise können Änderungen dieselben Personen involvieren, so dass diese nicht parallel umgesetzt werden können oder es sind Änderungen an denselben Artefakten vorgesehen, die sich gegenseitig widersprechen oder die eine Grundvoraussetzung einer anderen Änderung aufheben.

4.2. Begleitendes Beispiel

Zur Illustration des Ansatzes betrachten wir als begleitendes Beispiel ein vereinfachtes System eines Internet-Bestelldienstes. Anhand dieses Beispielsystems stellen wir ausgewählte Änderungsanfragen vor und zeigen auf, welche Fragen für die Projektverantwortlichen auftreten. In späteren Abschnitten zeigen wir, wie unser Verfahren die vorgestellten Beispielsituationen behandelt.

4.2.1. Systembeschreibung

Wie Abbildung 4.3 zeigt, ist der Internet-Bestelldienst als Client-Server-Architektur konzipiert. Konkret bedeutet das, das System besteht aus zwei Client-Anwendungen, einer zentralen Server-Applikation und einem Datenbank-Server. Es gibt zwei Arten von Clients. Zum Einen den Kunden-Client, der auf den Rechnern der Kunden zum Durchsuchen des Katalogs und zur Bestellung von Waren verwendet wird und zum Anderen der Händler-Client,

der auf den Arbeitsplatzrechnern der Händler zur Pflege des Katalogs und zur Abwicklung von Bestellungen eingesetzt wird.

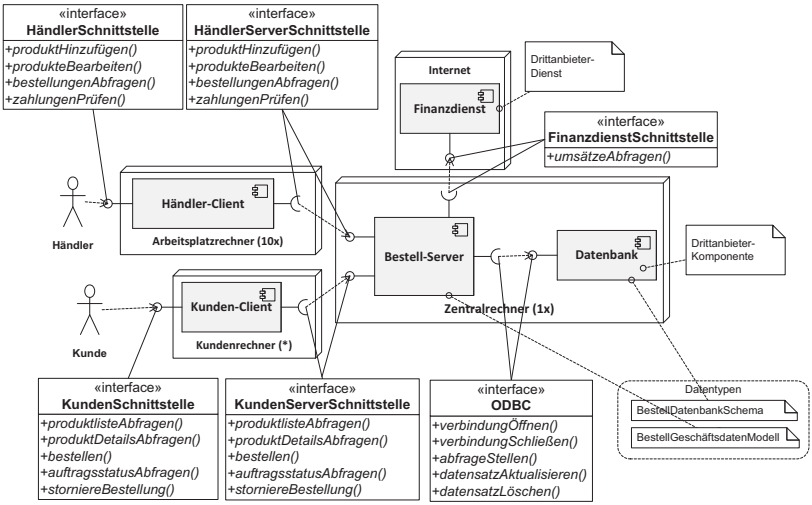


Abbildung 4.3.: Architektur des Internet-Bestell-Systems

Betrachtet man die vier verteilten Anwendungen vereinfacht als Komponenten, so umfasst die strukturelle Architekturbeschreibung vier Komponenten (*Kunden-Client*, *Händler-Client*, *Bestell-Server* und *Datenbank*), wobei jeweils entsprechende Schnittstellen angeboten und nachgefragt werden. Die Komponente *Kunden-Client* bietet eine Benutzerschnittstelle *KundenSchnittstelle* an und fragt die Schnittstelle *KundenServerSchnittstelle* nach. Die Komponente *Händler-Client* bietet eine Benutzerschnittstelle *HändlerSchnittstelle* an und fragt die Schnittstelle *HändlerServerSchnittstelle* nach. Das System verwendet zur Abwicklung von Finanzgeschäften eine Drittanbieter-Komponente *Finanzdienst*, welche die Schnittstelle *FinanzdienstSchnittstelle* anbietet. Die Komponente *Datenbank* bietet die Schnittstelle *ODBC* (engl. Open Database Connectivity) an. Die Komponente *Bestell-Server* bietet die Schnittstellen *KundenServerSchnittstelle* und *HändlerServerSchnittstelle* an und fragt die Schnittstellen *ODBC* und *FinanzdienstSchnittstelle* nach.

Der Kunden-Client wird auf vielen einzelnen Kunden-Rechnern verwendet. Demnach sind von ihm viele Laufzeitinstanzen verteilt auf unterschiedlichen Rechnern im Betrieb. Der Händler-Client läuft auf 10 Firmenrechnern. Die Server-Anwendung und das Datenbank-System laufen auf demselben Rechnerknoten. Der Finanzdienst ist ein externer Dienst, der über das Internet angesprochen wird.

4.2.2. Änderungsanfragen

Im Kontext dieses Beispielsystems treten einige Änderungsanfragen auf, welche wir im Folgenden vorstellen. Auf diese Beispielszenarien wenden wir unseren Ansatz in den nachfolgenden Abschnitten an.

Beispielszenario A: Performanzprobleme bei Datenabfragen

In Beispielszenario A liegen Performanzprobleme bei Datenbankabfragen vor. Die Architekten schlagen folgende Umsetzungswege vor zwischen denen abgewogen werden soll.

Umsetzungsweg A1: Das Datenbankschema wird umstrukturiert, so dass weniger Tabellen-Verknüpfungen (engl. joins) bei Datenbankabfragen berechnet werden.

Umsetzungsweg A2: Ein Zwischenspeicher (engl. cache) wird innerhalb der Server-Anwendung eingeführt, der die letzten Abfrageergebnisse vorhält.

Beispielszenario B: Die Kunden-Client-Anwendung soll unabhängiger vom Server werden

Die Kunden-Client-Anwendung kann bislang nur bei bestehender Verbindung zum Server verwendet werden. Die Projektverantwortlichen erörtern deshalb, wie die Kunden-Client-Anwendung so umgestaltet werden kann, dass ohne bestehende Internet-Verbindung der Katalog durchsucht und Bestellungen erfasst werden können. Das würde bedeuten, dass Teile der Geschäftslogik,

die derzeit auf dem Server lokalisiert sind, auf die Client-Seite übertragen werden.

Umsetzungsweg B: Die Kunden-Client-Anwendung wird erweitert und umstrukturiert, um den verbindungslosen Betrieb zu ermöglichen. Geschäftslogik wird vom Server auf den Client übertragen. Die Benutzerschnittstelle wird erweitert.

Beispielszenario C: Evolution einer Drittanbieterkomponente

Die Beispielanwendung verwendet zur Abwicklung von Zahlungen eine Anbindung an einen Finanzdienstleister. Die verwendete Finanzdienst-Schnittstelle verändert sich aufgrund von neuen internationalen Regelungen im Finanzsektor. Die Architekten wollen deshalb die Auswirkungen dieser Evolution auf die strukturelle Architektur untersuchen.

Umsetzungsweg C: Das System wird an die evolvierende Finanzdienst-Schnittstelle angepasst.

Beispielszenario D: Erweiterung der Benutzerschnittstellentechnologie

Aufgrund der zunehmenden Internetnutzung durch Mobilgeräte sind die Projektverantwortlichen gezwungen, für die Kunden-Client-Anwendung eine geeignete mobile Benutzerschnittstelle anzubieten. Mobile Geräte sind oft weniger leistungsfähig als Desktoprechner und besitzen außerdem kleinere Bildschirme. Demnach sollte ein entsprechendes Webangebot überflüssige Skriptausführungen vermeiden, sowie bei der Darstellung die eingeschränkten Bildschirmgrößen beachten. Demnach bestimmen die Projektverantwortlichen folgenden Umsetzungsweg.

Umsetzungsweg D: Die Benutzerschnittstelle der Kunden-Client-Anwendung wird für die mobile Nutzung erweitert. Für mobile Geräte wird eine schlankere Repräsentation des Webangebots mit speziellen Eingabeelementen und angepasster Nutzerführung angeboten.

In allen genannten Szenarien haben die Projektverantwortlichen Abwägungen vorzunehmen und Entscheidungen zu treffen. Dazu müssen sie analysieren, welche Auswirkungen aus den jeweiligen Umsetzungswegen resultieren.

4.2.3. Überblick über den Entscheidungsrahmen und die Herausforderungen

In der Regel sind die Projektverantwortlichen in der Lage, vorgeschlagene Umsetzungswege im strukturellen Architekturmodell zu beschreiben. Um von dort allerdings Einblick in die weiteren Auswirkungen zu erhalten und hin zu konkreten Aktivitätsbeschreibungen für die Arbeitsplanung zu kommen, müssen sie sich jedoch noch verschiedenen Herausforderungen stellen.

Strukturelle Änderungsausbreitung

Zunächst muss sicher gestellt werden, dass die Änderungsbeschreibung im Architekturmodell die Ausbreitung der Änderung innerhalb der Komponentenstruktur berücksichtigt. Beispielsweise breitet sich eine Veränderung am Datenbankschema über die angebotene Datenbankschnittstelle auf die Server-Applikation aus, falls dort das Datenbankschema innerhalb von SQL-Anfragen genutzt wird. Um solche Auswirkungen explizit zu erfassen, sieht unser Verfahren eine Änderungsausbreitungsanalyse vor.

Organisatorische Konsequenzen

Ist die strukturelle Änderung im Architekturmodell erfasst, stellt sich die Frage nach den organisatorischen Konsequenzen für die Umsetzung. Beispielsweise wollen die Projektverantwortlichen wissen, welche Arbeitspakete und Aufgaben für die einzelnen Tätigkeitsbereiche resultieren, wie viele Personengruppen oder Personen an der Änderungsumsetzung beteiligt werden müssen und welche weiteren Aufwände und Kosten für die Umsetzung eingeplant werden müssen.

Hierbei müssen sie insbesondere bei einem System, das bereits in Betrieb ist, darauf achten, dass eine Aktualisierung von Systemteilen auf vielen Laufzeitinstanzen nur begrenzt häufig durchgeführt werden sollte. Beispielsweise führt eine Modifikation der Client-Anwendung unseres Internet-Bestelldienstes zu einer Aktualisierung vieler Laufzeitinstanzen, was mit Unannehmlichkeiten für die Kunden verbunden sein kann. Unser Verfahren ermittelt explizit Auswirkungen auf verschiedene Tätigkeitsfelder.

4.3. Gesamtüberblick

Bevor wir die einzelnen Teilbeiträge des Verfahrens betrachten, geben wir einen Gesamtüberblick. Wie man der Abbildung 4.4 entnehmen kann, unterscheidet unser Verfahren eine Vorbereitungsphase und die Änderungsanfragenanalyse.

Als Eingaben in die *Vorbereitungsphase* dient ein bestehendes System. Außerdem werden für die Vorbereitungsphase Kontextinformationen zur Quelltextentsprechung und zu Technologiekonzepten, zur Baukonfiguration, zu Testfällen, zur Bereitstellung, zur Inbetriebnahme und zur Personalzuordnung benötigt. Die Phase besteht aus zwei Schritten in denen der Anwender das Architekturmodell erstellt und mit Kontextinformationen anreichert. Aus der Vorbereitungsphase resultiert ein Architekturmodell und ein dazugehöriges Anreicherungsmodell mit den Kontextinformationen.

Die *Änderungsanfragenanalyse* übernimmt als Eingabe das modellierte und angereicherte Architekturmodell aus den Vorbereitungsschritten, sowie eine Menge von Änderungsanfragen. Die Analyse befasst sich zunächst einzeln mit jeder Änderungsanfrage und berechnet jeweils einen Arbeitsplan pro Änderungsanfrage. Die Ergebnisse der Änderungsanfragenanalyse sind Zielarchitekturmodelle und Aktivitätslisten, welche die Umsetzungsschritte einer Änderungsanfrage bezogen auf verschiedene Tätigkeitsfelder beschreiben.

Das weitere Kapitel ist wie folgt aufgebaut. Abschnitt 4.4 beschreibt die Vorbereitungsschritte. In Abschnitt 4.5 gehen wir die einzelnen Schritte der Änderungsanfragenanalyse durch. Abschnitt 4.6 behandelt die Beiträge zur Bewertung und zum Vergleich von Aktivitätslisten.

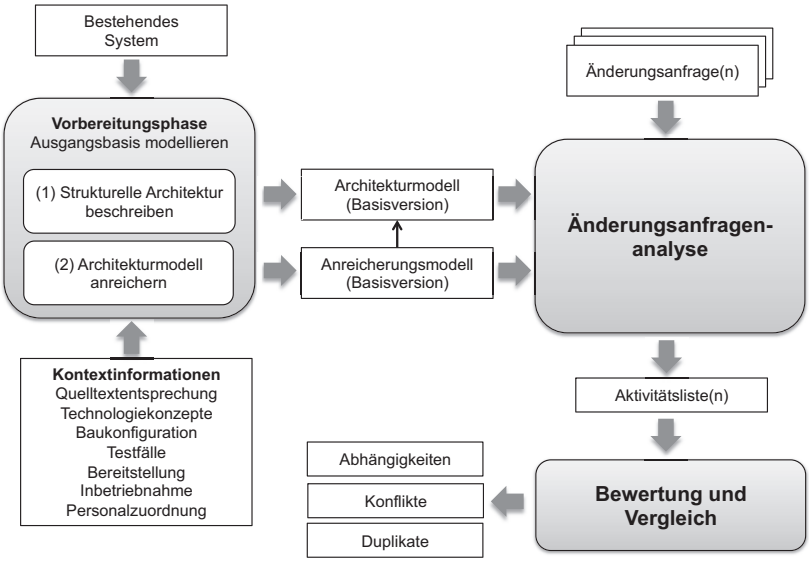


Abbildung 4.4.: Gesamtüberblick des Analyseverfahrens

4.4. Vorbereitungsschritte

Zu den Vorbereitungsschritten zählt die Architekturmodellierung, die in Abschnitt 4.4.1 beschrieben wird, und die Architekturmodellanreicherung, welche wir in Abschnitt 4.4.2 vorstellen.

4.4.1. Architekturmodellierung

Im ersten Vorbereitungsschritt, der *Architekturmodellierung*, beschreibt der Anwender die strukturelle Ausgangsarchitektur in Form eines Architekturmodells. Das Architekturmodell sollte den Ausgangszustand der Architektur vor der Umsetzung der Änderungsanfrage repräsentieren.

Für die Architekturmodellierung verwenden wir ein Architekturmetamodell. Genauer gesagt verwenden wir einen Ausschnitt des Palladio-Komponentenmodells (PCM, Palladio Component Model, [Reu+11]), der im Grundlagenkapitel in Abschnitt 2.2 beschrieben wird, sowie einige Erweiterungen, die wir in Kapitel 5 in Abschnitt 5.1.1.1 erläutern. Die Anwender verwenden für die Architekturmodellierung die Operationen, welche in den Abbildungen 4.1 und 4.2 aufgeführt sind. Eine formale Beschreibung der Operationen befindet sich in Kapitel 5 in Abschnitt 5.2.1.

Kollektions-Datentyp hinzufügen
Kollektions-Datentyp entfernen
Komposit-Datentyp hinzufügen
Komposit-Datentyp entfernen
Datentyp-Parameter zu Komposit-Datentyp hinzufügen
Datentyp-Parameter von Komposit-Datentyp entfernen
Schnittstelle hinzufügen
Schnittstelle entfernen
Operation zu Schnittstelle hinzufügen
Operation von Schnittstellen entfernen
Aufrufparameter zu Schnittstellenoperation hinzufügen
Aufrufparameter von Schnittstellenoperation entfernen
Rückgabeparameter zu Schnittstellenoperation hinzufügen
Rückgabeparameter von Schnittstellenoperation entfernen
Datentyp für Parameter spezifizieren
Basis-Komponente hinzufügen
Basis-Komponente entfernen
Komposit-Komponente hinzufügen
Komposit-Komponente entfernen

Tabelle 4.1.: Architekturmodellierungsoperationen (Teil 1)

Schnittstellenangebot zu Komponente (Basis-Komponente oder Komposit-Komponente) hinzufügen
Schnittstellenangebot von Komponente (Basis-Komponente oder Komposit-Komponente) entfernen
Schnittstellennachfrage zu Komponente (Basis-Komponente oder Komposit-Komponente) hinzufügen
Schnittstellennachfrage von Komponente (Basis-Komponente oder Komposit-Komponente) entfernen
Hinzufügen von Komponenten-Referenz zu Komposit-Komponente
Entfernen von Komponenten-Referenz von Komposit-Komponente
Assemblierungs-Konnektor zur Komposit-Komponente hinzufügen
Assemblierungs-Konnektor von Komposit-Komponente entfernen
Schnittstellenangebotsdelegation zur Komposit-Komponente hinzufügen
Schnittstellenangebotsdelegation von Komposit-Komponente entfernen
Schnittstellennachfragedelegation zur Komposit-Komponente hinzufügen
Schnittstellennachfragedelegation von Komposit-Komponente entfernen
Komponenten-Interne-Abhängigkeit zwischen Schnittstellenangebot und Schnittstellennachfrage zu Komponente hinzufügen
Komponenten-Interne-Abhängigkeit zwischen Schnittstellenangebot und Schnittstellennachfrage von Komponente entfernen

Tabelle 4.2.: Architekturmodellierungsoperationen (Teil 2)

Beispiel:

Die Anwender erstellen für den Internet-Bestelldienst ein Architekturmodell. Dieses enthält die fünf Komponenten für Händler-Client, Kunden-Client, Bestell-Server, Datenbank und Finanzdienst. Sie modellieren für diese Komponenten entsprechende Schnittstellenangebote und -nachfragen. Für jede Schnittstelle geben sie Schnittstellenoperationen an. Weiterhin legen sie für jede Komponente eine Instanz im System-Modell an und verbinden die Instanzen mit Verbindungselementen (Konnektoren). Ein Überblick über die modellierte Architektur für das Beispielsystem ist in Abbildung 4.3 dargestellt.

Modellierung komponenten-interner Abhängigkeitsbeziehungen

Für die Änderungsausbreitungsanalyse ist es notwendig zu wissen, wie sich eine Änderung an einer nachgefragten Schnittstelle einer Komponente innerhalb der Komponente ausbreiten kann und welche angebotenen Schnittstellen dieser Komponente dadurch betroffen sein könnten. Wir legen für die Ausbreitungsmöglichkeit die statischen Abhängigkeiten innerhalb der Komponente zu Grunde. Deshalb müssen die Anwender im Rahmen der Architekturmodellierung für jede Komponente die internen Abhängigkeiten von angebotenen Schnittstellen zu den nachgefragten Schnittstellen spezifizieren. Wir verwenden für die Analyse ein Architekturmodell, das die komponenten-internen Abhängigkeiten zwischen Schnittstellenangeboten und -nachfragen beschreibt. Falls möglich sollten die Abhängigkeiten auf Operationsebene beschrieben werden, d.h. man kann für die angebotenen Schnittstellenoperationen angeben, auf welche nachgefragten Schnittstellenoperationen diese direkt oder indirekt zugreifen.

Beispiel:

In unserem begleitenden Beispiel modelliert der Anwender die internen Abhängigkeiten wie im Komponenten-Diagramm in Abbildung 4.5 dargestellt. Die Abhängigkeiten sind im Diagramm durch gestrichelte Pfeile innerhalb der Komponenten-Box visualisiert. Innerhalb der Bestell-Server-Komponente hängt die angebotene HändlerServerSchnittstelle von der Finanzdienst-Schnittstelle und von der ODBC-Schnittstelle ab. Die angebotene KundenServerSchnittstelle hängt von der ODBC-Schnittstelle ab. Bei der Händler-Client-Komponente und bei der Kunden-Client-Komponente bestehen die internen Abhängigkeiten jeweils direkt zwischen einem Schnittstellenangebot und einer Schnittstellennachfrage. Tabelle 4.3 fasst die komponenteninternen Abhängigkeiten zusammen.

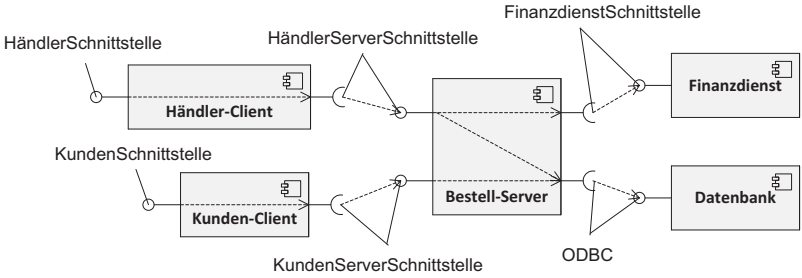


Abbildung 4.5.: Komponenten-Diagramm des Systems, welches zusätzlich die Abhängigkeiten innerhalb der Komponenten durch gestrichelte Pfeile beschreibt.

Komponente	Schnittstellenangebot	Schnittstellennachfrage
Kunden-Client-Komponente	Kunden-Schnittstelle	Kunden-Server-Schnittstelle
Händler-Client-Komponente	Händler-Schnittstelle	Händler-Server-Schnittstelle
Bestell-Server-Komponente	Kunden-Server-Schnittstelle	ODBC-Schnittstelle
Bestell-Server-Komponente	Händler-Server-Schnittstelle	ODBC-Schnittstelle
Bestell-Server-Komponente	Händler-Server-Schnittstelle	Finanzdienst-Schnittstelle

Tabelle 4.3.: Modellierung komponenten-interner Abhängigkeiten

Die Modellierung der internen Abhängigkeiten kann in großen Systemen sehr aufwändig sein. Eine Möglichkeit, um den Aufwand für die Modellierung von internen Abhängigkeitsbeziehungen in Komponenten zu reduzieren, besteht darin, systemweite Standardkonventionen zu definieren. Beispielsweise kann eine „pessimistische Abhängigkeitskonvention“ spezifiziert werden, nach der jede angebotene Schnittstelle von jeder nachgefragten Schnittstelle abhängt oder aber es wird eine „optimistische Abhängigkeitskonvention“ verwendet, nach der grundsätzlich keine Abhängigkeiten von angebotenen zu nachgefragten Schnittstellen angenommen werden. Beide Varianten führen zu Fehlern

bei der Änderungsausbreitungsanalyse in Form von Überschätzungen oder Unterschätzungen von abhängigen Artefakten. Für einzelne komplexere Komponenten sollte man die Abhängigkeiten genauer untersuchen und individuell modellieren.

Eine weitere prinzipielle Möglichkeit, die Modellierung interner Abhängigkeiten zu unterstützen, besteht darin, mit Hilfe einer statischen Quelltextanalyse die statischen Abhängigkeiten innerhalb von Komponenten zu untersuchen und ins Architekturmodell zu übertragen.

Das Architekturmodell und die Anreicherungen können für die Analyse mehrerer Änderungsanfragen verwendet werden. Bei einer großen Anzahl von Änderungsanfragen oder vorwiegend komplexen Änderungsanfragen wird der Modellierungsaufwand zunehmend durch die Wiederverwendung der Daten kompensiert. Auf der anderen Seite müssten die Informationen auch für eine manuelle Analyse bereitgestellt werden.

4.4.2. Architekturmodellanreicherung

Im zweiten Vorbereitungsschritt, der *Architekturmodellanreicherung*, wird das Architekturmodell mit Zusatzinformationen angereichert, die einen Bezug zu den verschiedenen Tätigkeitsfeldern im Software-Entwicklungsumfeld herstellen. Mit den Annotationen werden entscheidungsrelevante Informationen zur Projektplanung im Architekturmodell zugänglich gemacht und sind somit für die automatisierte szenarienbasierte Architekturanalyse und Aktivitätsherleitung verfügbar.

Die berücksichtigten Tätigkeitsfelder sind die Entwicklung (Programmierung im engeren und weiteren Sinn), das Bauen, das Testen, die Bereitstellung und die Inbetriebnahme. Entsprechende Angaben zu den Tätigkeitsfeldern werden an Architekturelemente (z. B. Komponenten, Schnittstellen und Konnektoren) annotiert.

Das Anreicherungsmetamodell wird in Kapitel 5 in Abschnitt 5.1.2 beschrieben. Die formale Beschreibung der Operationen für die Anreicherung findet sich in Kapitel 5 in Abschnitt 5.2.2.

Folgende Anreicherungen werden durch den Anwender bereitgestellt.

Quelltext-Repräsentation Für eine Komponente wird spezifiziert, durch wieviele Quelltextdateien sie implementiert wird. Die Modellannotationen erlauben sowohl die Spezifikation einzelner Dateien (mit Namen), als auch aggregierte Angaben, d.h. die Anzahl der Quelltextdateien. Können Quelltextdateien einzelnen Schnittstellenangeboten zugeordnet werden, so kann dies bei der Anreicherung angegeben werden.

Metadaten-Repräsentation Für eine Komponente wird spezifiziert, wieviele Metadatendateien zu ihr gehören. Das sind zum Beispiel Konfigurationsdateien, Schemadateien, Plugin-Deskriptoren, etc. Die Modellannotationen erlauben sowohl die Spezifikation einzelner Dateien (mit Namen), als auch aggregierte Angaben, d.h. die Anzahl der Metadatendateien. Falls Metadatendateien einzelnen Schnittstellenangeboten zugeordnet werden können, so kann dies bei der Anreicherung angegeben werden.

Baukonfiguration Für eine Komponente wird spezifiziert mit welcher Baukonfiguration sie gebaut wird. Die Baukonfiguration kann in Form eines Dateinamens, eines Dateipfades oder einer Netzwerk-Adresse angegeben werden.

Testfälle Für die Schnittstellenangebote einer Komponente spezifiziert der Anwender die Anzahl der Unit-Tests. Für Benutzerschnittstellenangebote spezifiziert der Anwender die Anzahl der Akzeptanztests. Für die Komponentenverbindungen (Konnektoren) spezifiziert der Anwender die Anzahl der Integrationstests.

Bereitstellungskonfiguration Für eine Komponente spezifiziert der Anwender die Bereitstellungskonfiguration in Form eines Dateipfades.

Inbetriebnahmekonfiguration Für eine Komponente spezifiziert der Anwender die Inbetriebnahmekonfiguration in Form eines Dateinamens.

Laufzeitinstanzen Für eine Komponente sollte im Modell spezifiziert werden, wieviele Laufzeitinstanzen von ihr im Betrieb sind (oder im Betrieb sein werden). Diese Information kann entweder über die Inbetriebnahme-Sicht (engl. deployment view) in Form einzelner Laufzeitinstanzelemente spezifiziert werden oder über eine aggregierte Annotation der Anzahl der Laufzeitinstanzen an die Komponente.

Technologie-Spezifikation Für Komponenten und Schnittstellenangebote spezifiziert der Anwender, welchen Technologiekonzepten sie entsprechen. Komponenten entsprechen aus Technologiesicht bspw. Plug-ins, Paketen oder Klassen. Schnittstellenangebote entsprechen zum Beispiel öffentlichen Klassen und deren öffentlichen Methoden, Schnittstellen-Konzepten der Programmiersprache oder Erweiterungsstellen (engl. extensions) für Plug-ins. Unser Anreicherungsmodell gibt einige Technologiekonzepte beispielhaft vor, erlaubt aber aufgrund der großen Bandbreite benutzerdefinierte Technologiekonzepte. Es können unterschiedliche Technologiekonzepte auf den verschiedenen Hierarchiestufen spezifiziert werden.

Entwurfsmuster-Rollen Für Komponenten spezifiziert der Anwender Entwurfsmusterrollen, welche eine Komponente einnimmt. Unser Anreicherungsmodell gibt einige architektur-basierte Entwurfsmusterrollen beispielhaft vor, erlaubt aber die Definition eigener Entwurfsmusterrollen.

Personalzuordnung Für Komponenten spezifiziert der Anwender das Personal, das für die Komponente zuständig ist. Unser Anreicherungsmodell erlaubt die Zuordnung von Teams und einzelnen Personen.

Einfluss-Spezifikation (Drittanbieter-Kennzeichnung) Für Komponenten spezifiziert der Anwender in welchem Einflussbereich diese stehen, d.h., ob die Komponente im vorliegenden System entwickelt wird, ob sie eine wiederverwendete Eigenentwicklung (engl. inhouse) ist oder ob es sich um eine Drittanbieterkomponente handelt.

Die Architekturbeschreibung bildet durch Komposit-Komponenten einen hierarchischen Aufbau des Systems. Zur Vereinfachung können Annotationen an Architekturelementen auf einer beliebigen Hierarchieebene angebracht werden. Annotationen auf höheren Hierarchieebenen gelten pauschal für alle untergeordneten Ebenen. Individuelle Annotationen auf unteren Ebenen überschreiben pauschale Regelungen.

Die Anreicherungen für das Internet-Bestell-System werden durch die Anwender aus dem Entwicklungskontext ermittelt, der in Abbildung 4.6 gezeigt wird. Die folgenden Abschnitte beschreiben die Anreicherungen für das Beispielsystem.

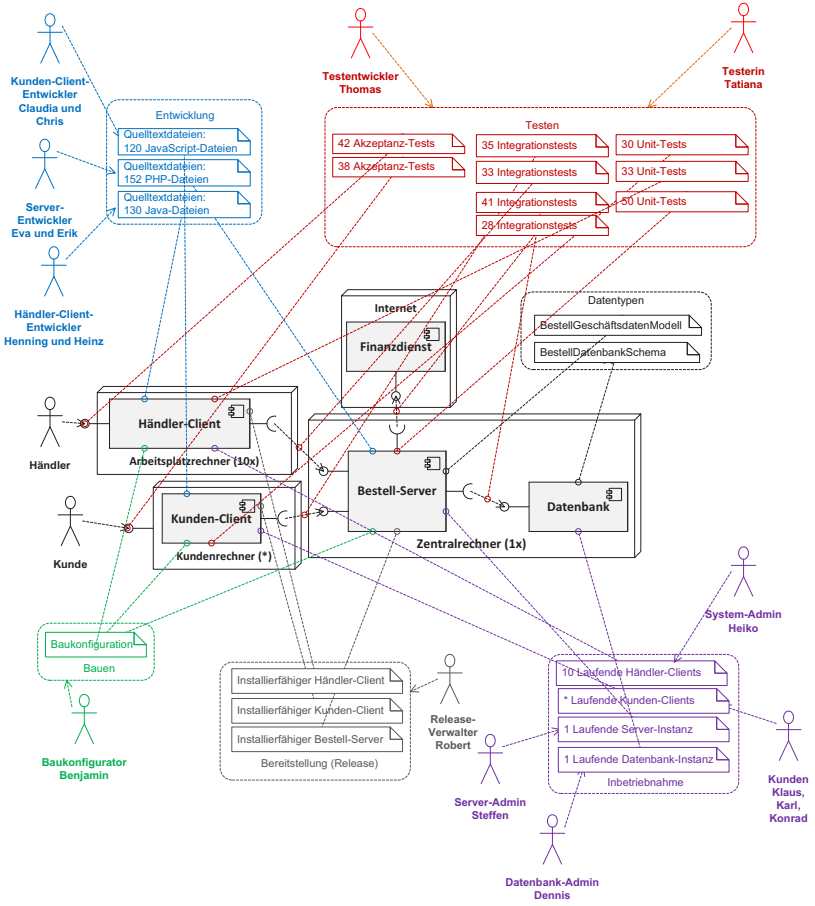


Abbildung 4.6.: Entwicklungsumfeld des Internet-Bestell-Systems

Anreicherungen zu Quelltextdateien und Technologiekonzepten

Die Kunden-Client-Komponente ist mit JavaScript implementiert und läuft in den Internet-Browsern der Kunden. Die Kunden-Client-Komponente ist durch 120 Quelltext-Dateien realisiert. Die Händler-Client-Komponente ist

mit Java implementiert und läuft auf den Arbeitsplatzrechnern der Händler. Die Händler-Client-Komponente ist durch 130 Quelltext-Dateien realisiert. Die Bestell-Server-Komponente ist mit PHP implementiert und läuft in einem Web-Server-Container. Die Bestell-Server-Komponente ist durch 152 Quelltext-Dateien realisiert.

Die Benutzerschnittstelle der Kunden-Client-Komponente ist mit graphischen Interaktionselementen von JavaScript realisiert. Ihre angebotene Schnittstelle des Bestell-Servers ist mit der REST-Technologie umgesetzt.

Die Datenbankkomponente ist eine Drittanbieterkomponente, deren Schnittstelle dem ODBC-Standard entspricht. Der Finanzdienst ist ein über das Internet angebotener Dienst eines Finanzinstituts.

Tabelle 4.4 beschreibt die Zuordnung der Entwicklungsartefakte zur Architektur.

Architektur-zuordnung	**Entwicklungs-artefakt**	**Anzahl**	**Technologiekonzept**
Kunden-Client-Komponente	Quelltext-dateien	120	JavaScript
Händler-Client-Komponente	Quelltext-dateien	130	Java
Bestell-Server-Komponente	Quelltext-dateien	152	PHP
Datenbank-Komponente	Datenbank-schemadateien	1	SQL-DDL

Tabelle 4.4.: Zuordnung von Entwicklungsartefakten zur Architektur

Die Anwender annotieren die Kunden-Client-Komponente mit einer Quelltextdatei-Aggregation und spezifizieren damit eine Anzahl von 120 Quelltextdateien. Die Anwender annotieren die Händler-Client-Komponente mit einer Quelltextdatei-Aggregation und spezifizieren damit eine Anzahl von 130 Quelltextdateien. Die Anwender annotieren die Bestell-Server-Komponente mit einer Quelltextdatei-Aggregation und spezifizieren damit eine Anzahl von 152 Quelltextdateien. Die Programmiersprache als Technologiekonzept wird mit in die Annotation aufgenommen.

Anreicherungen zur Baukonfiguration

Der Bau der Kunden-Client-Komponente, der Händler-Client-Komponente und der Bestell-Server-Komponente wird in einer gemeinsamen Baukonfiguration geregelt. Dementsprechend spezifizieren die Anwender im Anreicherungsmodell eine Baukonfiguration und ordnen sie der Händler-Client-Komponente, der Kunden-Client-Komponente und der Bestell-Server-Komponente zu.

Anreicherungen zu Testfällen

Tabelle 4.5 beschreibt die Zuordnung der Testfälle zur Architektur. Es gibt Unit-Tests, Integrationstests und Akzeptanztests.

Architekturzuordnung	**Testart**	**Anzahl**
Kunden-Client-Komponente	Unit-Tests	30
Händler-Client-Komponente	Unit-Tests	33
Bestell-Server-Komponente	Unit-Tests	50
Konnektor Kunden-Client-Schnittstellennachfrage → Bestell-Server-Schnittstellenangebot	Integrationstests	35
Konnektor Händler-Client-Schnittstellennachfrage → Bestell-Server-Schnittstellenangebot	Integrationstests	33
Konnektor Bestell-Server-Schnittstellennachfrage → Finanzdienst-Schnittstellenangebot	Integrationstests	41
Konnektor Bestell-Server-Schnittstellennachfrage → Datenbank-Schnittstellenangebot	Integrationstests	28
Kunden-Client-Schnittstellenangebot	Akzeptanztests	38
Händler-Client-Schnittstellenangebot	Akzeptanztests	42

Tabelle 4.5.: Architekturbasierte Testfall-Zuordnung

Die Anwender annotieren an der Kunden-Client-Komponente 30 Entwicklungstests (engl. Unit-Tests). Die Anwender annotieren an der Händler-Client-Komponente 33 Entwicklungstests (engl. Unit-Tests). Die Anwender annotieren an der Bestell-Server-Komponente 50 Entwicklungstests (engl. Unit-Tests).

Die Anwender annotieren 35 Integrationstests am Konnektor zwischen der Kunden-Client-Komponente und der Bestell-Server-Komponente. Die Anwender annotieren 33 Integrationstests am Konnektor zwischen der Händler-Client-Komponente und der Bestell-Server-Komponente. Die Anwender annotieren 41 Integrationstests am Konnektor zwischen der Bestell-Server-Komponente und der Datenbank-Komponente.

Die Anwender annotieren 38 Akzeptanztests an der Kunden-Client-Komponente. Die Anwender annotieren 42 Akzeptanztests an der Händler-Client-Komponente.

Anreicherungen zur Bereitstellung

Tabelle 4.6 fasst die Architekturzuordnung von Bereitstellungs-Informationen zusammen.

Architekturzuordnung	**Artefakte**
Kunden-Client	Installationsfähiger Kunden-Client
Händler-Client	Installationsfähiger Händler-Client
Bestell-Server	Installationsfähiger Bestell-Server

Tabelle 4.6.: Bereitstellungs-Informationen

Die Anwender spezifizieren im Anreicherungsmodell für die drei Komponenten (Kunden-Client, Händler-Client, Bestell-Server) jeweils eine Bereitstellungskonfigurations-Annotation, welche die Bezeichnung der installationsfähigen Komponente enthält.

Anreicherungen zur Inbetriebnahme

Die Komponente Kunden-Client ist auf vielen tausend Benutzerrechnern installiert, es gibt demnach entsprechend viele Laufzeitinstanzen dieser Komponente. Die Komponente Händler-Client ist auf 10 Firmenrechnern in Betrieb. Die Komponente Bestell-Server ist auf einem einzigen Server installiert und es gibt demnach nur eine Laufzeitinstanz. Für die Komponente Datenbank gibt es ebenfalls nur eine Laufzeitinstanz auf einem zentralen Datenbankserver.

Tabelle 4.7 fasst die Architekturzuordnung von Inbetriebnahme-Informationen zusammen.

Architektur-zuordnung	**Laufzeitknoten**	**Artefakte**
Kunden-Client-Komponente	Kundenrechner	* Laufzeitinstanzen
Händler-Client-Komponente	Firmenrechner	10 Laufzeitinstanzen
Bestell-Server-Komponente	Zentralrechner	1 Laufzeitinstanz
Datenbank-Komponente	Zentralrechner	1 Laufzeitinstanz

Tabelle 4.7.: Inbetriebnahme-Informationen

Die Anwender spezifizieren im Anreicherungsmodell für die Händler-Client-Komponente eine Installationskonfiguration. Die Anwender spezifizieren im Anreicherungsmodell für die Bestell-Server-Komponente eine Inbetriebnahmekonfiguration.

Beschreibung der Personalzuordnung

Tabelle 4.8 beschreibt die Zuordnung der Personals zur strukturellen Architektur differenziert nach einzelnen Tätigkeitsfeldern.

Architektur-zuordnung	**Tätigkeitsfeld**	**Personen**
Kunden-Client-Komponente	Entwickler	Claudia und Chris
Händler-Client-Komponente	Entwickler	Henning und Heinz
Bestell-Server-Komponente	Entwickler	Eva und Erik
Gesamtsystem	Testentwickler	Thomas
Gesamtsystem	Testerin	Tatiana
Gesamtsystem	Baukonfiguration	Benjamin
Gesamtsystem	Bereitstellung	Release-Verwalter Robert
Kunden-Client-Komponente	Inbetriebnahme	Kunden Klaus, Karl, Konrad, ...
Händler-Client-Komponente	Inbetriebnahme	System-Admin Heiko
Bestell-Server-Komponente	Inbetriebnahme	Server-Admin Steffen
Datenbank-Komponente	Inbetriebnahme	Datenbank-Admin Dennis

Tabelle 4.8.: Personalzuordnung

4.4.3. Mechanismus zur inkrementellen Verfeinerung des Analysemodells und Projektparametrisierung

Die komponenten-basierte Software-Entwicklung ermöglicht es, ein System entweder von den Teilen hin zum Ganzen oder vom Ganzen hin zu den Teilen zu entwickeln. Bei erster Variante werden zunächst die Basiskomponenten entwickelt und diese dann schrittweise zu Komposit-Komponenten kombiniert, um schließlich alle Bausteine zu einem Gesamtsystem zusammenzusetzen (d.h. zu assemblieren). Bei der zweiten Vorgehensweise wird das Gesamtsystem als Einheit entwickelt und je nach Bedarf in weitere Teilsysteme untergliedert. Will man ein System analysieren, so geht man in der Regel auch vom Ganzen hin zu den Teilen. Genauer gesagt nähert man sich aus der

Vogelperspektive und identifiziert zunächst die grobe Aufteilung in Subsysteme und verfeinert diese Beschreibung dann schrittweise nach Abwägung von geforderter Analysegenauigkeit und Modellierungsaufwand und abhängig von der zur Verfügung stehenden Information.

Unser Ansatz unterstützt daher gezielt die schrittweise Verfeinerung des Analysemodells und damit der zur Analyse verwendeten Informationen. Mit einem einfachen Mechanismus im Architekturmodellierungswerkzeug lässt sich das System zunächst vereinfacht durch assemblierte Basiskomponenten beschreiben. Eine Modelltransformation ermöglicht es dann, Basiskomponenten bei Bedarf in Komposit-Komponenten zu verwandeln und die Beschreibung des Innenlebens der Komponente zu verfeinern.

Es gibt in einem Software-System mehrere Abstraktionsebenen. Jede Ebene kann durch eine weitere Verfeinerungsebene innerhalb von Kompositkomponenten beschrieben werden. Auf der obersten Ebene steht das Gesamtsystem. Die jeweiligen unteren Ebenen hängen von der verwendeten Komponententechnologie und allgemein von den darunterliegenden Technologiekonzepten ab. Die angebotenen Schnittstellen einer Komponente geben einen abstrakten Blick auf das Innenleben der Komponente. Beim Verfeinern einer Basiskomponente hin zu einer Kompositkomponente werden Schnittstellen auf die inneren Komponenten übertragen. Mit Bezug auf die Technologieentsprechung von Komponenten und Schnittstellenangeboten kann man mit Hilfe dieser Hierarchisierung und Verfeinerung die unterschiedlichen Ausprägungen von Schnittstellen im Quelltext und in Metadaten widerspiegeln.

In den meisten Software-Projekten gibt es projekt-spezifische Regelungen und Merkmale, die generell für die Interpretation der Architektur genutzt werden können. Diese müssen nur einmal auf System-Ebene spezifiziert werden und gelten dann pauschal für alle untergeordneten Elemente. Dazu zählt zum Beispiel projekt-einheitliche Entwicklungstechnologie. Unterscheidet sich die Technologie zwischen einzelnen Subsystemen, so sollte jeweils auf abstraktester Ebene, auf der die Regelung gilt, die Angabe zur Technologie erfolgen.

Annotationen zu Testfällen sollten auf allen Abstraktionsebenen des Systems vorhanden sein und auch modelliert werden. Die Anzahl der Testfälle kann immer nur pro Komponente beschrieben werden. Eine pauschale Minimalbestimmung von Testfällen könnte man dadurch definieren, dass es pro Komponente jeweils pro angebotener Schnittstellenoperation mindestens

einen Testfall gibt. Unterschiedliche Testarten lassen sich unter Umständen auch einzelnen Abstraktionsebenen zuordnen. Integrationstests beispielsweise sollten erstellt werden, wenn zwei Komponenten über Konnektoren miteinander verbunden werden. Integrationstests sollten aktualisiert und durchgeführt werden, wenn sich die gemeinsame Schnittstelle verändert hat.

Die Baukonfiguration kann in der Regel systemweit definiert werden oder es gibt mehrere jeweils für einzelne Systemteile. In manchen plugin-basierten Systemen wird für kleinere Bausteine eine eigene Baukonfiguration gepflegt.

4.5. Änderungsanfragen-Analyse

In diesem Abschnitt gehen wir die einzelnen Schritte der Änderungsanfragenanalyse durch und illustrieren sie an den Beispielszenarien. Abbildung 4.7 stellt den Ablauf der Änderungsanfragenanalyse dar. Demnach besteht die Analyse aus sechs Teilschritten, die in den nachfolgenden Abschnitten einzeln behandelt werden.

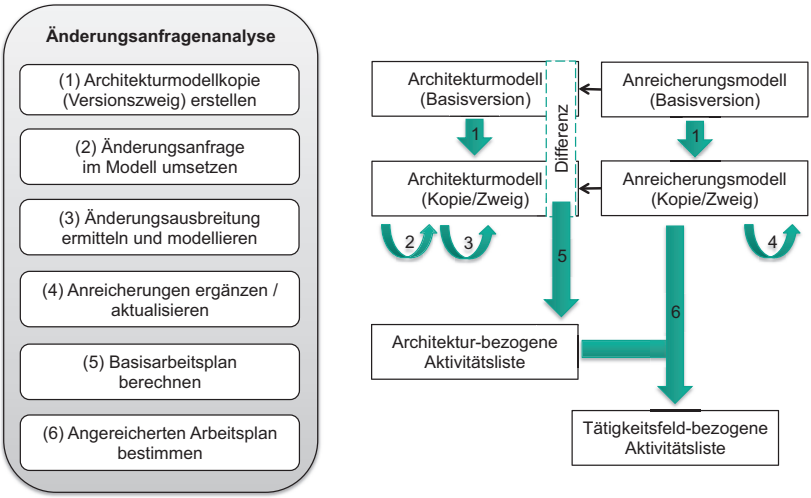


Abbildung 4.7.: Ablauf der Änderungsanfragenanalyse

4.5.1. Versionsbildung

Zu Beginn der Änderungsanfragenanalyse erstellen die Anwender mit Hilfe eines werkzeug-gestützten Versionierungsmechanismus für jede Änderungsanfrage eine Kopie des Ausgangsarchitekturmodells. Es werden sowohl das Architekturmodell als auch das Anreicherungsmodell kopiert. Die Kopie wird in den weiteren Analyseschritten durch Modellveränderungen und Annotationen in die Zielarchitektur überführt. Dieser Mechanismus ist wichtig, um die Veränderungen im Vergleich zur Ausgangsarchitektur automatisiert bestimmen zu können. Die formale Beschreibung des Duplizierungsverfahrens basierend auf den Modellgraphen befindet sich in Kapitel 5 in Abschnitt 5.3.1.

Bezogen auf unser begleitendes Beispiel bedeutet das, dass für jeden der fünf Umsetzungswege, wie in Abbildung 4.8 dargestellt, eine eigene Modellkopie erstellt wird.

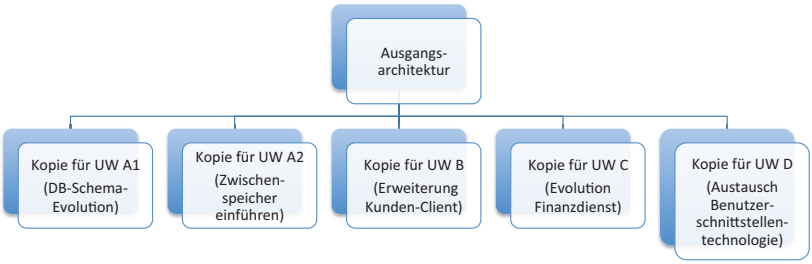


Abbildung 4.8.: Bei der Versionsbildung wird für jeden zu analysierenden Umsetzungsweg (UW) eine Kopie des Ausgangsarchitekturmodells erstellt

4.5.2. Umsetzungsweg modellieren

Wie bereits in Abschnitt 4.1 beschrieben, müssen die Projektverantwortlichen entscheiden, wie die Änderungsanfrage umgesetzt werden soll. Das entspricht einem kreativen Entscheidungsprozess, der nicht einfach automatisiert werden kann. Jedoch können Änderungen auf der abstrakten Sichtweise des Architekturmodells durchgespielt werden und dann automatisiert in konkrete Aktivitäten übersetzt werden. Dazu modelliert der Anwender in der

erstellten Kopie die Zielarchitektur für den jeweiligen Umsetzungsweg. Hierfür kann er die üblichen Operationen zur Architekturmodellierung (siehe auch Abschnitt 4.4.1 und Tabelle 4.1) verwenden. Der Anwender sollte den Umsetzungsweg möglichst vollständig und detailliert modellieren.

Modifikationen, die gekapselt in Architekturelementen vorgenommen werden müssen, die jedoch keine Veränderung der architektonischen Struktur vornehmen und auch Signaturen von Schnittstellenoperationen unverändert lassen, sollten ebenfalls erfasst werden. Dazu stellt unser Modellierungswerkzeug spezielle Markierungen für die Kennzeichnung interner Modifikationen bereit. Diese können in drei Detailgraden angebracht werden. Im Idealfall wird die interne Modifikation genau pro angebotener Schnittstellenoperation spezifiziert. Etwas weniger genau ist die Spezifikation an Schnittstellenangeboten einer Komponente. Der Anwender kann interne Modifikationen aber auch pauschal an einer Komponente spezifizieren, ohne sich genau auf ein Schnittstellenangebot oder eine Schnittstellenoperation zu beziehen. Des Weiteren ist eine Kennzeichnung von Modifikationen an Konnektoren möglich, um eine Änderungsausbreitung zwischen Komponenten explizit auszudrücken. Darüber hinaus kann mit einer Modifikationskennzeichnung an Schnittstellennachfragen die Änderungsausbreitung auf die Aufrufstellen von Schnittstellen explizit ausgedrückt werden. Datentypen und Schnittstellen können ebenfalls mit einer Modifikationskennzeichnung versehen werden. Eine Zusammenfassung dieser Operationen ist in Tabelle 4.9 zu finden. Die formale Beschreibung dieser Operationen befindet sich in Kapitel 5 in Abschnitt 5.3.2.

Interne Modifikation an Komponente kennzeichnen
Interne Modifikation an Schnittstellenangebot kennzeichnen
Interne Modifikation an angebotener Operation kennzeichnen
Interne Modifikation an Assemblierungskonnektor kennzeichnen
Interne Modifikation an Schnittstellennachfragen kennzeichnen
Interne Modifikation an Schnittstelle kennzeichnen
Interne Modifikation an Datentyp kennzeichnen

Tabelle 4.9.: Modellierungsoperationen zur Kennzeichnung interner Modifikationen

Die Umsetzungswege für die Beispielszenarien modellieren die Projektverantwortlichen wie folgt.

Umsetzungsweg A1: Datenbankschema optimieren

Der Anwender modelliert den Umsetzungsweg zur Optimierung des Datenbankschemas dadurch, dass er den Datentyp BestellDatenbankSchema, sowie die angebotene Schnittstelle der Datenbank-Komponente jeweils mit einer Modifikationskennzeichnung versieht. Abbildung 4.9 zeigt den modellierten Umsetzungsweg.

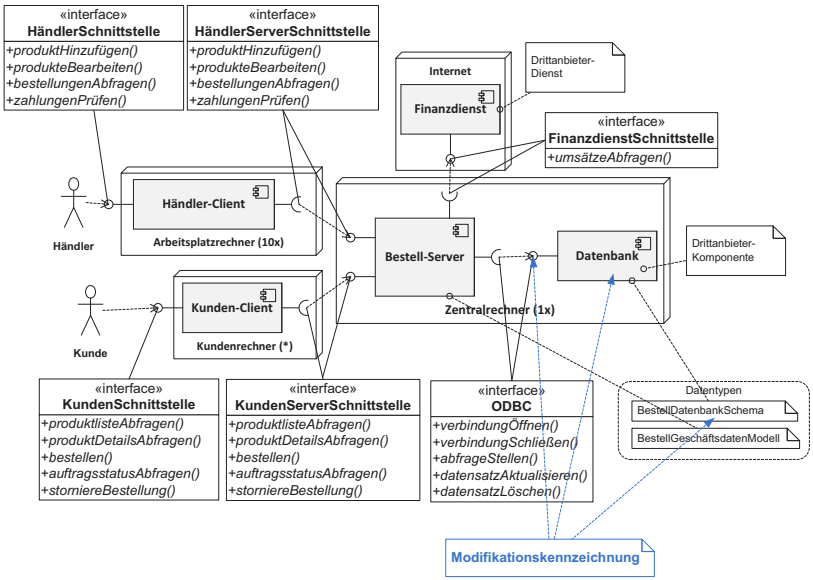


Abbildung 4.9.: Modellierung von Umsetzungsweg A1

Umsetzungsweg A2: Zwischenspeicher einführen

Der Anwender modelliert den Umsetzungsweg zur Einführung des Zwischenspeichers dadurch, dass er eine neue Komponente „Zwischenspeicher“ ins Modell einfügt, die eine eigene Schnittstelle „ZwischenspeicherSchnittstelle“ anbietet. Weiterhin fügt er der Server-Komponente eine Schnittstellennachfrage zu der ZwischenspeicherSchnittstelle hinzu. Er fügt in das Systemmodell eine Instanz der Zwischenspeicher-Komponente ein und verbindet die Server-Komponente und die Zwischenspeicher-Komponente mit einem Konnektor. Abbildung 4.10 illustriert den Umsetzungsweg.

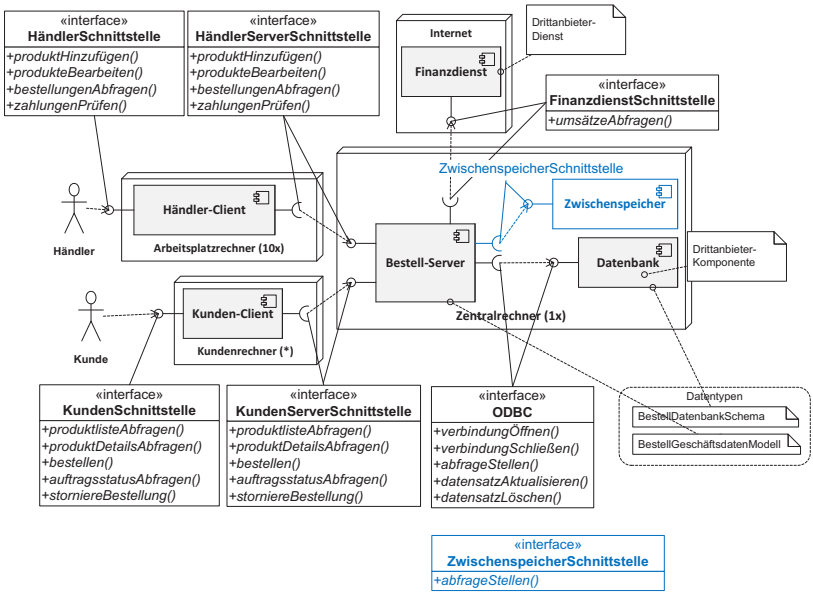


Abbildung 4.10.: Modellierung von Umsetzungsweg A2

Umsetzungsweg B: Erweiterung des Kunden-Clients

Der Anwender modelliert den Umsetzungsweg zur Überführung der Kunden-Client-Anwendung in eine vom Server unabhängigere Anwendung dadurch, dass er die Basiskomponente *Kunden-Client* in eine Komposit-Komponente umgestaltet und in diese mehrere Unterkomponenten einfügt. Funktionalität, die von der Server-Anwendung in den Kunden-Client verschoben wird, wird durch eine interne Modifikationskennzeichnung der Server-Komponente und eine Veränderung der angebotenen ServerSchnittstelle gekennzeichnet. Der Kunden-Client erhält eine zusätzliche Persistenz-Verwaltung, die Kommunikation mit dem Server muss überarbeitet werden und in der Benutzerschnittstelle des Kunden-Clients müssen die zusätzlichen Funktionen bereitgestellt werden. Abbildung 4.11 zeigt den modellierten Umsetzungsweg.

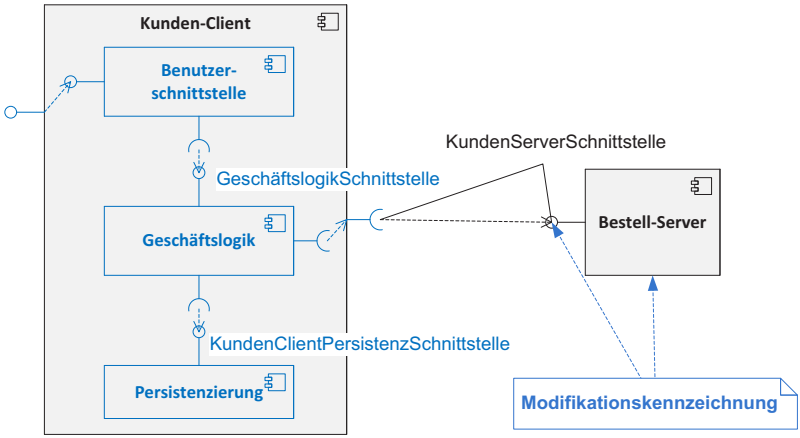


Abbildung 4.11.: Modellierung von Umsetzungsweg B

Umsetzungsweg C: Evolution der Finanzdienst-Schnittstelle

Der Anwender modelliert den Umsetzungsweg zur Evolution der Finanzdienst-Schnittstelle durch eine Modifikations-Kennzeichnung am Schnittstellenangebot der Finanzdienst-Komponente. Abbildung 4.12 zeigt den modellierten Umsetzungsweg.

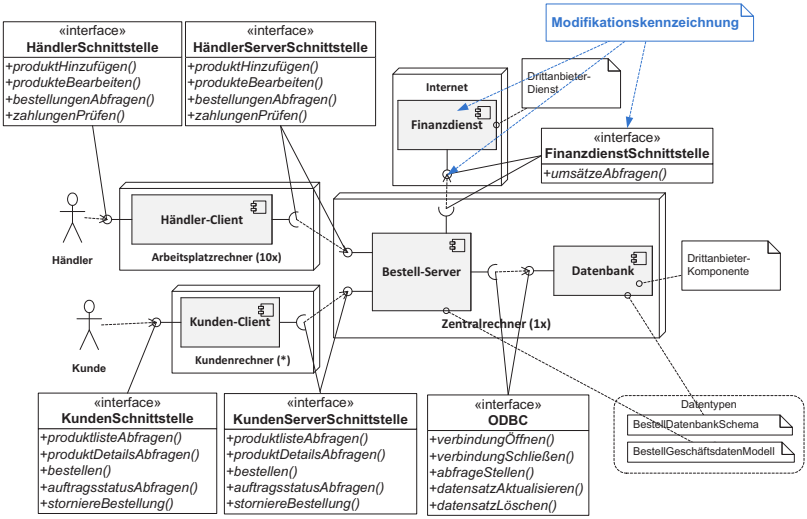


Abbildung 4.12.: Modellierung von Umsetzungsweg C

Umsetzungsweg D: Austausch der Benutzerschnittstellen-Technologie

Der Anwender modelliert den Umsetzungsweg zum Austausch der Benutzerschnittstellen-Technologie des Kunden-Clients dadurch, dass er eine Modifikations-Kennzeichnung am Schnittstellenangebot der Kunden-Client-Komponente annotiert. Abbildung 4.12 zeigt den modellierten Umsetzungsweg.

