HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

MSE | MASTER OF SCIENCE IN ENGINEERING

Ein werkzeugunterstütztes Knowledge Repository für Architectural Refactoring

von Anwendungs- und Integrationsarchitekturen

Masterarbeit
University of Applied Sciences Rapperswil
Information and Communication Technologies
Major Software and Systems
www.hsr.ch/mse

Autor:	Christian Bisig, cbisig@gmail.com
Version:	1.1
Datum:	17. Februar 2016
Advisor:	Prof. Dr. Olaf Zimmermann
Experte:	Dr. Hans-Peter Hoidn

Danksagung

Mein Dank geht an meinen Studienbetreuer Prof. Dr. Olaf Zimmermann, welcher mich während meines Masterstudiums an das Thema Softwarearchitektur herangeführt hat. Die Einblicke und das Wissen, das ich dank seiner Unterstützung erlangt habe, werden mir für die Zukunft von grossem Nutzen sein.

Ausserdem danke ich Carmelo Schumacher für die Durchführung des Anwendungstests.

Ein besonderer Dank möchte ich Joel Akeret und Michèle Fischer aussprechen für die Korrekturlesung und die Anregungen, mit welchen die Arbeit wertvolle Verbesserungen erfahren hat.

Und schliesslich geht ein herzlicher Dank an meine Eltern, meine Schwester und Helena Soti für ihre Unterstützung während dieser intensiven Zeit, in welcher diese Arbeit entstanden ist.

Zusammenfassung

Im Software Engineering wird beim Bau neuer Software oft auf bewährte Methoden wie Software Design Patterns oder Blueprints [Lie+08] zurückgegriffen. Die Literatur bietet gute Unterstützung bei der Entwicklung neuer Systeme. Dieses Wissen ist weit verbreitet, und Patterns wie jene der Gang Of Four [Gam+94] gelten als Quasi-Standard. Weniger geläufig sind dagegen Methoden zum Umbau von bestehenden Systemen. Ein derartiger Umbau kann nötig sein, wenn Leistung und Qualität eines aktuellen Systems nicht mehr den Erwartungen entsprechen. Die Behebung eines Problems soll vorzugsweise auf dem bestehenden System aufbauen, also keine Neuimplementation erfordern (aus Kosten- und Kompatibilitätsgründen). An dieser Stelle kommen Architectural Refactorings ins Spiel. Ein AR ist ein Leitfaden zur Implementation/Umsetzung einer qualitätsverbessernden Massnahme für ein Softwaresystem. Der Begriff Architectural Refactoring ist von Prof. Dr. Olaf Zimmermann [Zim15a] [Zim15b] geprägt worden und basiert auf den Ausführungen von Michael Stal zu Software Architecture Refactoring [Sta08]. Mittels sogenannter Smells, welche wertmindernde Symptome in einem Softwaresystem darstellen, verfolgt ein Leidtragender das Ziel, eine geeignete Architekturumbaumassnahme (AR) zu finden. Ein Architectural Refactoring ist keine komplett selbsterklärende Anleitung für den vorgeschlagenen Systemumbau, beinhaltet aber die wesentlichen Umsetzungsschritte und Referenzen, um vertieftes Wissen zu erlagen. Die Projektarbeit 1 [Bis15a] meines Masterstudiums hat das Konzept von Architectural Refactoring aufgegriffen und vertieft sowie ein Design und eine Machbarkeitsstudie für eine Werkzeugunterstützung erarbeitet. In der Projektarbeit 2 [Bis15b] sind das Detail Design sowie eine Implementation des Architectural Refactoring Werkzeugs entstanden.

Die Masterarbeit führt das Konzept von Architectural Refactoring sowie das unterstützende Werkzeug (AR-Werkzeug) ein und erweitert dieses anhand bereits gesammelter Erfahrungen in der Erstellung und Verwendung von ARs. Ein zweiter Hauptbeitrag der Arbeit besteht aus einer systematischen Zusammenstellung von Wissenselementen (Architectural Refactorings) konform zu definierten Modellierungsprinzipien. Zudem vollzieht die Arbeit eine Validierung der Forschungsergebnisse (Konzept, Wissen) sowie des Werkzeugprototyps. Der letzte Teil der Arbeit realisiert Änderungen im AR-Werkzeug, die aus der Validierung resultieren und setzt allgemeine qualitative Verbesserungen um.

Die Masterarbeit schafft mit zwanzig ARs aus dem Bereich der Applikations- und Integrationsarchitektur eine initiale Wissensgrundlage. Die entstandenen Modellierungsprinzipien für die Wissensobjekte bilden dafür und für alle weiteren ARs den beschreibenden Rahmen. Die Validierung des AR-Konzepts mittels allgemeiner Anforderungen für ein Architektur-Designverfahren, eine Architektur-Wissensverwaltung sowie ein Entwicklungs- bzw. Umsetzungsverfahren belegt die Praxistauglichkeit des neuen Verfahrens. Drei Anwendungstests, die Wissensschaffung und deren Validierungen liefern Erkenntnisse, welche in Form von Fehlerbehebungen oder neuen Funktionen im AR-Werkzeug umgesetzt wurden. Dadurch ist das AR-Werkzeug zu einem wertvollen unterstützenden Tool für Architectural Refactorings herangewachsen. Abschliessend zur Masterarbeit wurde das AR-Werkzeug auf Github.com publiziert und auf den Cloud-Service Heroku migriert. Diese Massnahmen sichern den Fortbestand des Werkzeugs als Open-Source-Projekt und bieten eine Referenz für weiterführende Arbeiten zum Thema Architectural Refactoring.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungsverzeichnis

Glossar

ACM Association for Computing Machinery, wissenschaftliche Gemeinschaft für Informatik mit Hauptsitz in New York City.

ADMentor Architecture Decision Mentor ist ein Werkzeug (.NET basiertes Plugin für die Software "Enterprise Architect"), welches am Institute for Software (IFS) der Hochschule für Technik Rapperswil (HSR) entwickelt wurde und zur Unterstützung von Architekturentscheidungen dient.