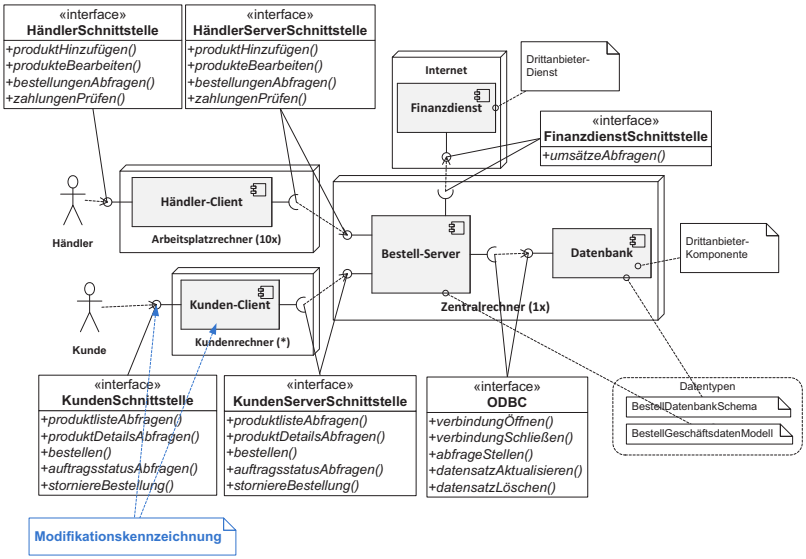


Abbildung 4.13.: Modellierung von Umsetzungsweg D

4.5.3. Änderungsausbreitungsanalyse

Im nächsten Schritt geht es darum, die Ausbreitung der bereits modellierten Änderungen zu analysieren und die Beschreibung des Umsetzungswegs im Architekturmodell zu erweitern, so dass Folgeänderungen in angrenzenden Systembereichen miteinbezogen werden. Dafür stellt das Verfahren eine einfache Vorgehensweise für die Änderungsausbreitungsanalyse bereit.

Horizontale Ausbreitung

Die Grundlage bietet das Architekturmodell und die darin beschriebenen Verbindungen zwischen den Komponenten, die Konnektoren. Die Konnektoren beschreiben, wie die Komponenten untereinander verbunden sind. Die Konnektoren verlaufen im Modell von den Schnittstellennachfragen einer

Komponente zu dem genutzten Schnittstellenangebot einer anderen Komponente. Eine Änderung innerhalb eines Schnittstellenangebots kann zu einer Nachfolgeänderung in den nachfragenden Komponenten führen.

Bei der Untersuchung der Änderungsausbreitung werden zwei Aspekte berücksichtigt. Erstens wie sich die Änderung zwischen den Komponenten ausbreitet, d.h. von einer Komponente auf die nutzenden Komponenten. Wir nennen diesen Aspekt *Inter-Komponenten-Ausbreitung*. Diese Ausbreitung trifft auf die Nutzungsstellen der Schnittstelle in den abhängigen Komponenten. Der zweite Aspekt, die *Intra-Komponenten-Ausbreitung*, befasst sich damit, wie sich diese Änderung der Nutzungsstellen innerhalb der Komponente ausbreitet. Wir untersuchen demnach, wie sich die Änderung an der Schnittstellennachfrage durch die Komponente hindurch propagiert und die angebotene Schnittstelle beeinflusst.

Der Startpunkt für die Inter-Komponenten-Ausbreitung ist eine Menge von modifizierten Schnittstellenangeboten. Dazu zählen zum einen Schnittstellenangebote, die strukturell verändert wurden oder mit internen Modifikationskennzeichnungen versehen sind, und zum anderen Schnittstellenangebote, die hinzugefügt oder entfernt wurden. Der Startpunkt für die Intra-Komponenten-Ausbreitung ist eine Menge von modifizierten Schnittstellennachfragen oder Konnektoren.

Unser Verfahren analysiert iterativ die Intra-Komponenten-Ausbreitung und die Inter-Komponenten-Ausbreitung. Der Anwender kann in das Ergebnis jederzeit eingreifen und Ausbreitungen manuell ausschließen.

Vertikale Ausbreitung

Bei geschachtelten Architekturen mit Komposit-Komponenten breitet sich eine Änderung vertikal von einer Ebene auf die andere aus. Diesen Effekt beachten wir ebenfalls bei der Änderungsausbreitungsanalyse. Dabei gilt eine übergeordnete Komponente als geändert falls eine Änderung einer Unterkomponente vorliegt. Umgekehrt muss beim Eintritt einer Änderung in eine Komposit-Komponente differenziert werden, zu welchen Unterkomponenten die Änderung delegiert wird. Es kann beispielsweise nur eine Teilmenge der Unterkomponenten betroffen sein. Unser Verfahren nutzt die modellierten

statischen Abhängigkeitsbeziehungen in Form von Delegationskonnektoren zur Zuordnung der Ausbreitung auf Unterkomponenten.

Die ausführliche Verfahrensbeschreibung der Änderungsausbreitungsanalyse im Pseudocode findet sich in Kapitel 5 im Abschnitt 5.3.3.

Betrachten wir die Änderungsausbreitungsanalysen für unsere Beispielszenarien, welche in Tabelle 4.10 zusammengefasst werden. Für einzelne Szenarien erläutern wir die Änderungsausbreitungsanalyse genauer.

A1:	In zwei Iterationen ermittelt die Änderungsausbreitungsanalyse mehrere betroffene Systemteile.
A2:	Mit der Änderungsausbreitungsanalyse kann die Ausbreitung der Änderung auf den Bestell-Server eingegrenzt werden.
B:	Die Änderungsausbreitungsanalyse findet keine Propagierung ausgehend vom umstrukturierten Kunden-Client, allerdings wird aufgrund der Bestell-Server-Modifikation der Assemblierungskonnektor beeinflusst.
C:	Die Änderungsausbreitungsanalyse bestimmt von der modifizierten Finanzdienstschnittstelle ausgehend die Ausbreitung auf Bestell-Server und auf den Händler-Client. Eine Ausbreitung auf den Kunden-Client kann den modellierten komponenten-internen Abhängigkeitsbeziehungen folgend automatisiert ausgeschlossen werden.
D:	Vom Kunden-Client ausgehend findet keine weitere Propagierung statt.

Tabelle 4.10.: Zusammenfassung der Änderungsausbreitungsanalyse für die Beispielszenarien

Änderungsausbreitung für Umsetzungsweg A1

Tabelle 4.11 zeigt die Modifikationen, die mit Hilfe der Änderungsausbreitungsanalyse in zwei Iterationen gefunden werden. Zu Beginn der Änderungsausbreitungsanalyse besteht in diesem Fall eine Modifikation des Datentyps BestellDatenbankSchema, sowie des ODBC-Schnittstellenangebots der Datenbank-Komponente. Das Zeichen $\equiv$ kennzeichnet jeweils eine Verfeinerung der übergeordneten Änderungszeile.

In der ersten Iteration führt das Verfahren eine Inter-Komponenten-Ausbreitungsanalyse durch und ermittelt, welche Konnektoren auf das ODBC-Schnittstellenangebot gerichtet sind. Über den Konnektor ermittelt es die ODBC-Schnittstellennachfrage der Bestell-Server-Komponente. Die Intra-Komponenten-Ausbreitungsanalyse ermittelt wie sich die Änderung der ODBC-Schnittstellennachfrage innerhalb der Bestell-Server-Komponente fortpflanzt. Sie nutzt dazu die modellierten komponenten-internen Abhängigkeiten. Demnach ermittelt das Verfahren die abhängigen Schnittstellenangebote HändlerServerSchnittstelle und KundenServerSchnittstelle.

In der zweiten Iteration bestimmt die Analyse die über Konnektoren verbundenen Client-Komponenten und deren jeweilige Schnittstellennachfragen. Schließlich führt die Intra-Komponenten-Ausbreitung bis zu den Benutzerschnittstellen HändlerSchnittstelle und KundenSchnittstelle.

Man mag an dieser Stelle einwenden, dass die Benutzerschnittstelle selbst vielleicht nicht durch das geänderte Datenbankschema beeinflusst wird, aber deren „gekapselte“ Implementierung schon, insofern dort Datenbankrelationen und -attribute verwendet werden. Daher kann diese Modifikation sehr wohl mit der Benutzerschnittstelle im Architekturmodell assoziiert werden.

Modellierter Umsetzungsweg: Modifiziere Datentyp BestellDatenbankSchema Modifiziere Komponente Datenbank ≡ Modifiziere Schnittstellenangebot ODBC
1. Iteration der Änderungsausbreitungsanalyse: **Inter-Komponenten-Ausbreitung:** Modifiziere Konnektor Bestell-Server → Datenbank ≡ Modifiziere Schnittstellennachfrage ODBC **Intra-Komponenten-Ausbreitung:** Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle
2. Iteration der Änderungsausbreitungsanalyse: **Inter-Komponenten-Ausbreitung:** Modifiziere Konnektor Kunden-Client → Bestell-Server ≡ Modifiziere Schnittstellennachfrage KundenServerSchnittstelle Modifiziere Konnektor Händler-Client → Bestell-Server ≡ Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle **Intra-Komponenten-Ausbreitung:** Modifiziere Komponente Kunden-Client ≡ Modifiziere Schnittstellenangebot KundenSchnittstelle Modifiziere Komponente Händler-Client ≡ Modifiziere Schnittstellenangebot HändlerSchnittstelle

Tabelle 4.11.: Aktivitäten von Umsetzungsweg A1 nach Änderungsausbreitungsanalyse

Änderungsausbreitung für Umsetzungsweg A2

Tabelle 4.12 zeigt Analyseergebnisse für Umsetzungsweg A2. Das Hinzufügen der Zwischenspeicher-Komponente und deren Nutzung im Bestell-Server wird durch die Änderungsausbreitungsanalyse mit den implementierten Schnittstellenangeboten des Bestell-Servers assoziiert. Die Aktivitäten zur Implementierung der Schnittstellennachfrage werden dadurch zu Teilaktivitäten oder auch Verfeinerungen der Modifikationen des Schnittstellenangebots.

Modellierter Umsetzungsweg: Implementiere Komponente Zwischenspeicher ≡ Implementiere Schnittstellenangebot ZwischenspeicherSchnittstelle ≡ Implementiere Schnittstellenoperation abfrageStellen() Modifiziere Komponente Bestell-Server ≡ Implementiere Schnittstellennachfrage zur ZwischenspeicherSchnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() Implementiere Konnektor Bestell-Server → Zwischenspeicher
1. Iteration der Änderungsausbreitungsanalyse: **Intra-Komponenten-Ausbreitung:** Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur ZwischenspeicherSchnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur ZwischenspeicherSchnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen()

Tabelle 4.12.: Aktivitäten von Umsetzungsweg A2 nach Änderungsausbreitungsanalyse

Änderungsausbreitung für Umsetzungsweg B

Tabelle 4.13 beschreibt den modellierten Umsetzungsweg und die ermittelte Ausbreitung für Umsetzungsweg B. In Szenario B ermittelt die Änderungsausbreitungsanalyse den Zusammenhang zwischen den zwei modellierten Teiländerungen in Form einer Modifikationskennzeichnung am Konnektor zwischen Kunden-Client und Bestell-Server.

Modellierter Umsetzungsweg:
Modifiziere Komponente Kunden-Client ≡ Implementiere Sub-Komponente Benutzerschnittstelle ≡ Implementiere Schnittstellenangebot Benutzerschnittstelle ≡ Implementiere Sub-Komponente Geschäftslogik ≡ Implementiere Schnittstellenangebot GeschäftslogikSchnittstelle ≡ Implementiere Sub-Komponente Persistenzierung ≡ Implementiere Schnittstellenangebot KundenClientPersistenzSchnittstelle Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle
1. Iteration der Änderungsausbreitungsanalyse: **Inter-Komponenten-Ausbreitung:** Modifiziere Konnektor Kunden-Client → Bestell-Server

Tabelle 4.13.: Aktivitäten von Umsetzungsweg B nach Änderungsausbreitungsanalyse

Änderungsausbreitung für Umsetzungsweg C

Tabelle 4.14 beschreibt den modellierten Umsetzungsweg und die ermittelte Ausbreitung für Umsetzungsweg C. Die Änderungsausbreitungsanalyse bestimmt von der modifizierten Finanzdienstschnittstelle die Ausbreitung auf Bestell-Server und auf den Händler-Client. Eine Ausbreitung auf den Kunden-Client kann dank der komponenten-internen Abhängigkeitsbeziehungen automatisiert ausgeschlossen werden.

Modellierter Umsetzungsweg: Modifiziere Schnittstelle FinanzdienstSchnittstelle Modifiziere Komponente Finanzdienst ≡ Modifiziere Schnittstellenangebot FinanzdienstSchnittstelle
1. Iteration der Änderungsausbreitungsanalyse: **Inter-Komponenten-Ausbreitung:** Modifiziere Konnektor Bestell-Server → Finanzdienst → Modifiziere Schnittstellennachfrage zur FinanzdienstSchnittstelle
Intra-Komponenten-Ausbreitung: Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle ≡ Modifiziere Schnittstellennachfrage zur FinanzdienstSchnittstelle
2. Iteration der Änderungsausbreitungsanalyse: **Inter-Komponenten-Ausbreitung:** Modifiziere Konnektor Händler-Client → Bestell-Server → Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle **Intra-Komponenten-Ausbreitung:** Modifiziere Komponente Händler-Client ≡ Modifiziere Schnittstellenangebot HändlerSchnittstelle ≡ Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle

Tabelle 4.14.: Aktivitäten von Umsetzungsweg C nach Änderungsausbreitungsanalyse

Änderungsausbreitung für Umsetzungsweg D

Tabelle 4.15 beschreibt den modellierten Umsetzungsweg nach Änderungsausbreitungsanalyse für Umsetzungsweg D. Demnach kann keine Ausbreitung identifiziert werden.

Modellierter Umsetzungsweg:
Modifiziere Komponente Kunden-Client
≡ Modifiziere Schnittstellenangebot KundenSchnittstelle
1. Iteration der Änderungsausbreitungsanalyse:
–

Tabelle 4.15.: Aktivitäten von Umsetzungsweg D nach Änderungsausbreitungsanalyse

4.5.4. Anreicherungen ergänzen und aktualisieren

Die Umsetzungswege inklusive der Änderungsausbreitung sind jetzt im Zielarchitekturmodell beschrieben. Betrachten wir jetzt die dazugehörigen Anreicherungsmodelle. Durch die Modellierung des Umsetzungswegs kommen zum Teil neue Elemente zum Architekturmodell hinzu, für die es noch keine Anreicherungen gibt. Diese müssen entsprechend ergänzt werden. Im modellierten Umsetzungsweg können zudem Architekturelemente wegfallen, so dass Anreicherungen ihren Bezugspunkt in der Architektur verlieren. Man muss demnach entscheiden, wie man mit diesen „freigesetzten“ Anreicherungen umgeht. Folglich müssen, bevor man mit der Aktivitätsherleitung beginnen kann, die Anreicherungen ergänzt und aktualisiert werden. Die Anreicherungsergänzungen und -aktualisierungen für die Beispielszenarien werden in Tabelle 4.16 zusammengefasst.

A1:	Es sind keine Ergänzungen oder Aktualisierungen notwendig, da nur Modifikationskennzeichnungen vorgenommen wurden.
A2:	Für die eingefügte Zwischenspeicher-Komponente müssen Anreicherungen ergänzt werden.
B:	Für die hinzugefügten Teilkomponenten der Kunden-Client-Komponente müssen die Anwender Anreicherungen ergänzen.
C:	Es sind keine Ergänzungen oder Aktualisierungen notwendig.
D:	Es sind keine Ergänzungen oder Aktualisierungen notwendig.

Tabelle 4.16.: Anreicherungsanpassungen für die Beispielszenarien

Automatisierte Vorschläge für Anreicherungsergänzungen

Für manche hinzugekommenen Architekturelemente lassen sich bestimmte Anreicherungen automatisiert aus dem Kontext ableiten. Unser Verfahren schlägt dem Anwender für solche Fälle zu ergänzende Anreicherungen vor. Tabelle 4.17 fasst die Vorschläge zusammen.

Vorschlag:	Anzahl und Art der zu entwickelnden Testfälle aus der Größe der Schnittstelle bzw. der Art des hinzugekommenen Elements (Benutzerschnittstellenangebot → Akzeptanztest, Schnittstellenangebot → Unittest, Assemblierungskonnektor → Integrationstest)
Vorschlag:	Eintragung in bestehende Baukonfiguration
Vorschlag:	Technologiekonzepte von anderen Komponenten derselben Hierarchiestufe übernehmen
Vorschlag:	Personalzuordnung von umgebender System- oder Komposit-Komponente übernehmen

Tabelle 4.17.: Vorschläge für automatisierte Anreicherungen

Im Falle unvollständiger Anreicherungen, die nicht automatisiert ermittelt werden können, weist das Verfahren den Anwender darauf hin, so dass dieser manuell eingreifen kann.

Automatisierte Behandlung ungültiger Anreicherungen

Werden Anreicherungen durch wegfallende Architekturmodellelemente bezugslos, so können diese mit den in Tabelle 4.18 genannten Strategien automatisiert behandelt werden.

Strategie 1:	Automatisches Entfernen ungültiger Anreicherungen
Strategie 2:	Vorhalten ungültiger Anreicherungen zur Übertragung auf andere Architekturelemente und Zuordnungsvorschläge über Komponenten-Hierarchie ermitteln.

Tabelle 4.18.: Automatisierte Behandlung bezugsloser Anreicherungen

4.5.5. Differenzen berechnen und architektur-bezogene Aktivitäten ermitteln

Hat der Anwender die Umsetzungswege modelliert, die Änderungsausbreitung bestimmt, sowie die Anreicherungen ergänzt und aktualisiert, so führt das Analysewerkzeug eine Differenzberechnung durch. Die Differenzberechnung ermittelt die Modifikationen, die der Anwender im Zielarchitekturmodell vorgenommen hat, sowie die Modifikationskennzeichnungen, die am Modell angebracht wurden. Aus diesen Unterschieden und Kennzeichnungen wird eine Menge von architektur-bezogenen Aktivitäten abgeleitet. Diese Aktivitäten beziehen sich auf Architekturelemente, welche hinzugefügt oder entfernt wurden, oder, deren Attribute oder Referenzen verändert wurden oder welche intern modifiziert werden sollen, was durch die Modifikationskennzeichnungen ausgedrückt wird.

Die Differenzberechnung liefert eine architektur-bezogene Aktivitätsliste, die lediglich die Architekturelemente adressiert ohne sich auf einzelne Tätigkeitsfelder zu beziehen. Eine solche architektur-bezogene Aktivitätsliste ist in Tabelle 4.19 schematisch dargestellt. In dieser schematischen Darstellung sieht man, dass übergeordnete Komponenten-Aktivitäten verfeinert werden durch Aktivitäten, die sich auf die Schnittstellenangebote dieser Komponenten beziehen. Unter Verfeinerung verstehen wir Konkretisierung oder das Gegenteil von Abstraktion. Diese Verfeinerungsbeziehung wird durch das Symbol $\equiv$ angedeutet. Im Beispiel bedeutet das, dass die Modifikation von Komponente „X“ bedeutet, dass von Komponente „X“ das Schnittstellenangebot „I1“ entfernt wird, zur Komponente das Schnittstellenangebot „I2“ hinzugefügt wird und das Schnittstellenangebot „I3“ modifiziert wird. Das Hinzufügen von Komponente „Y“ bedeutet konkret, dass das Schnittstellenangebot „I4“ zu implementieren ist. Die Entfernung der Komponente „Z“ bedeutet, dass die Implementierung des Schnittstellenangebots „I5“ zu entfernen ist.

Modifiziere Komponente „X“ ≡ Entferne Schnittstellenangebot „I1“ ≡ Füge Schnittstellenangebot „I2“ hinzu ≡ Modifiziere Schnittstellenangebot „I3“ Füge Komponente „Y“ hinzu ≡ Füge Schnittstellenangebot „I4“ hinzu Entferne Komponente „Z“ ≡ Entferne Schnittstellenangebot „I5“

Tabelle 4.19.: Schematische Darstellung einer architektur-bezogenen Aktivitätsliste

Die formale Beschreibung der Differenzberechnung und der Herleitung architektur-bezogener Aktivitäten befindet sich in Kapitel 5 in Abschnitt 5.3.5. Betrachten wir nun die Beispielszenarien.

Für Umsetzungsweg A1 wird die architektur-bezogene Aktivitätsliste in Tabelle 4.20 bestimmt.

Modifiziere Datentyp BestellDatenbankSchema Modifiziere Komponente Datenbank ≡ Modifiziere Schnittstellenangebot ODBC
Modifiziere Konnektor Bestell-Server → Datenbank
Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellennachfrage ODBC ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle
Modifiziere Konnektor Kunden-Client → Bestell-Server
Modifiziere Komponente Kunden-Client ≡ Modifiziere Schnittstellennachfrage KundenServerSchnittstelle ≡ Modifiziere Schnittstellenangebot KundenSchnittstelle
Modifiziere Konnektor Händler-Client → Bestell-Server
Modifiziere Komponente Händler-Client ≡ Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle ≡ Modifiziere Schnittstellenangebot HändlerSchnittstelle

Tabelle 4.20.: Architektur-Bezogene Aktivitäten von Umsetzungsweg A1

Für Umsetzungsweg A2 wird die architektur-bezogene Aktivitätsliste in Tabelle 4.21 bestimmt.

Implementiere Komponente Zwischenspeicher ≡ Implementiere Schnittstellenangebot ZwischenspeicherSchnittstelle ≡ Implementiere Schnittstellenoperation abfrageStellen()
Implementiere Konnektor Bestell-Server → Zwischenspeicher
Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur Zwischenspeicher-Schnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur Zwischenspeicher-Schnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen()

Tabelle 4.21.: Architektur-Bezogene Aktivitäten von Umsetzungsweg A2

Für Umsetzungsweg B wird die architektur-bezogene Aktivitätsliste in Tabelle 4.22 bestimmt.

Modifiziere Komponente Kunden-Client ≡ Implementiere Sub-Komponente Benutzerschnittstelle ≡ Implementiere Schnittstellenangebot Benutzerschnittstelle ≡ Implementiere Sub-Komponente Geschäftslogik ≡ Implementiere Schnittstellenangebot GeschäftslogikSchnittstelle ≡ Implementiere Sub-Komponente Persistenzierung ≡ Implementiere Schnittstellenangebot KundenClientPersistenz-Schnittstelle
Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle
Modifiziere Konnektor Kunden-Client → Bestell-Server

Tabelle 4.22.: Architektur-Bezogene Aktivitäten von Umsetzungsweg B

Für Umsetzungsweg C wird die architektur-bezogene Aktivitätsliste in Tabelle 4.23 bestimmt.

Modifiziere Komponente Finanzdienst ≡ Modifiziere Schnittstellenangebot FinanzdienstSchnittstelle
Modifiziere Konnektor Bestell-Server → Finanzdienst → Modifiziere Schnittstellennachfrage zur FinanzdienstSchnittstelle
Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle ≡ Modifiziere Schnittstellennachfrage zur FinanzdienstSchnittstelle
Modifiziere Konnektor Händler-Client → Bestell-Server → Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle
Modifiziere Komponente Händler-Client ≡ Modifiziere Schnittstellenangebot HändlerSchnittstelle ≡ Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle

Tabelle 4.23.: Architektur-Bezogene Aktivitäten von Umsetzungsweg C

Für Umsetzungsweg D wird die architektur-bezogene Aktivitätsliste in Tabelle 4.24 bestimmt.

Modifiziere Komponente Kunden-Client ≡ Modifiziere Schnittstellenangebot KundenSchnittstelle

Tabelle 4.24.: Architektur-Bezogene Aktivitäten von Umsetzungsweg D

4.5.6. Ableitung von Entwicklungs- und Folgetätigkeiten

Basierend auf den architektur-bezogenen Aktivitäten werden im nächsten Schritt Entwicklungs- und Folgeaktivitäten abgeleitet. *Entwicklungstätigkeiten* sind Aktivitäten, welche eine strukturelle Veränderung der Architektur bewirken oder welche Architekturelemente intern modifizieren. Zu den Entwicklungstätigkeiten gehört die *Programmierung im engeren Sinn* (d.h. die Bearbeitung von Quelltextdateien), sowie die *Programmierung im weiteren Sinn* (d.h. die Bearbeitung von Metadaten-Dateien, wie z.B. Plug-in-Deskriptoren, Datenbankschema-Spezifikationen, etc.).

Folgeaktivitäten sind Aktivitäten, welche im Anschluss an Entwicklungstätigkeiten notwendig sind, um wieder ein lauffähiges (oder laufendes) System zu erhalten. Zu den Folgeaktivitäten gehören Aktivitäten zum Bauen, zum Testen, zur Bereitstellung und zur Inbetriebnahme der Software. Folgeaktivitäten ergeben sich als Konsequenz aus Entwicklungstätigkeiten. Der konkrete Zusammenhang kann in Regeln ausgedrückt werden. Die Regeln drücken aus, wann welche Folgeaktivität aus einer Entwicklungstätigkeit resultiert. Mit Hilfe der Regeln ist es möglich, Folgeaktivitäten automatisiert aus Entwicklungstätigkeiten abzuleiten.

Beispielsweise gilt, wenn Quelltext verändert wird ist im Anschluss daran ein Bau notwendig. Falls Testfälle existieren müssen diese ausgeführt werden, um die Funktionsfähigkeit sicherzustellen. Wenn zum Beispiel Komponenten zum System hinzukommen, müssen diese in der Baukonfiguration erfasst werden, außerdem wird die Entwicklung neuer Testfälle notwendig.

Das Verfahren leitet mit Hilfe von Regeln und unter Berücksichtigung der Anreicherungen Aktivitäten zu den Tätigkeitsfeldern ab und verfeinert die Aktivitätsliste. Tabelle 4.25 zeigt eine angereicherte Aktivitätsliste. Die Einträge in der Aktivitätsliste konkretisieren, wie die Komponente modifiziert werden soll, d.h. ob die Bearbeitung von Quelltextdateien, Metadaten-Dateien oder Konfigurationsdateien erforderlich wird. Im Beispiel in Tabelle 4.25 sieht man beispielsweise, dass der Quelltext von Komponente X und Metadaten-Dateien zu bearbeiten sind. Außerdem wird ersichtlich, dass in der Folge die Komponente X anschließend neu gebaut werden muss und die erneute Inbetriebnahme dreier Instanzen durchzuführen ist.

Das Symbol $\equiv$ kam bereits im letzten Abschnitt vor. Es kennzeichnet äquivalente Aktivitäten, d.h. Konkretisierung oder Verfeinerungen einer Aktivität im Sinne von Teilaktivitäten. Als Verfeinerung verstehen wir das Gegenteil von Abstraktion. Das Pfeil-Symbol $\rightarrow$ hingegen kennzeichnet durch Regeln abgeleitete Folgeaktivitäten.

Modifiziere Komponente “X”
≡ Entferne Schnittstellenangebot “IP1”
≡ Modifiziere Quelltextdateien
≡ Implementiere Schnittstellenangebot “IP2”
≡ Modifiziere Quelltextdateien
≡ Modifiziere Schnittstellenangebot “IP3”
≡ Modifiziere Metadatendateien
→ Baue Komponente “X”
→ Aktualisiere Laufzeitinstanzen von Komponente “X” auf 3 Knoten
Implementiere Komponente “Y”
≡ Implementiere Schnittstellenangebot “IP4”
→ Modifiziere die Baukonfiguration von Komponente “Y”
→ Baue Komponente “Y”
Entferne Komponente “Z”
→ Modifiziere Baukonfiguration von Komponente “Z”

Tabelle 4.25.: Schematische Darstellung einer verfeinerten Aktivitätsliste

4.5.6.1. Ableitung von Entwicklungstätigkeiten (Programmierung im engeren und weiteren Sinn)

Aus den architektur-bezogenen Aktivitäten werden im nächsten Schritt Entwicklungstätigkeiten abgeleitet. Das heißt, das Verfahren ermittelt mit Hilfe der Annotationen Quelltextdateien, Metadatendateien und entwicklungsnahe Konfigurationsdateien, die bearbeitet werden müssen. Für die Ableitung von Entwicklungstätigkeiten gelten die Regeln in Tabelle 4.26.

Regel	Beschreibung
Regel E1	Wenn eine Komponente hinzugefügt oder modifiziert wird und es besteht eine Quelltext-Repräsentation, führt das zu einer Quelltext-Bearbeitungs-Aktivität
Regel E2	Wenn eine Komponente mit einer Metadaten-Repräsentation hinzugefügt oder modifiziert wird, führt das zu einer Metadaten-Bearbeitungs-Aktivität.

Tabelle 4.26.: Regeln zur Ableitung von Entwicklungsaktivitäten

4.5.6.2. Ableitung von Aktivitäten zur Baukonfiguration und Baudurchführung

Im nächsten Schritt werden Tätigkeiten zur Baukonfiguration und zur Baudurchführung ermittelt. Die Regeln für die Ableitung von Baukonfigurationsaktivitäten werden in Tabelle 4.27 aufgeführt und die Regeln für die Ableitung von Baudurchführungsaktivitäten werden in Tabelle 4.28 aufgeführt. Konkrete Beispiele für Baukonfigurationsaktivitäten sind die Anpassung von ant-Buildskripten und die Konfiguration der automatisierten Bauinfrastruktur (Build-Server) wie zum Beispiel CruiseControl, Buckminster, Jenkins oder TeamCity.

Regel	Beschreibung
Regel BK1	Wenn eine Komponente hinzugefügt wird, führt das zu einer Baukonfigurations-Aktivität. (Aktivität: „Komponente X in Baukonfiguration registrieren“)
Regel BK2	Wenn eine Schnittstellennachfrage zu einer Komponente hinzukommt, führt das zu einer Baukonfigurations-Aktivität. (Aktivität: „Hinzugekommene Abhängigkeiten für Komponente X in Baukonfiguration aktualisieren“)
Regel BK3	Wenn eine Schnittstellennachfrage von einer Komponente entfernt wird, führt das zu einer Baukonfigurations-Aktivität. (Aktivität: „Weggefallene Abhängigkeiten für Komponente X in Baukonfiguration aktualisieren“)

Tabelle 4.27.: Regeln zur Ableitung von Baukonfigurationsaktivitäten

Regel	Beschreibung
Regel BD1	Eine Quelltext-Bearbeitungsaktivität führt zu einer Baudurchführungsaktivität.
Regel BD2	Eine Metadaten-Bearbeitungsaktivität führt zu einer Baudurchführungsaktivität.
Regel BD3	Eine Baukonfigurations-Aktivität führt zu einer Baudurchführungsaktivität.

Tabelle 4.28.: Regeln zur Ableitung von Baudurchführungsaktivitäten

4.5.6.3. Ableitung von Aktivitäten zur Testentwicklung und Testdurchführung

Im nächsten Schritt werden Tätigkeiten zur Testentwicklung und zur Testdurchführung ermittelt. Dabei werden verschiedene Arten von Tests berücksichtigt und unterschieden. Die Regeln für die Ableitung von Testentwicklungsaktivitäten werden in Tabelle 4.29 aufgeführt und die Regeln für die Ableitung von Testdurchführungsaktivitäten werden in Tabelle 4.30 aufgeführt.

Wir unterscheiden Unit-Tests, die an den Schnittstellenangeboten der Komponente annotiert sind, Akzeptanztests, die an Benutzerschnittstellen annotiert sind, und Integrationstests, die an den Konnektoren annotiert sind.

Regel	**Beschreibung**
Regel TE1	Wenn ein Schnittstellenangebot hinzugefügt wird, führt das zu einer Testentwicklungsaktivität von Unit-Tests.
Regel TE2	Wenn ein Schnittstellenangebot modifiziert wird, führt das zu eine Testaktualisierungsaktivität der vorhandenen Unit-Tests.
Regel TE3	Wenn ein Assemblierungskonnektor hinzukommt, führt das zu einer Testentwicklungsaktivität für Integrationstests.
Regel TE4	Wenn ein Assemblierungskonnektor modifiziert wird, führt das zu einer Testaktualisierungsaktivität der vorhandenen Integrationstests.
Regel TE5	Wenn eine Benutzerschnittstelle hinzukommt, führt das zu einer Testentwicklungsaktivität für Akzeptanztests.
Regel TE6	Wenn eine Benutzerschnittstelle modifiziert wird, führt das zu einer Testaktualisierungsaktivität für vorhandene Akzeptanztests.

Tabelle 4.29.: Regeln zur Ableitung von Testentwicklungsaktivitäten

Regel	Beschreibung
Regel TD1	Wenn ein Schnittstellenangebot hinzugefügt oder modifiziert wird, führt das zu einer Testdurchführungsaktivität vorhandener Unit-Tests.
Regel TD2	Wenn ein Konnektor hinzugefügt oder modifiziert wird, führt das zu einer Testdurchführungsaktivität vorhandener Integrationstests.
Regel TD3	Wenn eine Benutzerschnittstelle hinzugefügt oder modifiziert wird, führt das zu einer Testdurchführungsaktivität vorhandener Akzeptanztests.

Tabelle 4.30.: Regeln zur Ableitung von Testdurchführungsaktivitäten

4.5.6.4. Ableitung von Aktivitäten zur Bereitstellungskonfiguration und -durchführung

Im nächsten Schritt werden Tätigkeiten zur Bereitstellungskonfiguration und zur Bereitstellungsdurchführung ermittelt. Die Regeln für die Ableitung von Bereitstellungskonfigurationsaktivitäten werden in Tabelle 4.31 aufgeführt und die Regeln für die Ableitung von Bereitstellungsdurchführungsaktivitäten werden in Tabelle 4.32 aufgeführt.

Regel	Beschreibung
Regel BSK1	Eine Baukonfigurationsaktivität führt zu einer Bereitstellungskonfigurationsaktivität.

Tabelle 4.31.: Regeln zur Ableitung von Bereitstellungskonfigurationsaktivitäten

Regel	Beschreibung
Regel BSD1	Eine Bauaktivität führt zu einer nachgelagerten Bereitstellungsaktivität. (Aktivität: „Installierbare Komponente X bereitstellen“)

Tabelle 4.32.: Regeln zur Ableitung von Bereitstellungsdurchführungsaktivitäten

4.5.6.5. Ableitung von Aktivitäten zur Installation und Inbetriebnahme

Im nächsten Schritt werden Tätigkeiten zur Installation und Inbetriebnahme ermittelt. Die Regeln für die Ableitung von Inbetriebnahmekonfigurationsaktivitäten werden in Tabelle 4.33 aufgeführt und die Regeln für die Ableitung von Inbetriebnahmeaktivitäten werden in Tabelle 4.34 aufgeführt.

Regel	Beschreibung
Regel IBK1	Eine Baukonfigurationsaktivität führt zu einer Inbetriebnahmekonfigurationsaktivität.

Tabelle 4.33.: Regeln zur Ableitung von Inbetriebnahmekonfigurationsaktivitäten

Regel	Beschreibung
Regel IBD1	Eine Bereitstellungsaktivität führt zu einer Inbetriebnahmeaktivität.

Tabelle 4.34.: Regeln zur Ableitung von Inbetriebnahmeaktivitäten

4.5.6.6. Regel-Anwendung auf Beispiel

Für Umsetzungsweg A1 werden die Entwicklungstätigkeiten wie in den Tabellen 4.35 und 4.36 abgeleitet.

Modifiziere Komponente Datenbank
≡ **Entwicklung: Metadaten bearbeiten von Komponente Datenbank**
≡ Modifiziere Schnittstellenangebot ODBC
≡ Modifiziere Datentyp BestellDatenbankSchema
≡ **Entwicklung: Bearbeite Metadatendatei „dbschema.sql“**
→ Modifiziere Konnektor Bestell-Server → Datenbank
→ Modifiziere Komponente Bestell-Server
≡ **Entwicklung: Quelltext bearbeiten von Komponente Bestell-Server**
≡ Modifiziere Schnittstellennachfrage ODBC
≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle
≡ **Entwicklung: Quelltext bearbeiten von KundenServerSchnittstelle der Komponente Bestell-Server**
≡ **Bearbeite Aufrufstellen von ODBC im Quelltext von KundenServerSchnittstelle**
≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle
≡ **Entwicklung: Bearbeite Aufrufstellen von ODBC im Quelltext von HändlerServerSchnittstelle**
≡ **Bearbeite Aufrufstellen von ODBC im Quelltext von Händler-ServerSchnittstelle**

Tabelle 4.35.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Entwicklungstätigkeiten (Teil 1)

→ Modifiziere Konnektor Kunden-Client → Bestell-Server
→ Modifiziere Komponente Kunden-Client
≡ **Entwicklung: Quelltext bearbeiten von Komponente Kunden-Client**
≡ Modifiziere Schnittstellennachfrage KundenServerSchnittstelle
≡ Modifiziere Schnittstellenangebot KundenSchnittstelle
≡ **Entwicklung: Bearbeite Aufrufstellen von KundenServer-Schnittstelle im Quelltext von KundenSchnittstelle**
→ Modifiziere Konnektor Händler-Client → Bestell-Server
→ Modifiziere Komponente Händler-Client
≡ **Entwicklung: Quelltext bearbeiten von Komponente Händler-Client**
≡ Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle
≡ Modifiziere Schnittstellenangebot HändlerSchnittstelle
≡ **Entwicklung: Bearbeite Aufrufstellen von HändlerServer-Schnittstelle im Quelltext von HändlerSchnittstelle**

Tabelle 4.36.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Entwicklungstätigkeiten (Teil 2)

Implementiere Komponente Zwischenspeicher ≡ Implementiere Schnittstellenangebot ZwischenspeicherSchnittstelle ≡ Implementiere Schnittstellenoperation abfrageStellen()
≡ **Entwicklung: Quelltext bearbeiten von Schnittstellenoperation abfrageStellen()** Implementiere Konnektor Bestell-Server → Zwischenspeicher
Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur Zwischenspeicher-Schnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() ≡ **Entwicklung: Quelltext bearbeiten von Schnittstellenangebot KundenServerSchnittstelle** ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur Zwischenspeicher-Schnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() ≡ **Entwicklung: Quelltext bearbeiten von Schnittstellenangebot HändlerServerSchnittstelle**

Tabelle 4.37.: Aktivitäten von Umsetzungsweg A2

Modifiziere Komponente Datenbank → *Entwicklung: Metadaten bearbeiten von Komponente Datenbank* ≡ Modifiziere Schnittstellenangebot ODBC ≡ Modifiziere Datentyp BestellDatenbankSchema → *Entwicklung: Bearbeite Metadatendatei „dbschema.sql“* → **Test-Entwicklung: Aktualisiere Unit-Tests für Schnittstellenangebot ODBC** → **Test-Durchführung: Führe Unit-Tests für Schnittstellenangebot ODBC aus** → Modifiziere Konnektor Bestell-Server → Datenbank → **Test-Entwicklung: Aktualisiere Integrationstests zwischen Bestell-Server und Datenbank** → **Test-Durchführung: Führe Integrationstests zwischen Bestell-Server und Datenbank aus** → Modifiziere Komponente Bestell-Server → *Entwicklung: Quelltext bearbeiten von Komponente Bestell-Server* ≡ Modifiziere Schnittstellennachfrage ODBC ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle → *Entwicklung: Quelltext bearbeiten von KundenServerSchnittstelle der Komponente Bestell-Server* ≡ *Bearbeite Aufrufschrtellen von ODBC im Quelltext von KundenServerSchnittstelle*
(Fortsetzung in Tabelle 4.39)

Tabelle 4.38.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Test-Aktivitäten (Teil 1)

(Fortsetzung von Tabelle 4.38)
→ **Test-Entwicklung: Aktualisiere Unit-Tests für Schnittstellenangebot KundenServerSchnittstelle** → **Test-Durchführung: Führe Unit-Tests für Schnittstellenangebot KundenServerSchnittstelle aus** ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle → *Entwicklung: Bearbeite Aufrufstellen von ODBC im Quelltext von HändlerServerSchnittstelle* → *Bearbeite Aufrufstellen von ODBC im Quelltext von HändlerServerSchnittstelle* → **Test-Entwicklung: Aktualisiere Unit-Tests für Schnittstellenangebot HändlerServerSchnittstelle** → **Test-Durchführung: Führe Unit-Tests für Schnittstellenangebot HändlerServerSchnittstelle aus** → Modifiziere Konnektor Kunden-Client → Bestell-Server → **Test-Entwicklung: Aktualisiere Integrationstests zwischen Kunden-Client und Bestell-Server** → **Test-Durchführung: Führe Integrationstests zwischen Kunden-Client und Bestell-Server aus** → Modifiziere Komponente Kunden-Client → *Entwicklung: Quelltext bearbeiten von Komponente Kunden-Client*
(Fortsetzung in Tabelle 4.40)

Tabelle 4.39.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Test-Aktivitäten (Teil 2)

(Fortsetzung von Tabelle 4.39)

≡ Modifiziere Schnittstellenangebot KundenSchnittstelle
≡ Modifiziere Schnittstellennachfrage KundenServerSchnittstelle
→ *Entwicklung: Bearbeite Aufrufstellen von KundenServerSchnittstelle im Quelltext von KundenSchnittstelle*
→ **Test-Entwicklung: Aktualisiere Akzeptanztests für Schnittstellenangebot KundenSchnittstelle**
→ **Test-Durchführung: Führe Akzeptanztests für Schnittstellenangebot KundenSchnittstelle aus**
→ Modifiziere Konnektor Händler-Client → Bestell-Server
→ **Test-Entwicklung: Aktualisiere Integrationstests zwischen Händler-Client und Bestell-Server**
→ **Test-Durchführung: Führe Integrationstests zwischen Händler-Client und Bestell-Server aus**
→ Modifiziere Komponente Händler-Client
→ *Entwicklung: Quelltext bearbeiten von Komponente Händler-Client*
≡ Modifiziere Schnittstellenangebot HändlerSchnittstelle
≡ Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle
→ *Entwicklung: Bearbeite Aufrufstellen von HändlerServerSchnittstelle im Quelltext von HändlerSchnittstelle*
→ **Test-Entwicklung: Aktualisiere Akzeptanztests für Schnittstellenangebot HändlerSchnittstelle**
→ **Test-Durchführung: Führe Akzeptanztests für Schnittstellenangebot HändlerSchnittstelle aus**

Tabelle 4.40.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Test-Aktivitäten (Teil 3)

Implementiere Komponente Zwischenspeicher ≡ Implementiere Schnittstellenangebot ZwischenspeicherSchnittstelle ≡ Implementiere Schnittstellenoperation abfrageStellen() ≡ *Entwicklung: Quelltext bearbeiten von Schnittstellenoperation abfrageStellen()* → **Test-Entwicklung: Entwickle Unit-Tests zum Schnittstellenangebot ZwischenspeicherSchnittstelle** → **Test-Durchführung: Führe Unit-Tests zum Schnittstellenangebot ZwischenspeicherSchnittstelle durch**
Implementiere Konnektor Bestell-Server → Zwischenspeicher → **Test-Entwicklung: Entwickle Integrationstests zwischen Bestell-Server und Zwischenspeicher** → **Test-Durchführung: Führe Integrationstests zwischen Bestell-Server und Zwischenspeicher durch**
Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur ZwischenspeicherSchnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() ≡ *Entwicklung: Quelltext bearbeiten von Schnittstellenangebot KundenServerSchnittstelle* → **Test-Entwicklung: Aktualisiere Unit-Tests zum Schnittstellenangebot KundenServerSchnittstelle** → **Test-Durchführung: Führe Unit-Tests zum Schnittstellenangebot KundenServerSchnittstelle durch**

Tabelle 4.41.: Aktivitäten von Umsetzungsweg A2 nach Ableitung von Test-Aktivitäten (Teil 1)

≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur ZwischenspeicherSchnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() ≡ *Entwicklung: Quelltext bearbeiten von Schnittstellenangebot HändlerServerSchnittstelle* → **Test-Entwicklung: Aktualisiere Unit-Tests zum Schnittstellenangebot HändlerServerSchnittstelle** → **Test-Durchführung: Führe Unit-Tests zum Schnittstellenangebot HändlerServerSchnittstelle durch**

Tabelle 4.42.: Aktivitäten von Umsetzungsweg A2 nach Ableitung von Test-Aktivitäten (Teil 2)

Modifiziere Komponente Datenbank → *Entwicklung: Metadaten bearbeiten von Komponente Datenbank* ≡ Modifiziere Schnittstellenangebot ODBC ≡ Modifiziere Datentyp BestellDatenbankSchema → *Entwicklung: Bearbeite Metadatendatei „dbschema.sql“* → *Test-Entwicklung: Aktualisiere Unit-Tests für Schnittstellenangebot ODBC* → *Test-Durchführung: Führe Unit-Tests für Schnittstellenangebot ODBC aus* → **Bau-Durchführung: kein Bau da Drittanbieter-Komponente!** → **Bereitstellung: Modifizierte Metadaten bereitstellen.** → **Inbetriebnahme-Konfiguration: Datenbank-Konfiguration aktualisieren, Datenmigration durchführen** → **Inbetriebnahme: Datenbank in Betrieb nehmen**
→ Modifiziere Konnektor Bestell-Server → Datenbank → *Test-Entwicklung: Aktualisiere Integrationstests zwischen Bestell-Server und Datenbank* → *Test-Durchführung: Führe Integrationstests zwischen Bestell-Server und Datenbank aus*
→ Modifiziere Komponente Bestell-Server → *Entwicklung: Quelltext bearbeiten von Komponente Bestell-Server* ≡ Modifiziere Schnittstellennachfrage ODBC

Tabelle 4.43.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Aktivitäten zu Bau, Bereitstellung und Inbetriebnahme (Teil 1)

≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle → *Entwicklung: Quelltext bearbeiten von KundenServerSchnittstelle der Komponente Bestell-Server* ≡ *Bearbeite Aufrufstellen von ODBC im Quelltext von KundenServer-Schnittstelle* → *Test-Entwicklung: Aktualisiere Unit-Tests für Schnittstellenangebot KundenServerSchnittstelle* → *Test-Durchführung: Führe Unit-Tests für Schnittstellenangebot KundenServerSchnittstelle aus* ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle → *Entwicklung: Bearbeite Aufrufstellen von ODBC im Quelltext von HändlerServerSchnittstelle* → *Bearbeite Aufrufstellen von ODBC im Quelltext von HändlerServerSchnittstelle* → *Test-Entwicklung: Aktualisiere Unit-Tests für Schnittstellenangebot HändlerServerSchnittstelle* → *Test-Durchführung: Führe Unit-Tests für Schnittstellenangebot HändlerServerSchnittstelle aus* → **Bau-Durchführung: Komponente Bestell-Server bauen** → **Bereitstellung: Installierbarer Bestell-Server bereitstellen** → **Inbetriebnahme: Bestell-Server in Betrieb nehmen**
→ Modifiziere Konnektor Kunden-Client → Bestell-Server → *Test-Entwicklung: Aktualisiere Integrationstests zwischen Kunden-Client und Bestell-Server* → *Test-Durchführung: Führe Integrationstests zwischen Kunden-Client und Bestell-Server aus*

Tabelle 4.44.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Aktivitäten zu Bau, Bereitstellung und Inbetriebnahme (Teil 2)

4.5.7. Personalzuordnung und Technologieentsprechung ermitteln

Den Aktivitäten kann je nach Tätigkeitsfeld über die Auswertung der Anreicherungen automatisiert das Personal zugeordnet werden. Darüber hinaus können den Aktivitäten über die Auswertung der Anreicherungen automatisiert die Technologieentsprechungen zugeordnet werden. So schafft das Verfahren den Übergang von der abstrakten Architekturbetrachtung zur konkreten Betrachtung der Entwicklungsartefakte.

4.5.8. Schablonen für Aktivitätsbeschreibungen

Die Beschreibung der Aktivitäten wird durch Textschablonen unterstützt, welche durch die Ableitungsregeln gefüllt werden. In den Tabellen 4.49 und 4.50 werden einige verwendete Schablonen aufgeführt. Diese Liste ist keineswegs vollständig. Vielmehr können die Schablonen an die Bedürfnisse der Anwender angepasst werden.

Entwicklung	Entwicklung: Quelltext bearbeiten von Basiskomponente [name]
Entwicklung	Entwicklung: Quelltext bearbeiten von Schnittstellenangebot [name]
Entwicklung	Entwicklung: Quelltext bearbeiten von angebotener Schnittstellenoperation [name]
Entwicklung	Entwicklung: Bearbeite Aufrufstellen von Schnittstellenangebot [name] im Quelltext von Komponente [name]
Entwicklung	Entwicklung: Metadaten bearbeiten von Basiskomponente [name]
Entwicklung	Entwicklung: Metadaten bearbeiten von Schnittstellenangebot [name]
Entwicklung	Entwicklung: Metadaten bearbeiten von angebotener Schnittstellenoperation [name] von Schnittstellenangebot [name] von Komponente [name]

Tabelle 4.49.: Textschablonen für Aktivitätsbeschreibungen (Teil 1)

→ Modifiziere Komponente Kunden-Client → *Entwicklung: Quelltext bearbeiten von Komponente Kunden-Client* ≡ Modifiziere Schnittstellenangebot KundenSchnittstelle ≡ Modifiziere Schnittstellennachfrage KundenServerSchnittstelle → *Entwicklung: Bearbeite Aufrufstellen von KundenServerSchnittstelle im Quelltext von KundenSchnittstelle* → *Test-Entwicklung: Aktualisiere Akzeptanztests für Schnittstellenangebot KundenSchnittstelle* → *Test-Durchführung: Führe Akzeptanztests für Schnittstellenangebot KundenSchnittstelle aus* → **Bau-Durchführung: Komponente Kunden-Client bauen** → **Bereitstellung: Installierbarer Kunden-Client bereitstellen**
→ Modifiziere Konnektor Händler-Client → Bestell-Server → *Test-Entwicklung: Aktualisiere Integrationstests zwischen Händler-Client und Bestell-Server* → *Test-Durchführung: Führe Integrationstests zwischen Händler-Client und Bestell-Server aus*

Tabelle 4.45.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Aktivitäten zu Bau, Bereitstellung und Inbetriebnahme (Teil 3)

→ Modifiziere Komponente Händler-Client → *Entwicklung: Quelltext bearbeiten von Komponente Händler-Client* ≡ Modifiziere Schnittstellenangebot HändlerSchnittstelle ≡ Modifiziere Schnittstellennachfrage HändlerServerSchnittstelle → *Entwicklung: Bearbeite Aufrufstellen von HändlerServerSchnittstelle im Quelltext von HändlerSchnittstelle* → *Test-Entwicklung: Aktualisiere Akzeptanztests für Schnittstellenangebot HändlerSchnittstelle* → *Test-Durchführung: Führe Akzeptanztests für Schnittstellenangebot HändlerSchnittstelle aus* → **Bau-Durchführung: Komponente Händler-Client bauen** → **Bereitstellung: Installierbarer Händler-Client bereitstellen** → **Inbetriebnahme: Händler-Client auf 10 Rechnern in Betrieb nehmen**

Tabelle 4.46.: Aktivitäten von Umsetzungsweg A1 nach Ableitung von Aktivitäten zu Bau, Bereitstellung und Inbetriebnahme (Teil 4)

Implementiere Komponente Zwischenspeicher ≡ Implementiere Schnittstellenangebot ZwischenspeicherSchnittstelle ≡ Implementiere Schnittstellenoperation abfrageStellen() ≡ *Entwicklung: Quelltext bearbeiten von Schnittstellenoperation abfrageStellen()* → *Test-Entwicklung: Entwickle Unit-Tests zum Schnittstellenangebot ZwischenspeicherSchnittstelle* → *Test-Durchführung: Führe Unit-Tests zum Schnittstellenangebot ZwischenspeicherSchnittstelle durch* → **Bau-Konfiguration: Trage Komponente Zwischenspeicher in Baukonfiguration ein** → **Bau-Durchführung: Komponente Zwischenspeicher bauen** → **Bereitstellung: Installierbarer Zwischenspeicher bereitstellen** → **Inbetriebnahme: Zwischenspeicher in Betrieb nehmen**
Implementiere Konnektor Bestell-Server → Zwischenspeicher → *Test-Entwicklung: Entwickle Integrationstests zwischen Bestell-Server und Zwischenspeicher* → *Test-Durchführung: Führe Integrationstests zwischen Bestell-Server und Zwischenspeicher durch*

Tabelle 4.47.: Aktivitäten von Umsetzungsweg A2 nach Ableitung von Aktivitäten zu Bau, Bereitstellung und Inbetriebnahme (Teil 1)

Modifiziere Komponente Bestell-Server ≡ Modifiziere Schnittstellenangebot KundenServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur ZwischenspeicherSchnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() ≡ *Entwicklung: Quelltext bearbeiten von Schnittstellenangebot KundenServerSchnittstelle*
→ *Test-Entwicklung: Aktualisiere Unit-Tests zum Schnittstellenangebot KundenServerSchnittstelle* → *Test-Durchführung: Führe Unit-Tests zum Schnittstellenangebot KundenServerSchnittstelle durch* ≡ Modifiziere Schnittstellenangebot HändlerServerSchnittstelle ≡ Implementiere Schnittstellennachfrage zur Zwischenspeicher-Schnittstelle ≡ Implementiere Operationsnachfrage abfrageStellen() ≡ *Entwicklung: Quelltext bearbeiten von Schnittstellenangebot HändlerServerSchnittstelle* → *Test-Entwicklung: Aktualisiere Unit-Tests zum Schnittstellenangebot HändlerServerSchnittstelle* → *Test-Durchführung: Führe Unit-Tests zum Schnittstellenangebot HändlerServerSchnittstelle durch* → **Bau-Konfiguration: Trage Abhängigkeiten zu Komponente Zwischenspeicher in Baukonfiguration von Komponente Bestell-Server ein** → **Bau-Durchführung: Komponente Bestell-Server bauen** → **Bereitstellung: Installierbarer Bestell-Server bereitstellen** → **Inbetriebnahme: Bestell-Server in Betrieb nehmen**

Tabelle 4.48.: Aktivitäten von Umsetzungsweg A2 nach Ableitung von Aktivitäten zu Bau, Bereitstellung und Inbetriebnahme (Teil 2)

Test-Entwicklung	Test-Entwicklung: Entwickle Akzeptanztests für Schnittstellenangebot [name] von Komponente [name]
Test-Entwicklung	Test-Entwicklung: Entwickle Unit-Tests für Schnittstellenangebot [name] von Komponente [name]
Test-Entwicklung	Test-Entwicklung: Entwickle Integrationstests zwischen Komponente [name] und Komponente [name]
Test-Entwicklung	Test-Entwicklung: Aktualisiere Akzeptanztests für Schnittstellenangebot [name] von Komponente [name]
Test-Entwicklung	Test-Entwicklung: Aktualisiere Unit-Tests für Schnittstellenangebot [name] von Komponente [name]
Test-Entwicklung	Test-Entwicklung: Aktualisiere Integrationstests zwischen Komponente [name] und Komponente [name]
Test-Durchführung	Test-Durchführung: Führe Unit-Tests für Schnittstellenangebot [name] durch
Test-Durchführung	Test-Durchführung: Führe Integrationstests zwischen Komponente [name] und Komponente [name] durch
Test-Durchführung	Test-Durchführung: Führe Akzeptanztests für Schnittstellenangebot [name] von Komponente [name] durch

Tabelle 4.50.: Textschablonen für Aktivitätsbeschreibungen (Teil 2)

Bau-Konfiguration	Bau-Konfiguration: Trage Komponente [name] in Baukonfiguration ein
Bau-Konfiguration	Bau-Konfiguration: Trage Abhängigkeiten von Komponente [name] zu Komponente [name] in Baukonfiguration ein
Bau-Durchführung	Bau-Durchführung: Komponente [name] bauen
Bereitstellung	Bereitstellung: Installierbare Komponente [name] bereitstellen
Inbetriebnahme	Inbetriebnahme: Komponente [name] in Betrieb nehmen
Inbetriebnahme	Inbetriebnahme: [anzahlLaufzeitinstanzen] Laufzeitinstanzen von Komponente [name] in Betrieb nehmen

Tabelle 4.51.: Textschablonen für Aktivitätsbeschreibungen (Teil 3)

4.6. Anwendungsszenarien von Aktivitätslisten

Aus der Änderungsanfragenanalyse resultieren einzelne Aktivitätslisten. In einem anschließenden Vergleichs- und Optimierungsschritt können diese zueinander in Bezug gesetzt werden. In diesem Abschnitt stellen wir Anwendungsszenarien für die Aktivitätslisten vor und zeigen wie das Verfahren zur Bewertung, zum Vergleich und zur Optimierung von Aktivitätslisten beitragen kann. Das Verfahren unterstützt folgende prinzipielle Anwendungsszenarien.

1) Bewertung einzelner Umsetzungswege: Einzelne Umsetzungswege werden unabhängig von anderen bewertet.

2) Abwägung alternativer Umsetzungswege: Es werden zwei Aktivitätslisten miteinander verglichen, die alternative Lösungen darstellen.

3) Kombination ähnlicher Umsetzungswege: Ähnliche oder überlappende Umsetzungswege können kombiniert werden.

4) Erkennung von Umsetzungskonflikten: Bestimmte Arten von Konflikten zwischen Umsetzungswegen können durch den Vergleich von Aktivitätslisten erkannt werden.

5) Unterstützung der Aufwands- und Kostenschätzung:
Aktivitätslisten können direkt zur Kostenschätzung, beispielsweise durch aufsteigende Schätzung (engl. bottom-up), verwendet werden.

4.6.1. Bewertung einzelner Umsetzungswege

Einzelne Umsetzungswege können mit dem Verfahren bewertet werden. Tabelle 4.52 fasst Metriken zur Bewertung von Umsetzungswegen zusammen, die aus einer Aktivitätsliste berechnet werden können. Diese Metriken liefern Einzelbewertungsergebnisse für Aktivitätslisten.

Metrik
Anzahl der modifizierten Komponenten
Anzahl der modifizierten Quelltextdateien
Anzahl der modifizierten Metadatendateien
Anzahl der beteiligten Personen
Anzahl zu erstellender oder zu modifizierender Testfälle differenziert nach Art der Tests
Anzahl der Testausführungen differenziert nach Art der Tests
Anzahl der Baudurchführungen pro Komponente
Anzahl der Bereitstellungen pro Komponente
Anzahl der aktualisierten Laufzeitinstanzen pro Komponente

Tabelle 4.52.: Metriken zur Bewertung von Umsetzungswegen

4.6.2. Abwägung alternativer Umsetzungswege

Zwei alternative Umsetzungswege können mit dem Verfahren gegeneinander abgewogen werden. Dies geschieht auf Basis der Einzelbewertungsergebnisse. Allerdings werden vor dem Vergleich der Einzelbewertungsergebnisse die strukturellen Gemeinsamkeiten und Unterschiede der Aktivitätsliste bestimmt. Für eine Abwägung reicht es aus, sich auf die Unterschiede zu konzentrieren. Demnach werden die Einzelbewertungsergebnisse der Unterschiede gegenübergestellt.

4.6.3. Kombination ähnlicher Umsetzungswege

Zwei Umsetzungswege können miteinander kombiniert werden. Bei der Kombination werden die Aktivitäten so nach Artefakten gruppiert, dass die Modifikationen an denselben Artefakten als Einheit geplant werden. Das führt in der Folge zu einer Reduzierung von Testausführungen, Baudurchführungen, Bereitstellungen und Inbetriebnahmen, da diese pro Artefakt nur einmal notwendig sind.

4.6.4. Erkennung von Umsetzungskonflikten

Aktivitätslisten ermöglichen es, verschiedene Arten von Konflikten und Abhängigkeiten zwischen Aktivitäten zu erkennen.

Konfliktbehaftet sind Aktivitäten, die in einem widersprüchlichen Verhältnis zueinander stehen. Beispielsweise wenn dasselbe Element in einer Aktivität entfernt und in einer anderen modifiziert werden soll. Aktivitäten auf denselben Artefakten stellen jedoch nicht zwangsläufig einen Konflikt dar. Allerdings kann dies ein Indiz für einen möglichen Konflikt sein.

Allgemein sollte man Aktivitäten untersuchen, zwischen denen Abhängigkeiten bestehen. Aktivitäten befinden sich in einem Abhängigkeitsverhältnis, wenn sie Voraussetzungen für die Durchführung einer anderen Aktivität schaffen oder wegnehmen.

4.6.5. Unterstützung der Aufwands- und Kostenschätzung

Für die einzelnen Aktivitäten können, der aufsteigenden Schätzmethodik (engl. bottom-up) folgend, Aufwände separat geschätzt werden. Der Gesamtaufwand wird dann durch die Aggregation der Teilaufwände bestimmt. Hierbei können insbesondere die Metrikwerten angereicherten Aktivitäten helfen und die Zusatzinformationen zur Technologieentsprechung und Personalzuordnung.

5. Formalisierung des Verfahrens

Das vorige Kapitel stellt unser Verfahren zur Bewertung und Planung von Änderungsanfragen vor. Der Ansatz wird dort aus der Anwenderperspektive präsentiert und die Analyseschritte werden anhand eines durchgängigen Beispiels illustriert. Dieses Kapitel befasst sich mit der Formalisierung des Verfahrens. Das heißt wir stellen die Modelle für das Analyseverfahren vor und beschreiben die Analyseschritte im Detail.

Zur Beschreibung der Eingabedaten, Zwischenergebnisse und Endergebnisse bedienen wir uns der Metamodellierung. Wir definieren mehrere Metamodelle mit denen das Verfahren arbeitet. Da sich manche Analyseschritte durch Graphoperationen gut beschreiben lassen, geben wir zusätzlich zu den Metamodellen eine dazu isomorphe Graphrepräsentation für die Datenmodelle an. Die Analyseschritte werden entweder durch Transformationen auf den Metamodellen beschrieben oder durch Graph-Algorithmen auf der Graphrepräsentation.

Für das Verfahren sind drei Datenmodelle von Bedeutung, zwei Eingabemodelle und ein Ausgabemodell. Als Eingabemodelle dienen zum Einen das *Architekturmodell*, welches die statische komponenten-basierte Struktur der Architektur beschreibt, und zum Anderen das *Anreicherungsmodell*, welches die Annotationen erfasst, welche Kontextinformationen über die Tätigkeitsbereiche mit den Architekturelementen in Bezug setzen lassen. Das *Aktivitätsmodell* spezifiziert die Aktivitätslisten, die als Zwischen- und Endergebnisse aus dem Verfahren resultieren und repräsentiert somit das Ausgabemodell.

Das Kapitel ist wie folgt aufgebaut. Abschnitt 5.1 beschreibt die Metamodelle und Graphdefinitionen für Architekturmodell, Anreicherungsmodell und

Aktivitätsmodell. Abschnitt 5.2 erläutert die Graphoperationen und Modelltransformationen zum Aufbau des Architekturmodells und des Anreicherungsmodells während der Vorbereitungsphase. Abschnitt 5.3 beschreibt für die einzelnen Schritte der Änderungsanfragenanalyse die jeweiligen Modelltransformationen und Graphoperationen.

5.1. Metamodelle und Graphdefinitionen

Wir beschreiben das Architekturmodell in Abschnitt 5.1.1, das Anreicherungsmodell in Abschnitt 5.1.2 und das Aktivitätsmodell in Abschnitt 5.1.3.

Zentrale Oberklasse `ModellElement` mit eindeutigem Identifikator-Attribut

Für alle definierten Metamodelle im Folgenden gilt, dass ihre Metaklassen von einer zentralen Metaklasse `ModellElement` abgeleitet sind, die ein Identifikator-Attribut (`id`) enthält. Bei der Erzeugung von Modellelementen wird sichergestellt, dass dieses Identifikator-Attribut universell eindeutig ist. Jedes Modellelement besitzt demnach einen universell und global eindeutigen Identifikator. Zur technischen Realisierung dieses Identifikators verwenden wir das Konzept des UUID-Standards (engl. Universally Unique Identifier) [LMS05].

5.1.1. Architekturmodell

5.1.1.1. Definition des Architektur-Metamodells

Wir verwenden als Metamodell für die Architekturmodellierung einen Ausschnitt des Palladio-Komponentenmodells (PCM, Palladio Component Model) wie es im Grundlagenkapitel in Abschnitt 2.2 beschrieben wird. Besondere Erweiterungen des PCMs, die für unser Verfahren notwendig sind, werden nachfolgend erläutert.

5.1.1.2. Modellierung komponenten-interner Abhängigkeiten

Für die Änderungsausbreitungsanalyse ist es notwendig, die Abhängigkeitsbeziehungen innerhalb einer Komponente zu kennen. Das Palladio-Komponentenmodell bietet sogenannte „Service Effect Specifications“ (SEFF) mit denen für eine angebotene Operation ein abstrakter Kontrollfluss modelliert werden kann. In den SEFFs werden auch Aufrufbeziehungen nachgefragter Schnittstellen modelliert und somit Abhängigkeitsbeziehungen zwischen Schnittstellenangeboten und -nachfragen ausgedrückt. Jedoch ist die Ausmodellierung aller SEFFs in einem Modell sehr aufwändig. Wir haben das Palladio-Komponentenmodell deshalb um ein vereinfachtes Abhängigkeitsmodell erweitert, wie im Klassendiagramm in Abbildung 5.1 dargestellt. Mit diesem Modell können Abhängigkeiten zwischen Schnittstellenangeboten und Schnittstellennachfragen (Metaklasse: `KomponentenInterneAbhängigkeit`) innerhalb einer Komponente modelliert werden. Wie man sieht, können diese Abhängigkeiten spezifisch für angebotene und nachgefragte Operationen beschrieben werden. Eine solche Abhängigkeitsbeziehung drückt aus, welche nachgefragten Operationen zur Umsetzung (d.h. Implementierung) einer angebotenen Operation verwendet werden.

5.1.1.3. Modellierung von Modifikationskennzeichnungen

Im Rahmen der Änderungsanfragenanalyse beschreibt der Anwender einen Umsetzungsweg im Architekturmodell. Um Umsetzungswege vollständig beschreiben zu können, muss der Anwender Modifikationen an Architekturelementen kennzeichnen können. Das betrifft insbesondere Modifikationen, die intern in Architekturelementen vorgenommen werden müssen. Mit der Erweiterung des Palladio-Komponentenmodells, die im Klassendiagramm in Abbildung 5.2 zu sehen ist, kann der Anwender Modifikationen (Metaklasse: `Modifikationskennzeichnung`) an Architekturelementen kennzeichnen. Das Modell unterstützt die Kennzeichnung von Datentypen, Schnittstellen, Komponenten, Schnittstellenangeboten, Schnittstellennachfragen und Assemblierungskonnektoren.

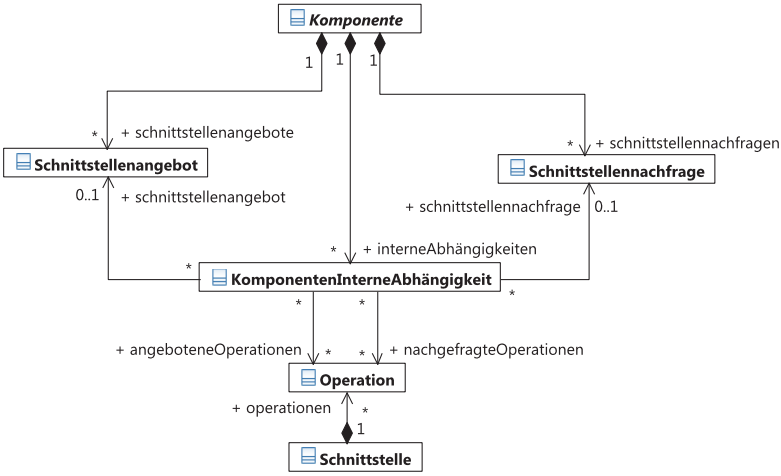


Abbildung 5.1.: Klassendiagramm zur Metamodellierung komponenten-interner Abhängigkeiten

5.1.1.4. Definition des Architektur-Graphen

Ausgehend vom Architekturmetamodell im Grundlagenkapitel in Abschnitt 2.2, sowie den gerade beschriebenen Erweiterungen zur Modellierung komponenten-interner Abhängigkeiten und zur Modellierung von Modifikationskennzeichnungen definieren wir den Architekturgraph $G_{Architektur}$, der die statische Software-Architektur repräsentiert. Der Graph besteht mit $G_{Architektur} = (V_{Architektur}, E_{Architektur})$ aus einer Knotenmenge $V_{Architektur}$ welche die statischen Architekturelemente beschreibt und einer Kantenmenge $E_{Architektur}$ welche die Beziehungen zwischen den Architekturelementen beschreibt. Im Folgenden werden die Eigenschaften der Knoten und Kanten definiert.

Definition 5.1 (Eigenschaften der Knoten)**.** *Die Knoten des Architekturgraphen besitzen einen Typ (v.typ), einen eindeutigen Identifikator (v.id) und eine Bezeichnung (v.name). Für die Knotenmenge $V_{Architektur}$ des Architekturgraphen gilt:*

$$\forall v \in V_{Architektur} : v.typ \in VT_{Architektur}$$

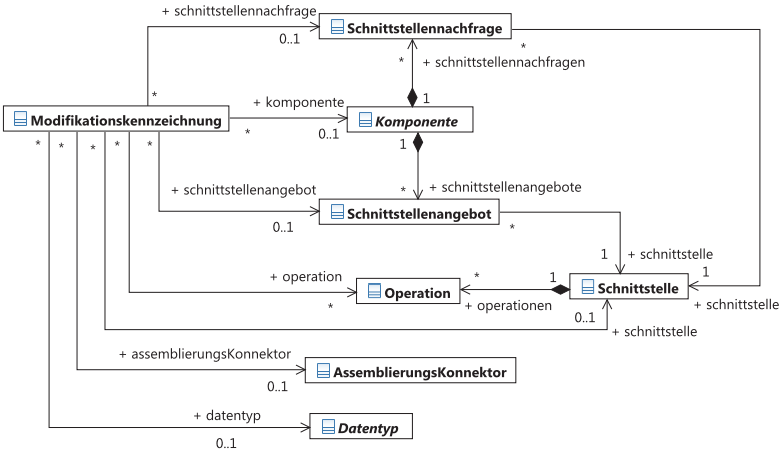


Abbildung 5.2.: Klassendiagramm zur Metamodellierung von Modifikationskennzeichnungen

Dabei ist $VT_{Architektur}$ die Menge der Knotentypen des Architekturgraphen, für die gilt: $VT_{Architektur} = \{$
`PrimitivDatentyp`, `KollektionsDatentyp`, `KompositDatentyp`,
`DatentypParameter`, `Schnittstelle`, `Operation`, `Parameter`,
`BasisKomponente`, `KompositKomponente`, `System`, `Schnittstellenangebot`,
`Schnittstellennachfrage`, `AssemblierungsKonnektor`,
`SchnittstellenAngebotsDelegation`,
`SchnittstellenNachfrageDelegation`,
`KomponentenInterneAbhängigkeit`,
`Modifikationskennzeichnung`
$\}$.

Definition 5.2 (Eigenschaften der Kanten)**.** *Die Kanten des Architekturgraphen besitzen einen Typ (e.typ). Für die Kantenmenge $E_{Architektur}$ des Architekturgraphen gilt:*

$$\forall e \in E_{Architektur} : e.typ \in ET_{Architektur}$$

Dabei ist $ET_{Architektur}$ die Menge der Kantentypen des Architekturgraphen, für die gilt: $ET_{Architektur} = \{$
`Operationsdefinition`,
`Aufrufparameterdefinition`,
`Rückgabeparameterdefinition`,
`Parameterdatentypdefinition`,
`KollektionsDatentypDatentypdefinition`,
`DatentypParameterDatentypdefinition`,
`Schnittstellenangebot`,
`Schnittstellennachfrage`,
`SchnittstellendefinitionFürSchnittstellenangebot`,
`SchnittstellendefinitionFürSchnittstellennachfrage`,
`Unterkomponentenbildung`,
`Assemblierung`,
`AssemblierungskonnektorSchnittstellenangebot`,
`AssemblierungskonnektorSchnittstellennachfrage`,
`SchnittstellenangebotsDelegationIntern`,
`SchnittstellenangebotsDelegationExtern`,
`SchnittstellennachfrageDelegationIntern`,
`SchnittstellennachfrageDelegationExtern`,
`KomponentenInterneAbhängigkeitSchnittstellenangebot`,
`KomponentenInterneAbhängigkeitSchnittstellennachfrage`,
`KomponentenInterneAbhängigkeitAngeboteneOperation`,
`KomponentenInterneAbhängigkeitNachgefragteOperation`,
`ModifikationskennzeichnungDatentyp`,
`ModifikationskennzeichnungSchnittstelle`,
`ModifikationskennzeichnungKomponente`,
`ModifikationskennzeichnungSchnittstellenangebot`,
`ModifikationskennzeichnungSchnittstellennachfrage`,
`ModifikationskennzeichnungOperation`,
`ModifikationskennzeichnungAssemblierungskonnektor`
$\}$.

Die Kantentypen werden nachstehend erläutert.

Definition 5.2.1 (Operationsdefinition)**.** *Der Kantentyp „Operationsdefinition“ verbindet einen Schnittstellenknoten mit einem Operationsknoten und beschreibt damit die Definition einer Operation durch diese Schnittstelle.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{Operationsdefinition} \Rightarrow$
$v_L.typ = \texttt{Schnittstelle} \wedge v_R.typ = \texttt{Operation}$

Definition 5.2.2 (Aufrufparameterdefinition)**.** *Der Kantentyp „Aufrufparameterdefinition“ verbindet einen Operationsknoten mit einem Parameterknoten und beschreibt damit die Definition eines Aufrufparameters für diese Operation.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{Aufrufparameterdefinition} \Rightarrow$
$v_L.typ = \texttt{Operation} \wedge v_R.typ = \texttt{Parameter}$

Definition 5.2.3 (Rückgabeparameterdefinition)**.** *Der Kantentyp „Rückgabeparameterdefinition“ verbindet einen Operationsknoten mit einem Parameterknoten und beschreibt damit die Definition eines Rückgabeparameters für diese Operation.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{Rückgabeparameterdefinition} \Rightarrow$
$v_L.typ = \texttt{Operation} \wedge v_R.typ = \texttt{Parameter}$

Definition 5.2.4 (Parameterdatentypdefinition)**.** *Der Kantentyp „Parameterdatentypdefinition“ verbindet einen Parameterknoten mit einem Datentypknoten und beschreibt damit die Definition des Datentyps für diesen Parameter.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{Parameterdatentypdefinition} \Rightarrow$
$v_L.typ = \texttt{Parameter} \wedge$
$v_R.typ \in \{\texttt{PrimitivDatentyp}, \texttt{KompositDatentyp}, \texttt{KollektionsDatentyp}\}$

Definition 5.2.5 (KollektionsDatentypDatentypdefinition)**.** *Der Kantentyp „KollektionsDatentypDatentypdefinition“ verbindet einen Kollektionsdatentypknoten mit einem Datentypknoten und beschreibt damit die Definition des inneren Datentyps für diesen Kollektionsdatentyp.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{KollektionsDatentypDatentypdefinition} \Rightarrow$
$v_L.typ = \texttt{KollektionsDatentyp} \wedge$
$v_R.typ \in \{\texttt{PrimitivDatentyp}, \texttt{KompositDatentyp}, \texttt{KollektionsDatentyp}\}$

Definition 5.2.6 (DatentypParameterDatentypdefinition)**.** *Der Kantentyp „DatentypParameterDatentypdefinition“ verbindet einen Datentypparameterknoten mit einem Datentypknoten und beschreibt damit die Definition des Datentyps für diesen Datentyp-Parameter.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{DatentypParameterDatentypdefinition} \Rightarrow$
$v_L.typ = \texttt{DatentypParameter} \wedge$
$v_R.typ \in \{\texttt{PrimitivDatentyp}, \texttt{KompositDatentyp}, \texttt{KollektionsDatentyp}\}$

Definition 5.2.7 (Schnittstellenangebot)**.** *Der Kantentyp „Schnittstellenangebot“ verläuft von einem Komponentenknoten (d.h. KompositKomponente, BasisKomponente oder System) zu einem Schnittstellenangebotsknoten und beschreibt damit das Angebot einer Schnittstelle durch eine Komponente.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{Schnittstellenangebot} \Rightarrow$
$v_L.typ \in \{\texttt{BasisKomponente}, \texttt{KompositKomponente}, \texttt{System}\} \wedge v_R.typ = \texttt{Schnittstellenangebot}$

Definition 5.2.8 (Schnittstellennachfrage)**.** *Der Kantentyp „Schnittstellennachfrage“ verläuft von einem Komponentenknoten (d.h. KompositKomponente, BasisKomponente oder System) zu einem Schnittstellennachfrageknoten und beschreibt damit die Nachfrage einer Schnittstelle durch eine Komponente.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{Schnittstellennachfrage} \Rightarrow$
$v_L.typ \in \{\texttt{BasisKomponente}, \texttt{KompositKomponente}, \texttt{System}\} \wedge v_R.typ = \texttt{Schnittstellennachfrage}$

Definition 5.2.9 (SchnittstellendefinitionFürSchnittstellenangebot)**.** *Der Kantentyp „SchnittstellendefinitionFürSchnittstellenangebot“ verläuft von einem Schnittstellenangebotsknoten zu einem Schnittstellenknoten und beschreibt damit die Definition der Schnittstelle für das Schnittstellenangebot.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{SchnittstellendefinitionFürSchnittstellenangebot} \Rightarrow$
$v_L.typ = \texttt{Schnittstellenangebot} \wedge v_R.typ = \texttt{Schnittstelle}$

Definition 5.2.10 (SchnittstellendefinitionFürSchnittstellennachfrage)**.** *Der Kantentyp „SchnittstellendefinitionFürSchnittstellennachfrage“ verläuft von einem Schnittstellennachfrageknoten zu einem Schnittstellenknoten und beschreibt damit die Definition der Schnittstelle für die Schnittstellennachfrage.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{SchnittstellendefinitionFürSchnittstellennachfrage} \Rightarrow$
$v_L.typ = \texttt{Schnittstellennachfrage} \wedge$
$v_R.typ = \texttt{Schnittstelle}$

Definition 5.2.11 (Unterkomponentenbildung)**.** *Der Kantentyp „Unterkomponentenbildung“ verläuft von einem Kompositkomponentenknoten zu einem Komponentenknoten und deklariert damit die Komponente als Unterkomponente der Kompositkomponente. Da Kompositkomponenten selbst Unterkomponenten sein können kann das Kantenziel auch eine Kompositkomponente sein.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{Unterkomponentenbildung} \Rightarrow$
$v_L.typ = \texttt{Kompositkomponente} \wedge$
$v_R.typ \in \{\texttt{BasisKomponente}, \texttt{KompositKomponente}, \texttt{System}\}$

Definition 5.2.12 (Assemblierung)**.** *Der Kantentyp „Assemblierung“ verläuft von einem KompositKomponenten-Knoten zu einem Assemblierungskonnektor-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{Assemblierung} \Rightarrow$
$v_L.typ = \texttt{KompositKomponente} \wedge$
$v_R.typ = \texttt{AssemblierungsKonnektor}$

Definition 5.2.13 (AssemblierungskonnektorSchnittstellenangebot)**.** *Der Kantentyp „AssemblierungskonnektorSchnittstellenangebot“ verläuft von einem Assemblierungskonnektor-Knoten zu einem Schnittstellenangebot-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{AssemblierungskonnektorSchnittstellenangebot} \Rightarrow$
$v_L.typ = \texttt{AssemblierungsKonnektor} \wedge$
$v_R.typ = \texttt{Schnittstellenangebot}$

Definition 5.2.14 (AssemblierungskonnektorSchnittstellennachfrage)**.** *Der Kantentyp „AssemblierungskonnektorSchnittstellennachfrage“ verläuft von einem Assemblierungskonnektor-Knoten zu einem Schnittstellennachfrage-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `AssemblierungskonnektorSchnittstellennachfrage` $\Rightarrow$
$v_L.typ =$ `AssemblierungsKonnektor` $\wedge$
$v_R.typ =$ `Schnittstellennachfrage`

Definition 5.2.15 (SchnittstellenangebotsDelegationIntern)**.** *Der Kantentyp „SchnittstellenangebotsDelegationIntern“ verläuft von einem SchnittstellenAngebotsDelegation-Knoten zu einem Schnittstellenangebot-Knoten, welcher ein Schnittstellenangebot innerhalb einer Komposit-Komponente repräsentiert.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `SchnittstellenangebotsDelegationIntern` $\Rightarrow$
$v_L.typ =$ `SchnittstellenAngebotsDelegation` $\wedge$
$v_R.typ =$ `Schnittstellenangebot`

Definition 5.2.16 (SchnittstellenangebotsDelegationExtern)**.** *Der Kantentyp „SchnittstellenangebotsDelegationExtern“ verläuft von einem SchnittstellenAngebotsDelegation-Knoten zu einem Schnittstellenangebot-Knoten, welcher ein Schnittstellenangebot einer Komposit-Komponente repräsentiert.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `SchnittstellenangebotsDelegationExtern` $\Rightarrow$
$v_L.typ =$ `SchnittstellenAngebotsDelegation` $\wedge$
$v_R.typ =$ `Schnittstellenangebot`

Definition 5.2.17 (SchnittstellennachfrageDelegationIntern)**.** *Der Kantentyp „SchnittstellennachfrageDelegationIntern“ verläuft von einem SchnittstellenNachfrageDelegation-Knoten zu einem Schnittstellennachfrage-Knoten, welcher eine Schnittstellennachfrage innerhalb einer Komposit-Komponente repräsentiert.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `SchnittstellennachfrageDelegationIntern` $\Rightarrow$
$v_L.typ =$ `SchnittstellenNachfrageDelegation` $\wedge$
$v_R.typ =$ `Schnittstellennachfrage`

Definition 5.2.18 (SchnittstellennachfrageDelegationExtern)**.** *Der Kantentyp „SchnittstellenangebotsDelegationExtern“ verläuft von einem Schnittstellen-NachfrageDelegation-Knoten zu einem Schnittstellennachfrage-Knoten, welcher eine Schnittstellennachfrage einer Komposit-Komponente repräsentiert.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `SchnittstellennachfrageDelegationExtern` $\Rightarrow$
$v_L.typ =$ `SchnittstellenNachfrageDelegation` $\wedge$
$v_R.typ =$ `Schnittstellennachfrage`

Definition 5.2.19 (KomponentenInterneAbhängigkeitSchnittstellenangebot)**.** *Der Kantentyp „KomponentenInterneAbhängigkeitSchnittstellenangebot“ verläuft von einem KomponentenInterneAbhängigkeit-Knoten zu einem Schnittstellenangebot-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `KomponentenInterneAbhängigkeitSchnittstellenangebot` $\Rightarrow$
$v_L.typ =$ `KomponentenInterneAbhängigkeit` $\wedge$
$v_R.typ =$ `Schnittstellenangebot`

Definition 5.2.20 (KomponentenInterneAbhängigkeitSchnittstellennachfrage)**.** *Der Kantentyp „KomponentenInterneAbhängigkeitSchnittstellennachfrage“ verläuft von einem KomponentenInterneAbhängigkeit-Knoten zu einem Schnittstellennachfrage-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `KomponentenInterneAbhängigkeitSchnittstellennachfrage` $\Rightarrow$
$v_L.typ =$ `KomponentenInterneAbhängigkeit` $\wedge$
$v_R.typ =$ `Schnittstellennachfrage`

Definition 5.2.21 (KomponentenInterneAbhängigkeitAngeboteneOperation)**.** *Der Kantentyp „KomponentenInterneAbhängigkeitAngeboteneOperation“ verläuft von einem KomponentenInterneAbhängigkeit-Knoten zu einem Operation-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `KomponentenInterneAbhängigkeitAngebotene-Operation` $\Rightarrow$
$v_L.typ =$ `KomponentenInterneAbhängigkeit` $\wedge$
$v_R.typ =$ `Operation`

Definition 5.2.22 (KomponentenInterneAbhängigkeitNachgefragteOperation)**.** *Der Kantentyp „KomponentenInterneAbhängigkeitNachgefragteOperation“ verläuft von einem KomponentenInterneAbhängigkeit-Knoten zu einem Operation-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{KomponentenInterneAbhängigkeitNachgefragteOperation} \Rightarrow$
$v_L.typ = \texttt{KomponentenInterneAbhängigkeit} \wedge$
$v_R.typ = \texttt{Operation}$

Definition 5.2.23 (ModifikationskennzeichnungDatentyp)**.** *Der Kantentyp „ModifikationskennzeichnungDatentyp“ verläuft von einem Modifikationskennzeichnung-Knoten zu einem KompositDatentyp-Knoten oder Kollektions-Datentyp-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{ModifikationskennzeichnungDatentyp} \Rightarrow$
$v_L.typ = \texttt{Modifikationskennzeichnung} \wedge$
$v_R.typ \in \{\texttt{KompositDatentyp}, \texttt{KollektionsDatentyp}\}$

Definition 5.2.24 (ModifikationskennzeichnungSchnittstelle)**.** *Der Kantentyp „ModifikationskennzeichnungSchnittstelle“ verläuft von einem Modifikationskennzeichnung-Knoten zu einem Schnittstellen-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{ModifikationskennzeichnungSchnittstelle} \Rightarrow$
$v_L.typ = \texttt{Modifikationskennzeichnung} \wedge$
$v_R.typ = \texttt{Schnittstelle}$

Definition 5.2.25 (ModifikationskennzeichnungKomponente)**.** *Der Kantentyp „ModifikationskennzeichnungKomponente“ verläuft von einem Modifikationskennzeichnung-Knoten zu einem BasisKomponente-Knoten, Komposit-Komponente-Knoten oder System-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ = \texttt{ModifikationskennzeichnungKomponente} \Rightarrow$
$v_L.typ = \texttt{Modifikationskennzeichnung} \wedge$
$v_R.typ \in \{\texttt{BasisKomponente}, \texttt{KompositKomponente}, \texttt{System}\}$

Definition 5.2.26 (ModifikationskennzeichnungSchnittstellenangebot)**.** *Der Kantentyp „ModifikationskennzeichnungSchnittstellenangebot“ verläuft von*

einem Modifikationskennzeichnung-Knoten zu einem Schnittstellenangebot-Knoten.

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$`ModifikationskennzeichnungSchnittstellenangebot`$\Rightarrow$
$v_L.typ =$ `Modifikationskennzeichnung` $\wedge$
$v_R.typ =$ `Schnittstellenangebot`

Definition 5.2.27 (ModifikationskennzeichnungSchnittstellennachfrage)**.** *Der Kantentyp „ModifikationskennzeichnungSchnittstellennachfrage“ verläuft von einem Modifikationskennzeichnung-Knoten zu einem Schnittstellennachfrage-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$`ModifikationskennzeichnungSchnittstellennachfrage`$\Rightarrow$
$v_L.typ =$ `Modifikationskennzeichnung` $\wedge$
$v_R.typ =$ `Schnittstellennachfrage`

Definition 5.2.28 (ModifikationskennzeichnungOperation)**.** *Der Kantentyp „ModifikationskennzeichnungOperation“ verläuft von einem Modifikationskennzeichnung-Knoten zu einem Operation-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$ `ModifikationskennzeichnungOperation` $\Rightarrow$
$v_L.typ =$ `Modifikationskennzeichnung` $\wedge$
$v_R.typ =$ `Operation`

Definition 5.2.29 (ModifikationskennzeichnungAssemblierungskonnektor)**.** *Der Kantentyp „ModifikationskennzeichnungAssemblierungskonnektor“ verläuft von einem Modifikationskennzeichnung-Knoten zu einem Assemblierungskonnektor-Knoten.*

Für eine Kante $e : (v_L, v_R) \in E_{Architektur}$ *gilt:*
$e.typ =$`ModifikationskennzeichnungAssemblierungskonnektor`
$\Rightarrow v_L.typ =$ `Modifikationskennzeichnung` $\wedge$
$v_R.typ =$ `AssemblierungsKonnektor`

5.1.2. Anreicherungsmodell

Im zweiten Vorbereitungsschritt reichert der Anwender das Architekturmodell mit zusätzlichen Informationen an. Das Metamodell und die Graphdefinition

für das Anreicherungsmodell werden in den folgenden Abschnitten beschrieben.

5.1.2.1. Definition des Anreicherungs-Metamodells

Wir definieren das Metamodell für die Anreicherungen wie in den Klassendiagrammen in den Abbildungen 5.3 und 5.4 zu sehen. Alle Metaklassen für Anreicherungen sind von der gemeinsamen abstrakten Oberklasse `ArchitekturmodellAnreicherung` abgeleitet. Anreicherungen werden pro Komponente oder pro Schnittstellenangebot einer Komponente vorgenommen. Die Beschreibung der Metaklassen folgt nachstehend im Zusammenhang mit der Graphdefinition.

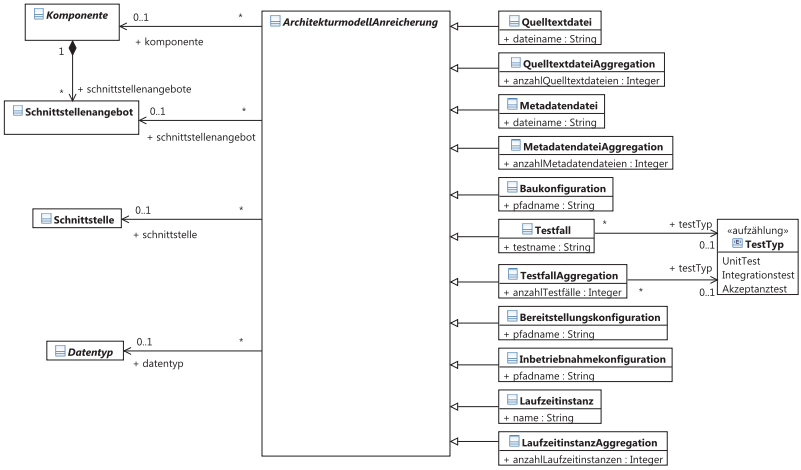


Abbildung 5.3.: Klassendiagramm zur Metamodellierung der Anreicherungen für Tätigkeitsfelder (Teil 1)

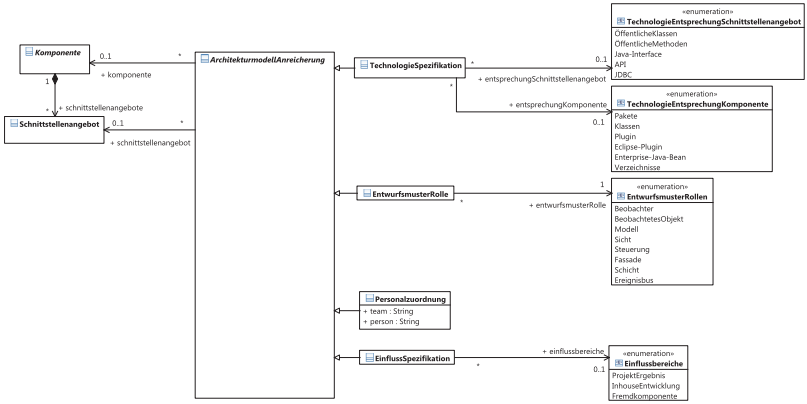


Abbildung 5.4.: Klassendiagramm zur Metamodellierung der Anreicherungen für Tätigkeitsfelder (Teil 2)

5.1.2.2. Graphdefinition

Auf der Basis des Metamodells definieren wir den Anreicherungsgraphen

$$G_{Anreicherungen} = (V_{Anreicherungen}, E_{Anreicherungen})$$

mit der Knotenmenge $V_{Anreicherungen}$ und der Kantenmenge $E_{Anreicherungen}$.

Eigenschaften der Knoten Die Knoten des Anreicherungsgraphen besitzen einen Typ ($v.typ$) und eine (optionale) Bezeichnung ($v.name$). Für die Knotenmenge $V_{Anreicherungen}$ des Anreicherungsgraphen gilt:

$$\forall v \in V_{Anreicherungen} : v.typ \in VT_{Anreicherungen}$$

Dabei ist $VT_{Anreicherungen}$ die Menge der Knotentypen des Anreicherungsgraphen, für die gilt:

$$VT_{Anreicherungen} = \left\{ \begin{array}{c} \texttt{Quelltextdatei}, \\ \texttt{QuelltextdateiAggregation}, \\ \texttt{Metadatendatei}, \\ \texttt{MetadatendateiAggregation}, \\ \texttt{Baukonfiguration}, \\ \texttt{Testfall}, \\ \texttt{TestfallAggregation}, \\ \texttt{Bereitstellungskonfiguration}, \\ \texttt{Inbetriebnahmekonfiguration}, \\ \texttt{Laufzeitinstanz}, \\ \texttt{LaufzeitinstanzAggregation}, \\ \texttt{TechnologieSpezifikation}, \\ \texttt{EntwurfsmusterRolle}, \\ \texttt{Personalzuordnung} \end{array} \right\}$$

Beschreibung und Eigenschaften der Metaklassen bzw. Knotentypen

Definition 5.2.30 (Quelltextdatei)**.** *Der Knotentyp* `Quelltextdatei` *(Metaklasse:* `Quelltextdatei`*) repräsentiert eine einzelne Quelltextdatei, deren Dateiname durch die Eigenschaft* `dateiname` *spezifiziert wird.*

Definition 5.2.31 (Quelltextdatei-Aggregation)**.** *Der Knotentyp* `Quelltextdatei-Aggregation` *(Metaklasse:* `QuelltextdateiAggregation`*) repräsentiert eine Menge von Quelltextdateien, deren Anzahl durch die Eigenschaft* `anzahlQuelltextdateien` *spezifiziert wird.*

Definition 5.2.32 (Metadatendatei)**.** *Der Knotentyp* `Metadatendatei` *(Metaklasse:* `Metadatendatei`*) repräsentiert eine einzelne Metadatendatei, deren Dateiname durch die Eigenschaft* `dateiname` *spezifiziert wird.*

Definition 5.2.33 (Metadatendatei-Aggregation)**.** *Der Knotentyp* `Metadatendatei-Aggregation` *(Metaklasse:* `MetadatendateiAggregation`*) repräsentiert eine Menge von Metadatendateien, deren Anzahl durch die Eigenschaft* `anzahlMetadatendateien` *spezifiziert wird.*

Definition 5.2.34 (Baukonfiguration)**.** *Der Knotentyp* `Baukonfiguration` *(Metaklasse:* `Baukonfiguration`*) repräsentiert die Konfiguration, die zum Bau einer Komponente oder mehrerer Komponenten notwendig ist. Der Pfadname der Baukonfiguration wird durch die Eigenschaft* `pfadname` *spezifiziert.*

Definition 5.2.35 (Testfall)**.** *Der Knotentyp* `Testfall` *(Metaklasse:* `Testfall`*) repräsentiert einen einzelnen Testfall. Der Name des Testfalls wird durch die Eigenschaft* `testname` *spezifiziert. Die Art des Testfalls wird durch die Eigenschaft* `testtyp` *angegeben, dabei gilt*

$v.\texttt{testtyp} \in \{\, \texttt{UnitTest}, \texttt{Integrationstest}, \texttt{Akzeptanztest} \}$

Definition 5.2.36 (Testfall-Aggregation)**.** *Der Knotentyp* `TestfallAggregation` *(Metaklasse:* `TestfallAggregation`*) repräsentiert eine Menge von Testfällen. Die Anzahl der Testfälle wird durch die Eigenschaft* `anzahlTestfälle` *spezifiziert.*

Definition 5.2.37 (Bereitstellungskonfiguration)**.** *Der Knotentyp* `Bereitstellungskonfiguration` *(Metaklasse:* `Bereitstellungskonfiguration`*) repräsentiert die Konfiguration, welche zur Bereitstellung einer Komponente oder mehrerer Komponenten notwendig ist. Der Pfadname der Bereitstellungskonfiguration wird durch die Eigenschaft* `pfadname` *spezifiziert.*

Definition 5.2.38 (Inbetriebnahmekonfiguration)**.** *Der Knotentyp* `Inbetriebnahmekonfiguration` *(Metaklasse:* `Inbetriebnahmekonfiguration`*) repräsentiert die Konfiguration, welche für die Inbetriebnahme einer Komponente oder mehrerer Komponenten notwendig ist. Der Pfadname der Bereitstellungskonfiguration wird durch die Eigenschaft* `pfadname` *spezifiziert.*

Definition 5.2.39 (Laufzeitinstanz)**.** *Der Knotentyp* `Laufzeitinstanz` *(Metaklasse:* `Laufzeitinstanz`*) repräsentiert eine in Betrieb genommene Instanz einer Komponente. Der Name der Laufzeitinstanz wird durch die Eigenschaft* `name` *spezifiziert.*

Definition 5.2.40 (Laufzeitinstanz-Aggregation)**.** *Der Knotentyp* `LaufzeitinstanzAggregation` *(Metaklasse:* `LaufzeitinstanzAggregation`*) repräsentiert eine Menge von in Betrieb genommener Instanzen einer Komponente. Die Anzahl der Laufzeitinstanzen wird durch die Eigenschaft* `anzahlLaufzeitinstanzen` *spezifiziert.*

Definition 5.2.41 (Technologie-Spezifikation)**.** *Der Knotentyp* `TechnologieSpezifikation` *(Metaklasse:* `TechnologieSpezifikation`*) definiert die technische Entsprechung von Komponenten oder Schnittstellenangeboten. Dieser Knotentyp besitzt die Eigenschaften* `entsprechungSchnittstelle` *und* `entsprechungKomponente` *für die gilt:*

$v.\texttt{entsprechungSchnittstelle} \in \{ \texttt{ÖffentlicheKlassen}, \texttt{ÖffentlicheMethoden}, \texttt{JavaInterface}, \texttt{API}, \texttt{JDBC}, \texttt{...} \}$

$v.\texttt{entsprechungKomponente} \in \{ \texttt{Pakete}, \texttt{Klassen}, \texttt{Plugin}, \texttt{Eclipse-Plugin}, \texttt{Enterprise-Java-Bean}, \texttt{Verzeichnisse}, \texttt{...} \}$

Die hier vorgeschlagenen Technologie-Entsprechungen sind ausgewählte Beispiele. Der Anwender kann selbst weitere technische Entsprechungen definieren.

Definition 5.2.42 (Entwurfsmuster-Rolle)**.** *Der Knotentyp* `EntwurfsmusterRolle` *(Metaklasse:* `EntwurfsmusterRolle`*) repräsentiert Entwurfsmuster-Rollen, die Architekturelementen zugeordnet werden. Knoten von diesem Typ besitzen die Eigenschaft* `rollenbezeichnung`*. Das Verfahren schlägt folgende Werte vor:* `Beobachter`*,* `BeobachtetesObjekt`*,* `Modell`*,* `Sicht`*,* `Steuerung`*,* `Fassade` *und* `Ereignisbus`*. Der Anwender kann selbst weitere Rollenbezeichnungen definieren und zuordnen.*

Definition 5.2.43 (Personalzuordnung)**.** *Der Knotentyp* `Personalzuordnung` *(Metaklasse:* `Personalzuordnung`*) definiert Teams und Personen, die Komponenten zugeordnet sind. Knoten von diesem Typ besitzen eine Team-Bezeichnung (Eigenschaft:* `team`*) und eine Personen-Bezeichnung (Eigenschaft:* `person`*).*

Definition 5.2.44 (Einfluss-Spezifikation)**.** *Der Knotentyp* `EinflussSpezifikation` *(Metaklasse:* `EinflussSpezifikation`*) beschreibt den Einfluss, den der Anwender auf das Architekturelement hat. Knoten von diesem Typ besitzen die Eigenschaft* `einflussbereich`*, welche folgende Werte annehmen kann:* `Projektergebnis`*,* `InHouseEntwicklung` *oder* `Fremdkomponente`*.*

Eigenschaften der Kanten Der Anreicherungsgraph annotiert den Architekturgraph. Daher ist die Kantenmenge $E_{Anreicherungen}$ graphübergreifend definiert, für sie gilt:

$\forall e : (v_L, v_R) \in E_{Anreicherungen} : e.v_L \in V_{Anreicherungen} \wedge e.v_R \in V_{Architektur} \wedge$
$e.v_R.typ \in \{\texttt{BasisKomponente}, \texttt{KompositKomponente}, \texttt{System},$
$\texttt{Schnittstellenangebot}\}$

5.1.3. Aktivitätsmodell

Die Änderungsanfragenanalyse liefert als Zwischen- und Endergebnisse Arbeitspläne in Form von Aktivitätslisten. Das Metamodell für die Aktivitätslisten wird im Folgenden beschrieben. In einer Aktivitätsliste werden alle Aktivitäten zur Erfüllung einer Änderungsanfrage gesammelt und sequentiell organisiert. Die Reihenfolge der Aktivitäten wird durch das Verfahren nicht berücksichtigt. Abbildung 5.5 zeigt das Grundmodell eines Arbeitsplans. Die Wurzel des Modells bildet der Arbeitsplan (Metaklasse: `Arbeitsplan`). Im Arbeitsplan sind Aktivitäten (Metaklasse: `Aktivität`) enthalten. Zwischen zwei Aktivitäten kann eine *Reihenfolge-Beziehung* (Vorgänger und Nachfolger) ausgedrückt werden. Zwischen zwei Aktivitäten kann eine *Schachtelungs-Beziehung* in Ober- und Unteraktivität ausgedrückt werden.

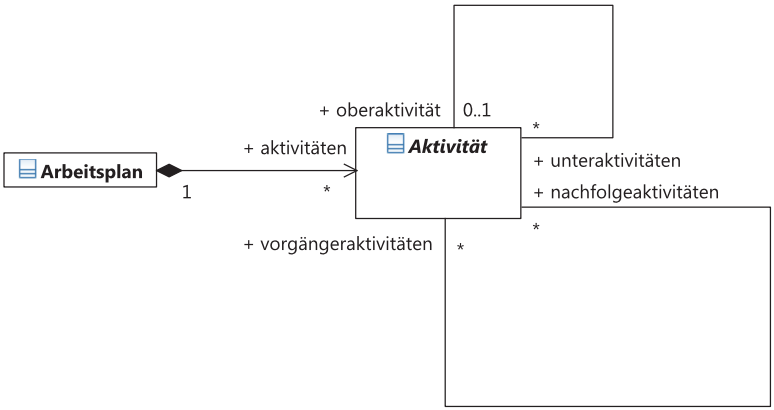


Abbildung 5.5.: Grundmodell des Arbeitsplans

Die Metaklasse `Aktivität` ist abstrakt definiert. Wie in Abbildung 5.6 zu sehen unterscheiden wir zwei grundsätzliche Typen von Aktivitäten. Zum einen *architekturbezogene Aktivitäten* (Metaklasse: `ArchitekturbezogeneAktivität`), die Modifikationen anhand von Elementen der statischen Architektur (d.h. Komponenten, Schnittstellen, etc.) beschreiben und zum anderen *tätigkeitsfeldbezogene Aktivitäten* (Metaklasse: `TätigkeitsfeldbezogeneAktivität`), die Vorgänge in den Tätigkeitsfeldern (d.h. Entwicklung, Bauen, Testen, etc.) beschreiben.

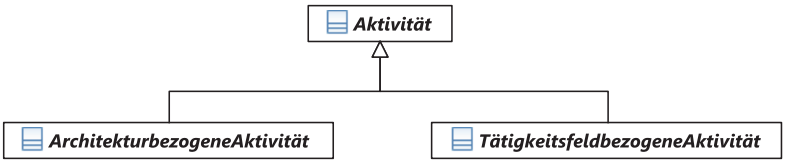


Abbildung 5.6.: Einteilung in architekturbezogene und tätigkeitsfeldbezogene Aktivitäten

5.1.3.1. Architekturbezogene Aktivitäten

Das Klassendiagramm in Abbildung 5.7 gibt einen Überblick der Arten von architekturbezogenen Aktivitäten. Demnach gibt es Aktivitäten, die sich auf Datentypen (Metaklasse: `DatentypAktivität`), Schnittstellen (Metaklasse: `SchnittstellenAktivität`), Operationen von Schnittstellen (Metaklasse: `SchnittstellenOperationAktivität`), Komponenten (Metaklasse: `KomponentenAktivität`), Schnittstellenangebote (Metaklasse: `SchnittstellenAngebotAktivität`), Schnittstellennachfragen (Metaklasse: `SchnittstellenNachfrageAktivität`), angebotene Operationen (Metaklasse: `AngeboteneOperationAktivität`), nachgefragte Operationen (Metaklasse: `NachgefragteOperationAktivität`) und Assemblierungskonnektoren (Metaklasse: `AssemblierungskonnektorAktivität`) beziehen. Diese Metaklassen sind alle abstrakt. Die konkreten Aktivitätstypen werden nachfolgend beschrieben.

Die Aktivitäten zu Datentypen werden im Klassendiagramm in Abbildung 5.8 aufgeführt. Wir unterscheiden die Aktivitäten „Datentyp hinzufügen“ (Meta-

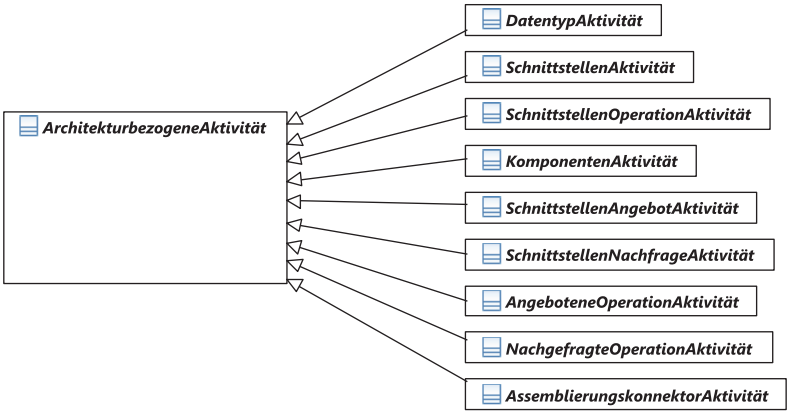


Abbildung 5.7.: Übersicht der architekturbezogenen Aktivitäten

klasse: `DatentypHinzufügen`), „Datentyp entfernen“ (Metaklasse: `DatentypEntfernen`) und „Datentyp modifizieren“ (Metaklasse: `DatentypModifizieren`).

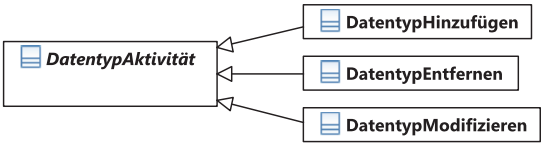


Abbildung 5.8.: Aktivitäten zu Datentypen

Das Klassendiagramm in Abbildung 5.9 führt die Aktivitäten zu Schnittstellen und zu Operationen von Schnittstellen auf. Zu den Schnittstellen-Aktivitäten gehören „Schnittstelle hinzufügen“ (Metaklasse: `SchnittstelleHinzufügen`), „Schnittstelle entfernen“
(Metaklasse: `SchnittstelleEntfernen`) und „Schnittstelle modifizieren“ (Metaklasse: `SchnittstelleModifizieren`). Zu Operationsdefinitionen von Schnittstellen enthält das Metamodell, wie das Klassendiagramm in Abbildung 5.10 zeigt, die Aktivitätstypen „Operationsdefinition hinzufügen“ (Metaklasse:

OperationsDefinitionHinzufügen), Operationsdefinition entfernen (Metaklasse: OperationsDefinitionEntfernen) und „Operationsdefinition modifizieren“ (Metaklasse: OperationsDefinitionModifizieren). Spezielle Untertypen von „Operationsdefinition modifizieren“ sind „Aufrufparameter hinzufügen“ (Metaklasse: AufrufparameterHinzufügen), „Aufrufparameter entfernen“ (Metaklasse: AufrufparameterEntfernen), „Aufrufparameter modifizieren“ (Metaklasse: AufrufparameterModifizieren), „Rückgabeparameter hinzufügen“ (Metaklasse: RückgabeparameterHinzufügen), „Rückgabeparameter entfernen“ (Metaklasse: RückgabeparameterEntfernen) und „Rückgabeparameter modifizieren“ (Metaklasse: RückgabeparameterModifizieren).

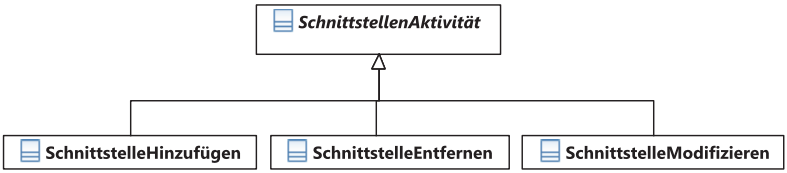


Abbildung 5.9.: Aktivitäten zu Schnittstellen

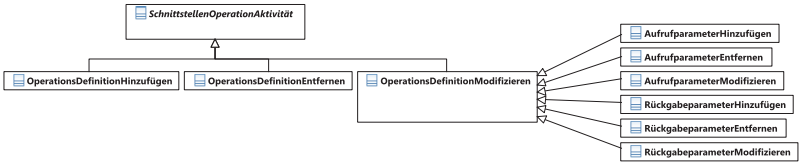


Abbildung 5.10.: Aktivitäten zu Schnittstellenoperationen

Im Klassendiagramm in Abbildung 5.11 sieht man die konkreten Aktivitäten für unterschiedliche Arten von Komponenten. Dazu gehören „Basiskomponente hinzufügen“ (Metaklasse: BasisKomponenteHinzufügen), „Basiskomponente entfernen“ (Metaklasse: BasisKomponenteEntfernen), „Basiskomponente modifizieren“ (Metaklasse: BasisKomponenteModifizieren), „Kompositkomponente hinzufügen“ (Metaklasse: KompositKomponenteHinzufügen), „Kompositkomponente entfernen“ (Metaklasse: KompositKomponenteEntfernen) und „Kompositkomponenten modifizieren“ (Metaklasse: KompositKomponenteModifizieren).

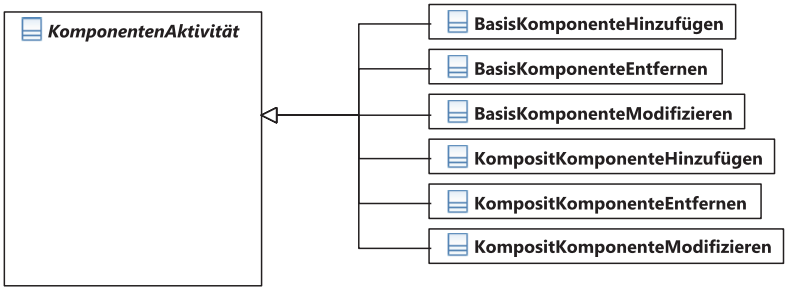


Abbildung 5.11.: Aktivitäten zu Komponenten

Das Klassendiagramm in Abbildung 5.12 nennt die Aktivitäten zu Schnittstellenangeboten. Es umfasst „Schnittstellenangebot hinzufügen“ (Metaklasse: `SchnittstellenAngebotHinzufügen`), „Schnittstellenangebot entfernen“ (Metaklasse: `SchnittstellenAngebotEntfernen`) und „Schnittstellenangebot modifizieren“ (Metaklasse: `SchnittstellenAngebotModifizieren`).

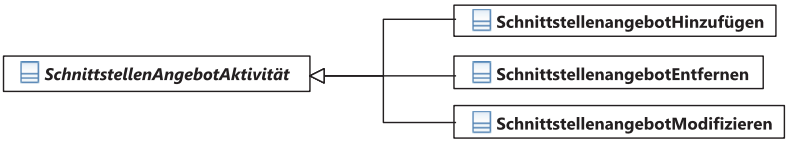


Abbildung 5.12.: Aktivitäten zu Schnittstellenangeboten

Das Klassendiagramm in Abbildung 5.13 nennt die Aktivitäten zu Schnittstellennachfragen. Dazu zählen „Schnittstellennachfrage hinzufügen“ (Metaklasse: `SchnittstellenNachfrageHinzufügen`), „Schnittstellennachfrage entfernen“ (Metaklasse: `SchnittstellenNachfrageEntfernen`) und „Schnittstellennachfrage modifizieren“ (Metaklasse: `SchnittstellenNachfrageModifizieren`).

Das Klassendiagramm in Abbildung 5.14 zeigt die Aktivitäten zu angebotenen Operationen. Hierunter fallen „Angebotene Operation hinzufügen“ (Metaklasse: `AngeboteneOperationHinzufügen`), „Angebotene Operation entfernen“

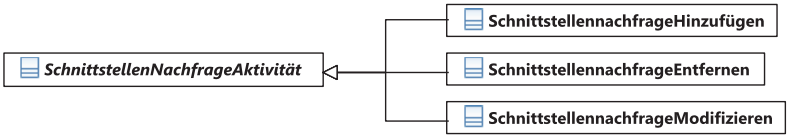


Abbildung 5.13.: Aktivitäten zu Schnittstellennachfragen

(Metaklasse: `AngeboteneOperationEntfernen`) und „Angebotene Operation modifizieren“ (Metaklasse: `AngeboteneOperationModifizieren`).

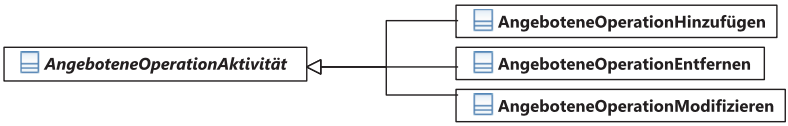


Abbildung 5.14.: Aktivitäten zu angebotenen Operationen

Das Klassendiagramm in Abbildung 5.15 zeigt die Aktivitäten zu nachgefragten Operationen. Dies umfasst „Nachgefragte Operation hinzufügen“ (Metaklasse: `NachgefragteOperationHinzufügen`), „Nachgefragte Operation entfernen“ (Metaklasse: `NachgefragteOperationEntfernen`) und „Nachgefragte Operation modifizieren“ (Metaklasse: `NachgefragteOperationModifizieren`).

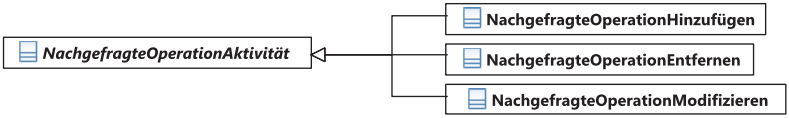


Abbildung 5.15.: Aktivitäten zu nachgefragten Operationen

Die Aktivitäten zu Assemblierungskonnektoren werden im Klassendiagramm in Abbildung 5.16 aufgeführt. Dazu zählen „Assemblierungskonnektor hinzufügen“, „Assemblierungskonnektor entfernen“ und „Assemblierungskonnektor modifizieren“.

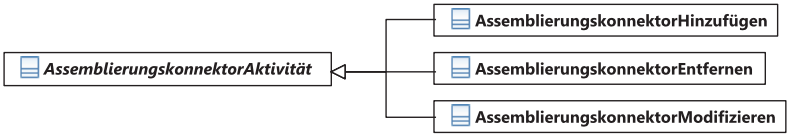


Abbildung 5.16.: Aktivitäten zu Assemblierungskonnektoren

5.1.3.2. Tätigkeitsfeldbezogene Aktivitäten

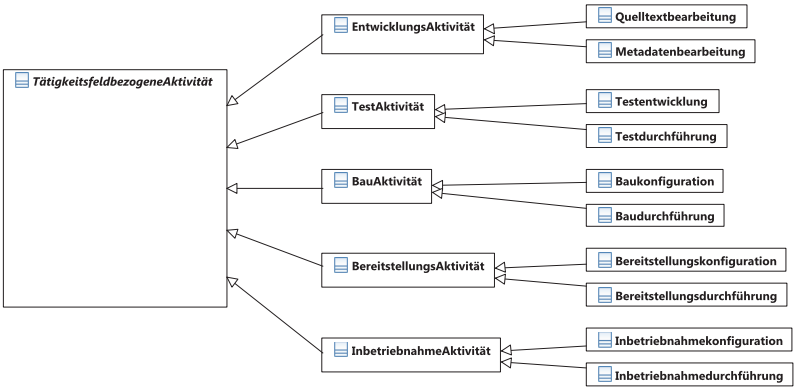


Abbildung 5.17.: Übersicht der tätigkeitsfeldbezogenen Aktivitäten

Das Klassendiagramm in Abbildung 5.17 fasst die tätigkeitsfeldbezogenen Aktivitätstypen zusammen. Zu den Entwicklungsaktivitätstypen (Metaklasse: `EntwicklungsAktivität`) zählen die Quelltextbearbeitung (Metaklasse: `Quelltextbearbeitung`) und die Metadatenbearbeitung (Metaklasse: `Metadatenbearbeitung`). Bei den Testaktivitätstypen (Metaklasse: `TestAktivität`) unterscheiden wir die Testentwicklung (Metaklasse: `Testentwicklung`) und die Testdurchführung (Metaklasse: `Testdurchführung`). Zu den Bauaktivitätstypen (Metaklasse: `BauAktivität`) gehören die Baukonfiguration (Metaklasse: `Baukonfiguration`) und die Baudurchführung (Metaklasse: `Baudurchführung`). Die Bereitstellungsaktivitätstypen (Metaklasse: `BereitstellungsAktivität`) teilen wir ein in Bereitstellungskonfiguration (Metaklasse: `Be-`

reitstellungskonfiguration) und Bereitstellungsdurchführung (Metaklasse: Bereitstellungsdurchführung). Zu den Inbetriebnahmeaktivitätstypen (Metaklasse: InbetriebnahmeAktivität) zählen wir die Inbetriebnahmekonfiguration (Metaklasse: Inbetriebnahmekonfiguration) und die Inbetriebnahmedurchführung (Metaklasse: Inbetriebnahmedurchführung).