ADRepo Architecture Decision Repository ist eine Erweiterung des ADMentors um Daten, welche aus dem "Enterprise Architect" exportiert wurden, als Serviceschnittstelle zur Verfügung zu stellen.

AR Architectural Refactoring ist ein Begriff, der von Prof. Dr. Olaf Zimmermann [Zim14] als Weiterentwicklung der Arbeiten von Michael Stal zum Thema Software Architecture Refactoring [Sta08] geprägt wurde.

AR-Werkzeug Architectural Refactoring Tool (ART).

Avatar ist die virtuelle Repräsentation eines Benutzers im Internet meist in Form eines Bildes.

Configuration Item ist ein Betriebsmittel, welches an allen führenden Geschäftsprozessen beteiligt ist und als Eintrag in einer Configuration Management Database existiert. Es können Hardwareelemente wie z. B. Server, Netzwerkkomponenten sein oder aber Softwarekomponenten und andere Werkzeuge.

Configuration Management Database ist eine Datenbank, die Configuration Items (Software-, Hardwarekomponenten) eines Unternehmens verwaltet. Es repräsentiert den Bestand (das Inventar) eines Unternehmens und zeigt die Abhängigkeiten der verschiedenen Configuration Items (CI) untereinander auf.

Cookie ist eine von der Webapplikation via Browser des Benutzers angelegte Datei. Sie wird dafür verwendet, um benutzerspezifische Attribute zu speichern und den Benutzer bei wiederkehrenden Anfragen erkennen zu können.

Cronjob ist ein Mechanismus in Unix Betriebssystemen zur zeitgesteuerten Ausführung von Prozessen.

Dependency Injection beschreibt ein Design Entwurfsmuster, bei welchem Abhängigkeiten in objektorientierten Programmen erst während der Laufzeit aufgelöst werden. Es gibt verschiedene Frameworks, welche Dependency Injection anbieten z. B. Spring oder das im Architectural Refactoring Werkzeug verwendete GUICE (entwickelt von Google).

EEPPI Entwurfsentscheidungen als Projektplanungsinstrument ist ein Werkzeug, welches an der HSR im Rahmen einer Bachelorarbeit entwickelt wurde. Es dient als Task Template Management Unterstützung.

Gang Of Four ist eine Gruppe von 4 Softwareentwicklern (Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides), welche durch das gemeinsam veröffentlichte Buch "Design Patterns" [Gam+94] bekannt wurde. Das Buch gilt als Standard in der Softwareentwicklung für Softwareentwurfsmuster.

Java EE Java Platform Enterprise Edition ist ein Industriestandard für Java Applikationen. Eine Hauptaufgabe ist z. B. der Applikationscontainer für transaktionsbasierte Webanwendungen.

n-Tier Multitier Architektur ist ein Architekturmodell, bei dem verschiedene Schichten eines Applikationssystems getrennt sind, z. B. Datenbank-, Backend- und Frontendschicht.

OOPSLA Object-Oriented Programming, Systems, Languages and Applications, ist eine jährliche Forschungskonferenz zum Thema Object Oriented Programming. Die Konferenz wird durchgeführt von der SIGPLAN, welche Teil der ACM ist.

REST Representational State Transfer, RESTful HTTP ist ein Architekturmodell für Webanwendungen, bei dem Ressourcen über eindeutige Uniform Resource Identifiers (URIs) via HTTP bereitgestellt werden. Eine URI repräsentiert einen Webservice mit einheitlicher Create Read Update Delete (CRUD) Schnittstelle.

SBT Scala Build Tool ist ein Open Source Build Werkzeug für Scala und Java.

Serviceorientierte Architektur (engl. Service Oriented Architecture) ist ein Architekturmodell, bei dem Software-Komponenten, einzelne Geschäftsprozesse oder auch technische Komponenten als Dienste angeboten werden.

SIGPLAN Special Interest Group on Programming Languages, ist Teil der ACM.

Smell sind Symptome für Fehlverhalten in Programmen oder Softwaresystemen.

Software Architecture Document ist ein zentrales Dokument, welches im Softwareentwicklungsprozess Rational Unified Process (RUP) von einem Softwarearchitekt geschrieben wird. Es enthält die Architekturbeschreibung für das zu entwickelnde Softwaresystem.

Softwarearchitekt ist eine Personen in einem Softwareentwicklungsteam. Er ist verantwortlich für Softwarelandschaft und deren Zusammenspiel in einem Unternehmen oder einem Teil davon [RW05][HS09].

SPARC Scalable Processor ARChitecture ist eine Prozessorarchitektur, die ursprünglich von der Firma Sun (heute Oracle) entwickelt wurde und den Reduced Instruction Set Computer (RISC) Befehlssatz verwendet.

Speed Read ist ein Lesevorgang, bei dem möglichst viele Informationen in kurzer Zeit von einem Leser verinnerlicht werden sollen.

Tagcloud ist eine Stichwortwolke, welche auf einem Webfrontend angezeigt wird. Die Worte sind meist alphabetisch geordnet und entsprechend ihrer Gewichtung (Häufigkeit) unterschiedlich gross dargestellt.

TDI Technical Debt Risk Index ist ein ein Wert, welcher die Eintretenswahrscheinlichkeit in Relation zur Auswirkung eines Smells stellt und so die technische Schuld am Softwaresystem bewertet [Fow03].

Thread ist eine ausführende Einheit in einem Programm. In einem Java Programm können mehrere solcher Threads erstellt werden, welche parallel Aufgaben erledigen können. Die Java Virtual Machine ist in der Lage, diese Threads als Betriebssystemprozesse zu abstrahieren, wodurch die Rechenzeitzuteilung durch das Betriebssystem erfolgt. Dies ermöglicht es, auf Multicore Systemen eine reale Parallelität in der Verarbeitung zu erlangen.

x86 ist eine Mikroprozessorarchitektur, welche ursprünglich von der Firma Intel entwickelt wurde und den Complex Instruction Set Computer (CISC) Befehlssatz verwendet. Intel bezeichnet diese Architektur auch als Intel Architecture 32-bit (IA-32).