5.1.3.3. Definition des Aktivitätsgraphen

Wir definieren den Aktivitätsgraphen $G_{Aktivitäten} = (V_{Aktivitäten}, E_{Aktivitäten})$ mit der Knotenmenge $V_{Aktivitäten}$ und der Kantenmenge $E_{Aktivitäten}$. Die Knoten des Aktivitätsgraphen besitzen einen Typ ($v.typ$) und eine Bezeichnung ($v.name$).

Für die Knotenmenge $V_{Aktivitäten}$ des Aktivitätsgraphen gilt: $\forall v \in V_{Aktivitäten} : v.typ \in VT_{ArchitekturbezogeneAktivitäten} \vee v.typ \in VT_{TätigkeitsfeldbezogeneAktivitäten}$

Dabei sind $VT_{ArchitekturbezogeneAktivitäten}$ und $VT_{TätigkeitsfeldbezogeneAktivitäten}$ die Mengen der Knotentypen des Aktivitätsgraphen, für die gelten:

$VT_{ArchitekturbezogeneAktivitäten} = \{$
DatentypHinzufügen,
DatentypEntfernen, DatentypModifizieren,
SchnittstelleHinzufügen,
SchnittstelleEntfernen, SchnittstelleModifizieren,
OperationsDefinitionHinzufügen,
OperationsDefinitionEntfernen,
AufrufparameterHinzufügen,
AufrufparameterEntfernen,
AufrufparameterModifizieren,
RückgabeparameterHinzufügen,
RückgabeparameterEntfernen,
RückgabeparameterModifizieren,
BasisKomponenteHinzufügen,
BasisKomponenteEntfernen,
BasisKomponenteModifizieren,
KompositKomponenteHinzufügen ,
KompositKomponenteEntfernen,
KompositKomponenteModifizieren,

SchnittstellenangebotHinzufügen,
SchnittstellenangebotEntfernen,
SchnittstellenangebotModifizieren,
SchnittstellennachfrageHinzufügen,
SchnittstellennachfrageEntfernen,
SchnittstellennachfrageModifizieren,
AngeboteneOperationHinzufügen,
AngeboteneOperationEntfernen,
AngeboteneOperationModifizieren,
NachgefragteOperationHinzufügen,
NachgefragteOperationEntfernen,
NachgefragteOperationModifizieren,
AssemblierungskonnektorHinzufügen,
AssemblierungskonnektorEntfernen,
AssemblierungskonnektorModifizieren $\}$

$VT_{TätigkeitsfeldbezogeneAktivitäten} = \{$ Quelltextbearbeitung, Metadatenbearbeitung, Testentwicklung, Testdurchführung, Baukonfiguration, Baudurchführung, Bereitstellungskonfiguration, Bereitstellungsdurchführung, Inbetriebnahmekonfiguration, Inbetriebnahmedurchführung $\}$

5.2. Vorbereitungsschritte

Zu den Vorbereitungsschritten gehören die Architekturmodellierung und die Architekturmodellanreicherung. Die beiden Schritte werden in einem jeweiligen nachfolgenden Abschnitt behandelt.

5.2.1. Architekturmodellierung

In diesem Abschnitt beschreiben wir die Graphoperationen, die zum Aufbau des Architekturgraphen $G_{Architektur} = (V_{Architektur}, E_{Architektur})$ notwendig sind. Eine Übersicht der Graphoperationen geben die Tabellen 5.1 und 5.2.

Die Pseudocode-Repräsentationen der Operationen folgen auf den nächsten Seiten.

Kollektions-Datentyp hinzufügen
Kollektions-Datentyp entfernen
Komposit-Datentyp hinzufügen
Komposit-Datentyp entfernen
Datentyp-Parameter zu Komposit-Datentyp hinzufügen
Datentyp-Parameter von Komposit-Datentyp entfernen
Schnittstelle hinzufügen
Schnittstelle entfernen
Operation zu Schnittstelle hinzufügen
Operation von Schnittstellen entfernen
Aufrufparameter zu Schnittstellenoperation hinzufügen
Aufrufparameter von Schnittstellenoperation entfernen
Rückgabeparameter zu Schnittstellenoperation hinzufügen
Rückgabeparameter von Schnittstellenoperation entfernen
Datentyp für Parameter spezifizieren
Basis-Komponente hinzufügen
Basis-Komponente entfernen
Komposit-Komponente hinzufügen
Komposit-Komponente entfernen

Tabelle 5.1.: Operationen für die Architekturmodellierung (Teil 1)

Operation 5.1 (Kollektions-Datentyp hinzufügen)**.** *Eingabeparameter:* α*=Name des Kollektions-Datentyps,* $v_{InternerDatentyp}$*=Knoten des internen Datentyps*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu} *mit* $v_{neu}.typ =$ `KollektionsDatentyp`*.*
2. *Setze das Namen-Attribut* $v_{neu}.name = \alpha$*.*
3. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*
4. *Erzeuge Kante zwischen Kollektions-Datentyp-Knoten* v_{neu} *und Knoten des internen Datentyps* $v_{InternerDatentyp}$*.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$*, aktualisierte Kantenmenge* $E_{Architektur}$

Operation 5.2 (Kollektions-Datentyp entfernen)**.** *Eingabeparameter:* α*=Name des Kollektions-Datentyps*

Verarbeitungsschritte:

1. *Ermittle Knoten* $v_{KollektionsDatentyp}$ *mit* $v_{KollektionsDatentyp}.typ =$ `KollektionsDatentyp` *und* $v_{KollektionsDatentyp}.name = \alpha$.
2. *Entferne Knoten* $v_{KollektionsDatentyp}$ *aus* $V_{Architektur}$.

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.3 (Komposit-Datentyp hinzufügen)**.**
Eingabeparameter: α*=Name des Komposit-Datentyps*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu} *mit* $v_{neu}.typ =$ `KompositDatentyp`.
2. *Setze das Namen-Attribut* $v_{neu}.name = \alpha$.
3. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.4 (Komposit-Datentyp entfernen)**.**
Eingabeparameter: α*=Name des Komposit-Datentyps*

Verarbeitungsschritte:

1. *Ermittle Knoten* $v_{KompositDatentyp}$ *mit* $v_{KompositDatentyp}.typ =$ `KompositDatentyp` *und* $v_{KompositDatentyp}.name = \alpha$.
2. *Entferne Knoten* $v_{KompositDatentyp}$ *aus* $V_{Architektur}$.

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.5 (Datentyp-Parameter zu Komposit-Datentyp hinzufügen)**.**
Eingabeparameter: α*=Name des Datentyp-Parameters,*
$v_{Datentyp}$*=Knoten des Parameter-Datentyps,*
$v_{KompositDatentyp}$*=Knoten des Komposit-Datentyps*

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu} mit $v_{neu}.typ = \texttt{DatentypParameter}$.*
2. *Setze das Namen-Attribut $v_{neu}.name = \alpha$.*
3. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
4. *Erzeuge Kante zwischen DatentypParameter-Knoten und Knoten des Parameter-Datentyps.*
5. *Erzeuge Kante zwischen DatentypParameter-Knoten und Knoten des Komposit-Datentyps.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.6 (Datentyp-Parameter von Komposit-Datentyp entfernen)**.**
Eingabeparameter: α=Name des Datentyp-Parameters,
$v_{KompositDatentyp}$=Knoten des Komposit-Datentyps

Verarbeitungsschritte:

1. *Ermittle Knoten $v_{DatentypParameter}$ mit $v_{DatentypParameter}.typ = \texttt{DatentypParameter}$ und $v_{DatentypParameter}.name = \alpha$.*
2. *Entferne Kante zwischen DatentypParameter-Knoten und Knoten des Parameter-Datentyps.*
3. *Entferne Kante zwischen DatentypParameter-Knoten und Knoten des Komposit-Datentyps.*
4. *Entferne Knoten $v_{DatentypParameter}$ aus $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.7 (Schnittstelle hinzufügen)**.**
Eingabeparameter: α=Name der Schnittstelle

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}.*
2. *$v_{neu}.typ = \texttt{Schnittstelle}$.*
3. *Setze $v_{neu}.name = \alpha$.*

4. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.8 (Schnittstelle entfernen)**.**
Eingabeparameter: α=Name der Schnittstelle

Verarbeitungsschritte:

1. *Ermittle Knoten $v_{Schnittstelle}$ mit $v_{neu}.typ =$ `Schnittstelle` und $v_{neu}.name = \alpha$.*
2. *Entferne Knoten $v_{Schnittstelle}$ aus $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.9 (Operation zu Schnittstelle hinzufügen)**.**
Eingabeparameter: α=Name der Operation, $v_{Schnittstelle}$=Knoten der Schnittstelle

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}.*
2. *$v_{neu}.typ =$ `Operation`.*
3. *Setze $v_{neu}.name = \alpha$.*
4. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
5. *Erzeuge Kante zwischen Operations-Knoten und Schnittstellen-Knoten.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.10 (Operation von Schnittstelle entfernen)**.**
Eingabeparameter: α=Name der Operation, $v_{Schnittstelle}$=Knoten der Schnittstelle

Verarbeitungsschritte:

1. *Ermittle Knoten $v_{Operation}$ in $V_{Architektur}$ mit $v_{Operation}.typ =$ `Operation` und $v_{Operation}.name = \alpha$ und bestehender Kante zum Schnittstellen-Knoten $v_{Schnittstelle}$.*

2. *Entferne Kante zwischen Operations-Knoten und Schnittstellen-Knoten.*
3. *Entferne Knoten $v_{Operation}$ aus $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.11 (Aufrufparameter zu Schnittstellenoperation hinzufügen)**.** *Eingabeparameter: α=Name des Parameters, $v_{Operation}$=Knoten der Schnittstellenoperation*

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}.*
2. *$v_{neu}.typ =$* `Parameter`*.*
3. *Setze $v_{neu}.name = \alpha$.*
4. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
5. *Erzeuge Kante zwischen Parameter-Knoten v_{neu} und Operations-Knoten $v_{Operation}$ mit Kantentyp* `Aufrufparameterdefinition`*.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.12 (Aufrufparameter von Schnittstellenoperation entfernen)**.** *Eingabeparameter: α=Name des Parameters, $v_{Operation}$=Knoten der Schnittstellenoperation*

Verarbeitungsschritte:

1. *Ermittle Knoten $v_{Parameter}$ in $V_{Architektur}$ mit $v_{Parameter}.typ =$* `Parameter`*. und $v_{Parameter}.name = \alpha$ und bestehender Kante vom Typ* `Aufrufparameterdefinition` *zum Operations-Knoten $v_{Operation}$.*
2. *Entferne Kante zwischen Parameter-Knoten $v_{Parameter}$ und Operations-Knoten $v_{Operation}$.*
3. *Entferne Knoten $v_{Parameter}$ aus $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.13 (Rückgabeparameter zu Schnittstellenoperation hinzufügen)**.**

Eingabeparameter: α=Name des Parameters, $v_{Operation}$=Knoten der Schnittstellenoperation

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}.*
2. *$v_{neu}.typ =$ `Parameter`.*
3. *Setze $v_{neu}.name = \alpha$.*
4. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
5. *Erzeuge Kante zwischen Parameter-Knoten v_{neu} und Operations-Knoten $v_{Operation}$ mit Kantentyp `Rückgabeparameterdefinition`.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.14 (Rückgabeparameter von Schnittstellenoperation entfernen)**.**

Eingabeparameter: α=Name des Parameters, $v_{Operation}$=Knoten der Schnittstellenoperation

Verarbeitungsschritte:

1. *Ermittle Knoten $v_{Parameter}$ in $V_{Architektur}$ mit $v_{Parameter}.typ =$ `Parameter` und $v_{Parameter}.name = \alpha$ und bestehender Kante vom Typ `Rückgabeparameterdefinition` zum Operations-Knoten $v_{Operation}$.*
2. *Entferne Kante zwischen Parameter-Knoten $v_{Parameter}$ und Operations-Knoten $v_{Operation}$.*
3. *Entferne Knoten $v_{Parameter}$ aus $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.15 (Datentyp für Parameter spezifizieren)**.**
Eingabeparameter: $v_{Parameter}$ *=Knoten des Parameters,*
$v_{Datentyp}$*=Knoten des Datentyps*

Verarbeitungsschritt:

> *Erzeuge Kante zwischen Parameter-Knoten* $v_{Parameter}$ *und Datentyp-Knoten* $v_{Datentyp}$ *vom Typ* `Parameterdatentypdefinition`*.*

Ausgabe: Aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.16 (Basis-Komponente hinzufügen)**.**
Eingabeparameter: α*=Name der Komponente*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu}*.*
2. $v_{neu}.typ =$ `BasisKomponente`*.*
3. *Setze* $v_{neu}.name = \alpha$*.*
4. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.17 (Basis-Komponente entfernen)**.**
Eingabeparameter: α*=Name der Komponente*

Verarbeitungsschritte:

1. *Ermittle Knoten* $v_{BasisKomponente}$ *in* $V_{Architektur}$ *mit* $v_{BasisKomponente}.typ =$ `BasisKomponente` *und* $v_{BasisKomponente}.name = \alpha$*.*
2. *Entferne Knoten* $v_{BasisKomponente}$ *aus* $V_{Architektur}$*.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.18 (Komposit-Komponente hinzufügen)**.**
Eingabeparameter: α*=Name der Komponente*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu}*.*
2. $v_{neu}.typ =$ `KompositKomponente`*.*

3. *Setze $v_{neu}.name = \alpha$.*
4. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.19 (Komposit-Komponente entfernen)**.**
Eingabeparameter: α=Name der Komponente

Verarbeitungsschritte:

1. *Ermittle Knoten $v_{KompositKomponente}$ in $V_{Architektur}$ mit $v_{KompositKomponente}.typ =$ `KompositKomponente` und $v_{KompositKomponente}.name = \alpha$.*
2. *Entferne Knoten $v_{KompositKomponente}$ aus $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$

Operation 5.20 (Schnittstellenangebot zu Komponente hinzufügen)**.**
Eingabeparameter: $v_{Schnittstelle}$ = Knoten der Schnittstelle, $v_{Komponente}$ = Knoten der Komponente

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}*
2. *$v_{neu}.typ =$ `Schnittstellenangebot`*
3. *Füge Knoten v_{neu} in $V_{Architektur}$ ein*
4. *Erzeuge Kante zwischen Schnittstellenangebotsknoten und Komponentenknoten mit Kantentyp `Schnittstellenangebot`.*
5. *Erzeuge Kante zwischen Schnittstellenangebotsknoten und Schnittstellenknoten mit Kantentyp `SchnittstellendefinitionFürSchnittstellenangebot`.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.21 (Schnittstellenangebot von Komponente entfernen)**.**
Eingabeparameter: $v_{Schnittstellenangebot}$ = Knoten des Schnittstellenangebots

Verarbeitungsschritte:

1. *Entferne Kante zwischen Schnittstellenangebotsknoten und Komponentenknoten mit Kantentyp* `Schnittstellenangebot`.
2. *Entferne Kante zwischen Schnittstellenangebotsknoten und Schnittstellenknoten mit Kantentyp* `SchnittstellendefinitionFürSchnittstellenangebot`.
3. *Entferne Knoten* $v_{Schnittstellenangebot}$ *aus* $V_{Architektur}$.

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$*, aktualisierte Kantenmenge* $E_{Architektur}$

Operation 5.22 (Schnittstellennachfrage zu Komponente hinzufügen)**.**
Eingabeparameter: $v_{Schnittstelle}$ *= Knoten der Schnittstelle,*
$v_{Komponente}$ *= Knoten der Komponente*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu}.
2. $v_{neu}.typ =$ `Schnittstellennachfrage`.
3. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*
4. *Erzeuge Kante zwischen Schnittstellennachfrageknoten und Komponentenknoten vom Typ* `Schnittstellennachfrage`.
5. *Erzeuge Kante zwischen Schnittstellennachfrageknoten und Schnittstellenknoten vom Typ* `SchnittstellendefinitionFürSchnittstellennachfrage`.

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$*, aktualisierte Kantenmenge* $E_{Architektur}$

Operation 5.23 (Schnittstellennachfrage von Komponente entfernen)**.**
Eingabeparameter: α*=Name der Schnittstellennachfrage*

Verarbeitungsschritte:

1. *Entferne Kante zwischen Schnittstellennachfrageknoten und Komponentenknoten vom Typ* `Schnittstellennachfrage`.
2. *Entferne Kante zwischen Schnittstellennachfrageknoten und Schnittstellenknoten vom Typ* `SchnittstellendefinitionFürSchnittstellennachfrage`.

3. *Entferne Knoten $v_{Schnittstellennachfrage}$ aus $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.24 (Hinzufügen v. Komponenten-Referenz zu Komposit-Komponente)**.**
Eingabeparameter: $v_{KompositKomponente}$ =Knoten der KompositKomponente, $v_{Komponente}$=Knoten der zu referenzierenden Komponente

Verarbeitungsschritt:

> *Erzeuge Kante zwischen KompositKomponenten-Knoten $v_{KompositKomponente}$ und Komponenten-Knoten $v_{Komponente}$ vom Typ* `Unterkomponentenbildung`*.*

Ausgabe: Aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.25 (Entfernen v. Komponenten-Referenz von Komposit-Komponente)**.**

Eingabeparameter: $v_{KompositKomponente}$ =Knoten der KompositKomponente, $v_{Komponente}$=Knoten der referenzierten Komponente

Verarbeitungsschritt:

> *Entferne Kante zwischen KompositKomponenten-Knoten $v_{KompositKomponente}$ und Komponenten-Knoten $v_{Komponente}$ mit Kantentyp* `Unterkomponentenbildung`*.*

Ausgabe: Aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.26 (Assemblierungs-Konnektor z. Komposit-Komponente hinzufügen)**.**
Eingabeparameter: $v_{KompositKomponente}$ = Knoten der Komposit-Komponente, $v_{Schnittstellenangebot}$ = Schnittstellenangebots-Knoten einer Unter-Komponente, $v_{Schnittstellennachfrage}$ = Schnittstellennachfrage-Knoten einer Unter-Komponente.

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}.*

2. $v_{neu}.typ = \texttt{AssemblierungsKonnektor}$.
3. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*
4. *Erzeuge Kante zwischen Komposit-Komponenten-Knoten* $v_{KompositKomponente}$ *und Assemblierungs-Konnektor-Knoten* v_{neu} *vom Typ* `Assemblierung`.
5. *Erzeuge Kante zwischen Schnittstellenangebots-Knoten* $v_{Schnittstellenangebot}$ *und Assemblierungs-Konnektor-Knoten* v_{neu} *vom Typ* `AssemblierungskonnektorSchnittstellenangebot`.
6. *Erzeuge Kante zwischen Schnittstellennachfrage-Knoten* $v_{Schnittstellennachfrage}$ *und Assemblierungs-Konnektor-Knoten* v_{neu} *vom Typ* `AssemblierungskonnektorSchnittstellennachfrage`.

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$*, aktualisierte Kantenmenge* $E_{Architektur}$

Operation 5.27 (Assemblierungs-Konnektor v. Komposit-Komponente entfernen)**.**
Eingabeparameter: $v_{AssemblierungsKonnektor}$ *= Knoten des Assemblierungs-Konnektors*

Verarbeitungsschritte:

1. *Entferne Kante zwischen Komposit-Komponenten-Knoten* $v_{KompositKomponente}$ *und Assemblierungs-Konnektor-Knoten* $v_{AssemblierungsKonnektor}$ *vom Typ* `Assemblierung`.
2. *Entferne Kante zwischen Schnittstellenangebots-Knoten* $v_{Schnittstellenangebot}$ *und Assemblierungs-Konnektor-Knoten* $v_{AssemblierungsKonnektor}$ *vom Typ* `Assemblierungskonnektor-Schnittstellenangebot`.
3. *Entferne Kante zwischen Schnittstellennachfrage-Knoten* $v_{Schnittstellennachfrage}$ *und Assemblierungs-Konnektor-Knoten* $v_{AssemblierungsKonnektor}$ *vom Typ* `Assemblierungskonnektor-Schnittstellennachfrage`.
4. *Entferne Knoten* $v_{AssemblierungsKonnektor}$ *aus* $V_{Architektur}$.

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.28 (Schnittstellenangebotsdelegation z. Komposit-Komp. hinzufügen)**.**
Eingabeparameter: $v_{KompositKomponente}$ = Knoten der Komposit-Komponente, $v_{SchnittstellenangebotExtern}$ = Schnittstellenangebots-Knoten einer Komposit-Komponente, $v_{SchnittstellenangebotIntern}$ = Schnittstellenangebots-Knoten einer Unterkomponente.

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}.*
2. *$v_{neu}.typ =$* `SchnittstellenAngebotsDelegation`*.*
3. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
4. *Erzeuge Kante zwischen Schnittstellenangebotsdelegations-Knoten $v_{SchnittstellenAngebotsDelegation}$ und internem Schnittstellenangebots-Knoten $v_{SchnittstellenangebotIntern}$ vom Typ* `SchnittstellenangebotsDelegationIntern`*.*
5. *Erzeuge Kante zwischen Schnittstellenangebotsdelegations-Knoten $v_{SchnittstellenAngebotsDelegation}$ und externem Schnittstellenangebots-Knoten $v_{SchnittstellenangebotExtern}$ vom Typ* `SchnittstellenangebotsDelegationExtern`*.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.29 (Schnittstellenangebotsdelegation v. Komposit-Komp. entfernen)**.**
Eingabeparameter: $v_{SchnittstellenAngebotsDelegation}$ = Knoten der Schnittstellenangebotsdelegation

Verarbeitungsschritte:

1. *Entferne Kante zwischen Schnittstellenangebotsdelegations-Knoten $v_{SchnittstellenAngebotsDelegation}$ und internem Schnittstellenangebots-Knoten $v_{SchnittstellenangebotIntern}$ vom Typ* `SchnittstellenangebotsDelegationIntern`*.*

2. *Entferne Kante zwischen Schnittstellenangebotsdelegations-Knoten $v_{SchnittstellenAngebotsDelegation}$ und externem Schnittstellenangebots-Knoten $v_{SchnittstellenangebotExtern}$ vom Typ* `Schnittstellenangebots-DelegationExtern`.

3. *Entferne Knoten $v_{SchnittstellenAngebotsDelegation}$ von $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.30 (Schnittstellennachfragedelegation z. Komposit-Komp. hinzufügen)**.**
Eingabeparameter: $v_{KompositKomponente}$ = Knoten der Komposit-Komponente, $v_{SchnittstellennachfrageExtern}$ = Schnittstellennachfrage-Knoten einer Komposit-Komponente, $v_{SchnittstellennachfrageIntern}$ = Schnittstellennachfrage-Knoten einer Unterkomponente.

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}.*

2. *$v_{neu}.typ =$* `SchnittstellenNachfrageDelegation`.

3. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*

4. *Erzeuge Kante zwischen Schnittstellennachfragedelegations-Knoten $v_{SchnittstellenNachfrageDelegation}$ und internem Schnittstellennachfrage-Knoten $v_{SchnittstellennachfrageIntern}$ vom Typ* `SchnittstellennachfrageDelegationIntern`.

5. *Erzeuge Kante zwischen Schnittstellennachfragedelegations-Knoten $v_{SchnittstellenNachfrageDelegation}$ und externem Schnittstellennachfrage-Knoten $v_{SchnittstellennachfrageExtern}$ vom Typ* `SchnittstellennachfrageDelegationExtern`.

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.31 (Schnittstellennachfragedelegation v Komposit-Komp. entfernen)**.**
Eingabeparameter: $v_{SchnittstellenNachfrageDelegation}$ = Knoten der Schnittstellennachfragedelegation

Verarbeitungsschritte:

1. *Entferne Kante zwischen Schnittstellennachfragedelegations-Knoten $v_{SchnittstellenNachfrageDelegation}$ und internem Schnittstellennachfragedelegations-Knoten $v_{Schnittstellennachfrage Intern}$ vom Typ* `SchnittstellennachfrageDelegationIntern`.
2. *Entferne Kante zwischen Schnittstellennachfragedelegations-Knoten $v_{SchnittstellenNachfrageDelegation}$ und externem Schnittstellennachfrage-Knoten $v_{Schnittstellennachfrage Extern}$ vom Typ* `SchnittstellennachfrageDelegationExtern`.
3. *Entferne Knoten $v_{SchnittstellenNachfrageDelegation}$ von $V_{Architektur}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.32 (Komponenten-Interne-Abhängigkeit hinzufügen)**.**
Eingabeparameter: $v_{BasisKomponente}$ = Knoten der Basis-Komponente, $v_{Schnittstellenangebot}$ = Schnittstellenangebots-Knoten, $v_{Schnittstellennachfrage}$ = Schnittstellennachfrage-Knoten.

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu}.*
2. *$v_{neu}.typ =$* `KomponentenInterneAbhängigkeit`.
3. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
4. *Erzeuge Kante zwischen KomponentenInterneAbhängigkeit-Knoten $v_{KomponentenInterneAbhängigkeit}$ und Schnittstellenangebots-Knoten $v_{Schnittstellenangebot}$ vom Typ* `KomponentenInterneAbhängigkeitSchnittstellenangebot`.
5. *Erzeuge Kante zwischen KomponentenInterneAbhängigkeit-Knoten $v_{KomponentenInterneAbhängigkeit}$ und Schnittstellennachfrage-Knoten $v_{Schnittstellennachfrage}$ vom Typ* `KomponentenInterneAbhängigkeitSchnittstellennachfrage`.

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.33 (Komponenten-Interne-Abhängigkeit entfernen)**.**
Eingabeparameter: $v_{KomponentenInterneAbhängigkeit}$ *= Knoten der Komponenten-Internen-Abhängigkeit*

Verarbeitungsschritte:

1. *Entferne Kante zwischen KomponentenInterneAbhängigkeit-Knoten* $v_{KomponentenInterneAbhängigkeit}$ *und Schnittstellenangebots-Knoten* $v_{Schnittstellenangebot}$ *vom Typ* `KomponentenInterneAbhängigkeitSchnittstellenangebot`*.*
2. *Entferne Kante zwischen KomponentenInterneAbhängigkeit-Knoten* $v_{KomponentenInterneAbhängigkeit}$ *und Schnittstellennachfrage-Knoten* $v_{Schnittstellennachfrage}$ *vom Typ* `KomponentenInterneAbhängigkeitSchnittstellennachfrage`*.*
3. *Entferne Knoten* $v_{KomponentenInterneAbhängigkeit}$ *aus* $V_{Architektur}$*.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$*, aktualisierte Kantenmenge* $E_{Architektur}$

5.2.2. Architekturmodellanreicherung

Im zweiten Vorbereitungsschritt wird das Anreicherungsmodell aufgebaut. Dazu stehen dem Anwender verschiedene Operationen zur Verfügung, die wir nachstehend erläutern. Der Anreicherungsgraph

$$G_{Anreicherungen} = (V_{Anreicherungen}, E_{Anreicherungen})$$

wird mit Hilfe der Operationen aufgebaut, die Tabelle 5.3 zusammengefasst sind.

Quelltextdatei spezifizieren
Quelltextdatei-Aggregation spezifizieren
Metadatendatei spezifizieren
Metadatendatei-Aggregation spezifizieren
Baukonfiguration spezifizieren
Testfälle einzeln spezifizieren
Testfall-Aggregation spezifizieren
Bereitstellungskonfiguration spezifizieren
Inbetriebnahmekonfiguration spezifizieren
Laufzeitinstanz spezifizieren
Laufzeitinstanz-Aggregation spezifizieren
Technologie-Spezifikation von Komponente
Technologie-Spezifikation von Schnittstellenangebot
Entwurfsmuster-Rollen spezifizieren
Personalzuordnung spezifizieren
Einfluss-Spezifikation (Drittanbieter-Kennzeichnung)

Tabelle 5.3.: Operationen zur Architektur-Anreicherung

Nun folgen die Pseudocode-Repräsentationen der Operationen.

Operation 5.34 (Quelltextdatei spezifizieren)**.** *Der Anwender spezifiziert die Quelltext-Repräsentation von Komponenten, indem er einen oder mehrere Quelltextdatei-Knoten erstellt. Der Annotationsknoten wird über eine Annotationskante mit dem zu annotierenden Komponentenknoten verbunden.*

Eingabeparameter: α*=Name der Quelltextdatei,*
$v_{Komponente}$*=Komponentenknoten*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{Quelltextdatei}$
2. *Setze* $v_{Quelltextdatei}.typ =$ `Quelltextdatei`
3. *Setze* $v_{Quelltextdatei}.name = \alpha$
4. *Füge Knoten* $v_{Quelltextdatei}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$

6. *Setze* $e_{Annotation}.v_L = v_{Quelltextdatei}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.35 (Quelltextdatei-Aggregation spezifizieren)**.** *Der Anwender spezifiziert die Quelltext-Repräsentation von Komponenten, indem er mit Hilfe eines Quelltextdateiaggregationsknoten eine Anzahl von Quelltextdateien definiert. Der Annotationsknoten wird über eine Annotationskante mit dem zu annotierenden Komponentenknoten verbunden.*

Eingabeparameter: α*=Anzahl der Quelltextdateien,*
$v_{Komponente}$*=Komponentenknoten*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{Quelltextdateiaggregation}$
2. *Setze* $v_{Quelltextdateiaggregation}.typ =$ `QuelltextdateiAggregation`
3. *Setze* $v_{Quelltextdateiaggregation}.anzahl = \alpha$
4. *Füge Knoten* $v_{Quelltextdateiaggregation}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{Quelltextdateiaggregation}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.36 (Metadatendatei spezifizieren)**.** *Der Anwender spezifiziert die Metadaten-Repräsentation von Komponenten, indem er einen oder mehrere MetadatendateiKnoten erstellt. Der Annotationsknoten wird über eine Annotationskante mit dem zu annotierenden Komponentenknoten verbunden.*

Eingabeparameter: α*=Name der Metadatendatei,*
$v_{Komponente}$*=Komponentenknoten*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{Metadatendatei}$
2. *Setze* $v_{Metadatendatei}.typ = \texttt{Metadatendatei}$
3. *Setze* $v_{Metadatendatei}.name = \alpha$
4. *Füge Knoten* $v_{Metadatendatei}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{Metadatendatei}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.37 (Metadatendatei-Aggregation spezifizieren)**.** *Der Anwender spezifiziert die Metadaten-Repräsentation von Komponenten, indem er mit Hilfe eines Metadatendateiaggregationsknoten eine Anzahl von Metadatendateien definiert. Der Annotationsknoten wird über eine Annotationskante mit dem zu annotierenden Komponentenknoten verbunden.*

Eingabeparameter: α*=Anzahl der Metadatendateien,*
$v_{Komponente}$*=Komponentenknoten*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{Metadatendateiaggregation}$
2. *Setze* $v_{Metadatendateiaggregation}.typ = \texttt{MetadatendateiAggregation}$
3. *Setze* $v_{Metadatendateiaggregation}.anzahl = \alpha$
4. *Füge Knoten* $v_{Metadatendateiaggregation}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{Metadatendateiaggregation}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ und aktualisierte Kantenmenge $E_{Anreicherungen}$

Operation 5.38 (Baukonfiguration spezifizieren)**.** *Der Anwender spezifiziert eine Baukonfiguration für eine Menge von Komponenten, indem er einen Knoten v_{neu} mit $v_{neu}.typ = Baukonfiguration$ erstellt, den Dateinamen (bzw. Pfad) der Baukonfiguration setzt und mit Annotationskanten mit den Komponenten verbindet.*

Eingabeparameter: α=Dateiname bzw. Pfad der Baukonfiguration, $V_{Komponenten}$=Gebaute Komponenten

Verarbeitungsschritte:

1. *Erzeuge Knoten $v_{Baukonfiguration}$*
2. *Setze $v_{Baukonfiguration}.typ =$ `Baukonfiguration`*
3. *Setze $v_{Baukonfiguration}.name = \alpha$*
4. *Füge Knoten $v_{Baukonfiguration}$ in $V_{Anreicherungen}$ ein*
5. *Für jede Komponente $v_{Komponente} \in V_{Komponenten}$*
   a) *Erzeuge Kante $e_{Annotation}$*
   b) *Setze $e_{Annotation}.v_L = v_{Baukonfiguration}$*
   c) *Setze $e_{Annotation}.v_R = v_{Komponente}$*
   d) *Füge Kante $e_{Annotation}$ in $E_{Anreicherungen}$ ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ und aktualisierte Kantenmenge $E_{Anreicherungen}$

Operation 5.39 (Testfälle einzeln spezifizieren)**.** *Der Anwender spezifiziert einzelne Testfälle von Komponenten, indem er einen oder mehrere Testfall-Knoten erstellt. Der Annotationsknoten wird über eine Annotationskante mit dem zu annotierenden Komponentenknoten verbunden.*

Eingabeparameter: α=Name des Testfalls, $v_{Komponente}$=Komponentenknoten

Verarbeitungsschritte:

1. *Erzeuge Knoten $v_{Testfall}$*

2. *Setze* $v_{Testfall}.typ = \texttt{Testfall}$
3. *Setze* $v_{Testfall}.name = \alpha$
4. *Füge Knoten* $v_{Testfall}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{Testfall}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.40 (Testfall-Aggregation spezifizieren)**.** *Der Anwender spezifiziert Testfälle von Komponenten, indem er mit Hilfe eines Testfallaggregationsknoten eine Anzahl von Testfällen definiert. Der Annotationsknoten wird über eine Annotationskante mit dem zu annotierenden Komponentenknoten verbunden.*

Eingabeparameter: α*=Anzahl der Testfälle,*
$v_{Komponente}$*=Komponentenknoten*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{Testfallaggregation}$
2. *Setze* $v_{Testfallaggregation}.typ = \texttt{TestfallAggregation}$
3. *Setze* $v_{Testfallaggregation}.anzahl = \alpha$
4. *Füge Knoten* $v_{Testfallaggregation}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{Testfallaggregation}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.41 (Bereitstellungskonfiguration spezifizieren)**.** *Der Anwender spezifiziert eine Bereitstellungskonfiguration für eine Menge von Komponenten, indem er einen Knoten v_{neu} mit $v_{neu}.typ = Bereitstellungskonfiguration$ erstellt, den Dateinamen (bzw. Pfad) der Baukonfiguration setzt und mit Annotationskanten mit den Komponenten verbindet.*

Eingabeparameter: α=Dateiname bzw. Pfad der Bereitstellungskonfiguration, $V_{Komponenten}$=Bereitgestellte Komponenten

Verarbeitungsschritte:

1. *Erzeuge Knoten $v_{Bereitstellungskonfiguration}$*
2. *Setze $v_{Bereitstellungskonfiguration}.typ =$* `Bereitstellungskonfiguration`
3. *Setze $v_{Bereitstellungskonfiguration}.name = \alpha$*
4. *Füge Knoten $v_{Bereitstellungskonfiguration}$ in $V_{Anreicherungen}$ ein*
5. *Für jede Komponente $v_{Komponente} \in V_{Komponenten}$*
   a) *Erzeuge Kante $e_{Annotation}$*
   b) *Setze $e_{Annotation}.v_L = v_{Bereitstellungskonfiguration}$*
   c) *Setze $e_{Annotation}.v_R = v_{Komponente}$*
   d) *Füge Kante $e_{Annotation}$ in $E_{Anreicherungen}$ ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ und aktualisierte Kantenmenge $E_{Anreicherungen}$

Operation 5.42 (Inbetriebnahmekonfiguration spezifizieren)**.** *Der Anwender spezifiziert eine Inbetriebnahmekonfiguration für eine Menge von Komponenten, indem er einen Knoten v_{neu} mit $v_{neu}.typ = Inbetriebnahmekonfiguration$ erstellt, den Dateinamen (bzw. Pfad) der Baukonfiguration setzt und mit Annotationskanten mit den Komponenten verbindet.*

Eingabeparameter: α=Dateiname bzw. Pfad der Inbetriebnahmekonfiguration,
$V_{Komponenten}$=In Betrieb genommene Komponenten

Verarbeitungsschritte:

1. *Erzeuge Knoten $v_{Inbetriebnahmekonfiguration}$*

2. *Setze* $v_{Inbetriebnahmekonfiguration}.typ = \texttt{Inbetriebnahmekonfiguration}$
3. *Setze* $v_{Inbetriebnahmekonfiguration}.name = \alpha$
4. *Füge Knoten* $v_{Inbetriebnahmekonfiguration}$ *in* $V_{Anreicherungen}$ *ein*
5. *Für jede Komponente* $v_{Komponente} \in V_{Komponenten}$
   a) *Erzeuge Kante* $e_{Annotation}$
   b) *Setze* $e_{Annotation}.v_L = v_{Inbetriebnahmekonfiguration}$
   c) *Setze* $e_{Annotation}.v_R = v_{Komponente}$
   d) *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.43 (Laufzeitinstanzen spezifizieren)**.** *Der Anwender spezifiziert Laufzeitinstanzen von Komponenten, indem er einen oder mehrere Laufzeitinstanz-Knoten erstellt. Der Annotationsknoten wird über eine Annotationskante mit dem zu annotierenden Komponentenknoten verbunden.*

Eingabeparameter: α*=Name der Laufzeitinstanz,*
$v_{Komponente}$*=Komponentenknoten*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{Laufzeitinstanz}$
2. *Setze* $v_{Laufzeitinstanz}.typ = \texttt{Laufzeitinstanz}$
3. *Setze* $v_{Laufzeitinstanz}.name = \alpha$
4. *Füge Knoten* $v_{Laufzeitinstanz}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{Laufzeitinstanz}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.44 (Laufzeitinstanz-Aggregation spezifizieren)**.** *Der Anwender spezifiziert Laufzeitinstanzen von Komponenten, indem er mit Hilfe eines Laufzeitinstanzaggregationsknoten eine Anzahl von Laufzeitinstanzen definiert. Der Annotationsknoten wird über eine Annotationskante mit dem zu annotierenden Komponentenknoten verbunden.*

Eingabeparameter: α*=Anzahl der Laufzeitinstanzen,* $v_{Komponente}$*=Komponentenknoten*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{Laufzeitinstanzaggregation}$
2. *Setze* $v_{Laufzeitinstanzaggregation}.typ =$ `LaufzeitinstanzAggregation`
3. *Setze* $v_{Laufzeitinstanzaggregation}.anzahl = \alpha$
4. *Füge Knoten* $v_{Laufzeitinstanzaggregation}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{Laufzeitinstanzaggregation}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.45 (Technologie-Spezifikation einer Komponente)**.** *Der Anwender spezifiziert die Technologie einer Komponente, indem er einen Technologie-Spezifikation-Knoten erstellt und mit einer Annotationskante mit dem Komponentenknoten verbindet.*

Eingabeparameter: α*=Technologie-Bezeichnung der Komponente,* $v_{Komponente}$*=Komponentenknoten,*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{TechnologieSpezifikation}$
2. *Setze* $v_{TechnologieSpezifikation}.typ =$ `TechnologieSpezifikation`
3. *Setze* $v_{TechnologieSpezifikation}.entsprechungKomponente = \alpha$

4. *Füge Knoten $v_{TechnologieSpezifikation}$ in $V_{Anreicherungen}$ ein*
5. *Erzeuge Kante $e_{Annotation}$*
6. *Setze $e_{Annotation}.v_L = v_{TechnologieSpezifikation}$*
7. *Setze $e_{Annotation}.v_R = v_{Komponente}$*
8. *Füge Kante $e_{Annotation}$ in $E_{Anreicherungen}$ ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ und aktualisierte Kantenmenge $E_{Anreicherungen}$

Operation 5.46 (Technologie-Spezifikation eines Schnittstellenangebots)**.** *Der Anwender spezifiziert die Technologie eines Schnittstellenangebots, indem er einen Technologie-Spezifikation-Knoten erstellt und mit einer Annotationskante mit dem Schnittstellenangebotsknoten verbindet.*

Eingabeparameter: α=Technologie-Bezeichnung des Schnittstellenangebots $v_{Schnittstellenangebot}$=Schnittstellenangebotsknoten,

Verarbeitungsschritte:

1. *Erzeuge Knoten $v_{TechnologieSpezifikation}$*
2. *Setze $v_{TechnologieSpezifikation}.typ =$* `TechnologieSpezifikation`
3. *Setze $v_{TechnologieSpezifikation}.entsprechungSchnittstellenangebot = \alpha$*
4. *Füge Knoten $v_{TechnologieSpezifikation}$ in $V_{Anreicherungen}$ ein*
5. *Erzeuge Kante $e_{Annotation}$*
6. *Setze $e_{Annotation}.v_L = v_{TechnologieSpezifikation}$*
7. *Setze $e_{Annotation}.v_R = v_{Komponente}$*
8. *Füge Kante $e_{Annotation}$ in $E_{Anreicherungen}$ ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ und aktualisierte Kantenmenge $E_{Anreicherungen}$

Operation 5.47 (Entwurfsmuster-Rollen spezifizieren)**.** *Der Anwender spezifiziert die*
Entwurfsmuster-Rolle einer Komponente, indem er einen Entwurfsmuster-Rolle-Knoten erstellt und mit einer Annotationskante mit dem Komponentenknoten verbindet.

Eingabeparameter: α*=Bezeichnung der Entwurfsmuster-Rolle* $v_{Komponente}$*=Komponentenknoten,*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{EntwurfsmusterRolle}$
2. *Setze* $v_{EntwurfsmusterRolle}.typ = \texttt{TechnologieSpezifikation}$
3. *Setze* $v_{EntwurfsmusterRolle}.entwurfsmusterRolle = \alpha$
4. *Füge Knoten* $v_{EntwurfsmusterRolle}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{EntwurfsmusterRolle}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.48 (Personalzuordnung spezifizieren)**.** *Der Anwender spezifiziert das zuständige Personal einer Komponente, indem er einen Personalzuordnungs-Knoten erstellt und mit einer Annotationskante mit dem Komponentenknoten verbindet.*

Eingabeparameter: α*=Name des Teams,* β*=Name der Person,* $v_{Komponente}$*=Komponentenknoten,*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{Personalzuordnung}$
2. *Setze* $v_{Personalzuordnung}.typ = \texttt{Personalzuordnung}$
3. *Setze* $v_{Personalzuordnung}.team = \alpha$

4. *Setze* $v_{Personalzuordnung}.person = \beta$
5. *Füge Knoten* $v_{Personalzuordnung}$ *in* $V_{Anreicherungen}$ *ein*
6. *Erzeuge Kante* $e_{Annotation}$
7. *Setze* $e_{Annotation}.v_L = v_{Personalzuordnung}$
8. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
9. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Operation 5.49 (Einfluss-Spezifikation)**.** *Der Anwender spezifiziert den Einflussbereich über eine Komponente, indem er einen EinflussSpezifikations-Knoten erstellt und mit einer Annotationskante mit dem Komponentenknoten verbindet.*

Eingabeparameter: α*=Einflussbereichs-Bezeichnung,* $v_{Komponente}$*=Komponentenknoten*

Verarbeitungsschritte:

1. *Erzeuge Knoten* $v_{EinflussSpezifikation}$
2. *Setze* $v_{EinflussSpezifikation}.typ =$ `EinflussSpezifikation`
3. *Setze* $v_{EinflussSpezifikation}.einflussbereich = \alpha$
4. *Füge Knoten* $v_{EinflussSpezifikation}$ *in* $V_{Anreicherungen}$ *ein*
5. *Erzeuge Kante* $e_{Annotation}$
6. *Setze* $e_{Annotation}.v_L = v_{EinflussSpezifikation}$
7. *Setze* $e_{Annotation}.v_R = v_{Komponente}$
8. *Füge Kante* $e_{Annotation}$ *in* $E_{Anreicherungen}$ *ein.*

Ausgabe: Aktualisierte Knotenmenge $V_{Anreicherungen}$ *und aktualisierte Kantenmenge* $E_{Anreicherungen}$

Schnittstellenangebot zu Komponente (Basis-Komponente oder Komposit-Komponente) hinzufügen
Schnittstellenangebot von Komponente (Basis-Komponente oder Komposit-Komponente) entfernen
Schnittstellennachfrage zu Komponente (Basis-Komponente oder Komposit-Komponente) hinzufügen
Schnittstellennachfrage von Komponente (Basis-Komponente oder Komposit-Komponente) entfernen
Hinzufügen von Komponenten-Referenz zu Komposit-Komponente
Entfernen von Komponenten-Referenz von Komposit-Komponente
Assemblierungs-Konnektor zur Komposit-Komponente hinzufügen
Assemblierungs-Konnektor von Komposit-Komponente entfernen
Schnittstellenangebotsdelegation zur Komposit-Komponente hinzufügen
Schnittstellenangebotsdelegation von Komposit-Komponente entfernen
Schnittstellennachfragedelegation zur Komposit-Komponente hinzufügen
Schnittstellennachfragedelegation von Komposit-Komponente entfernen
Komponenten-Interne-Abhängigkeit hinzufügen
Komponenten-Interne-Abhängigkeit entfernen

Tabelle 5.2.: Operationen für die Architekturmodellierung (Teil 2)

5.3. Änderungsanfragen-Analyse

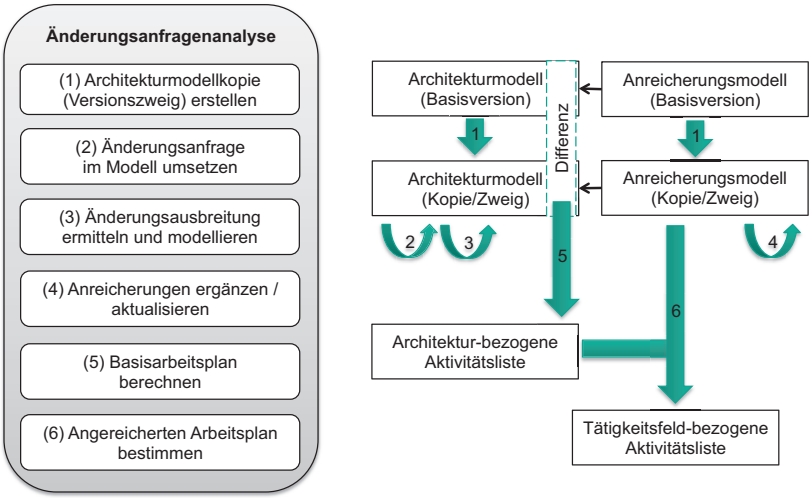


Abbildung 5.18.: Ablauf der Änderungsanfragenanalyse

In den folgenden Abschnitten beschreiben wir die Operationen, die im Rahmen der Änderungsanfragenanalyse eingesetzt werden. Wie in Abbildung 5.18 zu sehen, besteht die Änderungsanfragenanalyse aus sechs Schritten. Jeder Schritt wird nachstehend jeweils in einem eigenen Abschnitt behandelt.

5.3.1. Versionsbildung

Das Architekturmodell und das Anreicherungsmodell aus den Vorbereitungsschritten dienen als Eingabe in die Änderungsanfragenanalyse. Im ersten Schritt wird werkzeug-gestützt eine Kopie dieser Modelle erzeugt. Diese Kopie dient als neue Version der Ausgangsmodelle. In dieser neuen Version wird in den späteren Schritten die Zielarchitektur modelliert. Technisch betrachtet entspricht die Erzeugung der Kopie einer Duplizierung der Knoten und Kanten des Architektur- und Anreicherungsgraphen bzw. der Modellelemente, ihrer Attribute und Referenzen. Wir beschreiben das Duplizierungsverfahren

als Pseudocode-Algorithmus auf Basis der eingeführten Graphrepräsentation der Modelle.

Eingabeparameter:
Architekturgraph $G_{Architektur} = (V_{Architektur}, E_{Architektur})$ mit Knotenmenge $V_{Architektur}$ und Kantenmenge $E_{Architektur}$

Verarbeitungsschritte im Pseudocode:

1. Erzeuge leere Knotenmenge $V'_{Architektur}$ und leere Kantenmenge $E'_{Architektur}$
2. Erzeuge Hilfskantenmenge E_{Spur}
3. Knoten duplizieren:
   Für alle Knoten $v_{Architektur} \in V_{Architektur}$
   a) Erzeuge Knoten $v'_{Architektur}$
   b) Übertrage alle Attributwerte (incl. `id`) von $v_{Architektur}$ auf $v'_{Architektur}$
   c) Füge $v'_{Architektur}$ in $V'_{Architektur}$ ein
   d) Erzeuge Spur-Kante $(v_{Architektur}, v'_{Architektur})$ und füge sie in E_{Spur} ein
4. Kanten duplizieren:
   Für alle Kanten $e_{Architektur} \in E_{Architektur}$
   a) Bestimme mit Hilfe von E_{Spur} den korrespondierenden Knoten zu $e_{Architektur}.v_L \rightarrow v'_L$
   b) Bestimme mit Hilfe von E_{Spur} den korrespondierenden Knoten zu $e_{Architektur}.v_R \rightarrow v'_R$
   c) Erzeuge Kante $e'_{Architektur} = (v'_L, v'_R)$
   d) Übertrage alle Attributwerte (incl. `id`) von $e_{Architektur}$ auf $e'_{Architektur}$
   e) Füge $e'_{Architektur}$ in $E'_{Architektur}$ ein

Ausgabe: Duplizierter Architekturgraph $G'_{Architektur}$ mit Knotenmenge $V'_{Architektur}$ und Kantenmenge $E'_{Architektur}$

Das Verfahren wird analog auf $G_{Anreicherungen} = (V_{Anreicherungen}, E_{Anreicherungen})$ angewendet und liefert damit den duplizierten Anreicherungsgraphen $G'_{Anreicherungen} = (V'_{Anreicherungen}, E'_{Anreicherungen})$.

5.3.2. Umsetzungsweg modellieren

Im zweiten Schritt der Änderungsanfragenanalyse modelliert der Anwender die Zielarchitektur nach Umsetzung der Änderungsanfrage. Das heißt er modelliert den Umsetzungsweg in der duplizierten Version des Architekturmodells. Bezogen auf die Graphrepräsentation der Modelle bedeutet dies, dass Operationen auf dem duplizierten Architekturgraphen $G'_{Architektur}$ durchgeführt werden. Um die Änderungsanfrage im Architekturgraph umzusetzen, stehen dem Anwender zum Einen die Operationen aus der Architekturmodellierung in der Vorbereitungsphase zur Verfügung (siehe Abschnitt 5.2.1) und zum Anderen Operationen zur Kennzeichnung interner Modifikationen, die in Tabelle 5.4 aufgeführt werden.

Modifikationskennzeichnung an Komponente anbringen
Modifikationskennzeichnung an Schnittstellenangebot anbringen
Modifikationskennzeichnung an angebotener Operation anbringen
Modifikationskennzeichnung an Assemblierungskonnektor anbringen
Modifikationskennzeichnung an Schnittstellennachfrage anbringen
Modifikationskennzeichnung an Datentyp anbringen
Modifikationskennzeichnung an Schnittstelle anbringen

Tabelle 5.4.: Modellierungsoperationen zur Modifikationskennzeichnung

Operation 5.50 (Modifikationskennzeichnung an Komponente anbringen)**.**
Eingabeparameter: $v_{Komponente}$*=Knoten der zu kennzeichnenden Komponente*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu} *mit* $v_{neu}.typ =$ `Modifikationskennzeichnung`*.*
2. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*

3. *Erzeuge Kante zwischen Modifikationskennzeichnungs-Knoten v_{neu} und Knoten der zu kennzeichnenden Komponente $v_{Komponente}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.51 (Modifikationskennz. an Schnittstellenangebot anbringen)**.**
Eingabeparameter: $v_{Schnittstellenangebot}$=Knoten des zu kennzeichnenden Schnittstellenangebots

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu} mit $v_{neu}.typ =$* `Modifikationskennzeichnung`*.*
2. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
3. *Erzeuge Kante zwischen Modifikationskennzeichnungs-Knoten v_{neu} und Knoten des zu kennzeichnenden Schnittstellenangebots $v_{Schnittstellenangebot}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.52 (Modifikationskennz. an angebotener Operation anbringen)**.**

Eingabeparameter: $v_{Schnittstellenangebot}$=Knoten des zu kennzeichnenden Schnittstellenangebots,
$v_{Operation}$=Knoten der zu kennzeichnenden angebotenen Operation

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu} mit $v_{neu}.typ =$* `Modifikationskennzeichnung`*.*
2. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
3. *Erzeuge Kante zwischen Modifikationskennzeichnungs-Knoten v_{neu} und Knoten des zu kennzeichnenden Schnittstellenangebots $v_{Schnittstellenangebot}$.*
4. *Erzeuge Kante zwischen Modifikationskennzeichnungs-Knoten v_{neu} und Knoten der zu kennzeichnenden angebotenen Operation $v_{Operation}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.53 (Modifikationskennz. an Assemblierungskonnektor anbringen)**.**
Eingabeparameter: $v_{Assemblierungskonnektor}$*=Knoten des zu kennzeichnenden Assemblierungskonnektors*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu} *mit* $v_{neu}.typ =$ `Modifikationskennzeichnung`*.*
2. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*
3. *Erzeuge Kante zwischen Modifikationskennzeichnungs-Knoten* v_{neu} *und Knoten des Assemblierungskonnektors* $v_{Assemblierungskonnektor}$*.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$*, aktualisierte Kantenmenge* $E_{Architektur}$

Operation 5.54 (Modifikationskennz. an Schnittstellennachfrage anbringen)**.**

Eingabeparameter: $v_{Schnittstellennachfrage}$*=Knoten der zu kennzeichnenden Schnittstellennachfrage*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu} *mit* $v_{neu}.typ =$ `Modifikationskennzeichnung`*.*
2. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*
3. *Erzeuge Kante zwischen Modifikationskennzeichnungs-Knoten* v_{neu} *und Knoten der Schnittstellennachfrage* $v_{Schnittstellennachfrage}$*.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$*, aktualisierte Kantenmenge* $E_{Architektur}$

Operation 5.55 (Modifikationskennzeichnung an Datentyp anbringen)**.**
Eingabeparameter: $v_{Datentyp}$*=Knoten des zu kennzeichnenden Datentyps*

Verarbeitungsschritte:

1. *Erzeuge Knoten* v_{neu} *mit* $v_{neu}.typ =$ `Modifikationskennzeichnung`*.*
2. *Füge Knoten* v_{neu} *in* $V_{Architektur}$ *ein.*

3. *Erzeuge Kante zwischen Modifikationskennzeichnungs-Knoten v_{neu} und Knoten des Datentyps $v_{Datentyp}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

Operation 5.56 (Modifikationskennzeichnung an Schnittstelle anbringen)**.**
Eingabeparameter: $v_{Schnittstelle}$=Knoten der zu kennzeichnenden Schnittstelle

Verarbeitungsschritte:

1. *Erzeuge Knoten v_{neu} mit $v_{neu}.typ =$* `Modifikationskennzeichnung` *.*
2. *Füge Knoten v_{neu} in $V_{Architektur}$ ein.*
3. *Erzeuge Kante zwischen Modifikationskennzeichnungs-Knoten v_{neu} und Knoten der Schnittstelle $v_{Schnittstelle}$.*

Ausgabe: Aktualisierte Knotenmenge $V_{Architektur}$, aktualisierte Kantenmenge $E_{Architektur}$

5.3.3. Änderungsausbreitungsanalyse

Bei der Modellierung des Umsetzungswegs sollte der Anwender die strukturelle Ausbreitung der Änderung innerhalb von Komponenten und zwischen den Komponenten berücksichtigen. Dafür schlägt das Verfahren eine werkzeug-gestützte Änderungsausbreitungsanalyse vor. Die Änderungsausbreitungsanalyse weist ausgehend von einer vorgenommenen Modifikation an Komponenten oder an deren Schnittstellenangeboten auf weitere potenziell betroffene Komponenten und Schnittstellenangebote hin.

In unserem Modell sind Datentypen und Schnittstellen als Hauptentitäten vorgesehen, die wie Komponenten mit Modifikationskennzeichnungen versehen werden können oder an denen Modifikationen im Modell vorgenommen werden können. Da durch die Veränderung von Datentypen auch die davon abhängigen Schnittstellen betroffen sind, und Veränderungen an Schnittstellen wiederum zu möglichen Änderungen an davon abhängigen Komponenten führen, sieht unser Verfahren zwei weitere vorgelagerte Schritte vor. Erstens wird die Änderungsausbreitung von Datentypen zu Schnittstellen ermittelt und zweitens wird die Änderungsausbreitung von Schnittstellen zu Komponenten, d.h. zu Schnittstellenangeboten und -nachfragen berechnet.

Das Verfahren kann wie folgt im Pseudocode zusammengefasst werden.

Eingabe: Modifizierter Architekturgraph $G'_{Architektur} = (V'_{Architektur}, E'_{Architektur})$ mit Knotenmenge $V'_{Architektur}$ und Kantenmenge $E'_{Architektur}$

Vorgehensbeschreibung im Pseudocode:

1. Änderungsausbreitung von Datentypen zu Schnittstellen berechnen.
   a) Modifizierte Datentypen bestimmen.
   b) Für jeden modifizierten Datentyp: Schnittstellen ermitteln, welche einen Aufrufparameter oder Rückgabeparameter vom Typ des Datentyps haben und mit Modifikationskennzeichnungen versehen.
   c) Zwischenergebnis dem Anwender präsentieren. Optionale manuelle Korrektur des Vorschlags durch Anwender.
2. Änderungsausbreitung von Schnittstellen zu Schnittstellenangeboten und -nachfragen berechnen.
   a) Modifizierte Schnittstellen bestimmen.
   b) Für jede modifizierte Schnittstelle: Schnittstellenangebote und -nachfragen ermitteln, welche konform zur Schnittstelle sind und mit Modifikationskennzeichnungen versehen.
   c) Zwischenergebnis dem Anwender präsentieren. Optionale manuelle Korrektur des Vorschlags durch Anwender.
3. Änderungsausbreitung zwischen und innerhalb von Komponenten berechnen.

   Solange bis keine weiteren Modifikationen hinzukommen:
   a) Inter-Komponenten-Ausbreitung berechnen.

i. Modifizierte Schnittstellenangebote bestimmen. Dazu zählen alle Schnittstellenangebote, von denen Operationen entfernt werden oder zu denen Operationen hinzugefügt werden und solche, welche mit einer Modifikationskennzeichnung versehen sind.

ii. Verbundene Assemblierungskonnektoren ermitteln und mit Modifikationskennzeichnung versehen.

iii. Zwischenergebnis dem Anwender präsentieren. Optionale manuelle Korrektur des Vorschlags durch Anwender.

b) Intra-Komponenten-Ausbreitung berechnen.

i. Modifizierte Schnittstellennachfragen bestimmen. Dazu zählen alle Schnittstellennachfragen, die mit einem Assemblierungskonnektor verbunden sind, der mit einer Modifikationskennzeichnung versehen ist. Bereits betrachtete Schnittstellennachfragen werden nicht mehr beachtet.

ii. Bei Basis-Komponenten: Komponenten-Interne-Abhängigkeiten verfolgen und verbundene Schnittstellenangebote mit Modifikationskennzeichnungen versehen.
Bei Komposit-Komponenten: Delegationskonnektoren verfolgen und verbundene Schnittstellennachfragen der Unterkomponenten mit Modifikationskennzeichnungen versehen.

iii. Zwischenergebnis dem Anwender präsentieren. Optionale manuelle Korrektur des Vorschlags durch Anwender.

Ausgabe: Ein um weitere Modifikationskennzeichnungen ergänzter modifizierter Architekturgraph $G'_{Architektur} = (V'_{Architektur}, E'_{Architektur})$.

5.3.4. Anreicherungen ergänzen und aktualisieren

Durch die Modifikationen des Architekturgraphen $G'_{Architektur}$ in den vorherigen Schritten kann es vorkommen, dass Anreicherungen ungültig werden oder neue Anreicherungen ergänzt werden müssen. Deshalb werden im vierten Schritt Anreicherungen ergänzt und aktualisiert.

Zur Aktualisierung und Ergänzung des Anreicherungsmodells repräsentiert durch den Graphen $G'_{Anreicherungen}$ stehen dem Anwender dieselben Operationen zur Verfügung wie im zweiten Vorbereitungsschritt, siehe Abschnitt 5.2.2.

5.3.5. Differenzen berechnen

Zwischen dem ursprünglichen Architektur-Graphen $G_{Architektur} = (V_{Architektur}, E_{Architektur})$ und dem abgeleiteten und modifizierten Architektur-Graphen $G'_{Architektur} = (V'_{Architektur}, E'_{Architektur})$ wird die Differenz berechnet. Wir beschreiben die Differenz-Ermittlung im Folgenden deklarativ basierend auf Mengen.

Jeder Knoten (v) und jede Kante (e) der beiden Graphen besitzt einen eindeutigen Identifikator ($v.id$ bzw. $e.id$). Knoten bzw. Kanten in den beiden Graphen, die sich jeweils entsprechen, besitzen denselben Identifikator. Wir beschreiben im Folgenden, wie weggefallene und hinzugefügte Knoten und Kanten bestimmt werden.

5.3.5.1. Hilfsmengen mit Knoten- und Kanten-Identifikatoren

Folgende Menge enthält alle Knoten-Identifikatoren des Ursprungs-Architektur-Graphen $G_{Architektur}$: $VID = \{v.id | v \in V_{Architektur}\}$

Folgende Menge enthält alle Kanten-Identifikatoren des Ursprungs-Architektur-Graphen $G_{Architektur}$: $EID = \{e.id | e \in E_{Architektur}\}$

Folgende Menge enthält alle Knoten-Identifikatoren des Ziel-Architektur-Graphen $G'_{Architektur}$: $VID' = \{v'.id | v' \in V'_{Architektur}\}$

Folgende Menge enthält alle Kanten-Identifikatoren des Ziel-Architektur-Graphen $G'_{Architektur}$: $EID' = \{e'.id | e' \in E'_{Architektur}\}$

Diese Hilfsmengen werden in den nachfolgenden Berechnungen verwendet.

5.3.5.2. Bestimmung entfernter und hinzugefügter Knoten und Kanten

Folgende Menge beschreibt alle *Knoten*, die in $G'_{Architektur}$ im Vergleich zu $G_{Architektur}$ *entfernt* wurden:

$V_{entfernt} = \{v | v \in V_{Architektur} \wedge v.id \notin VID'\}$

Folgende Menge beschreibt alle *Kanten*, die in $G'_{Architektur}$ im Vergleich zu $G_{Architektur}$ *entfernt* wurden:

$E_{entfernt} = \{e | e \in E_{Architektur} \wedge e.id \notin EID'\}$

Folgende Menge beschreibt alle *Knoten*, die in $G'_{Architektur}$ im Vergleich zu $G_{Architektur}$ *hinzugefügt* wurden:

$V'_{hinzugefügt} = \{v' | v' \in V'_{Architektur} \wedge v'.id \notin VID\}$

Folgende Menge beschreibt alle *Kanten*, die in $G'_{Architektur}$ im Vergleich zu $G_{Architektur}$ *hinzugefügt* wurden:

$E'_{hinzugefügt} = \{e' | e' \in E'_{Architektur} \wedge e'.id \notin EID\}$

Mit Hilfe der Mengen $V_{entfernt}$, $E_{entfernt}$, $V'_{hinzugefügt}$ und $E'_{hinzugefügt}$ werden die Architekturmodell-Elemente und -Assoziationen bestimmt, die bei der Modellierung des Umsetzungswegs hinzugefügt bzw. entfernt wurden. Diese Mengen werden aus den Modellen berechnet.

5.3.5.3. Bestimmung von Modifikationskennzeichnungen

Modifikationskennzeichnungen werden wie folgt bestimmt:

$V_{Modifikationskennzeichnungen} = \{v' | v' \in V_{hinzugefügt} \wedge v'.typ = \text{Modifikationskennzeichnung}\}$

Die Modifikationskennzeichnungen für konkrete Typen werden über die Kantentypen eingeschränkt. Auf diesem Weg lassen sich die markierten Architekturelemente wie folgt identifizieren.

Gekennzeichnete Komponenten

Für einen Modifikationskennzeichnungs-Knoten $v_{Modifikationskennzeichnung} \in V_{Modifikationskennzeichnungen}$ wird die gekennzeichnete *Komponente* bestimmt als $e'.v_R$ aus

$$\{e' : (v_L, v_R) | e' \in E'_{hinzugefügt} \wedge e'.v_L = v_{Modifikationskennzeichnung} \wedge e'.typ = ModifikationskennzeichnungKomponente\}$$

Alle Komponenten, die mit einer Modifikationskennzeichnung versehen sind werden über folgende Menge bestimmt

$$V_{GekennzeichneteKomponenten} = \{e'.v_R | e' \in E'_{hinzugefügt} \wedge e'.v_L = v_{Modifikationskennzeichnungen} \wedge e'.typ = ModifikationskennzeichnungKomponente\}$$

Gekennzeichnete Schnittstellenangebote

Für einen Modifikationskennzeichnungs-Knoten $v_{Modifikationskennzeichnung} \in V_{Modifikationskennzeichnungen}$ wird das gekennzeichnete *Schnittstellenangebot* bestimmt als $e'.v_R$ aus

$$\{e' : (v_L, v_R) | e' \in E'_{hinzugefügt} \wedge e'.v_L = v_{Modifikationskennzeichnung} \wedge e'.typ = ModifikationskennzeichnungSchnittstellenangebot\}$$

Neben den direkt gekennzeichneten Schnittstellenangeboten gelten auch solche Schnittstellenangebote als „modifiziert“, zu denen Operationen hinzugekommen sind oder entfernt wurden, d.h.

$$V_{ModifizierteSchnittstellen} = \{e'.v_L | e' \in E'_{Architektur} \wedge e'.typ = Operationsdefinition \wedge (e'.v_R \in V_{hinzugefügt} \vee e'.v_R \in V_{entfernt})\}$$

Gekennzeichnete Operationen

Für einen Modifikationskennzeichnungs-Knoten $v_{Modifikationskennzeichnung} \in V_{Modifikationskennzeichnungen}$ wird die gekennzeichnete *Operation* bestimmt als $e'.v_R$ aus

$$\{e' : (v_L, v_R) | e' \in E'_{hinzugefügt} \wedge e'.v_L = v_{Modifikationskennzeichnung} \wedge e'.typ = ModifikationskennzeichnungOperation\}$$

Gekennzeichnete Schnittstellen

Für einen Modifikationskennzeichnungs-Knoten

$v_{Modifikationskennzeichnung} \in V_{Modifikationskennzeichnungen}$ wird die gekennzeichnete *Schnittstelle* bestimmt als $e'.v_R$ aus

$$\{e' : (v_L, v_R) | e' \in E'_{hinzugefügt} \wedge e'.v_L = v_{Modifikationskennzeichnung} \wedge e'.typ = ModifikationskennzeichnungSchnittstelle\}$$

Gekennzeichnete Datentypen

Für einen Modifikationskennzeichnungs-Knoten $v_{Modifikationskennzeichnung} \in V_{Modifikationskennzeichnungen}$ wird der gekennzeichnete *Datentyp* bestimmt als $e'.v_R$ aus

$$\{e' : (v_L, v_R) | e' \in E'_{hinzugefügt} \wedge e'.v_L = v_{Modifikationskennzeichnung} \wedge e'.typ = ModifikationskennzeichnungDatentyp\}$$

Insgesamt besteht die Differenz aus den Mengen der hinzugefügten und entfernten Knoten und Kanten ($V_{entfernt}$, $E_{entfernt}$, $V_{hinzugefügt}$ und $E_{hinzugefügt}$), sowie den Mengen der angebrachten Modifikationskennzeichnungen ($V_{Modifikationskennzeichnungen}$).

5.3.6. Architektur-bezogene Aktivitäten ermitteln

Aus den Differenzen werden die architekturbezogenen Aktivitäten abgeleitet. Die Differenzen werden auf der Graph-Repräsentation durch die Mengen $V_{entfernt}$, $E_{entfernt}$, $V_{hinzugefügt}$ und $E_{hinzugefügt}$ beschrieben.

Im Folgenden wird deklarativ beschrieben, für welche Fälle, welche Arten von architekturbezogenen Aktivitäten ermittelt werden.

Aktivität 5.1 (Datentyp-Hinzufügen-Aktivität)**.** *Eine Datentyp-Hinzufügen-Aktivität (Metaklasse:* `DatentypHinzufügen`*) wird abgeleitet für jedes Element der Menge*
$V_{DatentypHinzufügenAktivität} = \{v' | v' \in V_{hinzugefügt} \wedge v'.typ = Datentyp\}$

Aktivität 5.2 (Datentyp-Entfernen-Aktivität)**.** *Eine Datentyp-Entfernen-Aktivität (Metaklasse:* `DatentypEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{DatentypEntfernenAktivität} = \{v | v \in V_{entfernt} \wedge v.typ = Datentyp\}$

Aktivität 5.3 (Datentyp-Modifizieren-Aktivität)**.** *Eine Datentyp-Modifizieren-Aktivität (Metaklasse:* `DatentypModifizieren`*) wird für jede Interne-Modifikations-Kennzeichnung an einem Datentyp erzeugt, d.h. für jeden Datentyp bestimmt durch die Menge*
$V_{DatentypModifizierenAktivität} = \{e'.v_R | e' \in E'_{hinzugefügt} \wedge e'.typ = Modifikationskennzeichnung Datentyp\}$

Aktivität 5.4 (Schnittstelle-Hinzufügen-Aktivität)**.** *Eine Schnittstelle-Hinzufügen-Aktivität (Metaklasse:* `SchnittstelleHinzufügen`*) wird abgeleitet für jedes Element der Menge*
$V_{SchnittstelleHinzufügenAktivität} = \{v' | v' \in V_{hinzugefügt} \wedge v'.typ = Schnittstelle\}$

Aktivität 5.5 (Schnittstelle-Entfernen-Aktivität)**.** *Eine Schnittstelle-Entfernen-Aktivität (Metaklasse:* `SchnittstelleEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{SchnittstelleEntfernenAktivität} = \{v | v \in V_{entfernt} \wedge v.typ = Schnittstelle\}$

Aktivität 5.6 (Schnittstelle-Modifizieren-Aktivität)**.** *Eine Schnittstelle-Modifizieren-Aktivität (Metaklasse:* `SchnittstelleModifizieren`*) wird indirekt für hinzugefügte, entfernte und modifizierte Operationen erstellt.*

Aktivität 5.7 (Operationsdefinition-Hinzufügen-Aktivität)**.** *Eine Operationsdefinition-HinzufügenAktivität (Metaklasse:* `OperationsDefinitionHinzufügen`*) wird abgeleitet für jedes Element der Menge*
$V_{OperationsDefinitionHinzufügenAktivität} = \{v' | v' \in V_{hinzugefügt} \wedge v'.typ = Operation\}$

Aktivität 5.8 (Operationsdefinition-Entfernen-Aktivität)**.** *Eine Operationsdefinition-Entfernen-Aktivität (Metaklasse:* `OperationsDefinitionEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{OperationsDefinitionEntfernenAktivität} = \{v | v \in V_{entfernt} \wedge v.typ = Operation\}$

Aktivität 5.9 (Operationsdefinition-Modifizieren-Aktivität)**.** *Eine Operationsdefinition-Modifizieren-Aktivität (Metaklasse:* `OperationsDefinitionModifizieren`*) wird indirekt für hinzugefügte, entfernte und modifizierte Parameter erstellt.*

Aktivität 5.10 (Aufrufparameter-Hinzufügen-Aktivität)**.** *Eine Aufrufparameter-Hinzufügen-Aktivität (Metaklasse:* `AufrufparameterHinzufügen`*) wird abgeleitet für jedes Element der Menge*

$V_{AufrufparameterHinzufügenAktivität} = \{e'.v_R | e' \in E_{hinzugefügt} \wedge$
$e'.typ = Aufrufparameterdefinition\}$

Aktivität 5.11 (Aufrufparameter-Entfernen-Aktivität)**.** *Eine Aufrufparameter-Entfernen-Aktivität (Metaklasse:* `AufrufparameterEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{AufrufparameterEntfernenAktivität} = \{e.v_R | e \in E_{entfernt} \wedge$
$e.typ = Aufrufparameterdefinition\}$

Aktivität 5.12 (Rückgabeparameter-Hinzufügen-Aktivität)**.** *Eine Rückgabeparameter-Hinzufügen-Aktivität (Metaklasse:* `RückgabeparameterHinzufügen`*) wird abgeleitet für jedes Element der Menge*
$V_{RückgabeparameterHinzufügenAktivität} = \{e'.v_R | e' \in E_{hinzugefügt} \wedge$
$e'.typ = Rückgabeparameterdefinition\}$

Aktivität 5.13 (Rückgabeparameter-Entfernen-Aktivität)**.** *Eine Rückgabeparameter-Entfernen-Aktivität (Metaklasse:* `RückgabeparameterEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{RückgabeparameterEntfernenAktivität} = \{e.v_R | e \in E_{entfernt} \wedge$
$e.typ = Rückgabeparameterdefinition\}$

Aktivität 5.14 (Basiskomponente-Hinzufügen-Aktivität)**.** *Eine Basiskomponente-Hinzufügen-Aktivität (Metaklasse:* `BasisKomponenteHinzufügen`*) wird abgeleitet für jedes Element der Menge*
$V_{BasisKomponenteHinzufügen} = \{v' | v' \in V_{hinzugefügt} \wedge$
$v'.typ = BasisKomponente\}$

Aktivität 5.15 (Basiskomponente-Entfernen-Aktivität)**.** *Eine Basiskomponente-Entfernen-Aktivität (Metaklasse:* `BasisKomponenteEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{BasisKomponenteEntfernen} = \{v | v \in V_{entfernt} \wedge$
$v.typ = BasisKomponente\}$

Aktivität 5.16 (Basiskomponente-Modifizieren-Aktivität)**.** *Eine Basiskomponente-Modifizieren-Aktivität wird indirekt abgeleitet als Oberaktivität für folgende architekturbezogene Aktivitäten: Schnittstellenangebot hinzufügen, Schnittstellenangebot entfernen, Schnittstellenangebot modifizieren, Schnittstellennachfrage hinzufügen, Schnittstellennachfrage entfernen, Schnittstellennachfrage modifizieren, sowie für Basiskomponenten mit einer internen Modifikationskennzeichnung.*

Aktivität 5.17 (Kompositkomponente-Hinzufügen-Aktivität)**.** *Eine Kompositkomponente-Hinzufügen-Aktivität (Metaklasse:* `KompositKomponenteHinzufügen`*) wird abgeleitet für jedes Element der Menge*
$V_{KompositKomponenteHinzufügen} = \{v' | v' \in V_{hinzugefügt} \wedge$
$v'.typ = KompositKomponente\}$

Aktivität 5.18 (Kompositkomponente-Entfernen-Aktivität)**.** *Eine Kompositkomponente-Entfernen-Aktivität (Metaklasse:* `KompositKomponenteEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{KompositKomponenteEntfernen} = \{v | v \in V_{entfernt} \wedge$
$v.typ = KompositKomponente\}$

Aktivität 5.19 (Kompositkomponente-Modifizieren-Aktivität)**.** *Eine Kompositkomponente-Modifizieren-Aktivität wird indirekt durch hinzugefügte, entfernte oder modifizierte Unterkomponenten bestimmt.*

Aktivität 5.20 (Schnittstellenangebot-Hinzufügen-Aktivität)**.** *Eine Schnittstellenangebot-Hinzufügen-Aktivität (Metaklasse:* `SchnittstellenangebotHinzufügen`*) wird abgeleitet für jedes Element der Menge*
$V_{SchnittstellenangebotHinzufügen} = \{v' | v' \in V_{hinzugefügt} \wedge$
$v'.typ = Schnittstellenangebot\}$

Aktivität 5.21 (Schnittstellenangebot-Entfernen-Aktivität)**.** *Eine Schnittstellenangebot-Entfernen-Aktivität (Metaklasse:* `SchnittstellenangebotEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{SchnittstellenangebotEntfernen} = \{v | v \in V_{entfernt} \wedge$
$v.typ = Schnittstellenangebot\}$

Aktivität 5.22 (Schnittstellenangebot-Modifizieren-Aktivität)**.** *Eine Schnittstellenangebot-Modifizieren-Aktivität wird, wie oben beschrieben, aus Modifikationskennzeichnungen bestimmt, sowie indirekt durch das Hinzufügen und Entfernen von angebotenen Operationen.*

Aktivität 5.23 (Angebotene-Operation-Hinzufügen-Aktivität)**.** *Eine Angebotene-Operation-Hinzufügen-Aktivität (Metaklasse:* `AngeboteneOperationHinzufügen`*) wird abgeleitet für jedes Element der Menge*
$V_{AngeboteneOperationHinzufügen} = \{v' | v' \in V_{hinzugefügt} \wedge v'.typ = Operation\}$

Aktivität 5.24 (Angebotene-Operation-Entfernen-Aktivität)**.** *Eine Angebotene-Operation-Entfernen-Aktivität (Metaklasse:* `AngeboteneOperationEntfernen`*) wird abgeleitet für jedes Element der Menge*
$V_{AngeboteneOperationEntfernen} = \{v | v \in V_{entfernt} \wedge v.typ = Operation\}$

Aktivität 5.25 (Angebotene-Operation-Modifizieren-Aktivität)**.** *Eine Angebotene-Operation-Modifizieren-Aktivität wird, wie oben beschrieben, aus Modifikationskennzeichnungen bestimmt.*

Aktivität 5.26 (Schnittstellennachfrage-Hinzufügen-Aktivität)**.** *Eine Schnittstellennachfrage-Hinzufügen-Aktivität (Metaklasse:* `SchnittstellennachfrageHinzufügen`*) wird abgeleitet für jedes Element der Menge* $V_{SchnittstellennachfrageHinzufügen} = \{v' | v' \in V_{hinzugefügt} \wedge v'.typ = Schnittstellennachfrage\}$

Aktivität 5.27 (Schnittstellennachfrage-Entfernen-Aktivität)**.** *Eine Schnittstellennachfrage-Entfernen-Aktivität (Metaklasse:* `SchnittstellennachfrageEntfernen`*) wird abgeleitet für jedes Element der Menge* $V_{SchnittstellennachfrageEntfernen} = \{v | v \in V_{entfernt} \wedge v.typ = Schnittstellennachfrage\}$

Aktivität 5.28 (Schnittstellennachfrage-Modifizieren-Aktivität)**.** *Eine Schnittstellennachfrage-Modifizieren-Aktivität wird indirekt aus Aktivitäten für Nachgefragte-Operation-Hinzufügen und Nachgefragte-Operation-Entfernen bestimmt.*

5.3.7. Tätigkeitsfeldbezogene Aktivitäten ermitteln

Zur Ableitung von Aktivitäten in den Tätigkeitsfeldern ausgehend von den zuvor bestimmten architektur-bezogenen Aktivitäten werden Modelltransformationen verwendet.

Im vorigen Kapitel werden in Abschnitt 4.5.6 Ableitungsregeln für die einzelnen Tätigkeitsfelder formuliert. Diese Ableitungsregeln lassen sich in Form von Modelltransformationen beschreiben. Zur formalen Beschreibung der Modelltransformationen verwenden wir Triple-Graph-Grammatiken. Abbildung 5.19 zeigt beispielsweise die Transformationsregel, welche beschreibt, wie aus einer Komponenten-Aktivität für eine Komponente, die mit einer Quelltextdatei-Anreicherung versehen ist, eine Quelltextbearbeitungs-Aktivität abgeleitet wird.

Die Notation der Regeln ist wie folgt zu interpretieren. Die Knoten und Kanten welche nicht mit zwei Pluszeichen versehen sind bilden die Ausgangssituation vor der Regelanwendung, sozusagen die Vorbedingung der Regelanwendung.

Die Knoten und Kanten welche mit zwei Pluszeichen versehen sind werden durch die Regelanwendung dem Graph und damit dem Modell hinzugefügt.

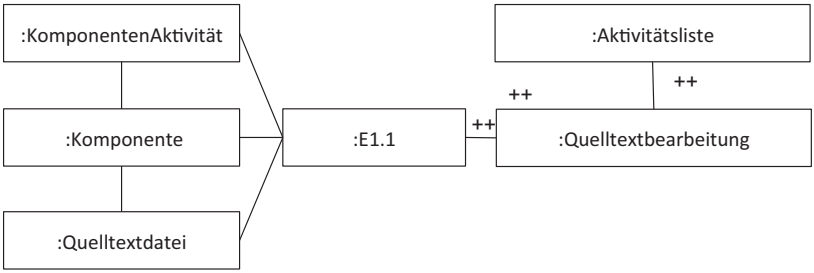


Abbildung 5.19.: Triple-Graph-Regel E1.1: Quelltextbearbeitung aus Komponenten-Aktivität mit Quelltextdatei-Anreicherung

Die Transformationsregeln beschreiben im Detail wie die Regeln im Modell umgesetzt werden. Aus Platzgründen führen wir die Transformationsregeln in Anhang A auf.

6. Validierung

In diesem Kapitel beschreiben wir die Validierung des Verfahrens zur architektur-basierten Bewertung und Planung von Änderungsanfragen. Das Verfahren automatisiert die Ableitung von Aktivitätslisten für beschriebene Umsetzungswege im Architekturmodell. Der Anwender kann mit dem Analysewerkzeug einen Umsetzungsweg in einem komponenten-basierten Architekturmodell beschreiben, indem er ein Ausgangsarchitekturmodell in ein Zielarchitekturmodell umgestaltet oder mit Hilfe von Annotationen notwendige interne Modifikationen kennzeichnet. Die modellierten Unterschiede und Annotationen werden durch das Verfahren ermittelt. Davon ausgehend wird die Änderungsausbreitung in der Komponentenstruktur bestimmt und mit Kontextinformationen angereicherte Aktivitäten in den Tätigkeitsfeldern abgeleitet. Zusammengefasst besteht der wissenschaftliche Beitrag der Dissertation in der automatisierten Ableitung von Aktivitäten aus Architekturmodellmodifikationen und -annotationen angereichert mit Kontextinformationen.

Aus der Analyse gehen Aktivitätslisten hervor, welche verwendet werden können, um schon vor der eigentlichen Umsetzung von Änderungsanfragen Auswirkungen zu untersuchen und mögliche Aufwände abzuschätzen. Ein Anwendungsfall ist die Aufwandsschätzung mit der sogenannten „Bottom-Up“-Methodik, bei der die Arbeit in einzelne Teilschritte heruntergebrochen wird und für die einzelnen Schritte Aufwände geschätzt werden. Der Gesamtaufwand wird durch die Aggregation der Teilaufwände bestimmt.

Das Kapitel ist wie folgt aufgebaut. Abschnitt 6.1 fasst die validierten Qualitätseigenschaften für die Dissertation zusammen. Abschnitt 6.2 beschreibt die Konzeption der empirischen Studie. Abschnitt 6.3 stellt die Ergebnisse der empirischen Studie vor. Abschnitt 6.4 interpretiert und diskutiert Validitätseigenschaften der Studie. Abschnitt 6.5 fasst das Kapitel zusammen und zieht das Fazit.

6.1. Validierte Qualitätseigenschaften

Im Rahmen der Validierung untersuchen wir die erwarteten Eigenschaften der automatisierten Aktivitätsherleitung. Der entscheidende Beitrag, den das Verfahren leistet, ist die automatisierte Auswertung und Zusammenführung von Kontextinformationen bei der Identifikation von Arbeitsschritten für Umsetzungswege. Als Vorteil der Automatisierung erwarten wir eine hohe Skalierbarkeit. Daneben erwarten wir eine stabile und hohe Ergebnisqualität selbst bei weniger erfahrenen Anwendern. Wir erwarten, dass das automatisierte Verfahren qualitativ gute Aktivitätslisten ermittelt und bei größeren Modellen und einer steigenden Anzahl von analysierten Umsetzungswegen insgesamt zu einem reduzierten Analyse-Aufwand führt.

Die zentrale These, die wir untersuchen, lautet:

> Durch die Anreicherung des Architekturmodells
> mit technischen und organisatorischen Informationen
> *ist es möglich,*
> Umsetzungswege im Architekturmodell zu beschreiben
> und automatisiert in Aktivitäten zu überführen
> *und dadurch,*
> die Skalierbarkeit zu verbessern,
> gleichzeitig Ergebnisse zu erzielen, welche mindestens
> so vollständig und genau sind wie bei manueller Herleitung.

In den nächsten Abschnitten betrachten wir die erwarteten Eigenschaften genauer.

Skalierbarkeit
: Durch die Automatisierung erwarten wir eine Verbesserung der Skalierbarkeit bei der Aktivitätsherleitung, das heißt, dass bei größeren Systemen mit vielen Kontextinformationen, das oft mühsame Zusammenführen der Kontextinformationen zur Bewertung eines Umsetzungsweges durch das automatische Verfahren übernommen wird. Die Kontextinformationen müssen im Modell nur einmal mit den Architekturelementen in Bezug gesetzt werden. Dann können die Umsetzungswege übersichtlich anhand der Architekturelemente beschrieben werden. Das

Verfahren wertet die Kontextinformationen aus, die für den konkreten Umsetzungsweg relevant sind und leitet daraus Aktivitäten ab. Wir erwarten, dass das Verfahren durch die Automatisierung skaliert bezüglich der Systemgröße und der damit verbundenen Menge der Annotationen.

Vollständigkeit und Genauigkeit der Aktivitäten
Wir erwarten, dass die durch die Automatisierung bestimmten Ergebnisse, d.h. die Aktivitätslisten, eine ausreichende Vollständigkeit und Genauigkeit aufweisen. Wir erwarten, dass insbesondere unerfahrene Anwender ein qualitativ besseres Ergebnis erhalten als vergleichbare unerfahrene Anwender und dass die Lösung der unerfahrenen Anwender gleich gut oder besser zum Ergebnis erfahrener Software-Entwickler ist.

Unabhängigkeit von der Anwendererfahrung
Wir erwarten, dass durch das Verfahren die Ergebnisse unabhängiger von der Erfahrung der Anwender werden, das heißt, dass sie stabiler werden und Aktivitätslisten daher vergleichbarer sind.

Der Aspekt der Skalierbarkeit von Automatisierungsverfahren ist allgemein bekannt und anerkannt. Daher wird die Skalierbarkeit nicht explizit im Rahmen von Studien untersucht. Wir untersuchen sie aber indirekt im Rahmen der Validierung der anderen zwei Aspekte, indem wir den Zeitbedarf für die Bearbeitung von Analysevorgängen betrachten.

Ausgehend von den genannten Erwartungen formulieren wir die folgenden konkreten Validierungsthesen:

1. Anwender *mit wenig Erfahrung* erhalten durch die automatisierte Analyse genauso vollständige und präzise oder sogar vollständigere und präzisere Ergebnisse wie Anwender mit *wenig* Erfahrung, welche eine manuelle Analyse durchführen.

2. Anwender *mit wenig Erfahrung* erhalten durch die automatisierte Analyse genauso vollständige und präzise oder sogar vollständigere und präzisere Ergebnisse wie *erfahrene* Anwender, welche eine manuelle Analyse durchführen.

Die Validierungsthesen legen eine Untersuchung mit Hilfe einer empirischen Studie nahe. Da unsere Validierungsthesen besonders Eigenschaften bei der

Anwendung durch wenig erfahrene Anwender erfassen, sehen wir die Durchführung von Experimenten mit studentischen Teilnehmern für angemessen. Wir entwickeln daher eine empirische Studie, welche wir im Rahmen eines Software-Entwicklungs-Praktikums von Informatik-Studenten durchführen. Die nachfolgenden Abschnitte beschreiben die Konzeption und die Ergebnisse der empirischen Studie.

6.2. Empirische Studie – Studienentwurf

6.2.1. Genauigkeit, Trefferquote und F_1-Maß

Bevor wir zum Studienentwurf kommen stellen wir Metriken vor, welche wir im Rahmen der Studie verwenden werden.

In der Studie werden zur Quantifizierung der Lösungen die Metriken *Genauigkeit* (engl. precision) und *Trefferquote* (engl. recall) bestimmt und daraus das gewichtete harmonische Mittel ([Bro+01], Seite 20), das sogenannte F_1-Maß bestimmt. Die Metriken stammen aus dem Information Retrieval ([Rij79]) und stellen gängige Metriken im Rahmen von Evaluierungen dar ([Pow11]). Das F_1-Maß wird allgemein durch folgende Berechnungsvorschrift definiert.

Gehen wir von einer Grundgesamtheit an Elementen aus. In dieser Grundgesamtheit gibt es eine Referenzteilmenge, die als „korrekt“ bezeichnet wird. Aus der Grundgesamtheit wird eine Auswahl getroffen, die mit der Referenzteilmenge verglichen wird. Dabei gibt es korrekt ausgewählte Elemente, vergessene Elemente, falsch ausgewählte Elemente und „zurecht nicht ausgewählte“ Elemente.

Seien t_p die Anzahl der korrekt ausgewählten Elemente, f_n die Anzahl der vergessenen Elemente und f_p die Anzahl der falsch ausgewählten Elemente, dann ergeben sich Genauigkeit (engl. precision) und Trefferquote (engl. recall) wie folgt.

$$\text{Genauigkeit} = \frac{t_p}{t_p + f_p}$$

Die Genauigkeit beschreibt den Anteil der korrekt ausgewählten Elemente im Verhältnis zu allen ausgewählten Elementen. Der Wertebereich der Genauigkeit bewegt sich zwischen 0 und 1.

$$\text{Trefferquote} = \frac{t_p}{t_p + f_n}$$

Die Trefferquote beschreibt das Verhältnis der korrekt ausgewählten Elemente zu allen erwarteten Elementen. Der Wertebereich der Trefferquote bewegt sich zwischen 0 und 1.

Bei gegebener *Genauigkeit* und *Trefferquote* und dem Gewichtungsfaktor α gilt für das gewichtete harmonische Mittel F_α:

$$F_\alpha = (1+\alpha^2)\frac{\text{Genauigkeit} * \text{Trefferquote}}{\alpha^2 * \text{Genauigkeit} + \text{Trefferquote}}$$

Der Gewichtungsfaktor α bestimmt, wie stark die Genauigkeit gegenüber der Trefferquote im Ergebnis gewichtet wird. Wir verwenden einen Gewichtungsfaktor α von 1 bei dem Genauigkeit und Trefferquote gleichstark berücksichtigt werden. Dies führt dann zum F_1-Maß.

$$F_1 = 2\frac{\text{Genauigkeit} * \text{Trefferquote}}{\text{Genauigkeit} + \text{Trefferquote}}$$

Das F_1-Maß wird in der Studie zur Bewertung von Aktivitätslisten herangezogen. Treffer und Fehler beziehen sich in diesem Fall auf Aktivitäten oder Anreicherungen, welche durch Studienteilnehmer ermittelt werden und welche im Vergleich zu einer Referenzlösung korrekt bestimmt (= Treffer), nicht angegeben (= Fehler) oder zuviel angegeben (= Falsch Positiv) werden.

6.2.2. GQM-Plan mit Studienzielen

Zur Erörterung und Beschreibung des Experimententwurfs verwenden wir die GQM-Methode nach V. Basili [BCR94]. Der GQM-Plan in den Tabellen 6.1 und 6.2 fasst das Studienziel, die damit verbundenen Forschungsfragen und die Metriken zur Evaluierung der jeweiligen Forschungsfragen zusammen.

Ziel:	Evaluierung der Unterschiede zwischen Aktivitätslisten bestimmt durch werkzeuggestützte Änderungsanfragenanalyse und Aktivitätslisten, welche manuell bestimmt werden.
Zweck: Problem: Objekt: Perspektive:	Evaluierung der Unterschiede von Aktivitätslisten im Vergleich zur Referenzlösung Vollständigkeit und Genauigkeit von Aktivitätslisten Aktivitätslisten Anwender des automatisierten Verfahrens
Frage 1:	Wie vollständig und präzise sind die Aktivitätslisten von weniger erfahrenen Anwendern mit automatisiertem Verfahren **(Behandlungsgruppe, BG)** im Vergleich zu Aktivitätslisten weniger erfahrener Anwender ohne automatisiertes Verfahren **(Kontrollgruppe, KG)**
Frage 1.1:	... bezüglich der **Aktivitätstypen**?
Metrik 1.1.1:	$F_1(Aktivitätstyp, BG)$: F_1-Metrik aus Genauigkeit und Trefferquote für Behandlungsgruppe (BG)
Metrik 1.1.2:	$F_1(Aktivitätstyp, KG)$: F_1-Metrik aus Genauigkeit und Trefferquote für Kontrollgruppe (KG)
Hypothese:	$F_1(Aktivitätstyp, BG) \geq F_1(Aktivitätstyp, KG)$, d.h. automatisiert bestimmte Lösung ist gleich oder besser
Frage 1.2:	... bezüglich der **Aktivitätsanreicherungen**?
Metrik 1.2.1:	$F_1(Anreicherung, BG)$: F_1-Metrik aus Genauigkeit und Trefferquote für Behandlungsgruppe (BG)
Metrik 1.2.2:	$F_1(Anreicherung, KG)$: F_1-Metrik aus Genauigkeit und Trefferquote für Kontrollgruppe (KG)
Hypothese:	$F_1(Anreicherung, BG) \geq F_1(Anreicherung, KG)$, d.h. automatisiert bestimmte Lösung ist gleich oder besser

Tabelle 6.1.: GQM-Plan für den Experimententwurf (Teil 1)

Frage 2:	Wie vollständig und präzise sind die Aktivitätslisten von weniger erfahrenen Anwendern mit automatisiertem Verfahren **(Behandlungsgruppe, BG)** im Vergleich zu Aktivitätslisten erfahrener Anwender ohne automatisiertes Verfahren **(Expertengruppe, EXP)** ...
Frage 2.1:	... bezüglich der **Aktivitätstypen**?
Metrik 2.1.1:	$F_1(Aktivitätstyp, BG)$: F_1-Metrik aus Genauigkeit und Trefferquote für Behandlungsgruppe (BG)
Metrik 2.1.2:	$F_1(Aktivitätstyp, EXP)$: F_1-Metrik aus Genauigkeit und Trefferquote für Expertengruppe (EXP)
Hypothese:	$F_1(Aktivitätstyp, BG) \geq F_1(Aktivitätstyp, EXP)$, d.h. automatisiert bestimmte Lösung ist gleich oder besser
Frage 2.2:	... bezüglich der **Aktivitätsanreicherungen**?
Metrik 2.2.1:	$F_1(Anreicherung, BG)$: F_1-Metrik aus Genauigkeit und Trefferquote für Behandlungsgruppe (BG)
Metrik 2.2.2:	$F_1(Anreicherung, EXP)$: F_1-Metrik aus Genauigkeit und Trefferquote für Expertengruppe (EXP)
Hypothese:	$F_1(Anreicherung, BG) \geq F_1(Anreicherung, EXP)$, d.h. automatisiert bestimmte Lösung ist gleich oder besser

Tabelle 6.2.: GQM-Plan für den Experimententwurf (Teil 2)

Die Frage 1 behandelt das Verhältnis von weniger erfahrenen Anwendern untereinander, wohingegen die Frage 2 das Verhältnis von weniger erfahrenen Anwendern zu erfahrenen Anwendern betrachtet.

Die Teilfragen 1.1 und 2.1 untersuchen, welche Aktivitätstypen identifiziert werden, unabhängig von den Kontextinformationen. Sie betrachten, ob die erforderlichen Tätigkeitsarten zur Umsetzung einer Änderungsanfrage grundsätzlich ermittelt werden.

Die Teilfragen 1.2 und 2.2 untersuchen, ob zu den Aktivitäten die Kontextinformationen vollständig und korrekt angegeben werden. Dazu gehören beispielsweise die Aufzählung von Quelltextdateien, Testfällen oder die Anzahl der Laufzeitinstanzen, welche in Betrieb zu nehmen sind.

Ausgehend von diesem GQM-Plan werden drei Teilexperimente mit einer jeweiligen Teilnehmergruppe konzipiert und durchgeführt. Insgesamt kom-

men drei Teilnehmergruppen zum Einsatz. 1) Die Behandlungsgruppe (BG) bestehend aus Studenten (weniger erfahrene Anwender) verwendet zur Bestimmung von Aktivitätslisten das werkzeug-gestützte Analyseverfahren. 2) Die Kontrollgruppe (KG) bestehend aus Studenten (weniger erfahrene Anwender) führt die Aktivitätsherleitung manuell durch. 3) Eine Expertengruppe (EXP) bestehend aus erfahrenen Anwendern führt die Aktivitätsherleitung wie die Kontrollgruppe manuell durch.

Für die Studie wird ein Software-System als Fallstudie herangezogen. Insgesamt sind für dieses Software-System vier Änderungsanfragen zu analysieren. In der Studie werden diese als Aufgaben bezeichnet. Aus den Ergebnissen der vier Teilaufgaben wird das Gesamtergebnis berechnet.

Wie der GQM-Plan in den Tabellen 6.1 und 6.2 zeigt, werden zum Vergleich der Behandlungsgruppe mit der Kontrollgruppe und zum Vergleich der Behandlungsgruppe mit der Expertengruppe zwei Aspekte analysiert:

1. F_1-Metrik zur Quantifizierung von Trefferquote und Genauigkeit bezüglich der genannten Aktivitätstypen
2. F_1-Metrik zur Quantifizierung von Trefferquote und Genauigkeit bezüglich der Aktivitätsanreicherungen

Als Vergleichsgrundlage für die individuellen Lösungen der Teilnehmer werden Referenzlösungen herangezogen. Diese werden im realen System bestimmt und in Abstimmung mit den Entwicklern des Systems erarbeitet.

Der erste Aspekt untersucht, ob der Teilnehmer die Aktivitätstypen der Referenzlösung richtig ermittelt hat. Korrekte Aktivitätstypen werden als „Treffer“ gezählt, fehlende Aktivitätstypen werden als „Fehler“ gezählt und zu viel genannte Aktivitätstypen werden als „Falsche Treffer“ gezählt.

Der zweite Aspekt betrachtet zusätzlich zum Aktivitätstyp die genannten Anreicherungen zu den Aktivitäten. Bei der Auswertung wird dann gezählt, welche Anreicherungen in der Teilnehmerlösung korrekt ermittelt werden („Treffer“), welche Anreicherungen nicht angegeben werden („Fehler“) und welche Anreicherungen zu viel angegeben werden („Falsche Treffer“).

6.2.3. Studienobjekt: Benutzerverwaltungssystem

Als Fallstudie dient ein Benutzerverwaltungssystem. Dabei handelt es sich um ein existierendes Software-System, welches zur Abspeicherung von Benutzerdaten verwendet werden kann. Ein Datenmodell gibt die Bestandteile der Benutzerdaten vor, welche in einer Datenbank persistiert werden und welche über bereitgestellte Dienste abgefragt und modifiziert werden können. Das Benutzerverwaltungssystem ist, was den Quelltext betrifft, nicht sehr groß, birgt aber durch die Verwendung verschiedener Technologien eine technologische Komplexität. Zur Persistierung der Benutzerdaten wird eine Datenbank verwendet, wobei Nutzerdaten über eine Objekt-Relationale-Abbildungskomponente automatisiert in die Datenbank geschrieben und aus ihr gelesen werden können. Darüber hinaus bietet das System eine REST-Benutzerschnittstelle (REST, Representational State Transfer), welche als Datenformat JSON (JSON, JavaScript Object Notation) verwendet.

Das Benutzerverwaltungssystem wird den Studierenden bereits zu Beginn des Praktikums zur Verwendung in ihrem eigenen Software-System zur Verfügung gestellt, jedoch ohne den Teilnehmern einen Einblick in den Quelltext und die Entwicklungsartefakte zu gewähren.

Für die Studie wird für das Benutzerverwaltungssystem ein werkzeug-gestütztes Architekturmodell erstellt und den Teilnehmern in der Studie zur Verfügung gestellt. Das Benutzerverwaltungssystem wird im Architekturmodell durch eine Menge von Komponenten, Schnittstellen und Datentypen beschrieben.

Abbildung 6.1 zeigt ein Systemdiagramm mit den Komponenten und deren Zusammensetzung zum Gesamtsystem. Demnach umfasst die komponentenbasierte Architektur des Benutzerverwaltungssystems acht Komponenten („UserServiceApacheProxy“, „UserServiceTomcat“, „Authentication“, „Reporting“, „UserManagement“, „UserDAO“, „DatabaseAccess“ und „UserDatabase“), sieben Datentypen („User“, „UserList“, „Report“, „SessionFactory“, „Session“, „Token“ und „WebServiceResponse“) und neun Schnittstellen („IUserServiceTomcat“, „IUserManagement“, „IUserManagementAdmin“, „IAuthentication“, „IReporting“, „IUserDAO“, „ISessionFactory“, „ISession“, „JDBCDriver“).

Die Komponenten „EventMngmt“ und „MensaServer“ gehören nicht zum Benutzerverwaltungssystem, sondern sind Teil der nutzenden Systeme. Diese zählen nicht zum Analyseumfang der Studie werden aber der Vollständigkeit halber mit aufgenommen, um die Integration mit dem Praktikum zu betonen und die Identifikation der Studenten mit der Studie zu erhöhen.

Die Kontextinformationen zum Benutzerverwaltungssystem werden in Form eines werkzeug-gestützten Anreicherungsmodells bereitgestellt. Das Anreicherungsmodell umfasst die folgenden Kontextinformationen. Das sind zum Einen Quelltext-Datei-Annotationen für die Dateien „UserRest.java“, „Reporting.java“, „UserService.java“, „UserDAO.java“, „HibernateUtil.java“ und „User.java“ und zum Anderen Metadaten-Datei-Annotationen für die Hibernate-Konfigurationsdateien („hibernate.cfg.xml“, „User.hbm.xml“) und die Datenbank-Schema-Datei („Userdatabase.sql“).

Weiterhin gehören dazu eine Baukonfigurations-Spezifikations-Annotation mit der Bezeichnung „Eclipse-Projekt-Einstellungen“ für die Komponenten „UserServiceApacheProxy“, „UserServiceTomcat“, „UserManagement“, „Reporting“, „UserDAO“, „DatabaseAccess“ und „UserDatabase“, sowie Annotationen zur Kennzeichnung von Drittanbieter-Komponenten und Bibliotheken für die Komponenten „UserServiceApacheProxy“, „DatabaseAccess“, „UserDatabase“ und „Authentication“.

Die Komponenten der Mensa- und EventManagement-Systeme („EventMngmtServer“, „MensaImport“, „MensaServer“, „MensaDatabase“) werden mit Hilfe von Drittanbieter-Annotationen aus dem Analyserahmen ausgeschlossen, da die Teilnehmer unterschiedliches Teilwissen über diese Systeme haben und wir die Konzentration auf das Benutzerverwaltungssystem an sich legen wollen.

Es werden Testfall-Annotationen für 7 Unit-Tests am Schnittstellenangebot „IUserServiceTomcat“ der Komponente „UserServiceTomcat“ angebracht. Es werden Testfall-Annotationen für 5 Unit-Tests am Schnittstellenangebot „IUserManagement“ der Komponente „UserManagement“ angebracht. Eine Testfall-Annotation für 1 Unit-Test wird am Schnittstellenangebot „IAuthentication“ der Komponente „Authentication“ angebracht. Alle 13 Testfälle werden jeweils einzeln mit Namen genannt.

Als Bereitstellungsartefakt wird die Datei „UserManagement.war“ spezifiziert mit den Komponenten „UserServiceApacheProxy“, „UserServiceTom-

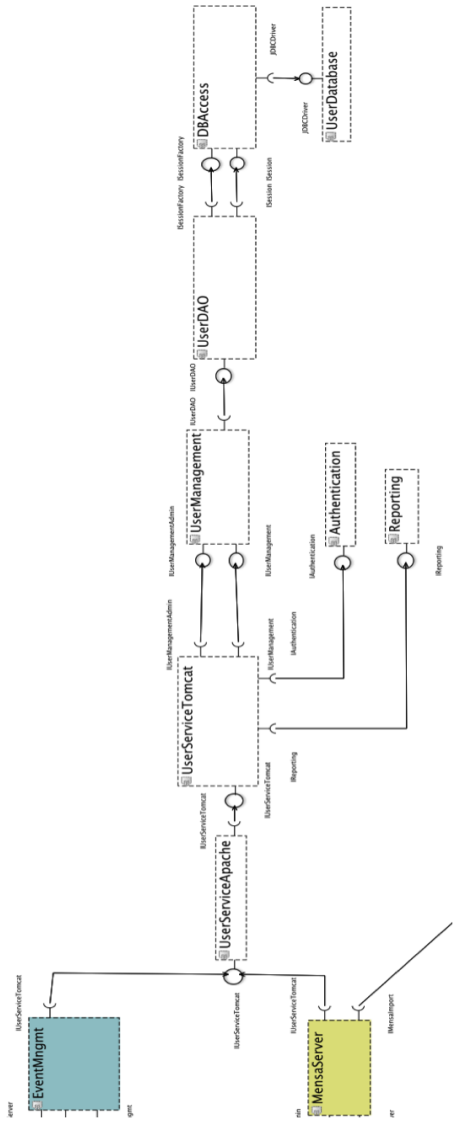


Abbildung 6.1.: Systemdiagramm des Benutzerverwaltungssystems

cat“, „UserManagement“, „Reporting“, „UserDAO“, „DatabaseAccess“ und „UserDatabase“. Durch eine Annotation wird eine Laufzeitinstanz spezifiziert mit den Komponenten „UserServiceApacheProxy“, „UserServiceTomcat“, „UserManagement“, „Reporting“, „UserDAO“, „DatabaseAccess“ und „UserDatabase“.

Die komponenten-internen Abhängigkeiten im Architekturmodell der Benutzerverwaltung werden pessimistisch modelliert. Das heißt, dass für jede Komponente für jedes Schnittstellenangebot Abhängigkeitsbeziehungen zu allen Schnittstellennachfragen der Komponente spezifiziert werden.

Für die Studien werden vier unterschiedliche Änderungsanfragen im Benutzerverwaltungssystem entwickelt, die von den Teilnehmern untersucht werden. Nachfolgend bezeichnen wir diese als „Aufgaben“. Die Aufgabe 1 behandelt die Erweiterung des Datentyps „User“ um ein Attribut. Die Aufgabe 2 behandelt die Evolution der Schnittstelle eines genutzten Drittanbieter-Dienstes zur Authentifizierung. Es soll ermittelt werden, welche Schritte zur Anpassung des Systems an die geänderte Schnittstelle notwendig sind. Die Aufgabe 3 fordert einen Wechsel des Datenformats der REST-Benutzerschnittstelle „IUserServiceTomcat“ von JSON zu XML. Die Aufgabe 4 umfasst die Ersetzung der genutzten Objekt-Relationalen-Abbildungskomponente durch eine eigene Implementierung.

6.2.4. Bildung der Gruppen

Für die Studie werden drei Gruppen gebildet, eine Behandlungsgruppe (BG), eine Kontrollgruppe (KG) und eine Expertengruppe (EXP).

Als Studienteilnehmer für die Kontroll- und Behandlungsgruppe werden die Teilnehmer eines Studentenpraktikums im Rahmen des Informatik-Studiums eingeladen. Die Studie wird zudem anonymisiert durchgeführt.

Im Rahmen des Praktikums haben vier Studentengruppen die Aufgabe, jeweils eine verteilte Software-Anwendung auf mobilen Endgeräten und auf Servern zu entwickeln und bearbeiten dabei alle typischen Phasen einer Neuentwicklung (Lastenhefterstellung, Pflichtenhefterstellung, Entwurf, Entwicklung, Testen, Abnahme).

Für die Expertengruppe werden wissenschaftliche Mitarbeiter mit mehrjähriger Erfahrung in der Software-Entwicklung eingeladen.

Zur Bildung von Behandlungsgruppe und Kontrollgruppe wird das folgende Verfahren verwendet. Im Rahmen des Praktikums zeigten die Teilnehmer unterschiedliche Leistungen, was auf unterschiedliche Erfahrungen und Fähigkeiten zurückzuführen ist. Um diese Unterschiede auszugleichen, wird zur Gruppenbildung von Behandlungsgruppe und Kontrollgruppe ein stratifiziertes Losverfahren eingesetzt. In der ersten Stufe werden alle Studenten aufgrund ihrer Leistung im Rahmen des Praktikums in zwei Qualifikationsgruppen Q1, Teilnehmer mit höherer Qualifikation, und Q2, Teilnehmer mit niedriger Qualifikation, eingeteilt. Im zweiten Schritt wird jede Qualifikationsgruppe durch Losverfahren gleichmäßig auf Behandlungsgruppe und Kontrollgruppe verteilt.

Die Behandlungsgruppe bestand aus sechs Studenten. Die Kontrollgruppe bestand aus acht Studenten. Die Expertengruppe bestand aus fünf Teilnehmern.

Die ungleiche Teilnehmerzahl von Behandlungs- und Kontrollgruppe ist auf den kurzfristigen Ausfall von Teilnehmern zurückzuführen. Eine Umverteilung war zu diesem Zeitpunkt nicht mehr möglich, da die Behandlungsgruppe bereits zuvor eine eigene Schulung erhalten hatte.

6.2.5. Materialien für die Teilnehmer

In diesem Abschnitt fassen wir die Materialien zusammen, welche die Teilnehmer der Studie erhalten. Tabelle 6.3 listet die Materialien auf und ordnet sie den entsprechenden Gruppen zu, welche sie erhalten.

Einige Materialien werden im Anhang B aufgeführt. Für das Experiment wird zudem ein Replikationspaket [Sta15] zur Verfügung gestellt. Das Replikationspaket enthält die kompletten Experimentunterlagen.

Alle Gruppen erhalten eine ausführliche textuelle Beschreibung des Systems mit allen benötigten Kontextinformationen.

Alle Gruppen erhalten ein werkzeug-gestütztes Architekturmodell des Benutzerverwaltungssystems.

Materialien	BG	KG	EXP
Textuelle Beschreibung des Systems mit Kontextinformationen	X	X	X
Werkzeuggestütztes Architekturmodell	X	X	X
Werkzeuggestütztes Anreicherungsmodell	X		
Analysewerkzeug	X		
Anleitung zur werkzeuggestützten Analyse	X		
Aufgabenbeschreibungen	X	X	X
Aktivitätstypenübersicht		X	X
Beispiel-Aktivitätslisten		X	X
Ergebnisfragebogen für Werkzeuganalyse	X		
Ergebnisfragebogen für manuelle Analyse		X	X
Orientierungsfragebogen	X	X	X
Abschlussfragebogen	X	X	X

Tabelle 6.3.: Übersicht der Materialien für die jeweiligen Gruppen

Die Teilnehmer der Behandlungsgruppe erhalten darüber hinaus das werkzeuggestützte Anreicherungsmodell mit Kontextinformationen sowie Zugriff auf eine spezielle Analysesicht, in der sie werkzeug-gestützt die Analyse durchführen können. Eine schrittweise Anleitung zur Analyse mit dem Werkzeug steht der Behandlungsgruppe ebenfalls zur Verfügung.

Die vier Aufgaben werden den Teilnehmern in einem Dokument zur Verfügung gestellt. Dieses umfasst die textuelle Beschreibung der Änderungsanfragen und eine ausführliche Arbeitsanweisung.

Die Kontrollgruppe und die Expertengruppe erhalten eine Übersichtstabelle mit Aktivitätstypen, deren Abkürzungen und Bedeutungserklärungen. Außerdem erhalten diese Teilnehmer eine umfassende Beispiel-Aktivitätsliste, an der sich ihre zu erarbeitende Lösung orientieren soll.

In Ergebnisfragebögen wird die Lösung erfasst. Neben den Ergebnisfragebögen erhalten die Gruppen einen Orientierungsfragebogen und einen Abschlussfragebogen. Der Orientierungsfragebogen dient den Teilnehmern zur Einarbeitung in das Software-System in einer Aufwärmphase vor der Studie. Der Abschlussfragebogen sammelt Rückmeldungen zur Stimmung, zum Verständnis und Verbesserungsvorschläge.

Den Teilnehmern ist es gestattet, Abkürzungen für Aktivitätstypen, Architekturelemente und Entwicklungsartefakte zu verwenden, solange eine eindeutige Zuordnung möglich ist. Ein Verweis auf Teillösungen in vorherigen Aufgaben ist zulässig, sofern die vorherige Lösung komplett übernommen werden kann.

Die Teilnehmer werden angehalten, Aktivitätstypen und insbesondere Entwicklungsartefakte vollständig und eindeutig zu erfassen. Insgesamt wird gefordert, dass pro Änderungsanfrage der komplette Umsetzungsweg bis zum erneut laufenden System beschrieben wird.

Die Teilnehmer der Behandlungsgruppe erfassen in einem Papierfragebogen die Einstiegspunkte der Analyse, welche im Analysewerkzeug als Eingabe dienen. Außerdem modellieren sie die Einstiegspunkte im Analysewerkzeug und führen die Änderungsausbreitungsanalyse und Aktivitätsherleitung im Werkzeug durch.

6.2.6. Schulungsmaßnahmen

Schulungsmaßnahmen	BG	KG	EXP
Architekturschulung	X	X	
Werkzeugschulung	X		

Tabelle 6.4.: Übersicht der Schulungsmaßnahmen

Schon während des Semesters werden alle Teilnehmer von Behandlungsgruppe und Kontrollgruppe in der Modellierung von Software-Architekturen geschult und erhalten das notwendige Wissen, um an den Studien teilnehmen zu können.

Die Architekturschulung vermittelt die grundsätzliche Motivation zur Verwendung von Software-Architekturen, die Begriffsdefinition und Sichten von Architekturen, konkrete Beispiele für Architekturen und deren Bestandteile, Definitionen von Begriffen wie Komponenten, Schnittstellen, Statischen Strukturen, Inbetriebnahme (engl. deployment) sowie architekturbasierte Entwurfsentscheidungen mit Beispielen.

Die Behandlungsgruppe wird vor der Studie in der Analysemethodik und in der Anwendung des Analysewerkzeugs geschult. Den Teilnehmern wird eine Anleitung zur werkzeug-gestützten Analyse ausgehändigt. Anhand eines Beispiels wird eine konkrete Analyse mit dem Werkzeug vorgeführt und dabei das nötige Wissen vermittelt. Auf Verständnisfragen wird direkt eingegangen.

Die Foliensätze für beide Schulungen sind im Replikationspaket [Sta15] beigefügt.

6.2.7. Studienverlauf

Für alle drei Gruppen läuft die Studie in drei Phasen ab, die Orientierungsphase, die Aufgabenbearbeitungsphase und die Abschlussphase.

Der Einstieg in das Experiment erfolgt mit Hilfe eines Orientierungsfragebogens, mit dem sich die Teilnehmer einen Überblick über die zur Verfügung gestellten Informationen verschaffen und die zielgerichtete Informationsrecherche in der textuellen Beschreibung und im Analysewerkzeug üben.

Nach der Einarbeitung mit dem Orientierungsfragebogen kommt der Hauptteil, die Aufgabenbearbeitung.

Jeder Teilnehmer erarbeitet pro Änderungsanfrage eine Aktivitätsliste. Die Aktivitätsliste besteht aus Aktivitäten, wobei jede Aktivität einen Aktivitätstyp, eine textuelle Beschreibung der Aktivität, eine Liste der mit der Aktivität verbundenen Entwicklungsartefakte und betroffene Architekturelemente umfasst.

Die Teilnehmer der Expertengruppe und der Kontrollgruppe ermitteln Aktivitätslisten manuell und tragen ihre Ergebnisse in einen Papierfragebogen ein.

Die Behandlungsgruppe verwendet das ins Modellierungswerkzeug integrierte Aktivitätsherleitungsverfahren um Aktivitätslisten für die Änderungsanfrage zu bestimmen.

Mit Hilfe eines Abschlussfragebogens wird eine Selbsteinschätzung bezüglich der Qualifikation und des Vertrauens in die ermittelten Ergebnisse eingeholt. Der Abschlussfragebogen erfasst bei den Studenten den Studiengang und die Fachsemesterzahl, und bei den Experten die Angabe der aktuellen beruflichen

Position, die Berufserfahrung in der Software-Entwicklung, sowie die Anzahl von Beteiligungen an Entwicklungsprojekten.