Abkürzungsverzeichnis

API Application Programming Interface.

CISC Complex Instruction Set Computer.

CPU Central Processing Unit, dt. Prozessor.

CRC Class-Responsibility-Collaboration.

CRUD Create Read Update Delete.

DAO Data Access Object.

DKS Decision Knowledge System.

ERD Entity Relationship Model.

GUI Graphical User Interface.

HSR Hochschule für Technik Rapperswil.

HTML Hyptertext Markup Language.

HTTP Hyptertext Transfer Protocol.

IFS Institute for Software.

ITIL IT Infrastructure Library.

JPA Java Persistence API.

JRE Java Runtime Environment.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

OS Operating System.

OSS Open Source Software.

RDBMS Relational Database Management System.

RISC Reduced Instruction Set Computer.

RUP Rational Unified Process.

SOAP Simple Object Access Protocol.

SQL Structured Query Language.

SSL Secure Sockets Layer.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

VM Virtual Machine.

WebUI Web-based User Interface.

WYSIWYG What You See Is What You Get.

1. Einleitung

1.1. Motivation

Im Software Engineering wird beim Bau neuer Software oft auf bewährte Methoden wie Software Design Patterns oder Blueprints [Lie+08] zurückgegriffen. Die Literatur bietet gute Unterstützung bei der Entwicklung neuer Systeme. Dieses Wissen ist weit verbreitet und Patterns wie jene der Gang Of Four [Gam+94] gelten als Quasi-Standard. Weniger geläufig sind dagegen Methoden zum Umbau von bestehenden Systemen. Ein derartiger Umbau kann nötig sein, wenn Leistung und Qualität eines aktuellen Systems nicht mehr den Erwartungen entsprechen. Die Behebung solcher Probleme soll vorzugsweise auf dem bestehenden System aufbauen, also keine Neuimplementation erfordern (aus Kosten- und Kompatibilitätsgründen). Hier kommen Architectural Refactorings (ARs) ins Spiel. Ein AR ist ein Leitfaden zur Implementation einer qualitätsverbessernden Massnahme für ein Softwaresystem. Der Begriff Architectural Refactoring ist von Prof. Dr. Olaf Zimmermann [Zim15a] [Zim15b] geprägt worden und basiert auf den Ausführungen von Michael Stal zu Software Architecture Refactoring [Sta08]. Mittels sogenannter Smells, welche wertmindernde Symptome in einem Softwaresystem darstellen, verfolgt ein Leidtragender das Ziel, eine geeignete Architekturumbaumassnahme (AR) zu finden. Ein AR ist keine selbsterklärende Anleitung für einen Systemumbau, beinhaltet aber die wesentlichen Umsetzungsschritte und Referenzen, um vertieftes Wissen zu erlangen. Bis anhin fehlte für den Umbau von bestehenden Softwaresystemen mit Hilfe von ARs ein Werkzeug, um Wegleitungen und Hilfestellungen für ein Projekt zu finden.

In der Projektarbeit 1 [Bis15a] wird das Konzept von Architectural Refactoring aufgegriffen und vertieft sowie ein Design und eine Machbarkeitsstudie für eine Werkzeugunterstützung erarbeitet. In der Projektarbeit 2 [Bis15b] ist das Detail Design sowie eine Implementation des Architectural Refactoring Werkzeugs (folgend AR-Werkzeug) entstanden.

Diese Masterarbeit stellt Wissen in Form von ARs aus dem Bereich der Applikations- und Integrationsarchitektur zusammen. Die Forschungsergebnisse (Wissen, Konzept und Werkzeug) werden validiert und anschliessend mit geeigneten Massnahmen verbessert. Das Konzept von Architectural Refactoring wird dabei als ein Architektur-Designverfahren, eine Methode zur Architektur-Wissensverwaltung sowie im Ansatz als ein Software-Entwicklungs- bzw. Umsetzungsverfahren betrachtet. Das AR-Werkzeug bietet hierfür die entsprechende Werkzeugunterstützung.

1.2. Aufbau

Die Arbeit ist in neun Teile gegliedert: Einer **Einleitung** (Kapitel 1) und die Beschreibung der **Projektziele** (Kapitel 2 auf Seite 3) sowie der **Einführung** ins Architectural Refactoring Modell und Werkzeug (Kapitel 3 auf Seite 5), basierend auf der Projektarbeit 1 und Projektarbeit 2. Die Definition von **Modellierungsprinzipien** (Kapitel 4 auf Seite 26), welche als Grundlage für die **Wissensschaffung** (Kapitel 5 auf Seite 34) von Architectural Refactorings sowie der Validierung der Wissensinhalte dient. Dem Festlegen von **Valdierungskriterien** (Kapitel 6 auf Seite 45) und der anschliessenden **Validierung** (Kapitel 7 auf Seite 53), welche die Bewertung des Konzeptes, der Wissensinhalte und des Werkzeugs für Architectural Refactoring umfasst. Ein weiterer Teil besteht aus einem durch eine Drittperson ausgeführten **Anwendungstest** (Kapitel 8 auf Seite 65) und zum Abschluss folgt die Beschreibung der **Werkzeugverbesserungen** auf Grund der Ergebnisse der Validierungen und Anwendungstests (Kapitel 9 auf Seite 70).

2. Projektziele

Die Masterarbeit setzt sich vier Ziele. Das erste Ziel dient der Aufarbeitung und Erklärung des bisher Erarbeiteten (aus Projektarbeit 1 [Bis15a] und 2 [Bis15b]) sowie den Anpassungen, welche diese Elemente in der Masterarbeit erfahren haben. Das zweite Ziel besteht aus der systematischen Erstellung von Wissenselementen anhand von Modellierungsprinzipien. Das dritte Ziel befasst sich mit der Validierung der Forschungsergebnisse und des Werkzeugprototyps. Das letzte Ziel beinhaltet die Realisierung von Änderungen am Konzept auf Grund von Erfahrungen aus der Validierung und die allgemeine qualitative Verbesserungen des Architectural Refactoring Werkzeugs.