Die Studien der Kontrollgruppe und der Behandlungsgruppe finden zeitgleich aber in getrennten Räumen statt. Die Studie der Expertengruppe findet an einem anderen Termin statt. Die Teilnehmer bearbeiten die Aufgaben individuell und nur unter Zuhilfenahme der bereitgestellten Informationen.

6.2.8. Datenauswertung

Alle Daten werden zur Auswertung in einer Tabellenkalkulation erfasst. Mit einer Statistik-Software werden Teilergebnisse zusätzlich aufbereitet.

Aus den Expertenlösungen und in Abstimmung mit der konkreten Implementierung der Fallstudie werden für die vier Änderungsanfragen Referenzlösungen ermittelt. Dazu werden die Lösungen der Experten miteinander verglichen und mit den Entwicklern der Benutzerverwaltung diskutiert. Die Lösungswege für die einzelnen Änderungsanfragen werden in der Implementierung und im Entwicklungsumfeld der Benutzerverwaltung nachgestellt und auf Plausibilität geprüft. Die Einzellösungen der Teilnehmer von Behandlungsgruppe, Kontrollgruppe und Expertengruppe werden zur Referenzlösung in Bezug gesetzt.

6.3. Empirische Studie – Ergebnisse

6.3.1. Behandlungsgruppe

Wir betrachten im Folgenden die Ergebnisse der Behandlungsgruppe. Die Teilnehmer sammelten im Rahmen der Analyse Einstiegspunkte, welche die primäre Eingabemodellierung der Änderungsanfrage im Analysewerkzeug repräsentieren und gaben diese Einstiegspunkte in das Werkzeug ein. Das Werkzeug führte dann die Änderungsausbreitungsanalyse automatisiert durch und berechnete Aktivitätslisten. Die folgenden Abschnitte beschreiben zusammenfassend die Einstiegspunkte und die daraus resultierenden Lösungsvarianten.

Aufgabe 1: In Aufgabe 1 sollte der Datentyp „User“ um ein Attribut erweitert werden. Alle Teilnehmer der Behandlungsgruppe haben den Datentyp „User“ mit einer Modifikationskennzeichnung versehen. Einige Teilnehmer der Behandlungsgruppe haben noch weitere Elemente gekennzeichnet, welche aber bei den übrigen Teilnehmern durch die Änderungsausbreitungsanalyse ebenfalls ausgehend vom Datentyp „User“ ermittelt wurden. Folglich wurde für diesen Umsetzungsweg von allen Teilnehmern der Behandlungsgruppe die gleiche Aktivitätsliste (Variante 1) ermittelt.

Aufgabe 2: In Aufgabe 2 sollte auf eine Schnittstellen-Änderung des externen Authentifizierungsdienstes reagiert werden. Eine Operation dieser Schnittstelle wurde um einen Aufrufparameter erweitert. Die Einstiegspunkte, welche die Teilnehmer der Behandlungsgruppe modellierten, umfassten Modifikationskennzeichnungen der Komponente „Authentication“ oder deren angebotene Schnittstelle „IAuthentication“. Ein Teilnehmer gab als Einstiegspunkt zusätzlich eine Modifikationskennzeichnung der Komponente „UserManagement“ an. Insgesamt resultierten die Einstiegspunkte in zwei Varianten von Aktivitätslisten.

Aufgabe 3: In Aufgabe 3 sollte das Datenformat der REST-Schnittstelle „IUserServiceTomcat“ von JSON nach XML gewechselt werden. Als Einstiegspunkte wurden hier von allen Teilnehmern der Behandlungsgruppe Modifikationskennzeichnungen an der Schnittstelle „IUserServiceTomcat“ oder an der Komponente „UserServiceTomcat“ ermittelt. Alle genannten Einstiegspunkte führten bei allen Teilnehmern der Behandlungsgruppe zur gleichen Aktivitätsliste, demnach gibt es für Aufgabe 3 nur eine einzige Lösungsvariante.

Aufgabe 4: Bei Aufgabe 4 sollte die Hibernate-Komponente aus dem System entfernt und durch eine eigene Implementierung ersetzt werden. Der Teilnehmer BG1 (erschien verspätet zur Studie und) hat diese Aufgabe (aus Zeitgründen) nicht mehr bearbeitet. Alle anderen Teilnehmer wählten als Einstiegspunkte Modifikationskennzeichnungen an den Komponenten „DatabaseAccess“ oder „UserDAO“, was zur Lösungsvariante 4.1 führte. Teilnehmer BG4 hatte zusätzlich als Einstiegspunkt eine Modifikationskennzeichnung an der Schnittstelle „JDBC-Driver“ angegeben, was zur Lösungsvariante 4.2 führte.

Tabelle 6.5 fasst die Lösungsvarianten der Behandlungsgruppe zusammen.

Teilnehmer	Aufgabe 1	Aufgabe 2	Aufgabe 3	Aufgabe 4
BG1	Variante 1	Variante 2.1	Variante 3	N/A
BG2	Variante 1	Variante 2.1	Variante 3	Variante 4.1
BG3	Variante 1	Variante 2.1	Variante 3	Variante 4.1
BG4	Variante 1	Variante 2.1	Variante 3	Variante 4.2
BG5	Variante 1	Variante 2.1	Variante 3	Variante 4.1
BG6	Variante 1	Variante 2.2	Variante 3	Variante 4.1

Tabelle 6.5.: Lösungsvarianten der Behandlungsgruppe

Kommen wir nun zu den Ergebnissen der Behandlungsgruppe. Wie im GQM-Plan festgelegt, betrachten wir zum einen das F1-Maß zu den Aktivitätstypen und zum anderen das F1-Maß zu den Aktivitätsanreicherungen.

Betrachten wir die Ergebnisse der Behandlungsgruppe zu den **Aktivitätstypen**.

Alle Aufgaben: Insgesamt werden durch die Behandlungsgruppe einzelne Aktivitätstypen sowohl nicht genannt als auch einzelne zuviel angegeben. Das führt zu einer Reduktion von Trefferquote und Genauigkeit. Es wird ein F1-Maß für alle Aufgaben im Durchschnitt von 0,914 erreicht.

Bezogen auf die einzelnen Aufgaben schneidet die Behandlungsgruppe wie folgt ab.

Aufgabe 1: Es gibt nur eine Lösungsvariante für Aufgabe 1. Bei Aufgabe 1 erreicht die Behandlungsgruppe ein F1-Maß von durchschnittlich 0,857. Dabei sind Abstriche bei der Trefferquote mit 0,750 maßgeblich. Das kommt daher, dass zwei verlangte Aktivitätstypen (Metadatenbearbeitung, Inbetriebnahmekonfiguration) nicht durch das Werkzeug angegeben werden.

Aufgabe 2: Es gibt zwei Lösungsvarianten für Aufgabe 2, welche aber bei den Aktivitätstypen gleich abschneiden. Bei Aufgabe 2 erreicht die Behandlungsgruppe ein F1-Maß von durchschnittlich 1,0.

Aufgabe 3: Es gibt nur eine Lösungsvariante für Aufgabe 3. Bei Aufgabe 3 erreicht die Behandlungsgruppe ein F1-Maß von durchschnittlich 1,0.

Aufgabe 4: Es gibt zwei Lösungsvarianten für Aufgabe 4, welche aber bei den Aktivitätstypen gleich abschneiden. Bei Aufgabe 4 erreicht die Behandlungsgruppe ein F1-Maß von durchschnittlich 0,8. Die Ursache liegt darin, dass der Aktivitätstyp Testaktualisierung/Testentwicklung zuviel angegeben wird und die Aktivitätstypen Baukonfiguration und Inbetriebnahmekonfiguration nicht angegeben werden.

Zusammenfassung: 1) Die Behandlungsgruppe schneidet bei den Aktivitätstypen mit fünf Fehlern insgesamt gut ab. 2) Die Ergebnisse der Behandlungsgruppe sind durch die Werkzeugunterstützung sehr homogen innerhalb der Gruppe. Pro Aufgabe gibt es jeweils nur wenige Lösungsvarianten.

Betrachten wir die Ergebnisse der Behandlungsgruppe zu den **Aktivitätsanreicherungen**.

Alle Aufgaben: Insgesamt werden durch die Behandlungsgruppe Anreicherungen sowohl nicht genannt als auch zuviel angeben. Das führt zu einer Reduktion von Trefferquote auf 0,944 und Genauigkeit auf 0,771. Es wird ein F1-Maß für alle Aufgaben von durchschnittlich 0,839 erreicht.

Bezogen auf die einzelnen Aufgaben schneidet die Behandlungsgruppe wie folgt ab.

Aufgabe 1: Es gibt nur eine Lösungsvariante für Aufgabe 1. Bei Aufgabe 1 erreicht die Behandlungsgruppe ein F1-Maß von durchschnittlich 0,852. Dabei sind Abstriche bei der Trefferquote mit 0,867 und bei der Genauigkeit mit 0,867 maßgeblich. Vier Annotationen fehlen in der Lösung, Fünf Annotationen werden zu viel angegeben. Ursachen: Die fehlenden Elemente hängen damit zusammen, dass eine Quelltextannotation zum Datentyp (User.java) fehlt oder nicht korrekt ausgewertet wird. Ebenso werden Annotationen zu den Metadaten User.hbm.xml und UserDatabase.sql nicht korrekt ausgewertet. Eine Annotation zur Laufzeitinstanz fehlt ebenfalls. Testannotationen zu ServiceAvailability und Authentication werden zu viel angegeben, sowohl bei der Testaktualisierung als auch bei der Testdurchführung.

Aufgabe 2: Bei Aufgabe 2 gibt es zwei Lösungsvarianten. Die Behandlungsgruppe erreicht bei Aufgabe 2 ein F1-Maß von durchschnittlich 0,790. Dabei sind Abstriche auf eine Genauigkeit von 0,659 zurückzuführen. Ursachen: Von allen Teilnehmern werden sechs Annotationen zu den Testfällen der

REST-Schnittstelle bei der Testaktualisierung zu viel angegeben. Ein Teilnehmer hat darüber hinaus weitere zehn Testannotationen bei Testaktualisierung und Testdurchführung zu viel.

Aufgabe 3: Es gibt nur eine Lösungsvariante für Aufgabe 3. Bei Aufgabe 3 erreicht die Behandlungsgruppe ein F1-Maß von durchschnittlich 1,0.

Aufgabe 4: Es gibt zwei Lösungsvarianten für Aufgabe 4. Bei Aufgabe 4 erreicht die Behandlungsgruppe ein F1-Maß von durchschnittlich 0,712. Dabei sind Abstriche bei der Trefferquote von 0,909 und bei der Genauigkeit von 0,585 maßgeblich. Zwei Quelltextannotationen sind in allen Lösungen zu viel angegeben, ebenso elf Testannotationen bezüglich Testaktualisierung. Eine Testannotation wird zu wenig angegeben sowie eine Annotation zur Baukonfiguration. In der Lösung eines Teilnehmers kommt eine Metadatenannotation zu viel vor.

Zusammenfassung: Fehlende Annotationen und Auswertungsfehler vorhandener Annotationen führen zu reduzierter Trefferquote und Genauigkeit. Insgesamt erreicht die Behandlungsgruppe jedoch eine hohes Gesamtergebnis bei den Aktivitätsanreicherungen.

6.3.2. Kontrollgruppe

Betrachten wir die Ergebnisse der Kontrollgruppe.

Im Hinblick auf die **Aktivitätstypen** schneidet die Kontrollgruppe wie folgt ab.

Alle Aufgaben: Insgesamt erreicht die Kontrollgruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,850. Dabei sind Abstriche sowohl bei der Genauigkeit von 0,911 als auch bei der Trefferquote von 0,814 maßgeblich.

Betrachten wir die Aufgaben individuell.

Aufgabe 1: Bei Aufgabe 1 erreicht die Kontrollgruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,907. Dabei fließen eine Genauigkeit von 1 und eine Trefferquote von 0,844 ein.

Aufgabe 2: Bei Aufgabe 2 erreicht die Kontrollgruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,894. Dies geht aus einer Genauigkeit von 0,909 und einer Trefferquote von 0,896 hervor.

Aufgabe 3: Bei Aufgabe 3 erreicht die Kontrollgruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,824. Dabei fließen eine Genauigkeit von 0,856 und eine Trefferquote von 0,813 ein.

Aufgabe 4: Bei Aufgabe 4 erreicht die Kontrollgruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,775. Dabei fließen eine Genauigkeit von 0,879 und eine Trefferquote von 0,703 ein.

Zusammenfassung: Bei den Aktivitätstypen kommen bei der Kontrollgruppe nur sehr wenig Fehler vor, so dass das Ergebnis hoch ist.

Betrachten wir, wie die Kontrollgruppe hinsichtlich der **Aktivitätsanreicherungen** abschneidet.

Alle Aufgaben: Insgesamt erreicht die Kontrollgruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,416. Dabei sind Abstriche sowohl bei der Genauigkeit von 0,865 als auch bei der Trefferquote von 0,312 maßgeblich. Die sehr niedrige Trefferquote deutet darauf hin, dass die Probleme der Kontrollgruppe schwerpunktmäßig bei nicht angegebenen Anreicherungen liegen.

Aufgabe 1: Bei Aufgabe 1 erreicht die Kontrollgruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,620. Dies geht aus einer Genauigkeit von 0,939 und einer Trefferquote von 0,475 hervor.

Aufgabe 2: Bei Aufgabe 2 erreicht die Kontrollgruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,389. Dabei fließen eine Genauigkeit von 0,948 und eine Trefferquote von 0,268 ein.

Aufgabe 3: Bei Aufgabe 3 erreicht die Kontrollgruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,352. Das geht aus einer Genauigkeit von 0,628 und einer Trefferquote von 0,299 hervor.

Aufgabe 4: Bei Aufgabe 4 erreicht die Kontrollgruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,306. Hierbei fließen eine Genauigkeit von 0,947 und eine Trefferquote von 0,205 mit ein.

Zusammenfassung:

Bei den Aktivitätsanreicherungen sind die Ergebnisse deutlich schlechter im Vergleich zu den Aktivitätstypen. Insbesondere fehlende Aktivitätsanreicherungen führen zu einer stark reduzierten Trefferquote.

6.3.3. Expertengruppe

Im Hinblick auf die **Aktivitätstypen** schneidet die Expertengruppe wie folgt ab.

Alle Aufgaben: Insgesamt erreicht die Expertengruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,941. Dabei fließen eine Genauigkeit von 0,965 und eine Trefferquote von 0,929 ein. Demnach ermittelt die Expertengruppe die Aktivitätstypen sehr vollständig und sehr genau.

Betrachten wir die Aufgaben individuell.

Aufgabe 1: Bei Aufgabe 1 erreicht die Expertengruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,958. Hier fließen eine Genauigkeit von 1,0 und eine Trefferquote von 0,925 ein.

Aufgabe 2: Bei Aufgabe 2 erreicht die Expertengruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,951. Das geht aus einer Genauigkeit von 0,938 und Trefferquote von 0,967 hervor.

Aufgabe 3: Bei Aufgabe 3 erreicht die Expertengruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,969. Das geht aus einer Genauigkeit von 0,943 und einer Trefferquote von 1,0 hervor.

Aufgabe 4: Bei Aufgabe 4 erreicht die Expertengruppe bezüglich Aktivitätstypen ein F1-Maß von durchschnittlich 0,887. Hier fließen eine Genauigkeit von 0,978 und eine Trefferquote von 0,825 mit ein.

Zusammenfassung: Bei den Aktivitätstypen erreicht die Expertengruppe ein hohes Ergebnis.

Betrachten wir, wie die Expertengruppe hinsichtlich der **Aktivitätsanreicherungen** abschneidet.

Alle Aufgaben: Insgesamt erreicht die Expertengruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,598. Dabei sind Abstriche sowohl bei der Genauigkeit von 0,947 als auch bei der Trefferquote von 0,506

maßgeblich. Die sehr niedrigen Werte bei der Trefferquote gehen aus einer großen Anzahl von nicht angegebenen Anreicherungen hervor.

Aufgabe 1: Bei Aufgabe 1 erreicht die Expertengruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,736. Hier fließen eine Genauigkeit von 0,929 und eine Trefferquote von 0,640 ein.

Aufgabe 2: Bei Aufgabe 2 erreicht die Expertengruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,488. Das geht aus einer Genauigkeit von 0,913 und einer Trefferquote von 0,371 hervor.

Aufgabe 3: Bei Aufgabe 3 erreicht die Expertengruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,758. Hier fließen eine Genauigkeit von 0,945 und eine Trefferquote von 0,722 ein.

Aufgabe 4: Bei Aufgabe 4 erreicht die Expertengruppe bezüglich angereicherter Aktivitäten ein F1-Maß von durchschnittlich 0,409. Das geht aus einer Genauigkeit von 1,0 und einer Trefferquote von 0,291 hervor.

Zusammenfassung: Bei den Aktivitätsanreicherungen fallen die Ergebnisse der Expertengruppe im Vergleich zu den Aktivitätstypen deutlich schlechter aus. Insbesondere fehlende Aktivitätsanreicherungen führen zu reduzierten Trefferquoten.

6.3.4. Vergleich der Gruppen

In diesem Abschnitt vergleichen wir die Ergebnisse der Gruppen untereinander.

Schauen wir zuerst auf das F1-Maß für die Aktivitätstypen. Die Abbildung 6.2 zeigt das F1-Maß für Aktivitätstypen für alle Aufgaben im Vergleich der drei Gruppen. Demnach ist die Behandlungsgruppe im Vergleich zur Expertengruppe etwas schlechter, allerdings besser als die Kontrollgruppe. Die Expertengruppe ist gegenüber der Kontrollgruppe besser. Die Streuung ist bei der Behandlungsgruppe sehr gering. Die Kontrollgruppe weist hingegen eine größere Streuung auf.

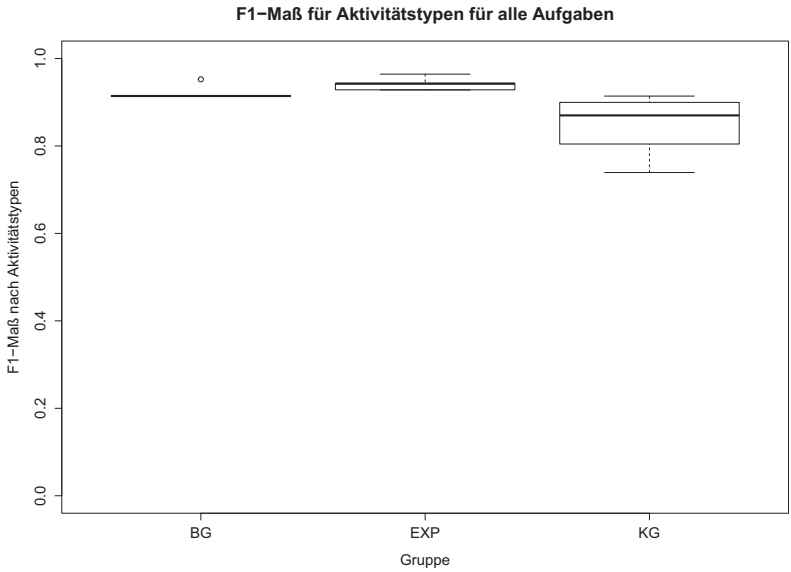


Abbildung 6.2.: F1-Metrik zu Aktivitätstypen für alle Aufgaben

Betrachten wir wie die Gruppen bei den einzelnen Aufgaben abschneiden. Abbildung 6.3 stellt das F1-Maß für die Aktivitätstypen für die drei Gruppen differenziert nach einzelnen Aufgaben dar. Bei den Aufgaben 1 und 4 schneidet die Behandlungsgruppe gegenüber der Expertengruppe und auch gegenüber der Kontrollgruppe schlechter ab. Bei den Aufgaben 2 und 3 ist die Behandlungsgruppe besser als Expertengruppe und Kontrollgruppe.

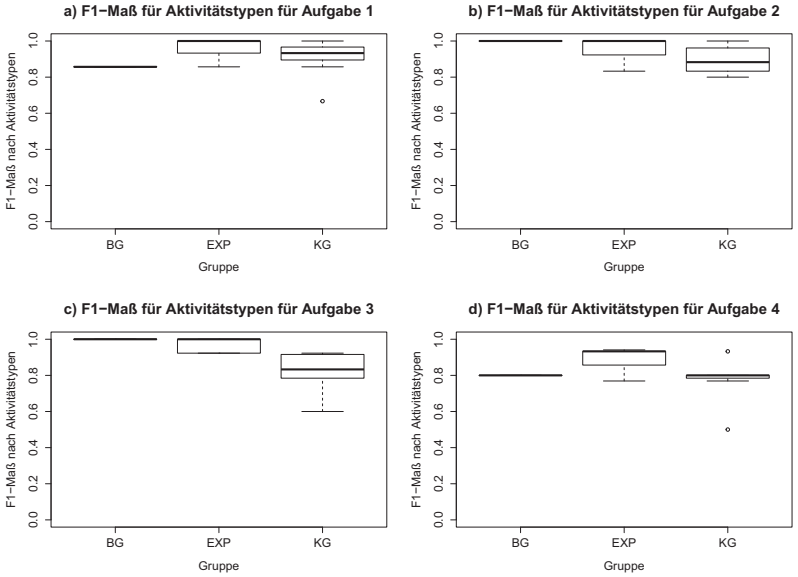


Abbildung 6.3.: F1-Metrik zu Aktivitätstypen für einzelne Aufgaben

Die Ursache für die schlechteren Ergebnisse der Behandlungsgruppe bei den Aufgaben 1 und 4 sind fehlende Annotationen im Anreicherungsmodell, sowie ein Auswertungsfehler im Analysewerkzeug. Trotz der Fehler schneidet die Behandlungsgruppe insgesamt gut ab.

Betrachten wir die Ergebnisse für die Aktivitätsanreicherungen.

Abbildung 6.4 stellt das F1-Maß für die Aktivitätsanreicherungen für die drei Gruppen durchschnittlich für alle Aufgaben dar. Demnach ist die Behandlungsgruppe deutlich besser als die Expertengruppe und die Kontrollgruppe. Die Expertengruppe ist besser als die Kontrollgruppe. Die Kontrollgruppe weist eine hohe Streuung auf.

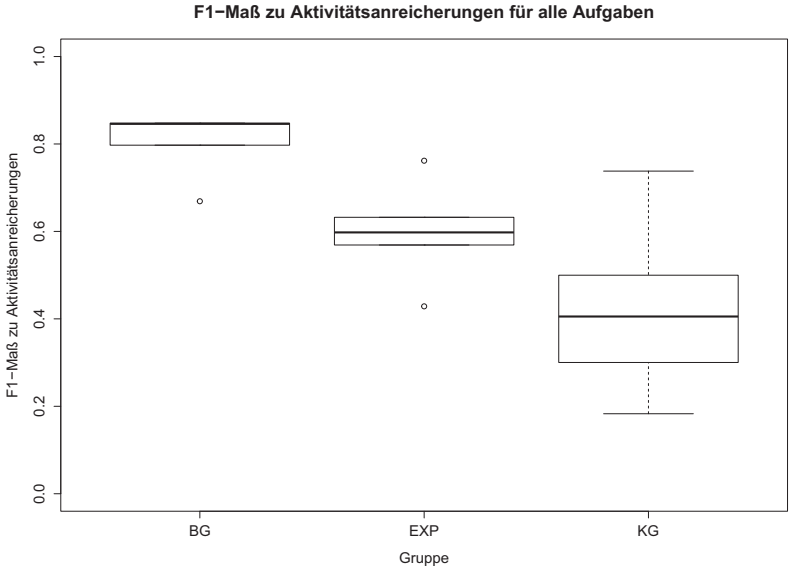


Abbildung 6.4.: F1-Metrik zu Aktivitätsanreicherungen für alle Aufgaben

Betrachten wir die Ergebnisse zu den Aktivitätsanreicherungen für die einzelnen Aufgaben. Abbildung 6.5 stellt das F1-Maß für die Aktivitätsanreicherungen für die drei Gruppen differenziert für die einzelnen Aufgaben dar. Bei allen vier Aufgaben ist die Behandlungsgruppe besser als Expertengruppe und Kontrollgruppe.

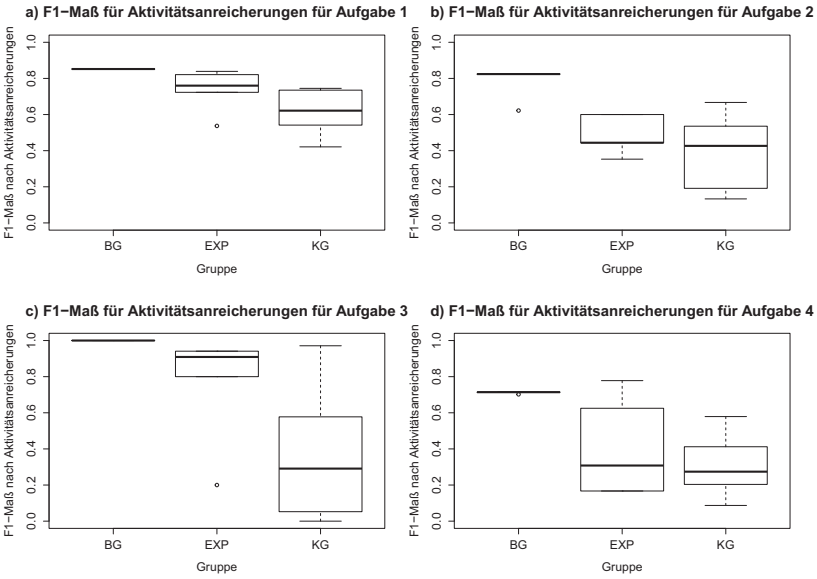


Abbildung 6.5.: F1-Metrik zu Aktivitätsanreicherungen für einzelne Aufgaben

Zusammenfassung des Gruppenvergleichs: 1) Bezüglich der Aktivitätstypen ist die Behandlungsgruppe etwas schlechter als die Expertengruppe, aber besser als die Kontrollgruppe. Die Expertengruppe ist demnach besser als die Kontrollgruppe. 2) Bezüglich der Aktivitätsanreicherungen ist die Behandlungsgruppe besser als die Expertengruppe und besser als die Kontrollgruppe. Die Expertengruppe ist besser als die Kontrollgruppe.

6.3.5. Gesamtergebnis

Nach Auswertung der Daten fließen die Gesamtergebnisse wieder in den GQM-Plan ein wie in den Tabellen 6.6 und 6.7 zu sehen ist. Bei den Werten handelt es sich um die Durchschnittswerte aus allen vier Aufgaben und über alle Teilnehmer der jeweiligen Gruppe.

Schlussfolgerungen:

Bei den Aktivitätstypen ist die Behandlungsgruppe besser als die Kontrollgruppe und etwas schlechter als die Expertengruppe. Dennoch erreicht die Behandlungsgruppe ein hohes Ergebnis.

Bei den Aktivitätsanreicherungen erfüllt die Behandlungsgruppe deutlich die Hypothesen, dass das automatisierte Verfahren eine gleiche oder sogar bessere Ergebnisqualität im Vergleich zur Kontrollgruppe und Expertengruppe erreichen kann.

6.3.6. Zeitbedarf

Tabelle 6.8 fasst die Zeitbedarfe der Teilnehmer pro Aufgabe in Minuten zusammen.

Beobachtungen:

Die Behandlungsgruppe benötigte für alle Aufgaben durchschnittlich am wenigsten Zeit im Vergleich zu Kontrollgruppe und Expertengruppe.

Für die Aufgabe 1 brauchen alle Gruppen durchschnittlich am längsten. Die Kontrollgruppe und die Expertengruppe brauchen für Aufgabe 1 fast doppelt so lange wie für die übrigen Aufgaben. Bei der Behandlungsgruppe ist der Unterschied von Aufgabe 1 zu den restlichen Aufgaben nicht so groß.

Die Expertengruppe benötigt im Vergleich zur Kontrollgruppe etwas länger für die Aufgaben.

Bei drei Teilnehmern fehlen Zeitangaben zur Aufgabe 4. Sie wurden für diese Aufgabe aus der Berechnung des Durchschnitts ausgeschlossen.

Interpretationen:

Die Automatisierung führt zu einer deutlichen Zeitersparnis bei der Analyse.

Mögliche Erklärungen zu Aufgabe 1: Der zusätzliche Zeitbedarf entsteht bei der ersten Aufgabe durch Einlesen in Beschreibung, Zusammensuchen der benötigten Anreicherungen und manuelles Aufschreiben. Außerdem konnten bei nachfolgenden Aufgaben Teillösungselemente aus vorigen Aufgaben referenziert werden, d.h. der Schreibaufwand wurde reduziert.

Schlussfolgerungen:

Insgesamt benötigt die Behandlungsgruppe für alle Aufgaben durchschnittlich deutlich weniger Zeit als die anderen beiden Gruppen.

Die Automatisierung bietet demnach einen deutlichen zeitlichen Vorteil.

Bei größeren Software-Systemen mit mehr Kontextinformationen dürfte sich dieser Effekt noch vergrößern, was auf Vorteile des Verfahrens im Sinne der Skalierbarkeit hindeutet.

Der Skalierungsvorteil verstärkt sich zusätzlich wenn größere Anzahlen von Änderungsanfragen analysiert werden.

6.3.7. Ergebnisse aus dem Orientierungsfragebogen

Die Ergebnisse des Orientierungsfragebogens werden im Anhang in Abschnitt C.3 zusammengefasst. Korrekte Antworten sind mit 1 gekennzeichnet, Fehler mit 0.

Die Behandlungsgruppe erreicht beim Orientierungsfragebogen durchschnittlich 94 Prozent. Teilnehmer BG4 kommt mit 8 falschen Antworten auf 69 Prozent. Die Probleme liegen hier bei der Anzahl von Komponenten, beim Verständnis externer Dienste und bei Anreicherungen zur Objekt-Relationalen-Abbildung. Trotz vermeintlicher Verständnisprobleme von Teilnehmer BG4 kommt dieser Teilnehmer bei den Aufgaben zu einem Gesamtergebnis von 0,914 beim F1-Maß zu Aktivitätstypen und 0,844 beim F1-Maß zu Aktivitätsanreicherungen.

Die Expertengruppe erreicht beim Orientierungsfragebogen ein Gesamtergebnis von durchschnittlich 96 Prozent. Teilnehmer EXP1 hat eine falsche Antwort (Anzahl Komponenten in Textueller Beschreibung) und erreicht damit 96 Prozent. Teilnehmer EXP5 kommt mit 3 falschen Antworten auf 88 Prozent.

Die Kontrollgruppe erreicht beim Orientierungsfragebogen durchschnittlich 95 Prozent. Teilnehmer KG8 schafft mit 4 falschen Antworten 84 Prozent. Die falschen Antworten liegen alle bei den Annotationen zur Objekt-Relationalen-Abbildung.

Schlussfolgerungen:

Insgesamt weisen alle Gruppen beim Orientierungsfragebogen hohe Ergebnisse aus. Damit kann von einem weitestgehend gleichen und vor allem ausreichenden Kenntnisstand der Teilnehmer ausgegangen werden.

6.3.8. Ergebnisse aus dem Abschlussfragebogen

Der Abschlussfragebogen erfasste die Aspekte Berufliche Qualifikation, Erfahrung mit der Software-Entwicklung, Erfahrung und Wissen über Modellierung, Verständnisprobleme bei den Aufgaben, Verständnisprobleme aufgrund der Sprache, Zufriedenheit mit der Lösung, Angemessenheit der Bearbeitungszeit und die allgemeine Befindlichkeit nach der Studie. Außerdem wurden allgemeine Anregungen und Verbesserungsvorschläge gesammelt. Die Behandlungsgruppe wurde zusätzlich zum Umgang mit dem Werkzeug befragt.

Die berufliche Qualifikation war bei allen studentischen Teilnehmern nahezu gleich. Bis auf vier Teilnehmer waren alle im dritten Fachsemester des Bachelor-Studiums Informatik. Vier Teilnehmer waren in höheren Semestern desselben Studiengangs. Die Expertengruppe setzte sich aus wissenschaftlichen Mitarbeitern zusammen.

Drei Teilnehmer der Kontrollgruppe, zwei Teilnehmer der Behandlungsgruppe und alle Teilnehmer der Expertengruppe haben bereits gegen Entgelt Software entwickelt. Für drei Teilnehmer der Kontrollgruppe und zwei Teilnehmer der Behandlungsgruppe war das Praktikum das erste Entwicklungsprojekt. Alle anderen Teilnehmer, auch die der Expertengruppe, waren entweder während des Studiums oder außerhalb des Studiums an anderen Software-Entwicklungs-Projekten beteiligt. Die Anzahl der vergangenen Projekte der Expertengruppe übersteigt deutlich die der anderen Gruppen. Behandlungsgruppe und Kontrollgruppe sind sehr ähnlich.

Kenntnisse von UML-Komponentendiagrammen wurden von allen Teilnehmern bestätigt. Kenntnisse des Palladio Component Models werden von fünf Teilnehmern der Kontrollgruppe und vier Teilnehmern der Behandlungsgruppe nicht bestätigt, obwohl alle Teilnehmer eine Architekturschulung erhielten, in der das Palladio Component Model vorgestellt wurde.

Verständnisprobleme mit Artefakten wurden von den Teilnehmern von Kontrollgruppe und Behandlungsgruppe nicht vermerkt. Ein Teilnehmer der Kontrollgruppe und zwei Teilnehmer der Behandlungsgruppe gaben an, dass ihre Verständnisprobleme geklärt werden konnten. Zwei Teilnehmer der Expertengruppe vermerkten Verständnisprobleme. EXP3 suchte nach Schnittstellen im Code. EXP4 kannte die Unterschiede zwischen JSON und XML nicht. Diese zwei Teilnehmer hatten im Studienverlauf diese Verständnisprobleme nicht zur Sprache gebracht, so dass diese während der Studie nicht geklärt werden konnten.

Die empfundene Zufriedenheit mit der Lösung zeichnet sich sehr heterogen ab. Klare Zufriedenheit mit der Lösung kommt bei der Behandlungsgruppe deutlich häufiger vor als bei der Kontrollgruppe und bei der Behandlungsgruppe. Bei der Behandlungsgruppe kommt „Eher Nein“ bei drei Teilnehmern bei Aufgabe 1 vor. Ansonsten schneiden die Lösungen der Behandlungsgruppe in der Mehrzahl mit „Ja“, in wenigen Fällen mit „Eher Ja“ ab. Bei der Kontrollgruppe dominiert das „Eher Ja“, vermehrt tritt auch das „Eher Nein“ auf. Teilnehmer KG3 bewertet seine Lösung für Aufgabe 3 als nicht zufriedenstellend. Bei der Expertengruppe dominiert das klare „Ja“, gefolgt vom „Eher Ja“, gefolgt vom „Eher Nein“. Teilnehmer EXP1 bewertet seine Lösung für Aufgabe 4 als nicht zufriedenstellend.

Die Bearbeitungszeit empfanden alle Teilnehmer als ausreichend. Vier Teilnehmer der Kontrollgruppe, drei Teilnehmer der Behandlungsgruppe und ein Teilnehmer der Expertengruppe empfanden die Bearbeitungszeit mehr als ausreichend.

Ziel:	Evaluierung der Unterschiede zwischen Aktivitätslisten bestimmt durch werkzeuggestützte Änderungsanfragenanalyse und Aktivitätslisten, welche manuell bestimmt werden.
Zweck: Problem: Objekt: Perspektive:	Evaluierung der Unterschiede von Aktivitätslisten im Vergleich zur Referenzlösung Vollständigkeit und Genauigkeit von Aktivitätslisten Aktivitätslisten Anwender des automatisierten Verfahrens
Frage 1:	Wie vollständig und präzise sind die Aktivitätslisten von weniger erfahrenen Anwendern mit automatisiertem Verfahren **(Behandlungsgruppe, BG)** im Vergleich zu Aktivitätslisten weniger erfahrener Anwender ohne automatisiertes Verfahren **(Kontrollgruppe, KG)**
Frage 1.1:	... bezüglich der **Aktivitätstypen**?
Metrik 1.1.1:	$F_1(Aktivitätstypen, BG)$: 0,914
Metrik 1.1.2:	$F_1(Aktivitätstypen, KG)$: 0,850
Hypothese:	$F_1(Aktivitätstypen, BG) \geq F_1(Aktivitätstypen, KG)$, d.h. automatisiert bestimmte Lösung ist gleich oder besser **ERFÜLLT**
Frage 1.2:	... bezüglich der **Aktivitätsanreicherungen**?
Metrik 1.2.1:	$F_1(Aktivitätsanreicherungen, BG)$: 0,839
Metrik 1.2.2:	$F_1(Aktivitätsanreicherungen, KG)$: 0,416
Hypothese:	$F_1(Aktivitätsanreicherungen, BG) \geq F_1(Aktivitätsanreicherungen, KG)$, d.h. automatisiert bestimmte Lösung ist gleich oder besser **ERFÜLLT**

Tabelle 6.6.: GQM-Plan mit Ergebnissen (Teil 1)

Frage 2:	Wie vollständig und präzise sind die Aktivitätslisten von weniger erfahrenen Anwendern mit automatisiertem Verfahren **(Behandlungsgruppe, BG)** im Vergleich zu Aktivitätslisten erfahrener Anwender ohne automatisiertes Verfahren **(Expertengruppe, EXP)** ...
Frage 2.1:	... bezüglich der **Aktivitätstypen**?
Metrik 2.1.1:	$F_1(Aktivitätstypen, BG)$: 0,914
Metrik 2.1.2:	$F_1(Aktivitätstypen, EXP)$: 0,941
Hypothese:	$F_1(Aktivitätstypen, BG) \geq F_1(Aktivitätstyp, EXP)$, d.h. automatisiert bestimmte Lösung ist gleich oder besser **NICHT ERFÜLLT**
Frage 2.2:	... bezüglich der **Aktivitätsanreicherungen**?
Metrik 2.2.1:	$F_1(Aktivitätsanreicherungen, BG)$: 0,839
Metrik 2.2.2:	$F_1(Aktivitätsanreicherungen, EXP)$: 0,598
Hypothese:	$F_1(Aktivitätsanreicherungen, BG) \geq F_1(Aktivitätsanreicherungen, EXP)$, d.h. automatisiert bestimmte Lösung ist gleich oder besser **ERFÜLLT**

Tabelle 6.7.: GQM-Plan mit Ergebnissen (Teil 2)

Minuten	**Aufg. 1**	**Aufg. 2**	**Aufg. 3**	**Aufg. 4**	**Summe**
Exp1	26	15	11	16	68
Exp2	26	13	6	9	54
Exp3	39	15	9	10	73
Exp4	32	12	12	10	66
Exp5	16	12	14	9	51
Durchschnitt Exp	**27,8**	**13,4**	**10,4**	**10,8**	**62,4**
KG1	25	12	8	10	55
KG2	16	10	14	9	49
KG3	17	11	9	8	45
KG4	15	6	6	6	33
KG5	22	10	8	9	49
KG6	20	13	11	k. A.	k. A.
KG7	18	10	15	8	51
KG8	19	14	12	9	54
Durchschnitt KG	**19**	**10,75**	**10,38**	**8,43**	**48**
BG1	16	8	13	k. A.	k. A.
BG2	8	6	9	6	29
BG3	9	5	5	4	23
BG4	7	4	6	5	22
BG5	13	13	6	k. A.	k. A.
BG6	9	7	2	5	23
Durchschnitt BG	**10,33**	**7,17**	**6,83**	**5**	**24,25**

Tabelle 6.8.: Zeitbedarf der Teilnehmer pro Aufgabe in Minuten

6.4. Validitätsbetrachtungen

In diesem Abschnitt diskutieren wir Aspekte, welche einen unerwünschten Einfluss auf die Gültigkeit der Ergebnisse haben können und zeigen auf, mit welchen Maßnahmen wir diesen Einflüssen entgegengewirkt haben.

6.4.1. Anforderungen an empirische Studien mit Studierenden

Bei der Konzeption der empirischen Studie orientierten wir uns an der Checkliste für empirische Studien von Carver et.al. [Car+10]. Carver et.al. identifizieren einige Anforderungen für erfolgreiche empirische Studien mit Studenten, welche wir in Tabelle 6.9 zusammenfassen.

Diesen Anforderungen kann man nach Carver et.al. mit der Checkliste in Tabelle 6.10 begegnen. Mit folgenden Maßnahmen wurden diese Aspekte bei der Experimentplanung und -durchführung berücksichtigt.

Das Praktikum wurde so ausgewählt, dass es thematisch sehr gut zur Studie passte. Die Fragestellungen der Studie wurden so formuliert, dass diese sich mit den Lernzielen des Praktikums und dem Qualifikationsgrad der Teilnehmer vereinbaren ließen.

Die Studienvorbereitung und -durchführung wurde eng mit dem Praktikumsbetrieb verzahnt. Schulungstermine wurden frühzeitig in die Terminplanung aufgenommen.

Die betrachtete Fallstudie wurde so gewählt, dass sie sich in das Software-System des Praktikums einfügte. Das Teilsystem, das als Studienobjekt diente, wurde bereits vor dem Praktikum konzipiert und implementiert und von den Studierenden als Drittanbieter-System in ihre Anwendung eingebunden. Das betonte sowohl den Aspekt der Integration der Studie mit dem Praktikum, als auch den Aspekt der Wiederverwendung einerseits im Praktikum als auch in der Studie.

Die Studierenden wurden frühzeitig über die Studien in Kenntnis gesetzt, jedoch ohne sie mit den Studienzielen vertraut zu machen. Die Daten wurden anonym erhoben.

A1	Herausforderungen der externen Validität müssen bewusst berücksichtigt werden
A2	Die Studie muss ordentlich in den Kurs integriert werden
A3	Ethische Fragen müssen angemessen durch den Studienentwurf aufgegriffen werden
A4	Die richtigen Studien-Ziele müssen basierend auf der Umgebung gewählt werden
A5	Der Studienaufbau muss angemessen sein in Bezug auf die Ziele, die Fähigkeiten, und die zu untersuchenden Aktivitäten
A6	Die Auswirkungen von Unterschieden zwischen Studien-Population und Ziel-Population müssen diskutiert werden
A7	Studenten sollten den Wert von empirischen Studien für die Bewertung von Produkten und Prozessen lernen und lernen, wie sie diese selbst anwenden können
A8	Gruppenarbeit und Zusammenarbeit sollten in einer Studie einbezogen werden
A9	Studien sollten Entwicklungsprojekte umfassen

Tabelle 6.9.: Anforderungen für erfolgreiche empirische Studien nach [Car+10]

Alle Experimentunterlagen wurden vorab durch sachverständige Dritte Korrektur gelesen. Außerdem wurden Experimentdurchläufe geprobt und dabei erkannte Verständnisprobleme und technische Schwierigkeiten behoben.

Strategien und Methoden zur Überwachung der Experiment-Variablen sind unter anderem die gezielten Schulungsmaßnahmen, die gleichmäßige Einarbeitung mit Hilfe des Orientierungsfragebogen und die Selbsteinschätzung mit Hilfe des Abschlussfragebogens.

Die Experimentunterlagen und -ergebnisse wurden in einem Replikationspaket [Sta15] zusammengefasst.

6.4.2. Interne Validität

Die interne Validität eines Experiments ist dann sichergestellt, wenn die beobachtete Wirkung des Experiments eindeutig auf die Behandlung zurückzuführen ist. Im Umkehrschluss bedeutet das, dass man außer der Behandlung

	Bevor der Kurs beginnt:
1.	Stelle eine angemessene Integration der Studie in das Kursthema sicher
2.	Integriere den Zeitplan der Studie mit dem Kurs-Zeitplan
3.	Verwende Artefakte und Werkzeuge wieder, soweit angemessen
4.	Schreibe ein Protokoll und lasse es überprüfen
	Sobald der Kurs beginnt:
5.	Erlange die Erlaubnis der Teilnehmer für ihre Beteiligung an der Studie
6.	Setze die Erwartungen der Teilnehmer
	Bei Beginn der Studie:
7.	Dokumentiere detaillierte Informationen über den Experiment-Kontext
8.	Etabliere „Strategien und Methoden“ zur Kontrolle und Überwachung der Experiment-Variablen
	Bei Abschluss der Studie:
9.	Plane nachfolgende Aktivitäten
10.	Erzeuge bzw. aktualisiere ein „Experiment-Paket“

Tabelle 6.10.: Checkliste für erfolgreiche empirische Studien nach [Car+10]

für alle Teilnehmer gleiche Grundvoraussetzungen schaffen muss und andere Einflüsse auf das Ergebnis ausschließen muss.

In unserem Experiment besteht die Behandlung aus der Anwendung des automatisierten Verfahrens der Änderungsanfragenanalyse. Dies umfasst zum einen die methodische Vorgehensweise, welche das Verfahren mit sich bringt als auch die konkrete Verwendung des Analysewerkzeugs.

Mit folgenden Maßnahmen wurde die interne Validität gestützt.

1) Bei der Bildung der Gruppen wurde auf eine gleichmäßige Besetzung geachtet. Größere Gruppen sind bei einer Studie vorteilhaft. In dieser Studie standen uns 19 Teilnehmer insgesamt zur Verfügung.

2) Mit Hilfe von Schulungsmaßnahmen wurden alle Teilnehmer auf den gleichen Kenntnisstand gebracht. Die Behandlungsgruppe nahm, spezifisch für die Behandlung, an einer Schulung für das Analyseverfahren und das Werkzeug teil.

3) Alle Teilnehmer erhielten - bis auf die spezifischen Unterschiede für die Behandlung - die gleichen Materialien und Informationen.

4) Der Orientierungsfragebogen synchronisierte den Wissensstand der Teilnehmer über das konkrete System.

5) Die Aufgaben wurden von den Teilnehmern individuell bearbeitet, d.h. deren Ergebnisse sind unabhängig voneinander.

6) Mit Hilfe des Abschlussfragebogens, wurden Unklarheiten und Missverständnisse abgegriffen, sowie die Einschätzung und das Vertrauen in die eigene Lösung abgefragt.

Nachfolgend diskutieren wir mögliche Einflussfaktoren auf die Ergebnisse.

Fehlende Annotationen und ein Fehler im Analysewerkzeug haben dazu geführt, dass in den Ergebnissen der Behandlungsgruppe Aktivitätstypen nicht auftauchten. Das führte zu reduzierten Ergebnissen der Behandlungsgruppe.

Die Teilnehmer der Kontrollgruppe und der Expertengruppe durften im Rahmen der manuellen Erfassung der Analyserergebnisse Abkürzungen verwenden oder auf Teilergebnisse anderer Aufgaben verweisen. Das sollte den Schreibaufwand reduzieren. Wir können einen negativen Einfluss auf die Ergebnisse daraus nicht erkennen.

Die Resultate der Orientierungsfragebögen sind hoch, was auf einen gleichmäßigen und ausreichenden Kenntnisstand der Teilnehmer hinweist.

Mit Hilfe des Abschlussfragebogens sind wir auf vereinzelte Verständnisprobleme aufmerksam geworden. Die exakten Auswirkungen auf die Ergebnisqualität können nicht nachvollzogen werden.

Die Bestimmung der Referenzlösung in Abstimmung mit den Entwicklern des Systems hat Einfluss auf die Ergebnisse. Bei der Erstellung der Referenzlösung sind wir objektiv vorgegangen und es wurde darauf geachtet, Mehrdeutigkeiten zu vermeiden.

6.4.3. Externe Validität

Die externe Validität eines Experiments bezieht sich auf die Übertragbarkeit von Ergebnissen aus dem Experiment auf andere Situationen.

In unserem Experiment gibt es sowohl Aussagen zu wenig erfahrenen Anwendern als auch zu erfahrenen Anwendern. Das Studienziel wurde so gewählt, dass es zu den vertretenen Anwendergruppen passt. Die Studienfragestellungen waren so konzipiert, dass die richtigen Zielgruppen miteinbezogen wurden.

Die Einschränkungen der Aktivitätstypen mit Hilfe von Listen und Beispielen war eine Hilfe, die in realen Umgebungen oftmals nicht vorhanden ist.

Die Architekturmodelle waren im Experiment vorgegeben. Die Erstellung und Aktualisierung von Architekturmodellen war nicht Teil der Experimentbetrachtungen. In realen Szenarien muss das Architekturmodell mit den Anreicherungen zuvor erst erstellt werden.

Das automatisierte Analyseverfahren wurde nicht durch die Expertengruppe eingesetzt. Demnach kann nichts über die Anwendung des Verfahrens durch erfahrene Anwender gesagt werden. Nach Aussage der Teilnehmer der Expertengruppe würde das automatisierte Verfahren ihnen erhebliche Vorteile bringen. Die Auswertung der Informationen für Änderungsanfragen per Hand stelle einen hohen manuellen Aufwand dar und sei sehr fehlerträchtig.

Darüber hinaus sind in der Praxis die Kontextinformationen meist verstreuter auf verschiedene Personen oder sogar Abteilungen. Im Experiment war in der textuellen Beschreibung bereits viel zusammengefasst. Das reduziert aber nicht die Aussagekraft der Ergebnisse. Vielmehr motiviert dies weitere Forschungsarbeiten, die untersuchen, wie die verteilten Informationen zusammengetragen und automatisiert in ein Modell integriert werden können.

6.5. Zusammenfassung und Schlussfolgerungen

Fassen wir die Validierung der Qualitätseigenschaften des automatisierten Änderungsanfragen-Analyse-Verfahrens zusammen.

Der Hauptvorteil der Automatisierung ist in der Skalierbarkeit zu sehen. Dem gegenüber muss sichergestellt werden, dass die automatisierten Ergebnisse eine ausreichende Qualität aufweisen. Die Qualität wird hierbei im Sinne von

Genauigkeit und Trefferquote im Vergleich zu manuellen Referenzlösungen betrachtet.

Um diese Ergebnisqualität zu evaluieren wurde eine empirische Studie konzipiert. Die Skalierbarkeit wurde nur indirekt durch den Zeitbedarf innerhalb der Studie untersucht. Das Kapitel präsentiert die Studienziele anhand eines GQM-Plans. Hierbei ist der Vergleich von automatisierten Analyseergebnissen mit manuell ermittelten Ergebnissen das Ziel, und zwar zum einen im Vergleich von weniger erfahrenen Anwendern zu weniger erfahrenen Anwendern und zum anderen von weniger erfahrenen Anwendern zu erfahrenen Anwendern.

Als Teilaspekte der Qualität von Aktivitätslisten wurde die Genauigkeit und Trefferquote für Aktivitätstypen und für Aktivitätsanreicherungen untersucht. Als Kombinationsmetrik für Genauigkeit und Trefferquote wird die F1-Metrik verwendet.

In Bezug auf den Studienentwurf wurden die Zielsetzung, die Aufgabenstellungen, das untersuchte Software-System, die Gruppen, die Materialien, sowie die Schulungsmaßnahmen vorgestellt. Zudem wurde der Studienverlauf und die Datenauswertung beschrieben.

Insgesamt lassen sich die Studienergebnisse wie folgt zusammenfassen.

Bei den manuellen Analysen wurden Anreicherungen mehrfach vergessen. Das führte dazu, dass die Behandlungsgruppe bezüglich den Anreicherungen besser als die Kontrollgruppe und die Expertengruppe war. Die Werkzeug-Unterstützung bringt Vorteile gegenüber der manuellen Analyse.

Die Ergebnisse der automatisierten Analyse innerhalb der Gruppe sind deutlich homogener. Die Konzentration auf einzelne Einstiegspunkte in die Analyse und die automatisierte Erfassung durch die Änderungsausbreitungsanalyse führen zu wenigen Ergebnisvarianten der Behandlungsgruppe. Ausgehend vom gleichen Einstiegspunkt der Modellierung eines Umsetzungswegs ermittelt das Verfahren gleiche Aktivitätslisten. Elemente die durch die Änderungsausbreitung indirekt betroffen sind werden dabei miterfasst unabhängig ob sie explizit bei der Einstiegsmodellierung dabei sind oder nicht. Dem gegenüber waren die manuellen Ergebnisse von Kontrollgruppe und Expertengruppe deutlich heterogener.

Das Verfahren berücksichtigt Aspekte (wie z.B. die strukturelle Änderungsausbreitung) explizit, die von weniger erfahrenen Anwenders übersehen werden können. Die resultierenden Ergebnisse sind somit unabhängiger von der Erfahrung.

Die Studie hat gezeigt, dass die Ergebnisqualität des automatisierten Verfahrens im Vergleich zu manuellen Analysen gleich oder besser ist. Allerdings können Modellierungsfehler oder Fehler im Analysewerkzeug zu einer reduzierten Ergebnisqualität führen. Wir empfehlen daher Analyseergebnisse regelmäßig auf Plausibilität zu überprüfen und die Modelle entsprechend zu verbessern.

Das Verfahren schafft durch die Automatisierung einen Skalierungseffekt bei größeren Modellen oder einer größeren Anzahl von analysierten Umsetzungswegen.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Der Kontext der Arbeit bildet die Zusammenarbeit von Projektverwaltern und Software-Architekten im Rahmen der Software-Evolution. Im Rahmen der Software-Evolution treten regelmäßig Änderungsanfragen auf. Für diese Änderungsanfragen müssen Umsetzungswege bestimmt werden.

Die Arbeit adressiert das Problem, dass die Beteiligten für ihre Entscheidungen wissen müssen, welche Konsequenzen aus einem gewählten Umsetzungsweg resultieren. Folglich wird eine Auswirkungsanalyse benötigt.

Die Projektverwantwortlichen sollen bei folgenden Fragen unterstützt werden:

- Was sind die strukturellen Auswirkungen eines geplanten Umsetzungswegs?
- Was sind die organisatorischen Auswirkungen eines geplanten Umsetzungswegs?
- Welche Aktivitäten sind zur Realisierung eines Umsetzungswegs, insbesondere in nachgelagerten Tätigkeitsfeldern und Lebenszyklusphasen erforderlich?
- Welche Artefakte (Dateien, Konfigurationen, etc.) müssen für die Umsetzung bearbeitet werden?

Die Arbeit stellt eine Reihe von gewünschten Eigenschaften vor, welche eine zufrieden stellende Lösung erfüllen soll.

Verknüpfung von Architekturmodellierung und Projektverwaltung

Das Verfahren sollte Architekturmodellierung und Projektverwaltung explizit miteinander verbinden.

Leichtgewichtigkeit

Es ist ein leichtgewichtiger Ansatz gewünscht, der mittels Werkzeugunterstützung einfach in verschiedene Arten von Entwicklungsprozessen eingebunden werden kann und schnelle Rückkopplung ermöglicht.

Lebenszyklusabdeckung

Es ist ein Verfahren gewünscht, welches den Lebenszyklus des Software-Systems ganzheitlich betrachtet und dabei in verschiedenen Phasen, d.h. Software-Entwurf, Software-Implementierung und Software-Evolution, eingesetzt werden kann.

Tätigkeitsfeldabdeckung

Es ist ein Verfahren gefordert, welches die gängigen Tätigkeitsfelder der Software-Entwicklung berücksichtigt, welche in vielen Entwicklungsvorhaben vorkommen. Dazu zählen insbesondere die Tätigkeitsfelder Programmierung, Fremdkomponenten-Konfiguration, Bauen, Testen, Bereitstellung und Inbetriebnahme.

Szenarien-Basiertheit

Das Verfahren sollte die Software-Evolution anhand konkreter Änderungsanfragen bewerten. Änderungsanfragen sind ein Spezialfall von Szenarien.

Berücksichtigung der strukturellen Änderungsausbreitung

Das Verfahren sollte die strukturelle Änderungsausbreitung ausgehend von Modifikationen im System berücksichtigen.

Berücksichtigung der organisatorischen Auswirkungen

Das Verfahren sollte die Auswirkungen von Modifikationen auf die Tätigkeitsfelder bestimmen können.

Skalierbarkeit durch Automatisierung

Das Verfahren sollte die entscheidungsrelevanten Daten automatisiert verarbeiten können.

Die in dieser Arbeit vorgestellte Lösung erfüllt diese Eigenschaften.

Diese Arbeit stellt eine automatisierte Methode zur Ableitung von Aktivitäten aus angereicherten Architekturmodellen vor. Umsetzungswege für Änderungsanfragen können in Architekturmodellen in einem Modellierungswerkzeug beschrieben werden und aus diesen Änderungen automatisiert Aktivitäten für verschiedene Tätigkeitsfelder und Lebenszyklusphasen abgeleitet werden.

Das Verfahren verbindet die Arbeit von Projektverwaltern und Architekten auf der Basis des Architekturmodells. Organisatorische und Technische Entscheidungen werden gezielt unterstützt.

Teilbeitrag Änderungsausbreitungsanalyse:

Mit dem Verfahren berücksichtigen Anwender explizit eine mögliche Ausbreitungen von Änderungen durch Abhängigkeiten innerhalb des Systems.

Teilbeitrag Abdeckung zahlreicher Tätigkeitsfelder im Lebenszyklus:

Das Verfahren ermittelt Aktivitäten für mehrere Tätigkeitsfelder im Lebenszyklus der Software-Entwicklung. Ermittelte Auswirkungen von Änderungen werden somit nicht nur auf die reine Entwicklung beschränkt. Auch die Aspekte der Qualitätssicherung (d.h. das Testen), sowie die Inbetriebnahme rücken durch das Verfahren ins Bewusstsein des Anwenders.

Teilbeitrag Skalierbarkeit durch Automatisierung und hohe Ergebnisqualität:

Die Validierung im Rahmen einer empirischen Studie zeigt, dass das automatisierte Verfahren eine vergleichbare oder sogar besser Ergebnisqualität erreicht, als eine manuelle Analyse. Die Automatisierung erhöht die Skalierbarkeit der Analyse.

7.2. Vorteile

Der Hauptvorteil des Ansatzes liegt in der zentralen Zusammenführung von Kontextinformationen im Architekturmodell und damit der Möglichkeit der Automatisierung der Auswertung dieser Kontextinformationen. Für einzelne Änderungsanfragen ist das Zusammensuchen von Kontextinformationen eine mühsame Angelegenheit, bei der man schnell Artefakte vergisst oder der Aufwand zur Ermittlung zu groß wird. Bei dem hier vorgestellten Ansatz werden diese Informationen einmal ermittelt und als Anreicherungen im Modell bereitgestellt. Der Aufwand für die Modellierung amortisiert sich dann, wenn häufiger und vermehrt Änderungsanfragen untersucht werden. Die Automatisierung bietet als Hauptvorteil eine erhöhte Skalierbarkeit.

Das Verfahren kann alle Änderungsanfragen untersuchen, unabhängig davon, ob die Architektur strukturell verändert wird oder die Änderungen gekapselt in den Komponenten auftreten.

7.3. Grenzen und Annahmen

Der hier vorgestellte Ansatz basiert auf einigen Annahmen, die wir nachfolgend beschreiben.

Komponenten-Orientierung:

Das Verfahren erfordert es, dass das Software-System in Form einer komponenten-basierten Architektur beschrieben werden kann.

Manuelle Abbildung von Umsetzungsweg auf Architektur:

Das Verfahren sieht vor, dass der Anwender zu einer Änderungsanfrage selbst den Umsetzungsweg bestimmen muss. Das bedeutet, dass der Anwender die Änderungsanfrage auf die Komponenten, Schnittstellen und Datentypen abbilden muss. Das Verfahren bietet keine automatisierte Ableitung von Umsetzungswegen aus Anforderungen oder Anforderungsänderungen.

Architekturrekonstruktion und Modellerstellung:

Der Anwender des Verfahrens muss die komponenten-basierte Architektur eines Software-Systems rekonstruieren und das Architekturmodell vor der Analyse erstellen.

Beschaffung der Kontextinformationen:

Das Verfahren erfordert es, dass die Kontextinformationen zu den Tätigkeitsbereichen und Lebenszyklusphasen zusammengetragen werden und das Architekturmodell mit den Kontextinformationen angereichert werden.

Detailgrad der Modellierung hat Einfluss auf Ergebnis:

Der Anwender kann den Detailgrad der Architekturmodellierung selbst entscheiden. Der Detailgrad der resultierenden Aktivitätslisten hängt dem entsprechend davon ab.

7.4. Zukünftige Arbeiten

In diesem Abschnitt betrachten wir weitere mögliche Forschungsfragen, die aus der hier vorgestellten Arbeit hervorgehen oder thematisch angrenzen.

Fragenblock: Abbildung von Anforderungen auf die Architektur

Ein Fragenblock ergibt sich bezüglich der Abbildung von Anforderungen auf das Architekturmodell. Wie lassen sich Anforderungsänderungen automatisiert auf das Architekturmodell abbilden? Wie lassen sich Entwurfsentscheidungen automatisieren? Wie bestimmt man automatisiert Umsetzungswege? Hier könnten zum Beispiel Verfahren zur Modellierung, Dokumentation und Analyse von Entwurfsentscheidungen, wie [DR13] oder [Küs13] eingebunden werden.

Fragenblock: Erstellung von Architekturmodellen

Ein Fragenblock besteht bei der Erstellung und Aktualisierung von Architekturmodellen aus vorhandenen Software-Systemen. Wie lassen sich reale Systeme systematisch mit komponenten-basierten Architekturmodellen beschreiben? Wie lassen sich solche Modelle automatisiert aus den Software-Artefakten erstellen? Verfahren aus dem Bereich Reverse Engineering, wie zum Beispiel [Kro10] und Faktenextraktoren könnten hier integriert und weiterentwickelt werden.

Fragenblock: Beschaffung und Modellierung von Kontextinformationen

Offene Fragen, an denen die Forschung anknüpfen kann existieren bei der Beschaffung und Modellierung von Kontextinformationen. Wie können Kontextinformationen automatisiert beschafft und mit Architekturmodellen verknüpft werden? Wie können vorhandene Datenquellen (beispielsweise Versionierungssysteme, Operativen Aufgabenverwaltung, Kontinuierliche Integration) effizient und effektiv ausgewertet werden?

Fragenblock: Integration mit Aufwandsschätzverfahren

Das Verfahren könnte mit Aufwandsschätzverfahren wie Function Points oder CoCoMO II [Boe+00] integriert werden.

Fragenblock: Automatisierte Anwendungen auf den Aktivitätslisten

Das Verfahren zur Änderungsanfragenanalyse liefert als Ergebnis Aktivitätslisten. Nachfolgende Forschungsarbeiten könnten sich damit befassen, wie aus den Aktivitätslisten Entscheidungen abgeleitet werden. Wie können Aktivitätslisten automatisiert verglichen und Entscheidungen abgeleitet werden? Das Verfahren könnte zu einem Simulator von Änderungsanfragen weiter entwickelt werden.

Teil I.

Anhang

A. Transformationsregeln

A.1. Ableitungsregeln zur Quelltextbearbeitung

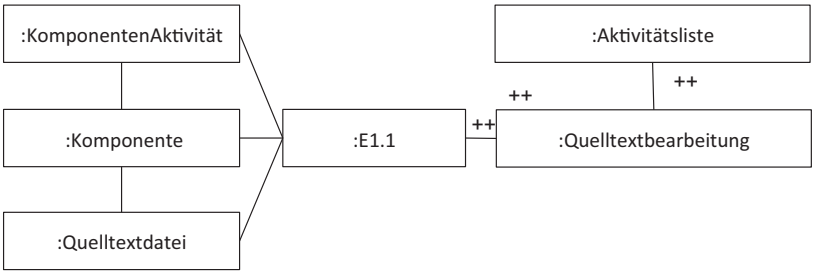


Abbildung A.1.: Triple-Graph-Regel E1.1: Quelltextbearbeitung aus Komponenten-Aktivität mit Quelltextdatei-Anreicherung

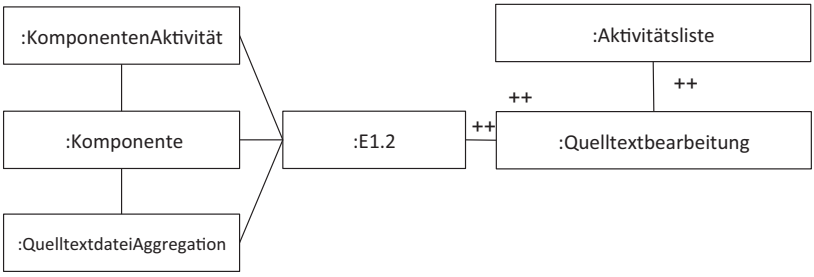


Abbildung A.2.: Triple-Graph-Regel E1.2: Quelltextbearbeitung aus Komponenten-Aktivität mit QuelltextdateiAggregation-Anreicherung

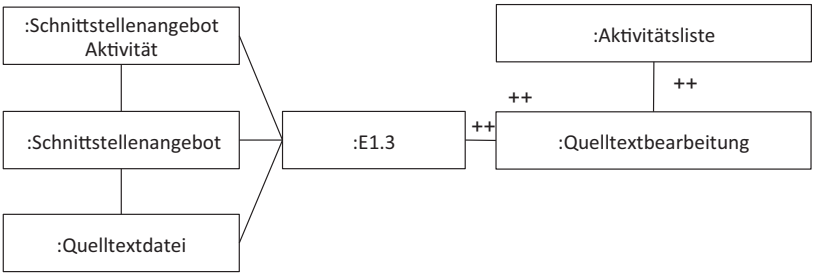


Abbildung A.3.: Triple-Graph-Regel E1.3: Quelltextbearbeitung aus Schnittstellenangebot-Aktivität mit Quelltextdatei-Anreicherung

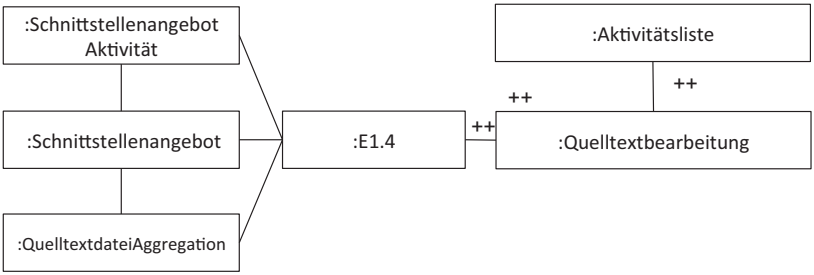


Abbildung A.4.: Triple-Graph-Regel E1.4: Quelltextbearbeitung aus Schnittstellenangebot-Aktivität mit QuelltextdateiAggregation-Anreicherung

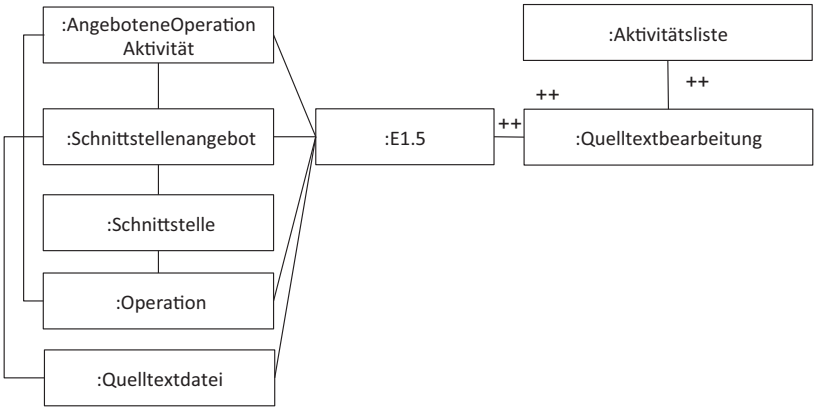


Abbildung A.5.: Triple-Graph-Regel E1.5: Quelltextbearbeitung aus AngeboteneOperation-Aktivität mit Quelltextdatei-Anreicherung

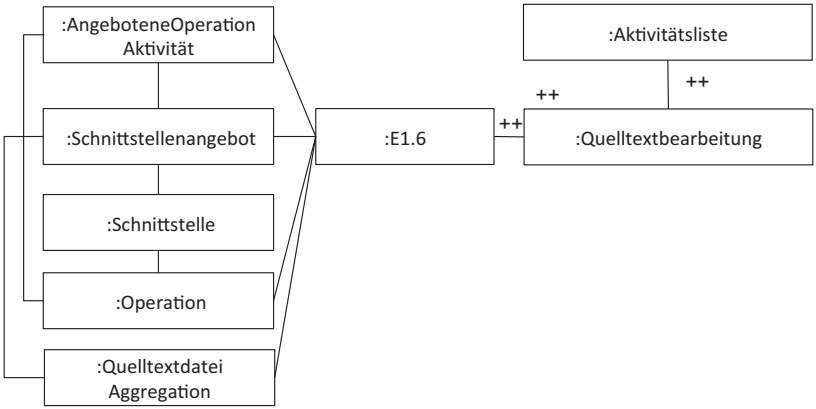


Abbildung A.6.: Triple-Graph-Regel E1.6: Quelltextbearbeitung aus AngeboteneOperation-Aktivität mit QuelltextdateiAggregation-Anreicherung

A.2. Ableitungsregeln zur Metadatenbearbeitung

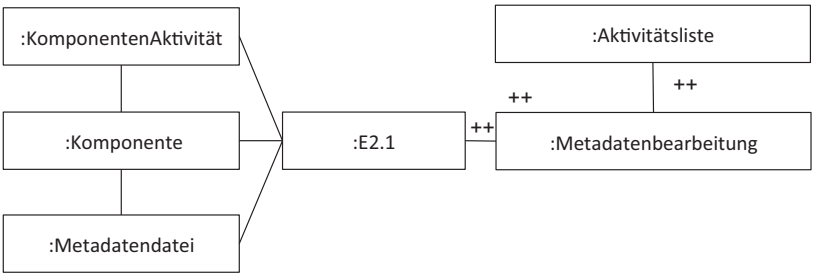


Abbildung A.7.: Triple-Graph-Regel E2.1: Metadatenbearbeitung aus Komponenten-Aktivität mit Quelltextdatei-Anreicherung

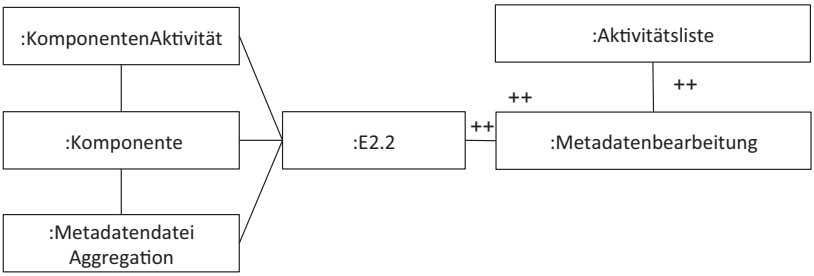


Abbildung A.8.: Triple-Graph-Regel E2.2: Metadatenbearbeitung aus Komponenten-Aktivität mit QuelltextdateiAggregation-Anreicherung

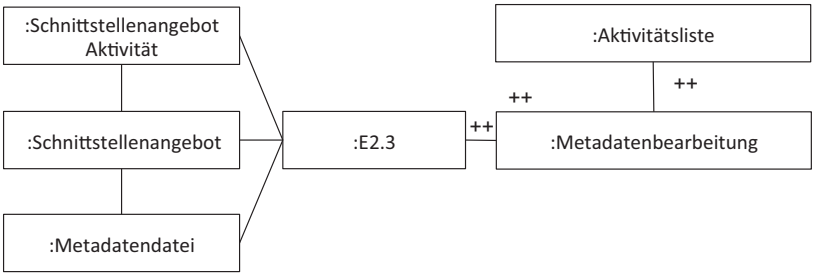


Abbildung A.9.: Triple-Graph-Regel E2.3: Metadatenbearbeitung aus Schnittstellenangebot-Aktivität mit Quelltextdatei-Anreicherung

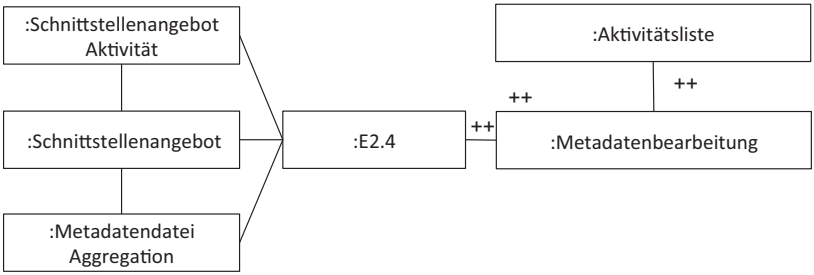


Abbildung A.10.: Triple-Graph-Regel E2.4: Metadatenbearbeitung aus Schnittstellenangebot-Aktivität mit QuelltextdateiAggregation-Anreicherung

A.3. Ableitungsregeln zum Bauen

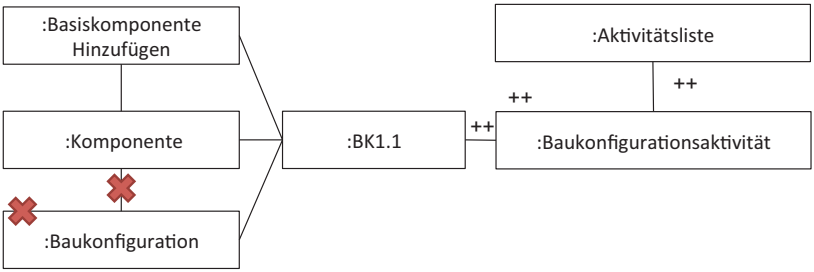


Abbildung A.11.: Triple-Graph-Regel BK1.1: Baukonfigurationsaktivität aus Basiskomponente-Hinzufügen-Aktivität

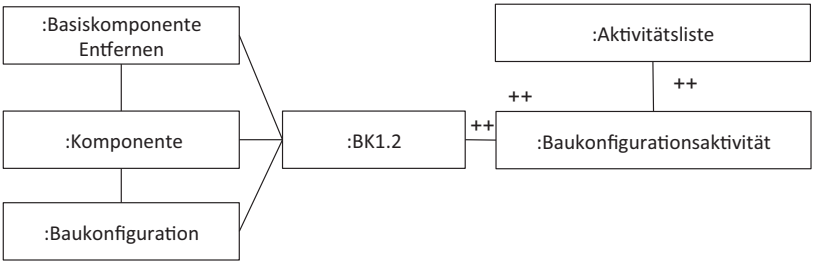


Abbildung A.12.: Triple-Graph-Regel BK1.2: Baukonfigurationsaktivität aus Basiskomponente-Entfernen–Aktivität

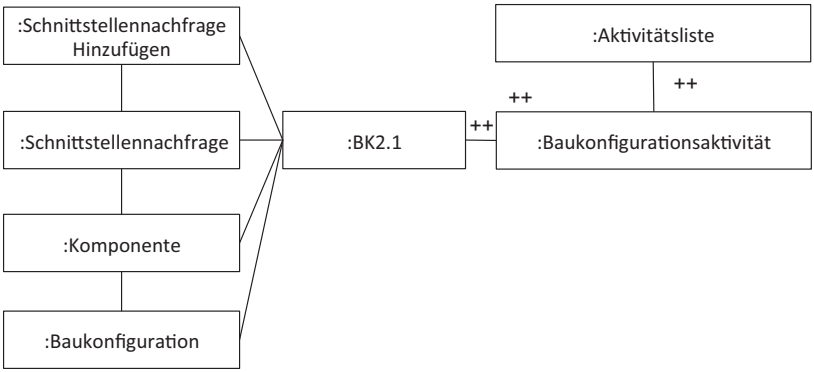


Abbildung A.13.: Triple-Graph-Regel BK2.1: Baukonfigurationsaktivität aus Schnittstellennachfrage-Hinzufügen-Aktivität

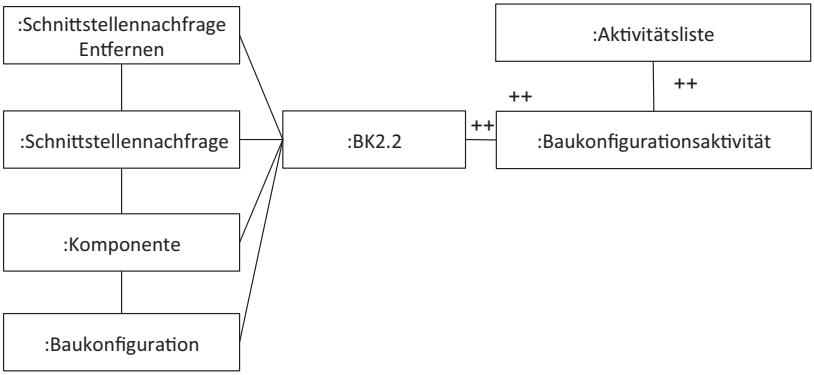


Abbildung A.14.: Triple-Graph-Regel BK2.2: Baukonfigurationsaktivität aus Schnittstellennachfrage-Entfernen–Aktivität

Abbildung A.15.: Triple-Graph-Regel BD1: Baudurchführungsaktivität aus Quelltextbearbeitung

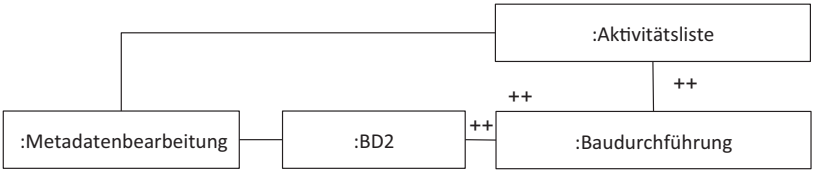


Abbildung A.16.: Triple-Graph-Regel BD2: Baudurchführungsaktivität aus Metadatenbearbeitung

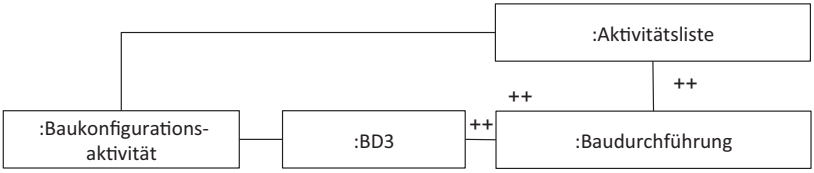


Abbildung A.17.: Triple-Graph-Regel BD3: Baudurchführungsaktivität aus Baukonfiguration

A.4. Ableitungsregeln zum Testen

Abbildung A.18.: Triple-Graph-Regel TE1: Testentwicklungsaktivität (Unit-Tests) aus Schnittstellenangebot-Hinzufügen-Aktivität

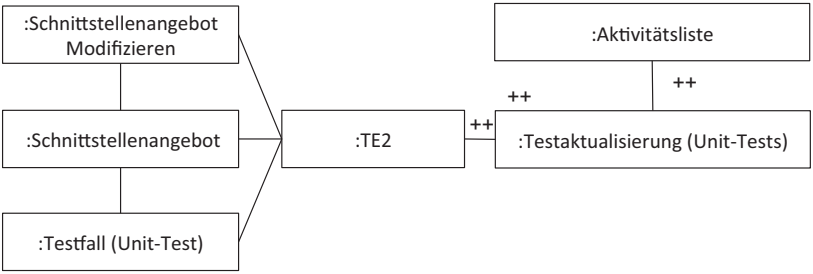


Abbildung A.19.: Triple-Graph-Regel TE2: Testaktualisierungsaktivität (Unit-Tests) aus Schnittstellenangebot-Modifizieren-Aktivität

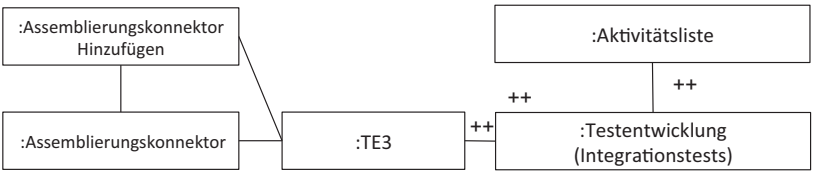


Abbildung A.20.: Triple-Graph-Regel TE3: Testentwicklungsaktivität (Integrationstests) aus Assemblierungskonnektor-Hinzufügen-Aktivität

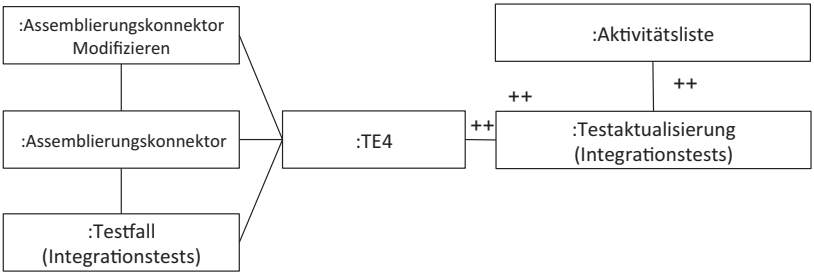


Abbildung A.21.: Triple-Graph-Regel TE4: Testaktualisierungsaktivität (Integrationstests) aus Assemblierungskonnektor-Modifizieren-Aktivität

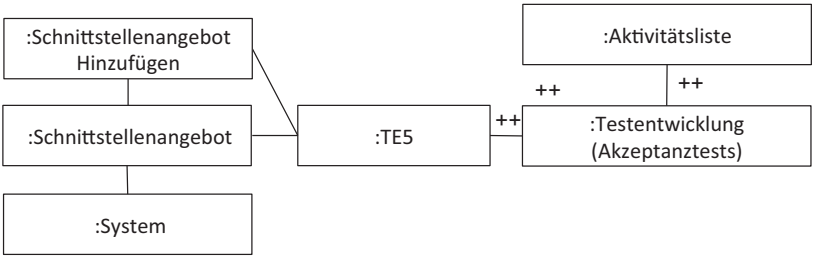


Abbildung A.22.: Triple-Graph-Regel TE5: Testentwicklungsaktivität (Akzeptanztests) aus (Benutzer)-Schnittstellenangebot-Hinzufügen-Aktivität

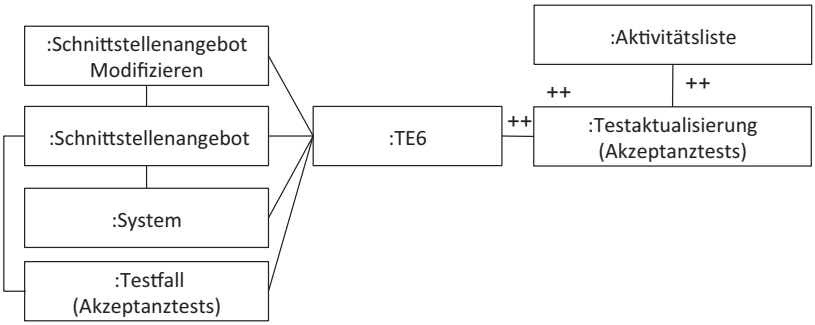


Abbildung A.23.: Triple-Graph-Regel TE6: Testaktualisierungsaktivität (Akzeptanztests) aus (Benutzer)-Schnittstellenangebot-Modifizieren-Aktivität

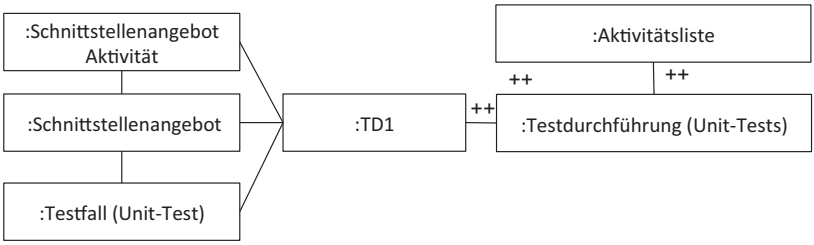


Abbildung A.24.: Triple-Graph-Regel TD1: Testdurchführungsaktivität (Unit-Tests) aus Schnittstellenangebots-Aktivität

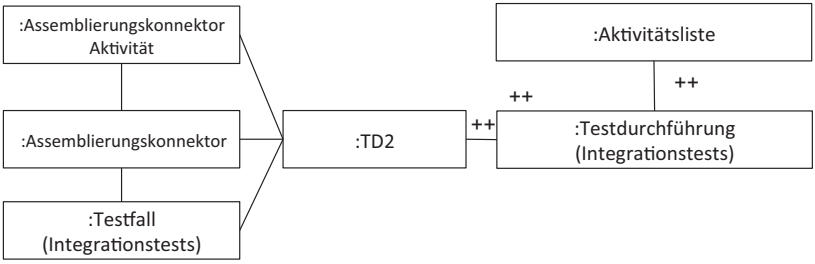


Abbildung A.25.: Triple-Graph-Regel TD2: Testdurchführungsaktivität (Integrationstests) aus Assemblierungskonnektor-Aktivität

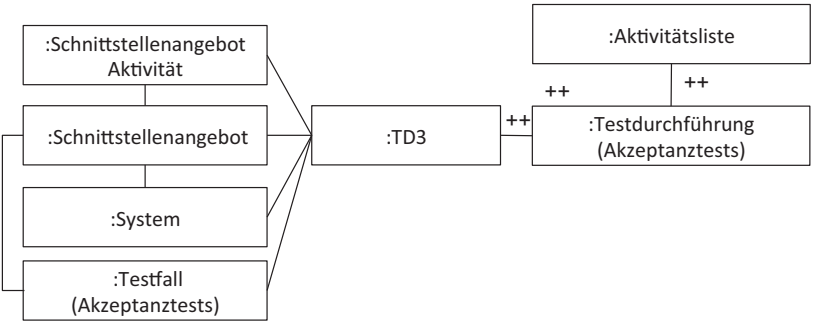


Abbildung A.26.: Triple-Graph-Regel TD3: Testdurchführungsaktivität (Akzeptanztests) aus Benutzer-Schnittstellenangebots-Aktivität

A.5. Ableitungsregeln zur Bereitstellung

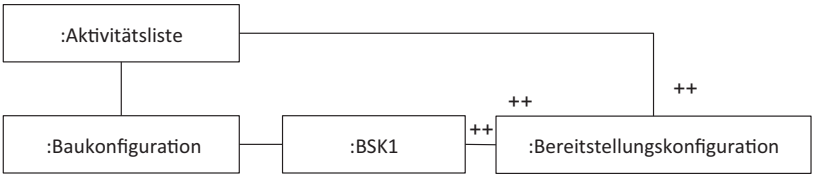


Abbildung A.27.: Triple-Graph-Regel BSK1: Bereitstellungskonfigurationsaktivität aus Baukonfigurationsaktivität

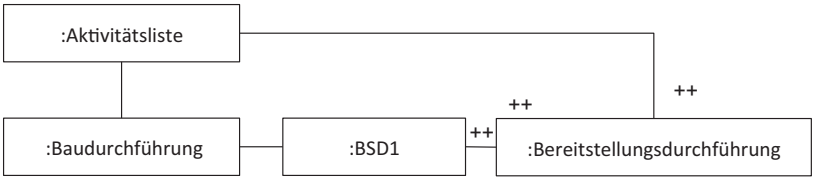


Abbildung A.28.: Triple-Graph-Regel BSD1: Bereitstellungsaktivität aus Bauaktivität

A.6. Ableitungsregeln zur Inbetriebnahme

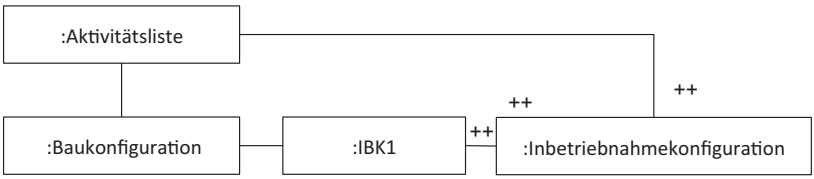


Abbildung A.29.: Triple-Graph-Regel IBK1: Inbetriebnahmekonfigurationsaktivität aus Baukonfigurationsaktivität

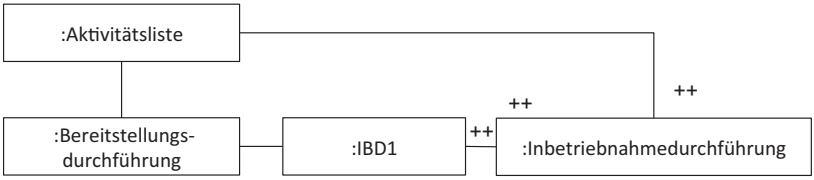


Abbildung A.30.: Triple-Graph-Regel IBD1: Inbetriebnahmedurchführungsaktivität aus Bereitstellungsdurchführungsaktivität

B. Experiment-Unterlagen

B.1. Textuelle Beschreibung

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Textuelle Beschreibung des Benutzerverwaltungssystems

Dieser Abschnitt beschreibt das Benutzerverwaltungssystem welches im Praktikum verwendet wurde. Die folgenden Abschnitte beschreiben zum einen die **Architekturelemente** (Datentypen, Schnittstellen und Komponenten) als auch das **Entwicklungsumfeld** (Quelltextdateien, Metadaten, Bauen, Testen, Bereitstellung und Inbetriebnahme)

Überblick über Datentypen, Schnittstellen und Komponenten

Das Benutzerverwaltungssystem besteht aus den folgenden Datentypen, Schnittstellen und Komponenten.

Datentypen

- **User:** Dieser Datentyp repräsentiert den Benutzer (mit Eigenschaften userId, nickname, firstName, lastName, dateOfBirth, email und password). Das Geburtsdatum „dateOfBirth" ist ein „verpflichtendes Attribut", das immer gesetzt sein muss.
- **UserList:** Dieser Datentyp repräsentiert eine Liste von Benutzern.
- **Report:** Dieser Datentyp repräsentiert einen Bericht.
- **SessionFactory:** Dieser Datentyp repräsentiert eine Fabrik-Klasse, die zur Erzeugung einer Hibernate-Datenbank-Session verwendet wird.
- **Session:** Dieser Datentyp repräsentiert eine Hibernate-Datenbank-Session.
- **Token:** Dieser Datentyp repräsentiert ein Token (Kennzeichen), das vom externen Authentifizierungsdienst für einen authentifizierten Benutzer geliefert wird.
- **WebServiceResponse:** Dieser Datentyp repräsentiert eine allgemeine Web-Service-Antwort.

Schnittstellen

- **IUserServiceTomcat:** Diese Schnittstelle wird durch den Anwender des Systems über REST angesprochen und von den Komponenten *UserServiceApacheProxy* und *UserServiceTomcat* angeboten. Als Datenformat wird JSON verwendet.
- **IUserManagement:** Diese Schnittstelle wird von der Komponente *UserManagement* angeboten und von der Komponente *UserServiceTomcat* nachgefragt und umfasst Operationen zum Speichern, Aktualisieren und Abrufen von Benutzerdaten.
- **IUserManagementAdmin:** Diese Schnittstelle wird von der Komponente *UserManagement* angeboten und von der Komponente *UserServiceTomcat* nachgefragt und umfasst administrative Operationen zum Löschen von Benutzern und Rücksetzen von Passwörtern.
- **IAuthentication:** Diese Schnittstelle wird von der Komponente *Authentication* angeboten und umfasst eine Operation *checkUser* zur Authentifizierung eines Benutzers beim externen Dienst. Die Komponente *UserServiceTomcat* nutzt diese Schnittstelle.
- **IReporting:** Diese Schnittstelle wird von der Komponenten Reporting angeboten und umfasst Operationen zum Erfassen von Berichtdaten (*log*) und zum Abfragen des Berichts (*getReport*). Die Schnittstelle wird von der Komponente *UserServiceTomcat* genutzt.

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

- **IUserDAO:** Diese Schnittstelle wird von der Komponente *UserDAO* angeboten und bietet Operationen zum Erzeugen, Aktualisieren, Abfragen und Löschen von Benutzerdaten in der Datenbank.
- **ISessionFactory:** Diese Schnittstelle wird von der Komponente *DBAccess* angeboten und enthält eine Operation zum Erzeugen/Öffnen einer Session. Die Schnittstelle wird der Komponente *UserDAO* genutzt.
- **ISession:** Diese Schnittstelle wird von der Komponente *DBAccess* angeboten und umfasst Operationen zum Starten, Abschließen und Rückrollen von Transaktionen; sowie zum Speichern, Aktualisieren, Abfragen, Löschen und Schließen von Objekten in der Datenbank. Die Schnittstelle wird der Komponente *UserDAO* genutzt.

Komponenten

- **UserServiceApacheProxy:** Diese Komponente bietet die REST-Schnittstelle *IUserServiceTomcat* an und leitet (d.h. delegiert) die Aufrufe an die nachgefragte Schnittstelle *IUserServiceTomcat* weiter. Diese Komponente dient der Port-Umstellung und reicht die Aufrufe ohne weitere Logik an den UserServiceTomcat weiter.
- **UserServiceTomcat:** Diese Komponente bietet die REST-Schnittstelle IUserServiceTomcat an und dient als Fassade in das Subsystem der Benutzerverwaltung. Sie fragt die Schnittstellen *IUserManagement* und *IUserManagementAdmin* nach. Sie bindet die Berichterstattung (Komponente *Reporting*) (über die nachgefragte Schnittstelle *IReporting*) und die externe Authentifikation (über die nachgefragte Schnittstelle *IAuthentication*) in die Geschäftslogik ein.
- **Authentication:** Diese Komponente repräsentiert einen externen Authentifikationsdienst. Benutzer, die bei diesem Dienst angemeldet sind, können sich über dessen Anmeldedaten authentifizieren.
- **Reporting:** Diese Komponente sammelt Berichte über Vorgänge in der Benutzerverwaltung.
- **UserManagement:** Diese Komponente stellt die Geschäftslogik zur Verwaltung von Benutzern zur Verfügung. Sie bietet die Schnittstellen *IUserManagement* und *IUserManagementAdmin* an.
- **UserDAO:** Diese Komponente repräsentiert die Datenzugriffsobjekte, die das Speichern, Laden und Aktualisieren von Benutzerdaten in der Datenbank regeln. Sie bietet die Schnittstelle *IUserDAO* an und nutzt den OR-Mapper *DBAccess* über die Schnittstellen *ISessionFactory* und *ISession*.
- **DBAccess:** Diese Komponente repräsentiert den Objekt-Relations-Mapper, der die Abbildung des Datentyps „User" auf die Datenbank vornimmt. Es handelt sich dabei um eine Drittanbieter-Komponente. Die Abbildung zwischen Java-Objekt und Datenbankschema erfolgt über eine Abbildungskonfiguration (Mapping-Konfiguration). Der OR-Mapper kommuniziert mit der Datenbank über die JDBCDriver-Schnittstelle und bietet selbst die Schnittstellen *ISessionFactory* und *ISession an.*
- **UserDatabase:** Diese Komponente repräsentiert die Datenbank, in der die Benutzerdaten gespeichert werden. Die Struktur der Datenbank wird über das Datenbankschema spezifiziert.

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Technologien und Entwicklungsartefakte

Folgende Tabelle beschreibt, durch welche Technologie die jeweilige Komponente umgesetzt ist und welche Entwicklungsartefakte ihr zugeordnet werden.

Komponente	**Technologie-Entsprechung**	**Entwicklungsartefakte**
UserServiceApacheProxy	Apache-Server-Proxy (Drittanbieter-Komponente)	Server-Konfiguration zur Port-Umstellung und -Weiterleitung
UserServiceTomcat	Java-Klassen	Quelltextdatei „UserRest.java“
Authentication	(externer Dienst)	
Reporting	Java-Klassen	Quelltextdatei „Reporting.java“
UserManagement	Java-Klassen	Quelltextdatei „UserService.java“
UserDAO	Java-Klassen	Quelltextdatei „UserDAO.java“ Quelltextdatei „HibernateUtil.java“
DBAccess	Hibernate „hibernate.jar“ (Drittanbieter-Komponente)	Konfigurationsdatei „hibernate.cfg.xml“ *(konfiguriert die allgemeinen Einstellung des OR-Mappers Hibernate)* Konfigurationsdatei „User.hbm.xml“ *(beschreibt die Abbildung des User-Datentyps auf das Datenbankschema)*
UserDatabase	MySQL-Datenbank (Drittanbieter-Komponente)	Konfigurationsdatei „UserDatabase.sql“ (definiert das Datenbankschema)

Datentyp	**Technologie-Entsprechung**	**Entwicklungsartefakte**
User	Java-Klasse, Datenobjekt	Quelltextdatei "User.java"

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Testen

Folgende **Unit-Testfälle** sind für das System vorhanden:

- **7 Testfälle** in „UserServiceTomcatTest.java“ (Testen die REST-Webschnittstelle „IUserServiceTomcat“ der Komponente *UserServiceTomcat*)
  - **testServiceAvailabilityREST:** Testet die Verfügbarkeit des Dienstes über die REST-Schnittstelle.
  - **testCreateUserREST:** Testet die Erstellung eines Benutzers über die REST-Schnittstelle.
  - **testUpdateUserREST:** Testet das Aktualisieren eines Benutzers über die REST-Schnittstelle.
  - **testDeleteUserREST:** Testet das Löschen eines Benutzers über die REST-Schnittstelle.
  - **testGetUserREST:** Testet die Abfrage eines Benutzers über die REST-Schnittstelle
  - **testPasswordResetREST:** Testet das Zurücksetzen eines Benutzerpassworts über die REST-Schnittstelle.
  - **testAuthenticationREST:** Testet die externe Authentifizierung über die REST-Schnittstelle.
- **5 Testfälle** in „UserManagementTest.java“ (Testen die Schnittstelle „IUserManagement“ der Komponente „UserManagement“)
  - **testCreateUser:** Testet die Erstellung eines Benutzers an der Service-Schnittstelle.
  - **testUpdateUser:** Testet die Aktualisierung eines Benutzers an der Service-Schnittstelle
  - **testDeleteUser:** Testet die Löschung eines Benutzers an der Service-Schnittstelle
  - **testGetUser:** Testet die Abfrage eines Benutzers an der Service-Schnittstelle
  - **testPasswordReset:** Testet das Zurücksetzen eines Benutzerpassworts an der Service-Schnittstelle
- **1 Testfall** in „AuthenticationTest.java“
  - **testAvailabilityOfAuthenticationService:** Testet die Verfügbarkeit des externen Authentifizierungsdienstes an der *IAuthentication*-Schnittstelle der Komponente *Authentication*.

Bauen

Die Baukonfiguration wird in den Java-Projekt-Eigenschaften (in Eclipse) vorgenommen.
Die Benutzerverwaltung wird lokal im Eclipse-Arbeitsbereich mit Hilfe von integrierten Buildern gebaut und dabei automatisch in einer war-Datei gebündelt.

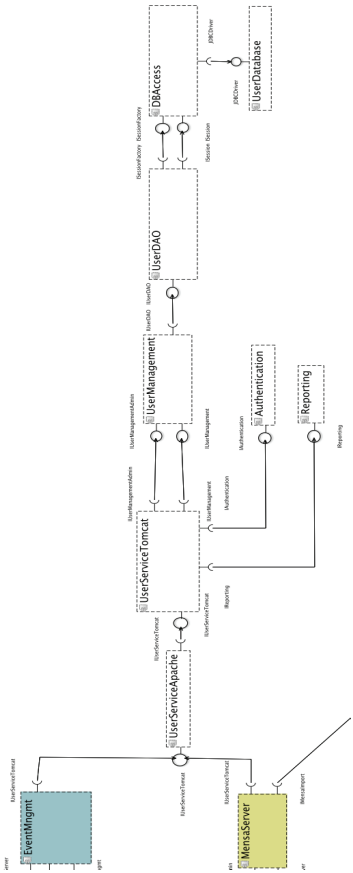
Bereitstellung

Das installationsfähige Web-Service-Paket wird über die Eclipse-Exportfunktion durch den Entwickler erstellt. Das resultierende Format ist eine war-Datei. Das installationsfähige Produkt wird auf einem zentralen Dateiserver zur Installation bereitgestellt. (➔ Datei: **UserManagement.war**)

Inbetriebnahme (Installation)

Einstellungen zum Webservice-Aufruf werden in Konfigurationsdatei „web.xml“ vorgenommen.
Eine Instanz des Systems wird auf einem Tomcat-Server betrieben (➔ Anzahl Laufzeitinstanzen: 1)
Die Inbetriebnahme erfolgt manuell über Netzwerkzugriff.

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel



Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Begriffserklärungen

- ***Datenbank-Schema:*** Ein Datenbank-Schema legt fest, welche Tabellen es in der Datenbank gibt und welche Spalten die jeweilige Tabelle besitzt.
- ***OR-Mapping:*** Mit Hilfe des OR-Mappings werden objektorientierte Datenstrukturen auf Datenbank-Schemas abgebildet. Diese Abbildung wird zum automatischen Lesen und Schreiben von Objekten aus der Datenbank verwendet. In einer Konfigurationsdatei wird beschrieben, welche Objekte auf welche Datenbank-Tabellen abgebildet werden und welche Attribute der Objekte in welcher Spalte der Tabelle gespeichert werden.
- ***JDBCDriver***: JDBC ist der Java-Sprachstandard zum Zugriff auf relationale Datenbanken. Er ist in den Paketen „java.sql" und „javax.sql" der Java-Standard-Bibliothek definiert. Für konkrete Datenbanken (z. B. MySql, Postgres, etc.) gibt es Treiber (Driver), welche diesen Standard implementieren und welche in das eigene Programm zum Zugriff auf die Datenbank eingebunden werden können.
- ***Drittanbieter-Komponente:*** Drittanbieter-Komponenten sind Teile des Programms, die man von anderen bezieht und in das eigene System integriert. In der Regel verfügt man nicht über den Quelltext der Drittanbieter-Komponente, sondern kann diese lediglich konfigurieren.
- ***Externer Dienst:*** Ein externer Dienst wird von Dritten angeboten und über das Netz angesprochen. Der Betrieb des Dienstes liegt nicht im Einflussbereich der Systementwickler oder –betreiber. Externe Dienste können ihre Schnittstelle verändern, so dass das nutzende System angepasst werden muss.

B.2. Orientierungsfragebogen

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

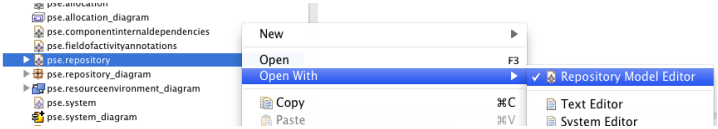
Orientierungsfragebogen

Teilnehmer: _____

Verschaffen Sie sich mit Hilfe dieses Fragebogens einen Überblick über das Benutzerverwaltungssystem.

Für die folgenden Fragen benötigen Sie

I. **die textuelle Beschreibung der Benutzerverwaltung und**
II. **das PCM-Repository-Modell in der Baum-Ansicht, das Sie wie folgt in Eclipse öffnen:**



Hinweis: Die unterschiedlichen Antworten erklären sich dadurch, dass in der textuellen Beschreibung nur Benutzerverwaltung beschrieben wird, das PCM-Modell auch Teile von Event-Management und Mensa enthält.

1) Wie viele Datentypen werden jeweils genannt?

Anzahl in textueller Beschreibung: ___
Anzahl im PCM-Repository-Baum: ___

(Tipp: Wenn Sie die Datentypen in der Baumdarstellung markieren wird unten die Anzahl angezeigt!)

2) Welche Attribute besitzt der Datentyp „Vote“ im PCM-Repository?

Attribut 1: _____
Attribut 2: _____

(Tipp: Klappen Sie den Datentyp in der Baumdarstellung auf!)

3) Wie viele Schnittstellen werden jeweils genannt?

Anzahl in textueller Beschreibung: ___
Anzahl im PCM-Repository-Baum: ___

(Tipp: Wenn Sie die Schnittstellen in der Baumdarstellung markieren wird unten die Anzahl angezeigt!)

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

4) Welche Operationen (Signaturen) besitzt die Schnittstelle „IReporting“?

Operation 1: _____
Operation 2: _____

(Tipp: Klappen Sie die Schnittstelle in der Baumdarstellung auf!)

5) Wie viele Komponenten werden jeweils genannt?

Anzahl in textueller Beschreibung: ___
Anzahl im PCM-Repository-Baum: ___

(Tipp: Wenn Sie die Komponenten in der Baumdarstellung markieren wird unten die Anzahl angezeigt!)

6) Welche Schnittstellenangebote und -nachfragen besitzt die Komponente „DBAccess “?

Schnittstellenangebot 1: _____
Schnittstellenangebot 2: _____
Schnittstellennachfrage: _____

(Tipp: Klappen Sie die Komponente in der Baumdarstellung auf!)

7) Für welche der Komponenten gibt es Quelltext? (siehe textuelle Beschreibung!)

Komponente 1: _____
Komponente 2: _____
Komponente 3: _____
Komponente 4: _____

(Hinweis: SQL zählt nicht als Quelltext!)

8) Welche Komponenten werden als „Externer Dienst oder Drittanbieter-Komponente“ im System verwendet? (siehe textuelle Beschreibung!)

Externer Dienst: _____
Drittanbieter-Komponente 1: _____
Drittanbieter-Komponente 2: _____
Drittanbieter-Komponente 3: _____

9) Was macht die Komponente DBAccess (Hibernate)? (siehe textuelle Beschreibung!)

Erklärung in zwei Sätzen:

_____

_____

10) Wie heißen die zwei Konfigurationsdateien zu DBAccess (Hibernate) und was wird darin konfiguriert?

Konfigurationsdatei 1: _____ Zweck: _____

Konfigurationsdatei 2: _____ Zweck: _____

B.3. Abschlussfragebogen Behandlungsgruppe

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

Abschlussfragebogen

Teilnehmer: _____

1. Sie sind derzeit:

- ☐ Bachelor (Informatik)-Student
- ☐ Master (Informatik)-Student
- ☐ Diplom (Informatik)-Student
- ☐ Sonstiges und zwar _____

2. In welchem Fachsemester sind Sie jetzt?

Ich bin im ____ Fachsemester

3. Haben Sie je gegen Entgelt Software entwickelt?

- ☐ Ja
- ☐ Nein

4. Haben Sie bereits praktische Programmiererfahrungen sammeln können? (Mehrfachnennung möglich)

- ☐ Nein, PSE ist mein erstes Software-Entwicklungsprojekt
- ☐ Ja, und zwar ich habe bereits in ____ (Anzahl) Entwicklungs-Projekten während des Studiums mitgearbeitet
- ☐ Ja, und zwar ich habe bereits in ____ (Anzahl) Entwicklungs-Projekten außerhalb des Studiums mitgearbeitet

Davon waren ____ (Anzahl) Projekte aus der Domäne der mobilen Anwendungen

5. Welche Architekturmodellierungssprachen kennen Sie? Haben Sie diese praktisch in Projekten verwendet?

Ich kenne (aus Vorlesung oder Büchern):

- ☐ UML-Komponentendiagramme
- ☐ Palladio Component Model
- ☐ Sonstiges und zwar _____

Ich habe praktisch verwendet (Modelle erstellt oder geändert):

- ☐ UML-Komponentendiagramme
- ☐ Palladio Component Model
- ☐ Sonstiges und zwar _____

6. Gab es Artefakte, bei denen Sie Verständnisprobleme hatten?

- ☐ Nein, keine
- ☐ Ja, mit folgenden Artefakten _____

Konnten diese geklärt werden?

- ☐ Ja, die wurden geklärt
- ☐ Nein, folgende sind noch offen geblieben _____

Bitte wenden ➔

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

7. Sind Sie mit Ihrer ermittelten Lösung zufrieden?

Aufgabe 1	Ja	Eher Ja	Eher Nein	Nein	Nicht dazu gekommen
Aufgabe 2	Ja	Eher Ja	Eher Nein	Nein	Nicht dazu gekommen
Aufgabe 3	Ja	Eher Ja	Eher Nein	Nein	Nicht dazu gekommen
Aufgabe 4	Ja	Eher Ja	Eher Nein	Nein	Nicht dazu gekommen

8. War die Bearbeitungszeit angemessen?

- ☐ Nein, ich hätte mehr Zeit gebraucht
- ☐ Ja, sie hat geradeso ausgereicht.
- ☐ Ja, ich hatte ausreichend Zeit.
- ☐ Ja, ich hatte mehr als ausreichend Zeit.

9. Hatten Sie Verständnisprobleme mit den Aufgaben oder mit der englischen Sprache während der Studie?

- ☐ Nein, keine
- ☐ Ja, mit folgende Aufgabe _____

Konnten diese geklärt werden?

- ☐ Ja, die wurden geklärt
- ☐ Nein, folgende sind noch offen geblieben _____

10. Ich habe ...

... mich mit den Werkzeugen einfach zurechtgefunden.	Ja	Eher Ja	Eher Nein	Nein
... mich durch das Werkzeug unterstützt gefühlt.	Ja	Eher Ja	Eher Nein	Nein
... genug Zeit gehabt für die Beantwortung der Aufgaben.	Ja	Eher Ja	Eher Nein	Nein
... mich überfordert gefühlt von den Aufgaben.	Ja	Eher Ja	Eher Nein	Nein
... Spaß gehabt bei der Bearbeitung.	Ja	Eher Ja	Eher Nein	Nein

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

12. Würden Sie gerne uns irgendetwas mitteilen?
(Kommentare, Wünsche, Anregungen, Fragen, Probleme, etc.)

_____
_____
_____
_____
_____

Wir bedanken uns ganz herzlich!

Bitte lassen Sie die Rechner eingeloggt! Wir kümmern uns um die Abmeldung!

B.4. Abschlussfragebogen Experten- und Kontrollgruppe

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

Abschlussfragebogen

Teilnehmer: ____

1. Sie sind derzeit:

- ☐ Bachelor (Informatik)-Student
- ☐ Master (Informatik)-Student
- ☐ Diplom (Informatik)-Student
- ☐ Sonstiges und zwar _____

2. In welchem Fachsemester sind Sie jetzt?

Ich bin im ____ Fachsemester

3. Haben Sie je gegen Entgelt Software entwickelt?

- ☐ Ja
- ☐ Nein

4. Haben Sie bereits praktische Programmiererfahrungen sammeln können? (Mehrfachnennung möglich)

- ☐ Nein, PSE ist mein erstes Software-Entwicklungsprojekt
- ☐ Ja, und zwar ich habe bereits in ____ (Anzahl) Entwicklungs-Projekten während des Studiums mitgearbeitet
- ☐ Ja, und zwar ich habe bereits in ____ (Anzahl) Entwicklungs-Projekten außerhalb des Studiums mitgearbeitet

Davon waren ____ (Anzahl) Projekte aus der Domäne der mobilen Anwendungen

5. Welche Architekturmodellierungssprachen kennen Sie? Haben Sie diese praktisch in Projekten verwendet?

Ich kenne (aus Vorlesung oder Büchern):

- ☐ UML-Komponentendiagramme
- ☐ Palladio Component Model
- ☐ Sonstiges und zwar _____

Ich habe praktisch verwendet (Modelle erstellt oder geändert):

- ☐ UML-Komponentendiagramme
- ☐ Palladio Component Model
- ☐ Sonstiges und zwar _____

6. Gab es Artefakte, bei denen Sie Verständnisprobleme hatten?

- ☐ Nein, keine
- ☐ Ja, mit folgenden Artefakten _____

Konnten diese geklärt werden?

- ☐ Ja, die wurden geklärt
- ☐ Nein, folgende sind noch offen geblieben _____

Bitte wenden ➔

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

7. Sind Sie mit Ihrer ermittelten Lösung zufrieden?

Aufgabe 1	Ja	Eher Ja	Eher Nein	Nein	Nicht dazu gekommen
Aufgabe 2	Ja	Eher Ja	Eher Nein	Nein	Nicht dazu gekommen
Aufgabe 3	Ja	Eher Ja	Eher Nein	Nein	Nicht dazu gekommen
Aufgabe 4	Ja	Eher Ja	Eher Nein	Nein	Nicht dazu gekommen

8. War die Bearbeitungszeit angemessen?

- ☐ Nein, ich hätte mehr Zeit gebraucht
- ☐ Ja, sie hat geradeso ausgereicht.
- ☐ Ja, ich hatte ausreichend Zeit.
- ☐ Ja, ich hatte mehr als ausreichend Zeit.

9. Hatten Sie Verständnisprobleme mit den Aufgaben oder mit der englischen Sprache während der Studie?

- ☐ Nein, keine
- ☐ Ja, mit folgende Aufgabe _____

Konnten diese geklärt werden?

- ☐ Ja, die wurden geklärt
- ☐ Nein, folgende sind noch offen geblieben _____

10. Ich habe ...

... genug Zeit gehabt für die Beantwortung der Aufgaben.	Ja	Eher Ja	Eher Nein	Nein
... mich überfordert gefühlt von den Aufgaben.	Ja	Eher Ja	Eher Nein	Nein
... Spaß gehabt bei der Bearbeitung.	Ja	Eher Ja	Eher Nein	Nein

12. Würden Sie gerne uns irgendetwas mitteilen?
(Kommentare, Wünsche, Anregungen, Fragen, Probleme, etc.)

Wir bedanken uns ganz herzlich!

B.5. Aufgaben Behandlungsgruppe

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Aufgabe 1

Teilnehmer:

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Änderungsanfrage 1

Uhrzeit Beginn: _____

Beschreibung der Änderungsanfrage

Im Benutzerverwaltungssystem werden bislang für jeden Benutzer (User) der Vorname (first name), der Spitzname (nickname), der Nachname (name), das Geburtsdatum (date of birth), die E-Mail-Adresse (email) und das Passwort (password) gespeichert. Um bessere statistische Auswertungen vornehmen zu können, soll ein weiteres Attribut **Postleitzahl (PLZ)** für jeden Benutzer gespeichert werden können.

Schritt 1 – Minimal Betroffene Architekturelemente sammeln:

Im ersten Schritt bitten wir Sie, die Architekturelemente zu identifizieren, die als **Einstiegspunkte** in die nachfolgende Analyse verwendet werden sollen.

Überlegen Sie, welche Architekturelemente **„auf jeden Fall"** von der Änderungsanfrage betroffen sind und notieren Sie diese in nachfolgender Tabelle. Schauen Sie nach Datentypen, Schnittstellen und Komponenten. Sie können dazu ins **PCM-Modell in Eclipse** hineinschauen oder die **textuelle Beschreibung** (wie in der Orientierungsphase) verwenden.
(Hinweis: Die Anzahl der Zeilen sagt nichts über die Anzahl der Einstiegspunkte aus.)

Einstiegspunkte: Datentypen, Schnittstellen oder Komponenten	**Kurze Begründung**

In den weiteren Schritten berechnen Sie die konkreten Auswirkungen mit Hilfe des Werkzeugs. Sie können das Ergebnis dann betrachten und selbständig auf Plausibilität prüfen. Bei Bedarf können Sie noch Verbesserungen in Schritt 1 vornehmen und die Analyse wiederholen.

- **Schritt 2: Änderungen modellieren**
- **Schritt 3: Auswirkung in der Architektur berechnen**
- **Schritt 4: Aktivitäten im Entwicklungsumfeld ableiten.**

Folgen Sie den Arbeitsanweisungen für die Werkzeug-Analyse.

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

Fragen zur Änderungsausbreitung: (im Baumeditor)

Sind die ermittelten Ausbreitungsschritte für Sie plausibel?

- ☐ Ja, sie sind alle plausibel.
- ☐ Nein, folgende(n) Schritt(e) sind nicht plausibel:
  _____
  _____
- ☐ Ich weiß nicht, ich verstehe folgende(n) Schritt(e) nicht:
  _____
  _____

Fragen zu den ermittelten Aktivitäten (in Excel-Tabelle):

Sind die ermittelten Aktivitäten für Sie plausibel?

- ☐ Ja, sie sind alle plausibel.
- ☐ Nein, folgende(n) Aktivitäten(e) sind nicht plausibel:
  _____ Grund: _____
  _____ Grund: _____
  _____ Grund: _____
- ☐ Ich weiß nicht, ich verstehe folgende(n) Aktivitäten(e) nicht:
  _____
  _____

Uhrzeit Ende: _____

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Aufgabe 2

Teilnehmer:

___

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Änderungsanfrage 2

Uhrzeit Beginn: _____

Beschreibung der Änderungsanfrage

Im Benutzerverwaltungssystem wird ein externer **Authentifizierungsdienst (Authentication)** verwendet, der über die REST-Schnittstelle **IAuthentication** angesprochen wird. Bislang bietet diese Schnittstelle eine Operation **checkUser** (mit den Parametern **nickname** und **password**) an. Die Schnittstelle wird vom Anbieter des Dienstes wie folgt angepasst: Die Operation **checkUser** wird um einen neuen Aufrufparameter „**dateOfBirth**“ (vom Typ „STRING“) erweitert.

Schritt 1 – Minimal Betroffene Architekturelemente sammeln:

Im ersten Schritt bitten wir Sie, die Architekturelemente zu identifizieren, die als **Einstiegspunkte** in die nachfolgende Analyse verwendet werden sollen.