2.1. Einführung in das Architectural Refactoring Konzept und Werkzeug

Das erste Ziel der Masterarbeit ist die Aufarbeitung der Ergebnisse aus der Projektarbeit 1 [Bis15a] und 2 [Bis15b]. Die Arbeit erklärt zusammenfassend das Konzept und das Werkzeug sowie daran vorgenommene Änderungen auf Grund der in dieser Masterarbeit vorgeschlagenen Verbesserungen.

2.2. Wissensschaffung

Das zweite Ziel dieser Masterarbeit ist es, eine Wissensbasis aus Architectural Refactorings aufzubauen. Dabei stellt die Arbeit die Praxistauglichkeit des Konzeptes von Architectural Refactoring sowie der Webapplikation (AR-Werkzeug) anhand von Architekturumbauvorschlägen aus dem Bereich der Anwendungs- und Integrationsarchitektur her.

2.3. Validierung der Forschungsergebnisse

Das dritte Ziel ist die Validierung der Forschungsergebnisse. Die Forschungsergebnisse sind in diesem Fall das bestehende und überarbeitete Konzept von Architectural Refactoring, die erschaffenen Wissensinhalte in Form von ARs sowie die Webapplikation als unterstützendes Werkzeug des Konzeptes. Daher ist die Validierung der Forschungsergebnisse in folgende drei Schritte unterteilt:

1. **Konzept-Validierung**
   Das Konzept von Architectural Refactoring ist ein Architektur-Designverfahren und in Zusammenarbeit mit dem AR-Werkzeug eine Methode zur Architektur-Wissensverwaltung sowie im Ansatz ein Entwicklungs- bzw. Umsetzungsverfahren. Die Arbeit validiert das Konzept mittels allgemeinen Anforderungen für diese drei Kategorien nach einem Leitfaden zur Bewertung von Forschungsergebnissen [Zim09].

2. **Wissens-Validierung**
   Ein Hauptteil dieser Masterarbeit erschafft Wissensinhalte und trägt bestehendes Wissen in Form von ARs zusammen. Um die Wissensinhalte zu beurteilen, ist ein beschreibendes Modell nötig. Die Arbeit entwickelt hierfür Modellierungsprinzipien für die Wissensobjekte von Architectural Refactoring und validiert die Inhalte in einem zweiten Schritt anhand dieser Prinzipien.

3. **Werkzeug-Validierung**
   Als dritter Validierungsschritt bewertet die Arbeit die Unterstützung durch das AR-Werkzeug. Dazu entwickelt die Arbeit einen szenariobasierten Anwendungstest, welcher eine Drittperson ausführt. Zudem ist der Anwendungsfall des Erfassens eines neuen ARs sowie das Finden und Implementieren eines bestehenden ARs als Selbsttest ausgeführt und bewertet.

2.4. Funktionale und qualitative Verbesserungen für das Werkzeug

Das vierte und letzte Ziel umfasst die funktionalen und qualitativen Verbesserungen des AR-Werkzeugs. Die Punkte zur Verbesserung des Werkzeugs sind auf Grund der Anwendungstests (durch verschiedene Personen) sowie der Benutzung des Tools während der Wissenserfassung entstanden. Die Punkte sind in drei Kategorien unterteilt und gemäss folgender Auflistung in der Umsetzung priorisiert:

1. **Robustheit**
   Beinhaltet Softwarefehler und Instabilitäten, welche das Arbeiten mit dem AR-Werkzeug stark beeinträchtigen.

2. **Usability**
   Zusätzlich Funktionen, welche das Arbeiten mit dem AR-Werkzeug erleichtern und verbessern.

3. **Code-Qualität und Wartbarkeit**
   Beinhaltet Punkte, welche die Qualität des Quellcodes sowie der Wartbarkeit des AR-Werkzeugs steigern.

Diese Verbesserungen erhöhen die Praxistauglichkeit, das Anwendungserlebnis sowie die einfachere Wartbarkeit und Weiterentwicklung des AR-Werkzeugs.

3. Architectural Refactoring Modell und Werkzeug: Einführung

Dieses Kapitel beschreibt das Design und Konzept von Architectural Refactorings sowie die Architektur des Knowledge Repository (Architectural Refactoring Werkzeug). Das Konzept und Werkzeug entstammen ursprünglich aus der Projektarbeit 1 [Bis15a] bzw. der Projektarbeit 2 [Bis15b]. Das Kapitel enthält als Erstes eine Zusammenfassung der Projektarbeit 1 und 2, geht aber bei jenen Punkten, welche auf Grund der Forschungsarbeiten in dieser Arbeit eine Änderung erfahren haben, genauer ins Detail.

Das Konzept des Architectural Refactoring ist als ein Architektur-Designverfahren mit Softwareentwicklungs-Elementen und einem aktiven Wissensmanagement im Kontext von Software Evolution und Wartung zu beschreiben. Das AR-Werkzeug stellt dabei die entsprechende Werkzeugunterstützung bereit.

Das zentrale Datenobjekt des AR-Werkzeugs ist das Architectural Refactoring. Tabelle 3.1 zeigt und beschreibt die Attribute eines ARs. In Kapitel 5 auf Seite 34 sind die in dieser Masterarbeit erarbeiteten ARs enthalten. Die Struktur der ARs basiert auf der AR-Tabelle von Prof. Dr. Olaf Zimmermann [Zim14]. Die Projektarbeiten 1 und 2 sowie die Masterarbeit nehmen an dieser Tabelle verschiedene Anpassungen in der Namensgebung, der Anordnung und den Attributen vor.

3.1. Die Hauptobjekte des Architectural Refactoring Werkzeugs

Dieser Abschnitt beschreibt den Aufbau eines ARs sowie dessen weitere Hauptobjekte Smell, Execution Task und Model Elements, welche in enger Verbindung zum AR stehen. Abbildung 3.1 zeigt die Abhängigkeiten dieser Hauptobjekte in einem Klassendiagramm.

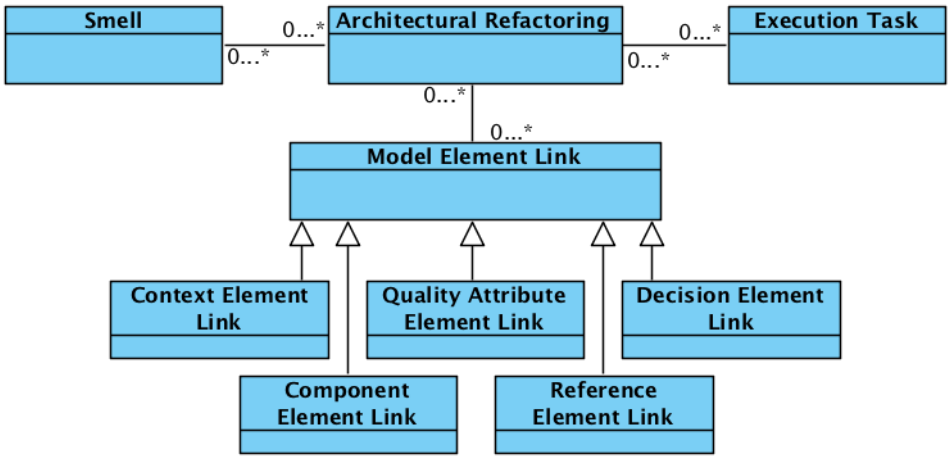


Abbildung 3.1: Klassendiagramm Hauptobjekte im AR-Werkzeug

Das AR steht mit allen Objekten in Mehrfachbeziehungen. Diese Beziehungen erlauben es, das AR mit beliebig vielen Objekten (Smell, Tasks, Modell Elemente) zu verknüpfen und gewährleisten dabei die Wiederverwendbarkeit der einzelnen Objekte.

3.1.1. Architectural Refactoring

Ein AR definiert einen Leitfaden zur Implementation/Umsetzung einer qualitätsverbessernden Massnahme für ein Softwaresystem. Mittels Symptomen (Smells) wird versucht, die Ursache für die Wertminderung im System und somit ein geeignete Verbesserungsmassnahme (AR) zu finden. Ein Architectural Refactoring besteht nicht aus einer vollständig selbsterklärenden Anleitung zur Umsetzung eines kompletten Architekturumbaus, sondern beinhaltet eine Zusammenfassung des Umbaus, eine Auflistung der meist relevanten Ausführungsschritte zur Umsetzung sowie eine Anzahl von Literaturreferenzen.

Die folgende Tabelle 3.1 beschreibt den Aufbau eines ARs. Die Reihenfolge und Anordnung entspricht der Darstellung wie sie auch im AR-Werkzeug implementiert ist. Aus den Anwendungstests mit dem AR-Werkzeug ist diese Gliederung als die praktikabelste Ansicht zur Einarbeitung in ein AR hervorgegangen.

<table>
<tr><td>Name
Eine kurze, prägnante Beschreibung des ARs</td><td>Status
Ein AR kann folgende Status haben:
draft, in Review, accepted, published, deprecated, unspecified</td></tr>
<tr><td>Context (viewpoint, refinement level)
Enthält eine Auflistung von betroffenen Viewpoints (z. B. Functional View von 4+1 View Model von Kruchten [Kru95]) oder Refinement Levels, z. B. Software Level.</td><td>Quality attributes (forces)
Eine Auflistung der vom AR adressierten Qualitätsattribute.</td></tr>
<tr><td colspan="2">Smells (refactoring drivers)
Eine Smell ist ein Symptom in einem Softwaresystem, welcher auf einen Missstand hinweist. Ein Architectural Smell bezieht sich immer auf das oben erwähnte und im Softwaresystem beeinträchtigte Qualitätsattribut.</td></tr>
<tr><td colspan="2">Description
Die Beschreibung enthält eine Zusammenfassung des Problems (Initial Position), das Vorgehen zur Verbesserung des Designs (Revised Design) sowie eine Auflistung von Punkten, auf welche beim Umsetzen des ARs zu achten sind (Pitfalls to avoid).</td></tr>
<tr><td>Architectural decision(s) to be revisited
Eine Auflistung von vormals gefällten Architekturentscheidungen, welche mit der Umsetzung des spezifischen ARs neu zu überdenken sind, z. B. die Wahl einer Middleware Komponente.</td><td>Affected components and connectors (if modelled explicitly)
Ein Auflistung der Komponenten, welche von der Umsetzung des ARs betroffen sind, z. B. die Business Software.</td></tr>
<tr><td colspan="2">Execution tasks
Die Ausführungsschritte sind die Schnittstellen zur projektorientierten Umsetzung des Architectural Refactorings. Es gibt verschiedene Arten von Ausführungsschritten z. B. “Design Task”, “Development Task” oder “Testing Task”.</td></tr>
<tr><td colspan="2">References (links)
Referenzen sind ein wichtiger Teil des ARs. Da ein AR eine nicht exhaustive Beschreibung eines Architekturumbaus wiedergibt, ist es nötig, entsprechende Referenzen bereitzustellen, welche zu vertiefendem Wissen führen.</td></tr>
</table>

Tabelle 3.1: Architectural Refactoring Attribute

Ursprünglich (Projektarbeit 1 [Bis15a]) enthielt das AR ein Attribut “Refactoring (solution sketch/evolution outline)”, welches während der Masterarbeit in die “Description” integriert wurde, um Redundanzen zu vermeiden.

3.1.2. Smell

Der Smell ist ein wichtiges Objekt im AR Konzept. Er ist das Element, über welches ein Benutzer eine Lösung für ein Problem in seinem Softwaresystem sucht. Der Smell ist das Symptom für ein Problem in einem bestehenden Softwaresystem. ARs referenzieren einen oder mehrere Smells. Ein Smell bezieht sich immer auf ein Qualitätsattribut, welches durch das verursachende Problem beeinträchtigt ist.