Überlegen Sie, welche Architekturelemente **„auf jeden Fall“** von der Änderungsanfrage betroffen sind und notieren Sie diese in nachfolgender Tabelle. Schauen Sie nach Datentypen, Schnittstellen und Komponenten. Sie können dazu ins **PCM-Modell in Eclipse** hineinschauen oder die **textuelle Beschreibung** (wie in der Orientierungsphase) verwenden.
(Hinweis: Die Anzahl der Zeilen sagt nichts über die Anzahl der Einstiegspunkte aus.)

Einstiegspunkte: Datentypen, Schnittstellen oder Komponenten	**Kurze Begründung**

In den weiteren Schritten berechnen Sie die konkreten Auswirkungen mit Hilfe des Werkzeugs. Sie können das Ergebnis dann betrachten und selbständig auf Plausibilität prüfen. Bei Bedarf können Sie noch Verbesserungen in Schritt 1 vornehmen und die Analyse wiederholen.

- **Schritt 2: Änderungen modellieren**
- **Schritt 3: Auswirkung in der Architektur berechnen**
- **Schritt 4: Aktivitäten im Entwicklungsumfeld ableiten.**

Folgen Sie den Arbeitsanweisungen für die Werkzeug-Analyse.

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

Fragen zur Änderungsausbreitung: (im Baumeditor)

Sind die ermittelten Ausbreitungsschritte für Sie plausibel?

- ☐ Ja, sie sind alle plausibel.
- ☐ Nein, folgende(n) Schritt(e) sind nicht plausibel:
  _____
  _____
- ☐ Ich weiß nicht, ich verstehe folgende(n) Schritt(e) nicht:
  _____
  _____

Fragen zu den ermittelten Aktivitäten (in Excel-Tabelle):

Sind die ermittelten Aktivitäten für Sie plausibel?

- ☐ Ja, sie sind alle plausibel.
- ☐ Nein, folgende(n) Aktivitäten(e) sind nicht plausibel:
  _____ Grund: _____
  _____ Grund: _____
  _____ Grund: _____
- ☐ Ich weiß nicht, ich verstehe folgende(n) Aktivitäten(e) nicht:
  _____
  _____

Uhrzeit Ende: _____

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Aufgabe 3

Teilnehmer:

———

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Änderungsanfrage 3

Uhrzeit Beginn: _____

Beschreibung der Änderungsanfrage

Im Benutzerverwaltungssystem soll das Datenformat der REST-Schnittstelle *IUserServiceTomcat* von *JSON* auf *XML* umgewandelt werden.

Schritt 1 – Minimal Betroffene Architekturelemente sammeln:

Im ersten Schritt bitten wir Sie, die Architekturelemente zu identifizieren, die als **Einstiegspunkte** in die nachfolgende Analyse verwendet werden sollen.

Überlegen Sie, welche Architekturelemente **„auf jeden Fall"** von der Änderungsanfrage betroffen sind und notieren Sie diese in nachfolgender Tabelle. Schauen Sie nach Datentypen, Schnittstellen und Komponenten. Sie können dazu ins **PCM-Modell in Eclipse** hineinschauen oder die **textuelle Beschreibung** (wie in der Orientierungsphase) verwenden.
(Hinweis: Die Anzahl der Zeilen sagt nichts über die Anzahl der Einstiegspunkte aus.)

Einstiegspunkte: Datentypen, Schnittstellen oder Komponenten	**Kurze Begründung**

In den weiteren Schritten berechnen Sie die konkreten Auswirkungen mit Hilfe des Werkzeugs. Sie können das Ergebnis dann betrachten und selbständig auf Plausibilität prüfen. Bei Bedarf können Sie noch Verbesserungen in Schritt 1 vornehmen und die Analyse wiederholen.

- **Schritt 2: Änderungen modellieren**
- **Schritt 3: Auswirkung in der Architektur berechnen**
- **Schritt 4: Aktivitäten im Entwicklungsumfeld ableiten.**

Folgen Sie den Arbeitsanweisungen für die Werkzeug-Analyse.

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

Fragen zur Änderungsausbreitung: (im Baumeditor)

Sind die ermittelten Ausbreitungsschritte für Sie plausibel?

- ☐ Ja, sie sind alle plausibel.
- ☐ Nein, folgende(n) Schritt(e) sind nicht plausibel:
  _____
  _____
- ☐ Ich weiß nicht, ich verstehe folgende(n) Schritt(e) nicht:
  _____
  _____

Fragen zu den ermittelten Aktivitäten (in Excel-Tabelle):

Sind die ermittelten Aktivitäten für Sie plausibel?

- ☐ Ja, sie sind alle plausibel.
- ☐ Nein, folgende(n) Aktivitäten(e) sind nicht plausibel:
  _____ Grund: _____
  _____ Grund: _____
  _____ Grund: _____
- ☐ Ich weiß nicht, ich verstehe folgende(n) Aktivitäten(e) nicht:
  _____
  _____

Uhrzeit Ende: _____

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Aufgabe 4

Teilnehmer:

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Änderungsanfrage 4

Uhrzeit Beginn: _____

Beschreibung der Änderungsanfrage

Bislang verwendet das System den OR-Mapper Hibernate. Aus Lizenzgründen darf Hibernate nicht mehr verwendet werden. Daher soll die Komponente DBAccess/Hibernate aus dem System entfernt werden und die JDBC-Schnittstelle der Datenbank direkt anprogrammiert werden.

Schritt 1 – Minimal Betroffene Architekturelemente sammeln:

Im ersten Schritt bitten wir Sie, die Architekturelemente zu identifizieren, die als **Einstiegspunkte** in die nachfolgende Analyse verwendet werden sollen.

Überlegen Sie, welche Architekturelemente **„auf jeden Fall"** von der Änderungsanfrage betroffen sind und notieren Sie diese in nachfolgender Tabelle. Schauen Sie nach Datentypen, Schnittstellen und Komponenten. Sie können dazu ins **PCM-Modell in Eclipse** hineinschauen oder die **textuelle Beschreibung** (wie in der Orientierungsphase) verwenden.
(Hinweis: Die Anzahl der Zeilen sagt nichts über die Anzahl der Einstiegspunkte aus.)

Einstiegspunkte: Datentypen, Schnittstellen oder Komponenten	**Kurze Begründung**

In den weiteren Schritten berechnen Sie die konkreten Auswirkungen mit Hilfe des Werkzeugs. Sie können das Ergebnis dann betrachten und selbständig auf Plausibilität prüfen. Bei Bedarf können Sie noch Verbesserungen in Schritt 1 vornehmen und die Analyse wiederholen.

- **Schritt 2: Änderungen modellieren**
- **Schritt 3: Auswirkung in der Architektur berechnen**
- **Schritt 4: Aktivitäten im Entwicklungsumfeld ableiten.**

Folgen Sie den Arbeitsanweisungen für die Werkzeug-Analyse.

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: _____

Fragen zur Änderungsausbreitung: (im Baumeditor)

Sind die ermittelten Ausbreitungsschritte für Sie plausibel?

- ☐ Ja, sie sind alle plausibel.
- ☐ Nein, folgende(n) Schritt(e) sind nicht plausibel:
  _____
  _____
- ☐ Ich weiß nicht, ich verstehe folgende(n) Schritt(e) nicht:
  _____
  _____

Fragen zu den ermittelten Aktivitäten (in Excel-Tabelle):

Sind die ermittelten Aktivitäten für Sie plausibel?

- ☐ Ja, sie sind alle plausibel.
- ☐ Nein, folgende(n) Aktivitäten(e) sind nicht plausibel:
  _____ Grund: _____
  _____ Grund: _____
  _____ Grund: _____
- ☐ Ich weiß nicht, ich verstehe folgende(n) Aktivitäten(e) nicht:
  _____
  _____

Uhrzeit Ende: _____

B.6. Aufgaben Experten- und Kontrollgruppe

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ___

Aufgabe 1

Teilnehmer:

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Änderungsanfrage 1

Uhrzeit Beginn: _____

Beschreibung der Änderungsanfrage

Im Benutzerverwaltungssystem werden bislang für jeden Benutzer (User) der Vorname (first name), der Spitzname (nickname), der Nachname (name), das Geburtsdatum (date of birth), die E-Mail-Adresse (email) und das Passwort (password) gespeichert. Um bessere statistische Auswertungen vornehmen zu können, soll ein weiteres Attribut **Postleitzahl (PLZ)** für jeden Benutzer gespeichert werden können.

Arbeitsanweisungen

- Ermitteln Sie die Auswirkungen auf das System und bestimmen Sie die Aktivitäten, die zur Umsetzung dieser Änderungsanfrage durchzuführen sind. Füllen Sie dazu das nachfolgende Formular aus.
- Sie sollten die Aktivitätstypen unten verwenden. Verwenden Sie die Abkürzungen (QB, MB, TE, TD, TA, BK, BD, BS, IK, ID) um Zeit und Tinte zu sparen.
- Beschreiben Sie jede Aktivität in zwei Sätzen und nennen Sie die betroffenen Entwicklungsartefakte (Quelltextdateien, Metadatendateien, Testfälle, Anzahl der Laufzeitinstanzen), sowie betroffene Architekturelemente (Datentypen, Schnittstellen, Komponenten)
- Versuchen Sie dabei möglichst den kompletten Weg bis zum „laufenden" Software zu erfassen.
- Bei der Auswertung Ihrer Antworten werden wir die Reihenfolge der Aktivitäten **nicht** berücksichtigen.

Aktivitätstyp	ABKÜRZUNG	Beschreibung
Quelltextbearbeitung	**QB**	Programmierung, Überprüfung oder Anpassung von Code
Metadatenbearbeitung	**MB**	Deskriptor-Anpassungen, Konfigurationsdatei-Anpassungen
Testentwicklung Unit-Tests	**TE**	Entwicklung neuer Unit-Tests
Testaktualisierung Unit-Tests	**TA**	Aktualisierung oder Erweiterung bestehender Unit-Tests
Testdurchführung Unit-Tests	**TD**	Durchführen von Unit-Tests
Baukonfiguration	**BK**	Anpassung des Build-Skripts oder der Build-Einstellungen
Baudurchführung	**BD**	Kompilieren und Erstellen eines installierbaren Software-Produkts
Bereitstellung	**BS**	Installierbares Software-Release bereitstellen
Inbetriebnahmekonfiguration	**IK**	Änderung von Laufzeitparametern bzw. Setup-Einstellungen etc.
Inbetriebnahmedurchführung	**ID**	Starten von Laufzeitinstanzen

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Beispiel:

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1 Satz)**	**Betroffene Entwicklungs-artefakte (Quelltext-Dateien, Metadaten-Dateien, Testfälle, Releases, Anzahl Neu-Inbetriebnahme, etc.)**	**Betroffene Architektur-elemente**
QB	Prüfen und Anpassen von Code	auto.java, lenkrad.java (= 2 Dateien)	Komponente „Auto“
MB	Prüfen und Anpassen von Metadaten / Konfigurationsdateien	motor.config init.config (= 2 Dateien)	Komponente „Motor“
TE	Entwicklung von Unit-Tests	5 Unit-Tests	Schnittstellen-angebot „Auto.IAuto“
TA	Aktualisierung/Erweiterung bestehender Unit-Tests	testAutoStarten testAutoStoppen testAutoBremsen (= 3 Tests)	Schnittstellen-angebot „Auto.IAuto“
TD	Durchführen von Unit-Tests	testAutoStarten testAutoStoppen testAutoBremsen (= 3 Tests)	Schnittstellen-angebot „Auto.IAuto“
BK	Anpassung des Build-Skripts	build.ant	Komponenten „Auto“, „Motor“
BD	Build-Skript manuell starten und laufen lassen.	build.ant	Komponenten „Auto“, „Motor“
BS	Neues Release erstellen	Release „simulation.jar“	Komponenten „Auto“, „Motor“
IK	Konfiguration der Laufzeitinstanzen	10 Laufzeitinstanzen (Konfigurationsdatei: init.config)	Komponenten „Auto“, „Motor“
ID	Aktualisierung und Start von Laufzeitinstanzen	10 Laufzeitinstanzen	Komponenten „Auto“, „Motor“

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Analyseformular zu Änderungsanfrage 1

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1 Satz)**	**Betroffene Entwicklungs-artefakte (Quelltext-Dateien, Metadaten-Dateien, Testfälle, Releases, Anzahl Neu-Inbetriebnahme, etc.)**	**Betroffene Architektur-elemente**

(Fortsetzung auf nächster Seite, falls Platz nicht ausreicht...)

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

(...Fortsetzung)

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1 Satz)**	**Betroffene Entwicklungs-artefakte (Quelltext-Dateien, Metadaten-Dateien, Testfälle, Releases, Anzahl Neu-Inbetriebnahme, etc.)**	**Betroffene Architektur-elemente**

Uhrzeit Ende: _____

.
.

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Aufgabe 2

Teilnehmer:

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Änderungsanfrage 2

Uhrzeit Beginn: _____

Beschreibung der Änderungsanfrage

Im Benutzerverwaltungssystem wird ein externer **Authentifizierungsdienst (Authentication)** verwendet, der über die REST-Schnittstelle **IAuthentication** angesprochen wird. Bislang bietet diese Schnittstelle eine Operation **checkUser** (mit den Parametern **nickname** und **password**) an. Die Schnittstelle wird vom Anbieter des Dienstes wie folgt angepasst: Die Operation **checkUser** wird um einen neuen Aufrufparameter „**dateOfBirth**“ (vom Typ „STRING“) erweitert.

Arbeitsanweisungen

- Ermitteln Sie die Auswirkungen auf das System und bestimmen Sie die Aktivitäten, die zur Umsetzung dieser Änderungsanfrage durchzuführen sind. Füllen Sie dazu das nachfolgende Formular aus.
- Sie sollten die Aktivitätstypen unten verwenden. Verwenden Sie die Abkürzungen (QB, MB, TE, TD, TA, BK, BD, BS, IK, ID) um Zeit und Tinte zu sparen.
- Beschreiben Sie jede Aktivität in zwei Sätzen und nennen Sie die betroffenen Entwicklungsartefakte (Quelltextdateien, Metadatendateien, Testfälle, Anzahl der Laufzeitinstanzen), sowie betroffene Architekturelemente (Datentypen, Schnittstellen, Komponenten)
- Versuchen Sie dabei möglichst den kompletten Weg bis zum „laufenden“ Software zu erfassen.
- Bei der Auswertung Ihrer Antworten werden wir die Reihenfolge der Aktivitäten **nicht** berücksichtigen.

Aktivitätstyp	ABKÜRZUNG	Beschreibung
Quelltextbearbeitung	**QB**	Programmierung, Überprüfung oder Anpassung von Code
Metadatenbearbeitung	**MB**	Deskriptor-Anpassungen, Konfigurationsdatei-Anpassungen
Testentwicklung Unit-Tests	**TE**	Entwicklung neuer Unit-Tests
Testaktualisierung Unit-Tests	**TA**	Aktualisierung oder Erweiterung bestehender Unit-Tests
Testdurchführung Unit-Tests	**TD**	Durchführen von Unit-Tests
Baukonfiguration	**BK**	Anpassung des Build-Skripts oder der Build-Einstellungen
Baudurchführung	**BD**	Kompilieren und Erstellen eines installierbaren Software-Produkts
Bereitstellung	**BS**	Installierbares Software-Release bereitstellen
Inbetriebnahmekonfiguration	**IK**	Änderung von Laufzeitparametern bzw. Setup-Einstellungen etc.
Inbetriebnahmedurchführung	**ID**	Starten von Laufzeitinstanzen

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Beispiel:

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1 Satz)**	**Betroffene Entwicklungs-artefakte (Quelltext-Dateien, Metadaten-Dateien, Testfälle, Releases, Anzahl Neu-Inbetriebnahme, etc.)**	**Betroffene Architektur-elemente**
QB	Prüfen und Anpassen von Code	auto.java, lenkrad.java (= 2 Dateien)	Komponente „Auto“
MB	Prüfen und Anpassen von Metadaten / Konfigurationsdateien	motor.config init.config (= 2 Dateien)	Komponente „Motor“
TE	Entwicklung von Unit-Tests	5 Unit-Tests	Schnittstellen-angebot „Auto.IAuto“
TA	Aktualisierung/Erweiterung bestehender Unit-Tests	testAutoStarten testAutoStoppen testAutoBremsen (= 3 Tests)	Schnittstellen-angebot „Auto.IAuto“
TD	Durchführen von Unit-Tests	testAutoStarten testAutoStoppen testAutoBremsen (= 3 Tests)	Schnittstellen-angebot „Auto.IAuto“
BK	Anpassung des Build-Skripts	build.ant	Komponenten „Auto“, „Motor“
BD	Build-Skript manuell starten und laufen lassen.	build.ant	Komponenten „Auto“, „Motor“
BS	Neues Release erstellen	Release „simulation.jar“	Komponenten „Auto“, „Motor“
IK	Konfiguration der Laufzeitinstanzen	10 Laufzeitinstanzen (Konfigurationsdatei: init.config)	Komponenten „Auto“, „Motor“
ID	Aktualisierung und Start von Laufzeitinstanzen	10 Laufzeitinstanzen	Komponenten „Auto“, „Motor“

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Analyseformular zu Änderungsanfrage 2

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	Beschreibung der Aktivität (1-2 Sätze)	Betroffene Entwicklungs-artefakte (Dateien, Testfälle, etc.)	Betroffene Architektur-elemente

(Fortsetzung auf nächster Seite, falls Platz nicht ausreicht...)

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

(...Fortsetzung)

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1-2 Sätze)**	**Betroffene Entwicklungs-artefakte (Dateien, Testfälle, etc.)**	**Betroffene Architektur-elemente**

Uhrzeit Ende: _____

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ___

Aufgabe 3

Teilnehmer:

___

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Änderungsanfrage 3

Uhrzeit Beginn: _____

Beschreibung der Änderungsanfrage

Im Benutzerverwaltungssystem soll das Datenformat der REST-Schnittstelle *IUserServiceTomcat* von *JSON* in *XML* umgewandelt werden.

Arbeitsanweisungen

- Ermitteln Sie die Auswirkungen auf das System und bestimmen Sie die Aktivitäten, die zur Umsetzung dieser Änderungsanfrage durchzuführen sind. Füllen Sie dazu das nachfolgende Formular aus.
- Sie sollten die Aktivitätstypen unten verwenden. Verwenden Sie die Abkürzungen (QB, MB, TE, TD, TA, BK, BD, BS, IK, ID) um Zeit und Tinte zu sparen.
- Beschreiben Sie jede Aktivität in zwei Sätzen und nennen Sie die betroffenen Entwicklungsartefakte (Quelltextdateien, Metadatendateien, Testfälle, Anzahl der Laufzeitinstanzen), sowie betroffene Architekturelemente (Datentypen, Schnittstellen, Komponenten)
- Versuchen Sie dabei möglichst den kompletten Weg bis zum „laufenden" Software zu erfassen.
- Bei der Auswertung Ihrer Antworten werden wir die Reihenfolge der Aktivitäten **nicht** berücksichtigen.

Aktivitätstyp	ABKÜRZUNG	Beschreibung
Quelltextbearbeitung	**QB**	Programmierung, Überprüfung oder Anpassung von Code
Metadatenbearbeitung	**MB**	Deskriptor-Anpassungen, Konfigurationsdatei-Anpassungen
Testentwicklung Unit-Tests	**TE**	Entwicklung neuer Unit-Tests
Testaktualisierung Unit-Tests	**TA**	Aktualisierung oder Erweiterung bestehender Unit-Tests
Testdurchführung Unit-Tests	**TD**	Durchführen von Unit-Tests
Baukonfiguration	**BK**	Anpassung des Build-Skripts oder der Build-Einstellungen
Baudurchführung	**BD**	Kompilieren und Erstellen eines installierbaren Software-Produkts
Bereitstellung	**BS**	Installierbares Software-Release bereitstellen
Inbetriebnahmekonfiguration	**IK**	Änderung von Laufzeitparametern bzw. Setup-Einstellungen etc.
Inbetriebnahmedurchführung	**ID**	Starten von Laufzeitinstanzen

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Beispiel:

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1 Satz)**	**Betroffene Entwicklungs-artefakte (Quelltext-Dateien, Metadaten-Dateien, Testfälle, Releases, Anzahl Neu-Inbetriebnahme, etc.)**	**Betroffene Architektur-elemente**
QB	Prüfen und Anpassen von Code	auto.java, lenkrad.java (= 2 Dateien)	Komponente „Auto“
MB	Prüfen und Anpassen von Metadaten / Konfigurationsdateien	motor.config init.config (= 2 Dateien)	Komponente „Motor“
TE	Entwicklung von Unit-Tests	5 Unit-Tests	Schnittstellen-angebot „Auto.IAuto“
TA	Aktualisierung/Erweiterung bestehender Unit-Tests	testAutoStarten testAutoStoppen testAutoBremsen (= 3 Tests)	Schnittstellen-angebot „Auto.IAuto“
TD	Durchführen von Unit-Tests	testAutoStarten testAutoStoppen testAutoBremsen (= 3 Tests)	Schnittstellen-angebot „Auto.IAuto“
BK	Anpassung des Build-Skripts	build.ant	Komponenten „Auto“, „Motor“
BD	Build-Skript manuell starten und laufen lassen.	build.ant	Komponenten „Auto“, „Motor“
BS	Neues Release erstellen	Release „simulation.jar“	Komponenten „Auto“, „Motor“
IK	Konfiguration der Laufzeitinstanzen	10 Laufzeitinstanzen (Konfigurationsdatei: init.config)	Komponenten „Auto“, „Motor“
ID	Aktualisierung und Start von Laufzeitinstanzen	10 Laufzeitinstanzen	Komponenten „Auto“, „Motor“

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Analyseformular zu Änderungsanfrage 3

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1-2 Sätze)**	**Betroffene Entwicklungs-artefakte (Dateien, Testfälle, etc.)**	**Betroffene Architektur-elemente**

(Fortsetzung auf nächster Seite, falls Platz nicht ausreicht...)

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

(...Fortsetzung)

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1-2 Sätze)**	**Betroffene Entwicklungs-artefakte (Dateien, Testfälle, etc.)**	**Betroffene Architektur-elemente**

Uhrzeit Ende: _____

Studie: Auswirkungen von
Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Aufgabe 4

Teilnehmer:

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Änderungsanfrage 4

Uhrzeit Beginn: _____

Beschreibung der Änderungsanfrage

Bislang verwendet das System den OR-Mapper Hibernate. Aus Lizenzgründen darf Hibernate nicht mehr verwendet werden. Daher soll die Komponente DBAccess/Hibernate aus dem System entfernt werden und die JDBC-Schnittstelle der Datenbank direkt anprogrammiert werden.

Arbeitsanweisungen

- Ermitteln Sie die Auswirkungen auf das System und bestimmen Sie die Aktivitäten, die zur Umsetzung dieser Änderungsanfrage durchzuführen sind. Füllen Sie dazu das nachfolgende Formular aus.
- Sie sollten die Aktivitätstypen unten verwenden. Verwenden Sie die Abkürzungen (QB, MB, TE, TD, TA, BK, BD, BS, IK, ID) um Zeit und Tinte zu sparen.
- Beschreiben Sie jede Aktivität in zwei Sätzen und nennen Sie die betroffenen Entwicklungsartefakte (Quelltextdateien, Metadatendateien, Testfälle, Anzahl der Laufzeitinstanzen), sowie betroffene Architekturelemente (Datentypen, Schnittstellen, Komponenten)
- Versuchen Sie dabei möglichst den kompletten Weg bis zum „laufenden" Software zu erfassen.
- Bei der Auswertung Ihrer Antworten werden wir die Reihenfolge der Aktivitäten **nicht** berücksichtigen.

Aktivitätstyp	ABKÜRZUNG	Beschreibung
Quelltextbearbeitung	**QB**	Programmierung, Überprüfung oder Anpassung von Code
Metadatenbearbeitung	**MB**	Deskriptor-Anpassungen, Konfigurationsdatei-Anpassungen
Testentwicklung Unit-Tests	**TE**	Entwicklung neuer Unit-Tests
Testaktualisierung Unit-Tests	**TA**	Aktualisierung oder Erweiterung bestehender Unit-Tests
Testdurchführung Unit-Tests	**TD**	Durchführen von Unit-Tests
Baukonfiguration	**BK**	Anpassung des Build-Skripts oder der Build-Einstellungen
Baudurchführung	**BD**	Kompilieren und Erstellen eines installierbaren Software-Produkts
Bereitstellung	**BS**	Installierbares Software-Release bereitstellen
Inbetriebnahmekonfiguration	**IK**	Änderung von Laufzeitparametern bzw. Setup-Einstellungen etc.
Inbetriebnahmedurchführung	**ID**	Starten von Laufzeitinstanzen

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Beispiel:

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1 Satz)**	**Betroffene Entwicklungs-artefakte (Quelltext-Dateien, Metadaten-Dateien, Testfälle, Releases, Anzahl Neu-Inbetriebnahme, etc.)**	**Betroffene Architektur-elemente**
QB	Prüfen und Anpassen von Code	auto.java, lenkrad.java (= 2 Dateien)	Komponente „Auto“
MB	Prüfen und Anpassen von Metadaten / Konfigurationsdateien	motor.config init.config (= 2 Dateien)	Komponente „Motor“
TE	Entwicklung von Unit-Tests	5 Unit-Tests	Schnittstellen-angebot „Auto.IAuto“
TA	Aktualisierung/Erweiterung bestehender Unit-Tests	testAutoStarten testAutoStoppen testAutoBremsen (= 3 Tests)	Schnittstellen-angebot „Auto.IAuto“
TD	Durchführen von Unit-Tests	testAutoStarten testAutoStoppen testAutoBremsen (= 3 Tests)	Schnittstellen-angebot „Auto.IAuto“
BK	Anpassung des Build-Skripts	build.ant	Komponenten „Auto“, „Motor“
BD	Build-Skript manuell starten und laufen lassen.	build.ant	Komponenten „Auto“, „Motor“
BS	Neues Release erstellen	Release „simulation.jar“	Komponenten „Auto“, „Motor“
IK	Konfiguration der Laufzeitinstanzen	10 Laufzeitinstanzen (Konfigurationsdatei: init.config)	Komponenten „Auto“, „Motor“
ID	Aktualisierung und Start von Laufzeitinstanzen	10 Laufzeitinstanzen	Komponenten „Auto“, „Motor“

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

Analyseformular zu Änderungsanfrage 4

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	Beschreibung der Aktivität (1-2 Sätze)	Betroffene Entwicklungs-artefakte (Dateien, Testfälle, etc.)	Betroffene Architektur-elemente

(Fortsetzung auf nächster Seite, falls Platz nicht ausreicht...)

Studie: Auswirkungen von Änderungsanfragen in Software-Systemen
Autor: Johannes Stammel

Teilnehmer-ID: ____

(...Fortsetzung)

Aktivitätstyp (abgekürzt) (QB, MB, TE, TA, TD, BK, BD, BS, IK, ID)	**Beschreibung der Aktivität (1-2 Sätze)**	**Betroffene Entwicklungs-artefakte (Dateien, Testfälle, etc.)**	**Betroffene Architektur-elemente**

Uhrzeit Ende: _____

C. Experimentergebnisse

C.1. Metrikwerte zu Aktivitätstypen

Aktivitätstypen																													
	Aufgabe 1						Aufgabe 2						Aufgabe 3						Aufgabe 4						Ø aller Aufgaben		
	t_p	f_n	f_p	G	T	F1	t_p	f_n	f_p	G	T	F1	t_p	f_n	f_p	G	T	F1	t_p	f_n	f_p	G	T	F1	G	T	F1
Exp1	8,000	0,000	0,000	1,000	1,000	**1,000**	5,000	1,000	1,000	0,833	0,833	**0,833**	6,000	0,000	0,000	1,000	1,000	**1,000**	8,000	0,000	1,000	0,889	1,000	**0,941**	0,931	0,958	0,944
Exp2	7,000	1,000	0,000	1,000	0,875	**0,933**	6,000	0,000	1,000	0,857	1,000	**0,923**	6,000	0,000	1,000	0,857	1,000	**0,923**	7,000	1,000	0,000	1,000	0,875	**0,933**	0,929	0,938	0,928
Exp3	8,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	2,000	0,000	1,000	0,750	**0,857**	1,000	0,938	0,964
Exp4	8,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	5,000	3,000	0,000	1,000	0,625	**0,769**	1,000	0,906	0,942
Exp5	6,000	2,000	0,000	1,000	0,750	**0,857**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	1,000	0,857	1,000	**0,923**	7,000	1,000	0,000	1,000	0,875	**0,933**	0,964	0,906	0,928
Ø Exp	**7,400**	**0,600**	**0,000**	**1,000**	**0,925**	**0,958**	**5,800**	**0,200**	**0,400**	**0,938**	**0,967**	**0,951**	**6,000**	**0,000**	**0,400**	**0,943**	**1,000**	**0,969**	**6,600**	**1,400**	**0,200**	**0,978**	**0,825**	**0,887**	**0,965**	**0,929**	**0,941**
KG1	7,000	1,000	0,000	1,000	0,875	**0,933**	6,000	0,000	0,000	1,000	1,000	**1,000**	3,000	3,000	1,000	0,750	0,500	**0,600**	5,000	3,000	0,000	1,000	0,625	**0,769**	0,938	0,750	0,826
KG2	8,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	1,000	0,857	1,000	**0,923**	6,000	0,000	1,000	0,857	1,000	**0,923**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,893	0,938	0,912
KG3	6,000	2,000	0,000	1,000	0,750	**0,857**	4,000	2,000	0,000	1,000	0,667	**0,800**	4,000	2,000	0,000	1,000	0,667	**0,800**	3,000	5,000	1,000	0,750	0,375	**0,500**	0,938	0,615	0,739
KG4	7,000	1,000	0,000	1,000	0,875	**0,933**	5,000	1,000	1,000	0,833	0,833	**0,833**	5,000	1,000	1,000	0,833	0,833	**0,833**	7,000	1,000	0,000	1,000	0,875	**0,933**	0,917	0,854	0,883
KG5	7,000	1,000	0,000	1,000	0,875	**0,933**	5,000	1,000	0,000	1,000	0,833	**0,909**	5,000	1,000	0,000	1,000	0,833	**0,909**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,964	0,823	0,888
KG6	4,000	4,000	0,000	1,000	0,500	**0,667**	5,000	1,000	1,000	0,833	0,833	**0,833**	5,000	1,000	1,000	0,833	0,833	**0,833**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,881	0,729	0,783
KG7	7,000	1,000	0,000	1,000	0,875	**0,933**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	1,000	0,857	1,000	**0,923**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,929	0,906	0,914
KG8	8,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	2,000	0,750	1,000	**0,857**	5,000	1,000	2,000	0,714	0,833	**0,769**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,830	0,896	0,857
Ø KG	**6,750**	**1,250**	**0,000**	**1,000**	**0,844**	**0,907**	**5,375**	**0,625**	**0,625**	**0,909**	**0,896**	**0,894**	**4,875**	**1,125**	**0,875**	**0,856**	**0,813**	**0,824**	**5,625**	**2,375**	**0,750**	**0,879**	**0,703**	**0,775**	**0,911**	**0,814**	**0,850**
BG1	6,000	2,000	0,000	1,000	0,750	**0,857**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	nicht bearbeitet						1,000	0,917	0,952
BG2	6,000	2,000	0,000	1,000	0,750	**0,857**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,964	0,875	0,914
BG3	6,000	2,000	0,000	1,000	0,750	**0,857**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,964	0,875	0,914
BG4	6,000	2,000	0,000	1,000	0,750	**0,857**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,964	0,875	0,914
BG5	6,000	2,000	0,000	1,000	0,750	**0,857**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,964	0,875	0,914
BG6	6,000	2,000	0,000	1,000	0,750	**0,857**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	0,000	0,000	1,000	1,000	**1,000**	6,000	2,000	1,000	0,857	0,750	**0,800**	0,964	0,875	0,914
Ø BG	**6,000**	**2,000**	**0,000**	**1,000**	**0,750**	**0,857**	**6,000**	**0,000**	**0,000**	**1,000**	**1,000**	**1,000**	**6,000**	**0,000**	**0,000**	**1,000**	**1,000**	**1,000**	**6,000**	**2,000**	**1,000**	**0,857**	**0,750**	**0,800**	**0,964**	**0,875**	**0,914**

Legende:	
t_p	Treffer
f_n	falsch negativ
f_p	falsch positiv
G	Genauigkeit
T	Trefferquote
Fl	F1-Maß aus Genauigkeit und Trefferquote

C.2. Metrikwerte zu angereicherten Aktivitäten

Angereicherte Aktivitäten																														
	Aufgabe 1						Aufgabe 2						Aufgabe 3						Aufgabe 4						Alle Aufgaben		
	t_p	f_n	f_p	G	T	F1	t_p	f_n	f_p	G	T	F1	t_p	f_n	f_p	G	T	F1	t_p	f_n	f_p	G	T	F1	G	T	F1
Exp1	26,000	4,000	6,000	0,813	0,867	**0,839**	4,000	10,000	0,000	1,000	0,286	**0,444**	16,000	2,000	0,000	1,000	0,889	**0,941**	2,000	20,000	0,000	1,000	0,091	**0,167**	**0,953**	**0,533**	**0,598**
Exp2	23,000	7,000	3,000	0,885	0,767	**0,821**	9,000	5,000	7,000	0,563	0,643	**0,600**	16,000	2,000	6,000	0,727	0,889	**0,800**	4,000	18,000	0,000	1,000	0,182	**0,308**	**0,794**	**0,620**	**0,632**
Exp3	19,000	11,000	1,000	0,950	0,633	**0,760**	6,000	8,000	0,000	1,000	0,429	**0,600**	15,000	3,000	0,000	1,000	0,833	**0,909**	14,000	8,000	0,000	1,000	0,636	**0,778**	**0,988**	**0,633**	**0,762**
Exp4	17,000	13,000	0,000	1,000	0,567	**0,723**	4,000	10,000	0,000	1,000	0,286	**0,444**	16,000	2,000	0,000	1,000	0,889	**0,941**	2,000	20,000	0,000	1,000	0,091	**0,167**	**1,000**	**0,458**	**0,569**
Exp5	11,000	19,000	0,000	1,000	0,367	**0,537**	3,000	11,000	0,000	1,000	0,214	**0,353**	2,000	16,000	0,000	1,000	0,111	**0,200**	10,000	12,000	0,000	1,000	0,455	**0,625**	**1,000**	**0,287**	**0,429**
Ø Exp	**19,200**	**10,800**	**2,000**	**0,929**	**0,640**	**0,736**	**5,200**	**8,800**	**1,400**	**0,913**	**0,371**	**0,488**	**13,000**	**5,000**	**1,200**	**0,945**	**0,722**	**0,758**	**6,400**	**15,600**	**0,000**	**1,000**	**0,291**	**0,409**	**0,947**	**0,506**	**0,598**
KG1	19,000	11,000	2,000	0,905	0,633	**0,745**	1,000	13,000	0,000	1,000	0,071	**0,133**	0,000	18,000	0,000	0,000	0,000	**0,000**	4,000	18,000	0,000	1,000	0,182	**0,308**	**0,726**	**0,222**	**0,297**
KG2	13,000	17,000	1,000	0,929	0,433	**0,591**	3,000	11,000	0,000	1,000	0,214	**0,353**	1,000	17,000	0,000	1,000	0,056	**0,105**	2,000	20,000	0,000	1,000	0,091	**0,167**	**0,982**	**0,199**	**0,304**
KG3	11,000	19,000	2,000	0,846	0,367	**0,512**	1,000	13,000	0,000	1,000	0,071	**0,133**	0,000	18,000	1,000	0,000	0,000	**0,000**	1,000	21,000	0,000	1,000	0,045	**0,087**	**0,712**	**0,121**	**0,183**
KG4	8,000	22,000	0,000	1,000	0,267	**0,421**	2,000	12,000	0,000	1,000	0,143	**0,250**	7,000	11,000	0,000	1,000	0,389	**0,560**	4,000	18,000	0,000	1,000	0,182	**0,308**	**1,000**	**0,245**	**0,385**
KG5	18,000	12,000	1,000	0,947	0,600	**0,735**	7,000	7,000	0,000	1,000	0,500	**0,667**	17,000	1,000	0,000	1,000	0,944	**0,971**	11,000	11,000	5,000	0,688	0,500	**0,579**	**0,909**	**0,636**	**0,738**
KG6	18,000	12,000	1,000	0,947	0,600	**0,735**	6,000	8,000	2,000	0,750	0,429	**0,545**	4,000	14,000	5,000	0,444	0,222	**0,296**	8,000	14,000	1,000	0,889	0,364	**0,516**	**0,758**	**0,404**	**0,523**
KG7	12,000	18,000	0,000	1,000	0,400	**0,571**	5,000	9,000	1,000	0,833	0,357	**0,500**	11,000	7,000	8,000	0,579	0,611	**0,595**	3,000	19,000	0,000	1,000	0,136	**0,240**	**0,853**	**0,376**	**0,477**
KG8	15,000	15,000	1,000	0,938	0,500	**0,652**	5,000	9,000	0,000	1,000	0,357	**0,526**	3,000	15,000	0,000	1,000	0,167	**0,286**	3,000	19,000	0,000	1,000	0,136	**0,240**	**0,984**	**0,290**	**0,426**
Ø KG	**14,250**	**15,750**	**1,000**	**0,939**	**0,475**	**0,620**	**3,750**	**10,250**	**0,375**	**0,948**	**0,268**	**0,389**	**5,375**	**12,625**	**1,750**	**0,628**	**0,299**	**0,352**	**4,500**	**17,500**	**0,750**	**0,947**	**0,205**	**0,306**	**0,865**	**0,312**	**0,416**
BG1	26,000	4,000	5,000	0,839	0,867	**0,852**	14,000	0,000	6,000	0,700	1,000	**0,824**	18,000	0,000	0,000	1,000	1,000	**1,000**	nicht bearbeitet						**0,846**	**0,956**	**0,892**
BG2	26,000	4,000	5,000	0,839	0,867	**0,852**	14,000	0,000	6,000	0,700	1,000	**0,824**	18,000	0,000	0,000	1,000	1,000	**1,000**	20,000	2,000	14,000	0,588	0,909	**0,714**	**0,782**	**0,944**	**0,848**
BG3	26,000	4,000	5,000	0,839	0,867	**0,852**	14,000	0,000	6,000	0,700	1,000	**0,824**	18,000	0,000	0,000	1,000	1,000	**1,000**	20,000	2,000	14,000	0,588	0,909	**0,714**	**0,782**	**0,944**	**0,848**
BG4	26,000	4,000	5,000	0,839	0,867	**0,852**	14,000	0,000	6,000	0,700	1,000	**0,824**	18,000	0,000	0,000	1,000	1,000	**1,000**	20,000	2,000	15,000	0,571	0,909	**0,702**	**0,778**	**0,944**	**0,844**
BG5	26,000	4,000	5,000	0,839	0,867	**0,852**	14,000	0,000	6,000	0,700	1,000	**0,824**	18,000	0,000	0,000	1,000	1,000	**1,000**	20,000	2,000	14,000	0,588	0,909	**0,714**	**0,782**	**0,944**	**0,848**
BG6	26,000	4,000	5,000	0,839	0,867	**0,852**	14,000	0,000	17,000	0,452	1,000	**0,622**	18,000	0,000	0,000	1,000	1,000	**1,000**	20,000	2,000	14,000	0,588	0,909	**0,714**	**0,720**	**0,944**	**0,797**
Ø BG	**26,000**	**4,000**	**5,000**	**0,839**	**0,867**	**0,852**	**14,000**	**0,000**	**7,833**	**0,659**	**1,000**	**0,790**	**18,000**	**0,000**	**0,000**	**1,000**	**1,000**	**1,000**	**20,000**	**2,000**	**14,200**	**0,585**	**0,909**	**0,712**	**0,771**	**0,944**	**0,839**

Legende:	
t_p	Treffer
f_n	falsch negativ
f_p	falsch positiv
G	Genauigkeit
T	Trefferquote
Fl	F1-Maß aus Genauigkeit und Trefferquote

C.3. Ergebnisse Orientierungsfragebogen

	Frage	Erwartete Antwort	EXP1	EXP2	EXP3	EXP4	EXP5	KG1	KG2	KG3	KG4	KG5	KG6	KG7	KG8	BG1	BG2	BG3	BG4	BG5	BG6
1.1	Anzahl Datentypen in textueller Beschreibung	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1.2	Anzahl Datentypen im PCM-Modell	17	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0
2.1	Attribute von Datentyp	voteFor	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2.2	"Vote"	value	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3.1	Anzahl Schnittstellen in textueller Beschreibung	8	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
3.2	Anzahl Schnittstellen im PCM-Modell	15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4.1	Operationen von Schnittstelle "IReporting"	getReport	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4.2		log	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5.1	Anzahl Komponenten in textueller Beschreibung	8	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
5.2	Anzahl Komponenten in PCM-Modell	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
6.1	Schnittstellenangebote von Komponente "DBAccess"	ISession	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6.2		ISessionFactory	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6.3	Schnittstellenanfrage von Komponente "DBAccess"	JDBCDriver	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7.1	Für welche	UserServiceTomcat	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7.2	Komponenten gibt es	Reporting	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7.3	Quelltext?	UserManagement	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
7.4		UserDAO	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8.1	Welche Komponenten werden als "Externer Dienst oder Drittanbieter-Komponente" im System verwendet?	Authentication	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8.2	Welche Komponenten werden als "Externer	UserServiceApacheProxy	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	0	1	1
8.3	Dienst oder Drittanbieter-Komponente" im	DBAccess	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8.4	System verwendet?	UserDatabase	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
9.1	Was macht die Komponente "DBAccess" (Hibernate)"?	Die Komponente mappt den Datentyp "User" auf die Datenbank. [simgemail]	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	Frage	Erwartete Antwort	EXP1	EXP2	EXP3	EXP4	EXP5	KG1	KG2	KG3	KG4	KG5	KG6	KG7	KG8	BG1	BG2	BG3	BG4	BG5	BG6
10.1	Wie heißen die zwei Konfigurationsdateien zu DBAccess (Hibernate) und was wird darin konfiguriert?	hibernate.cfg.xml	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1
10.2	Wie heißen die zwei Konfigurationsdateien zu DBAccess (Hibernate) und was wird darin konfiguriert?	enthält allg. Einstellungen des OR Mappers	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1
10.3	Wie heißen die zwei Konfigurationsdateien zu DBAccess (Hibernate) und was wird darin konfiguriert?	User.hbm.xml	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	1	1
10.4	Wie heißen die zwei Konfigurationsdateien zu DBAccess (Hibernate) und was wird darin konfiguriert?	Beschreibt Abb. des Users auf DBSchema	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1
Anzahl richtiger Antworten			25	26	26	26	23	25	23	26	26	26	26	25	22	26	26	26	18	26	25
Anteil richtiger Antworten (Prozent)			96,2%	100,0%	100,0%	100,0%	88,5%	96,2%	88,5%	100,0%	100,0%	100,0%	100,0%	96,2%	84,6%	100,0%	100,0%	100,0%	69,2%	100,0%	96,2%
Durchschnitt pro Gruppe			25,20					24,50													
Anteil pro Gruppe			96,92%					94,23%													

Literatur

[Bas+07] Matthew Bass, Vesna Mikulovic, Len Bass, James D. Herbsleb und Marcelo Cataldo. “Architectural Misalignment: An Experience Report.” In: *WICSA*. IEEE Computer Society, 2007, S. 17.

[BB99] Per-Olof Bengtsson und Jan Bosch. “Architecture Level Prediction of Software Maintenance.” In: *CSMR*. IEEE Computer Society, 1999, S. 139–147.

[BC00] Carliss Baldwin und Kim Clark. *Design Rules vol. 1: The Power of Modularity*. Cambridge, MA: MIT Press, 2000, S. 471.

[BCR94] Victor R. Basili, Gianluigi Caldiera und Dieter H. Rombach. “The Goal Question Metric Approach”. In: Bd. I. John Wiley & Sons, 1994.

[Ben+04] Per-Olof Bengtsson, Nico Lassing, Jan Bosch und Hans van Vliet. “Architecture-level modifiability analysis (ALMA)”. In: *Journal of Systems and Software* 69.1-2 (2004).

[BKR09] Steffen Becker, Heiko Koziolek und Ralf Reussner. “The Palladio component model for model-driven performance prediction”. In: *Journal of Systems and Software* 82.1 (2009). Special Issue: Software Performance – Modeling and Analysis, S. 3–22.

[Bla01] Sue Black. “Computing ripple effect for software maintenance.” In: *Journal of Software Maintenance* 13.4 (2001), S. 263–279.

[Boe+00] Barry W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, Ray Madachy und Bert Steece. *Software Cost Estimation with Cocomo II*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[Bro+01] Bronstein, Semendjajew, Musiol und Mühlig. *Taschenbuch der Mathematik*. 5. Aufl. Harri Deutsch, 2001.

[BZJ04] Muhammad Ali Babar, Liming Zhu und D. Ross Jeffery. “A Framework for Classifying and Comparing Software Architecture Evaluation Methods.” In: *Australian Software Engineering Conference*. IEEE Computer Society, 2004, S. 309–319.

[Car+10] Jeffrey C. Carver, Maria Letizia Jaccheri, Sandro Morasca und Forrest Shull. “A checklist for integrating student empirical studies with research and teaching goals.” In: *Empirical Software Engineering* 15.1 (2010), S. 35–59.

[Car12] Ralf Carbon. “Architecture-centric software producibility analysis”. Diss. 2012.

[CB10] Lyra Colfer und Carliss Y. Baldwin. “The Mirroring Hypothesis: Theory, Evidence and Exceptions”. In: *working paper* (Feb. 2010). Hrsg. von Harvard Business School.

[CD01] John Cheesman und John Daniels. *UML Components - A Simple Process for Specifying Component-Based Software*. Hrsg. von Clemens Szyperski. Component Software Series. Addison-Wesley, 2001.

[CKK05] Paul Clements, Rick Kazman und Mark Klein. *Evaluating software architectures*. Boston [u.a.]: Addison-Wesley, 2005.

[Con68] M. Conway. “How do Committees Invent?” In: *Datamation Journal* (Apr. 1968), S. 28–31.

[Cru15] Cruise Control Open Source Project. *Cruise Control*. `http://cruisecontrol.sourceforge.net/`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[DN02] Liliana Dobrica und Eila Niemelä. “A Survey on Software Architecture Analysis Methods.” In: *IEEE Trans. Software Eng.* 28.7 (2002), S. 638–653.

[DR13] Zoya Durdik und Ralf Reussner. “On the Appropriate Rationale for Using Design Patterns and Pattern Documentation”. In: *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*. QoSA ’13. Vancouver, British Columbia, Canada: ACM, 2013, S. 107–116.

[Gar+09] David Garlan, Jeffrey M. Barnes, Bradley R. Schmerl und Orieta Celiku. “Evolution styles: Foundations and tool support for software architecture evolution.” In: *WICSA/ECSA*. IEEE, 2009, S. 131–140.

[Gla06] R. L. Glass. “The Standish Report: Does It Really Describe a Software Crisis?” In: *Communications of the ACM* 49.8 (Aug. 2006), S. 15–16.

[Gra15] Gradleware Inc. *Gradle*. `https://gradle.org/`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[HC01] George T. Heineman und William T. Councill, Hrsg. *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, Juni 2001.

[HG99] J.D. Herbsleb und R.E. Grinter. “Architectures, coordination, and distance: Conway’s law and beyond ”. In: *Software, IEEE* 16.5 (Sep. 1999), S. 63–70.

[HNS00] C. Hofmeister, R. Nord und D. Soni. *Applied Software Architecture*. The Addison-Wesley Object Technology Series. Addison-Wesley, 2000.

[ISO11] ISO/IEC/IEEE. “Systems and software engineering – Architecture description”. In: *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)* (Jan. 2011), S. 1–46.

[Jen15] Jenkins CI community. *Jenkins CI*. `http://jenkins-ci.org/`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[Jet15] JetBrains. *TeamCity*. `https://www.jetbrains.com/teamcity/`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[Jør04] Magne Jørgensen. “Top-down and bottom-up expert estimation of software development effort”. In: *Information and Software Technology* 46.1 (2004), S. 3–16.

[JUn15] JUnit. *JUnit*. `http://junit.org/`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[KCD12] Irwin Kwan, Marcelo Cataldo und Daniela Damian. “Conway’s Law Revisited: The Evidence for a Task-Based Perspective.” In: *IEEE Software* 29.1 (2012), S. 90–93.

[KH06] Heiko Koziolek und Jens Happe. “A QoS Driven Development Process Model for Component-Based Software Systems.” In: *CBSE*. Hrsg. von Ian Gorton, George T. Heineman, Ivica Crnkovic, Heinz W. Schmidt, Judith A. Stafford, Clemens A. Szyperski und Kurt C. Wallnau. Bd. 4063. Lecture Notes in Computer Science. Springer, 2006, S. 336–343.

[KRH08] M. Kütz, R. Reussner und W. Hasselbring. *Handbuch der Software-Architektur: Werkzeuge für Controlling und Management*. Dpunkt-Verlag, 2008.

[Kro10] Klaus Krogmann. “Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis”. Diss. Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 2010.

[Küs13] Martin Küster. “Architecture-Centric Modeling of Design Decisions for Validation and Traceability”. In: *Proceedings of the 7th European Conference on Software Architecture (ECSA ’13)*. Hrsg. von Khalil Drira. Bd. 7957. Lecture Notes in Computer Science. Montpellier, France: Springer Berlin Heidelberg, 2013, S. 184–191.

[KW07] Ekkart Kindler und Robert Wagner. *Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios*. Techn. Ber. University of Paderborn, Department of Computer Science, 2007.

[Lie09] Daniel Liebhart. “Das Märchen von den gescheiterten IT-Projekten”. In: *netzwoche* (Juni 2009). Hrsg. von netzmedien ag.

[LMS05] P. Leach, M. Mealling und R. Salz. “RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace”. In: (2005).

[McI68] Doug McIlroy. “Mass-Produced Software Components”. In: *Proceedings of NATO Software Engineering Conference*. Hrsg. von P. Naur und B. Randell. Garmisch, Germany, Okt. 1968, S. 138–155.

[Mic15] Microsoft Corporation. *Microsoft COM Webseite*. `http://www.microsoft.com/com/`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[Naa12] Matthias Naab. “Enhancing architecture design methods for improved flexibility in long-living information systems”. Diss. Kaiserslautern: Fraunhofer IESE, 2012.

[Nor03] M.E. Nordberg III. “Managing Code Ownership.” In: *IEEE Software* 20.2 (2003), S. 26–33.

[NS12] Matthias Naab und Johannes Stammel. “Architectural Flexibility in a Software-system’s Life-cycle: Systematic Construction and Exploitation of Flexibility”. In: *Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures*. QoSA ’12. Bertinoro, Italy: ACM, 2012, S. 13–22.

[NUn15] NUnit. *NUnit*. `http://www.nunit.org`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[Obj06] Object Management Group (OMG). *CORBA Component Model 4.0 Specification*. Specification Version 4.0. Object Management Group, Apr. 2006.

[Ora15] Oracle. *Enterprise Java Beans Webseite*. `http://www.oracle.com/technetwork/java/javaee/ejb`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[ORM03] Pivi Ovaska, Matti Rossi und Pentti Marttiin. “Architecture as a coordination tool in multi-site software development.” In: *Software Process: Improvement and Practice* 8.4 (2003), S. 233–247.

[OSG15] OSGi Alliance. *OSGi Webseite*. `http://www.osgi.org`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

[Par72] David Lorge Parnas. “On the Criteria To Be Used in Decomposing Systems into Modules.” In: *Commun. ACM* 15.12 (1972), S. 1053–1058.

[Pau01] Daniel J. Paulish. *Architecture-Centric Software Project Management: A Practical Guide*. Addison-Wesley, Reading, MA, USA, 2001.

[PC94] Daniel J. Paulish und Anita D. Carleton. “Case Studies of Software-Process-Improvement Measurement”. In: *IEEE Computer* 27.9 (1994), S. 50–57.

[PNS96] Daniel J. Paulish, Robert L. Nord und Dilip Soni. “Experience with architecture-centered software project planning”. In: *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints ’96) on SIGSOFT ’96 workshops*. ISAW ’96. San Francisco, California, United States: ACM, 1996, S. 126–129.

[Pow11] David Martin Ward Powers. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation”. In: *International Journal of Machine Learning Technology* 2.1 (2011), S. 37–63.

[PR11] R. Pichler und S. Roock. *Agile Entwicklungspraktiken mit Scrum*. Dpunkt.Verlag GmbH, 2011.

[Reu+11] Ralf Reussner, Steffen Becker, Erik Burger, Jens Happe, Michael Hauck, Anne Koziolek, Heiko Koziolek, Klaus Krogmann und Michael Kuperberg. *The Palladio Component Model*. Karlsruhe Reports in Informatics, ISSN: 2190-4782. Karlsruhe, 2011.

[Rij79] C.J. van Rijsbergen. *Information retrieval. 2nd ed.* Hrsg. von Butterworths. 1979.

[Som12] Ian Sommerville. *Software Engineering*. 9. Aufl. München: Pearson, 2012.

[SR11] Cleidson R. B. de Souza und David F. Redmiles. “The Awareness Network, To Whom Should I Display My Actions? And, Whose Actions Should I Monitor?” In: *IEEE Trans. Software Eng.* 37.3 (2011), S. 325–340.

[Sta15] Stammel, Johannes. *Replikationspaket Empirische Studie*. `http://www.johannes-stammel.de/rp.html`. 2015.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[The15] The Apache Software Foundation. *Maven*. `http://maven.apache.org/`. [Online; zuletzt aufgerufen am 13.01.2015]. 2015.

The Karlsruhe Series on Software Design and Quality

Edited by Prof. Dr. Ralf Reussner // ISSN 1867-0067

Band 1 Steffen Becker
Coupled Model Transformations for QoS Enabled Component-Based Software Design.
ISBN 978-3-86644-271-9

Band 2 Heiko Koziolek
Parameter Dependencies for Reusable Performance Specifications of Software Components.
ISBN 978-3-86644-272-6

Band 3 Jens Happe
Predicting Software Performance in Symmetric Multi-core and Multiprocessor Environments.
ISBN 978-3-86644-381-5

Band 4 Klaus Krogmann
Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis.
ISBN 978-3-86644-804-9

Band 5 Michael Kuperberg
Quantifying and Predicting the Influence of Execution Platform on Software Component Performance.
ISBN 978-3-86644-741-7

Band 6 Thomas Goldschmidt
View-Based Textual Modelling.
ISBN 978-3-86644-642-7

Die Bände sind unter www.ksp.kit.edu als PDF frei verfügbar oder als Druckausgabe bestellbar.

The Karlsruhe Series on Software Design and Quality

Edited by Prof. Dr. Ralf Reussner // ISSN 1867-0067

Band 7 Anne Koziolek
Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes.
ISBN 978-3-86644-973-2

Band 8 Lucia Happe
Configurable Software Performance Completions through Higher-Order Model Transformations.
ISBN 978-3-86644-990-9

Band 9 Franz Brosch
Integrated Software Architecture-Based Reliability Prediction for IT Systems.
ISBN 978-3-86644-859-9

Band 10 Christoph Rathfelder
Modelling Event-Based Interactions in Component-Based Architectures for Quantitative System Evaluation.
ISBN 978-3-86644-969-5

Band 11 Henning Groenda
Certifying Software Component Performance Specifications.
ISBN 978-3-7315-0080-3

Band 12 Dennis Westermann
Deriving Goal-oriented Performance Models by Systematic Experimentation.
ISBN 978-3-7315-0165-7

Die Bände sind unter www.ksp.kit.edu als PDF frei verfügbar oder als Druckausgabe bestellbar.

The Karlsruhe Series on Software Design and Quality

Edited by Prof. Dr. Ralf Reussner // ISSN 1867-0067

Band 13 Michael Hauck
Automated Experiments for Deriving Performance-relevant Properties of Software Execution Environments.
ISBN 978-3-7315-0138-1

Band 14 Zoya Durdik
Architectural Design Decision Documentation through Reuse of Design Patterns.
ISBN 978-3-7315-0292-0

Band 15 Erik Burger
Flexible Views for View-based Model-driven Development.
ISBN 978-3-7315-0276-0

Band 16 Benjamin Klatt
Consolidation of Customized Product Copies into Software Product Lines.
ISBN 978-3-7315-0368-2

Band 17 Andreas Rentschler
Model Transformation Languages with Modular Information Hiding.
ISBN 978-3-7315-0346-0

Band 18 Omar-Qais Noorshams
Modeling and Prediction of I/O Performance in Virtualized Environments.
ISBN 978-3-7315-0359-0

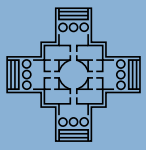
Die Bände sind unter www.ksp.kit.edu als PDF frei verfügbar oder als Druckausgabe bestellbar.

The Karlsruhe Series on Software Design and Quality

Edited by Prof. Dr. Ralf Reussner // ISSN 1867-0067

Band 19 Johannes Josef Stammel
Architekturbasierte Bewertung und Planung von Änderungsanfragen.
ISBN 978-3-7315-0524-2

Die Bände sind unter www.ksp.kit.edu als PDF frei verfügbar oder als Druckausgabe bestellbar.



The Karlsruhe Series on Software Design and Quality

Edited by Prof. Dr. Ralf Reussner

Die Software-Architektur umfasst die technische Organisation eines Software-Systems und die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen. Die Problemstellung ergibt sich aus der Software-Evolution, wenn das System an neue Anforderungen oder technische Weiterentwicklungen angepasst werden muss. In dieser Phase werden die Projektverantwortlichen mit Änderungsanfragen konfrontiert und haben die Aufgabe zu entscheiden, wie diese im System umgesetzt werden. Umsetzungswege wirken sich unterschiedlich stark auf Software-Artefakte, Tätigkeitsbereiche und Lebenszyklusphasen aus. Der Beitrag dieser Arbeit ist ein Verfahren zur Änderungsanfragenanalyse im Architekturmodell, welches die Ableitung von Tätigkeiten in nachgelagerten Tätigkeitsfeldern und Lebenszyklusphasen ermöglicht. Damit wird es möglich, Auswirkungen von Änderungen vor deren Umsetzung abzuschätzen.

ISSN 1867-0067
ISBN 978-3-7315-0524-2

ISBN 978-3-7315-0524-2

Gedruckt auf FSC-zertifiziertem Papier