<table>
<tr><td>Name
Eine kurze prägnante Beschreibung des Smells</td><td>Status
Ein Smell kann folgende Status haben (analog AR):
draft, in Review, accepted, published, deprecated, unspecified</td></tr>
<tr><td>Keywords
Die Keywords enthalten 2-3 der meist beschreibenden Worte des Smells.</td><td rowspan="2">Technical Debt Risk Index (TDI)
Der TDI setzt die Eintretens-Wahrscheinlichkeit in Relation zur Auswirkung des Smells. Der Wert wird aus einer 3x3 Matrix ausgewählt, in der die Y-Achse die Eintretens-Wahrscheinlichkeit low, mid, high und die X-Achse die Auswirkung low, mid, high enthält. Daraus resultiert der Kombinationswert für die technische Schuld [Fow03].</td></tr>
<tr><td>Group
Die Smell Gruppe ist eine Kategorisierung des Smells nach einem Qualitätsattribut.</td></tr>
<tr><td colspan="2">Description
Die Beschreibung des Smells beinhaltet 1-2 Sätze, welche das Symptom erläutern und einen Benutzer dazu befähigt, in kurzer Zeit zu ermitteln, ob das beschriebene Symptom in seinem Softwaresystem auftritt oder nicht.</td></tr>
<tr><td colspan="2">Questions
Die Fragen zu den Smells sind wichtig, da diese im “Smell Self-Assessment” des AR-Werkzeugs in einem Fragebogen Verwendung finden. Die Auswertung dieses Fragebogens ermittelt mögliche Problemlösungen in Form von ARs. Die Fragen sind mit Ja oder Nein zu beantworten. Es müssen fachliche und keine rhetorischen oder Suggestivfragen sein.</td></tr>
<tr><td colspan="2">Referenced ARs
Ist kein eigentliches Attribut sondern zeigt im AR-Werkzeug lediglich an, welche ARs den angezeigten Smell referenzieren.</td></tr>
</table>

Tabelle 3.2: Smell Attribute

3.1.2.1. Smell Gruppen

Smells sind in Gruppen von Qualitätsattributen [Sta11] eingeteilt. Die Gruppierung ist wichtig für die übersichtliche Gestaltung des “Smell Self-Assessment” Fragebogens. Im Administrationsbereich des AR-Werkzeugs ist es möglich, neue Smell Gruppen hinzuzufügen oder zu verändern. Es existieren die folgenden Smell Gruppen:

- Performance
- Complexity
- Dependency
- Security
- Standards
- Maintainability
- Affordability
- Functionality

3.1.3. Execution Task

Die Ausführungsschritte bilden die Schnittstellen zur projektorientierten Abwicklung des Architekturumbaus. Sie enthalten die nötigen Punkte zur Umsetzung der Problemlösung. Sie lassen sich kategorisieren und einer Projektrolle zuordnen.

Name Eine kurze prägnante und spezifische Beschreibung des Execution Tasks. Der Execution Task soll nicht generisch sein.
Assignee Projektrolle, welche in der Verantwortung steht den Task auszuführen. Z. B. ein Softwarearchitekt, ein Softwareentwickler, ein Testmanager, etc.
Type Ist ein Exeuction Task Type (Details siehe Abschnitt 3.1.3.1 auf Seite 10), welcher den Execution Task kategorisiert.
Description Die Beschreibung führt den Arbeitsschritt im Detail aus und beinhaltet mögliche Teilschritte oder Bedingungen, welche erfüllt sein müssen, um den Task ausführen zu können.
Priority Die Priorität des Ausführungsschrittes kann entweder low, mid oder high sein.
Due Date Das Due Date ist sehr spezifisch und nur in seltenen Fällen anzugeben.

Estimated Duration
Zeitangabe über die geschätzte Dauer zur Ausführung des Execution Tasks. Angabe in Personen-Stunden, -Tagen oder -Monaten.

Tabelle 3.3: Execution Task Attribute

3.1.3.1. Execution Task Type

Das Klassendiagramm in Abbildung 3.2 zeigt alle zur Zeit der Masterarbeit vorhandenen Execution Task Types. Im Administrationsbereich des AR-Werkzeugs können neue Execution Task Types erfasst oder bestehende angepasst werden. Eine erste Zusammenstellung von Execution Task Types ist in der Projektarbeit 1 entstanden. Die Masterarbeit ergänzt die Typen um den “Architectural Decision Task”, den “Modelling Task” sowie den “Hold Meeting Task”.

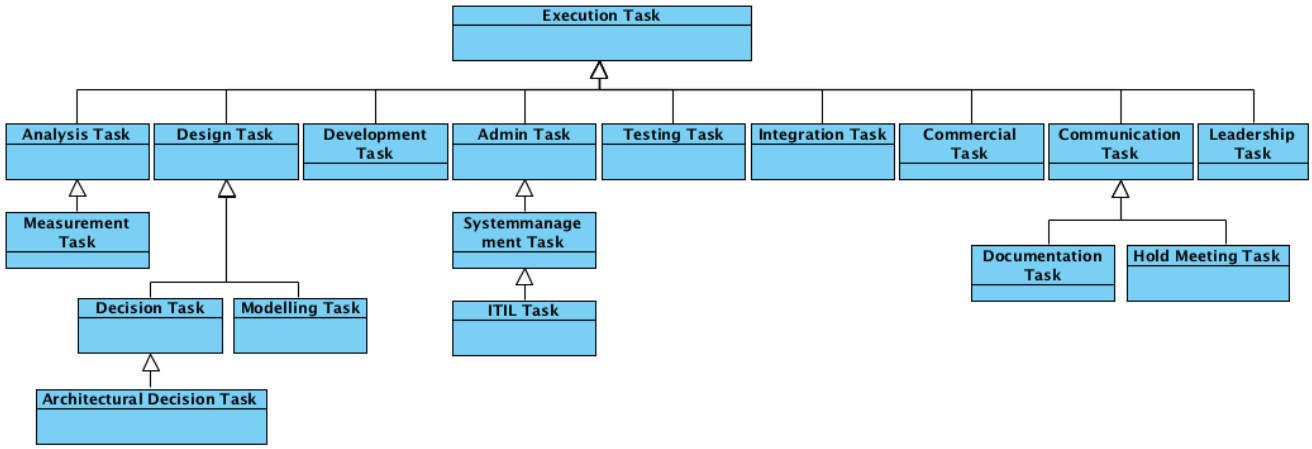


Abbildung 3.2: Klassendiagramm Execution Task Typen

Tabelle 3.4 enthält Beispiele für jeden Execution Task Type:

Type	**Beispiel**
Analysis Task	Top-Down Analyse des bestehenden Softwaresystems
Measurement Task	Messen der aktuellen Systemkapazität
Design Task	Entwerfen der zu verwendenden Tier-Architektur
Decision Task	Entscheidung über die Verwendung einer bestimmten Middleware-Komponente
Architectural Decision Task	Entscheidung über das zu verwendende Architekturpattern

Modelling Task	Entwerfen des Domain-Models für ein Softwaresystem
Development Task	Implementieren einer REST-Schnittstelle
Admin Task	Einrichten von Arbeitsplätzen für neue Mitarbeiter
Systemmanagement Task	Ersetzen von SSL Zertifikaten auf einem Webserver
ITIL Task	Bestellen von neuen Servern für ein produktives System
Testing Task	Ausführen eines Lasttests
Integration Task	Paketieren einer Third-Party Software für die Ausbreitung auf ein Testsystem
Commercial Task	Erneuern von Software-Lizenzen einer Third-Party Applikation
Communication Task	Aktualisieren der Projektwebseite im Intranet
Documentation Task	Aktualisieren des Softwarearchitekturdokumentes nach der Umsetzung eines ARs
Hold Meeting Task	Organisieren eines Projektkickoff-Meetings
Leadership Task	Einstellen von neuen Softwareentwicklern

Tabelle 3.4: Beispiele für Execution Task Types

3.1.4. AR-Meta Data / Model Element Links

Unter dem Begriff AR-Meta Data bzw. Model Element Links sind die fünf im AR verlinkbaren Attribute “Context”, “Quality attributes”, “Architectural decisions”, “Affected components and connectors” und “References” vereint. Diese Attribute kann ein Benutzer mittels den Multiselektionslisten im AR-Werkzeug einem AR zuteilen.

Die Auflistung zeigt die verschiedenen Typen (technischer Name, welcher im AR-Werkzeug verwendet wird) in der Reihenfolge, wie diese in der AR Übersicht auftreten:

- ContextElementLink
  Der Kontext, in welchem das AR steht, kann ein Viewpoint (z. B. “4+1” von Kruchten [Kru95]), eine Refinement Level oder eine Enterprise Architektur sein.
- QASElementLink
  “Quality attributes and stories”, wobei während dieser Arbeit entschieden wurde, den Teil “and stories” in der Beschreibung wegzulassen. Ein Qualitätsattribut ist eine Eigenschaft eines Softwaresystems, welches durch die Umsetzung des ARs verbessert werden soll.
- DecisionElementLink
  Eine Design Entscheidung, welche mit dem Umsetzen des ARs neu evaluiert

werden muss. Kann auch ein Link auf ein Decision Knowledge System (DKS) enthalten.

- ComponentElementLink
  Der Name einer Komponente, welche durch die Umsetzung eins ARs geändert wird oder anderweitig betroffen ist. Kann ein Link auf eine genauere Beschreibung oder ein Configuration Item in einer Configuration Management Database sein.

- ReferenceElementLink
  Eine Referenz auf weiterführendes Wissen, welches der Umsetzung des ARs dient. Kann ein Buch (mit ISBN-Nummer), ein Artikel, ein Report oder ein Link auf eine Internet Ressource sein.

Der nachfolgende Abschnitt 3.2 stellt die Model Element Links im Domain Modell in Abhängigkeit zu den weiteren Elementen des AR-Werkzeugs dar.

3.2. Domain Model Architectural Refactoring

Das Domain Model bietet als Teil eines Business Modeling Prozesses eine Übersicht über alle Objekte einer Domain (Definitionsbereich des Softwaresystems). Es stellt konzeptionelle Klassen und ihre Beziehungen in einem UML Klassendiagramm [Lar08] dar. Die Klassen repräsentieren reale Situationen, sind aber unabhängig von der physischen Implementierung (Java-Klassen). Die abgebildeten Klassen stellen somit keine Programmklassen dar. Die Abbildung 3.3 zeigt das Domain Model der AR-Werkzeug Applikation. Das Model entstammt ursprünglich der Projektarbeit 1 [Bis15a]. In der Masterarbeit hat das Model kleinere Anpassungen erfahren. In der jetzigen Version zeigt das Domain Model alles, was auch in der Applikation abgebildet ist.

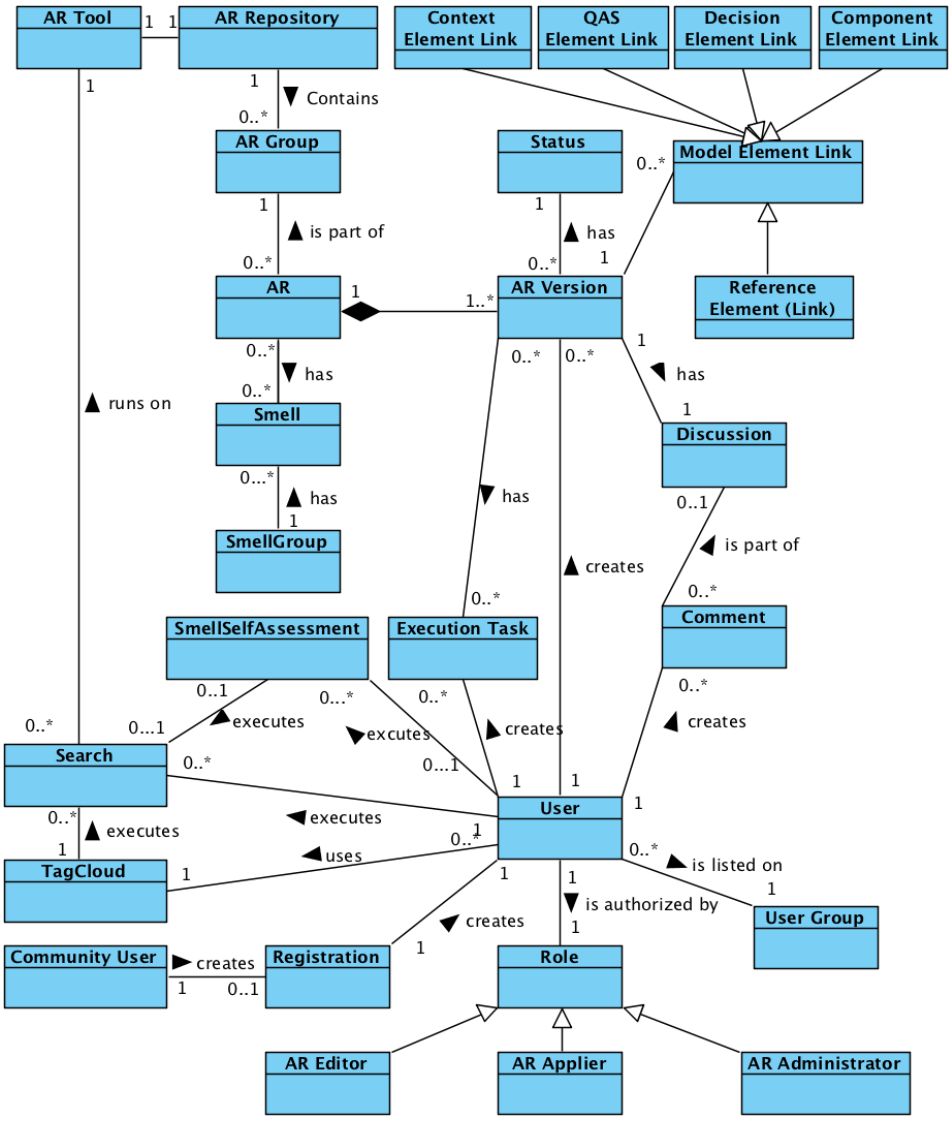


Abbildung 3.3: Domain Model Klassendiagramm

Die Änderungen gegenüber der Vorversion sind folgende:

- Ursprünglich war vorgesehen, nicht nur das AR zu versionieren, sondern auch der Execution Task und der Smell. Davon wurde aber auf Grund des erhöhten Entwicklungsaufwands abgesehen und als Feature-Request für die Zukunft erfasst (siehe Abschnitt A.4 auf Seite 124 Punkt Nr. 68).
- Die Smells wurden in Gruppen eingeteilt und somit ist die Klasse SmellGroup neu hinzugekommen.
- Auch das SmellSelfAssessment ist als Mittel zur Suche von ARs hinzugekommen.

- Ursprünglich waren für das Diskutieren und Kommentieren der AR-Versionen zwei getrennte Bereiche vorgesehen. Eine für die wissenserfassenden Personen (AR-Editoren) und eine für die wissensanwendenden Personen (AR-Applier). Dies wurde zur Vereinfachung weglassen, aber als Feature-Request erfasst (siehe Abschnitt A.4 auf Seite 124 Punkt Nr. 71).
- Der “Design Element Link” wurde in “Component Element Link” umbenannt und der “Context Element Link” ist neu hinzugekommen. Zudem ist der “Reference Element Link” nun ebenfalls vom “Model Element Link” abgeleitet und nicht direkt, wie zuvor von der “Ar Version” referenziert.

3.2.1. Konzeptionelle Klassen

Tabelle 3.5 zeigt die Beschreibung der in Abbildung 3.3 dargestellten konzeptionellen Klassen. Eine solche Klasse kann verschiedene Arten von Objekten repräsentieren z. B. eine Transaktion, eine Beschreibung, ein physisches Objekt, ein Daten Container oder eine Liste, eine Person oder Rolle etc.

3.2.2. Klassenbeschreibung

Klasse	**Beschreibung**
AR Tool	Das AR-Werkzeug als Hauptobjekt des Models.
AR Repository	Das Repository als Repräsentation des Speicherorts in dem der Inhalt (Content) des AR-Werkzeugs abgelegt ist.
AR Group	Die Gruppe definiert einen Typ von ARs.
AR	Das zentrale Inhaltsobjekt des Systems.
AR Version	Eine Instanz des ARs mit Versionsnummer, es bestehen die aktuelle und beliebig viele weitere Instanzen.
Smell	Das Symptom für ein Problem, mit welchem eine geeignete Lösung (AR) im System gefunden werden soll (Refactoring Driver).
Smell Group	Eine Einteilung von Smells in Kategorien von Qualitätsattributen.
Smell Self-Assessment	Eine Funktion zur Ermittlung von geeigneten ARs mittels eines Fragebogens.
Status	Der Status eines ARs.
Model Element Link	Basis Klasse für Element Links.
Context Element Link	Kontext, Architecture Viewpoints
QAS Element Link	Qualitätsattribute sind Eigenschaften eines ARs, welche durch das Umsetzen des ARs verbessert werden sollen.

Decision Element Link	Design Entscheidungen, die bei der Umsetzung des ARs neu zu evaluieren sind. Z. B. eine Referenz auf ein Decision Knowledge System (DKS).
Component Element Link	Von der Änderung betroffene Komponenten.
Reference (Link)	Ein Link auf weiterführendes Wissen (z. B. Web-URI, Buchreferenz (ISBN) etc.).
Discussion	Eine Diskussion von Architekturexperten über eine AR-Version. Das Discussion Objekt repräsentiert die ganze Konversation zu einem AR.
Comment	Objekt, das einen einzelnen Beitrag eines Benutzers zu einer Diskussion repräsentiert.
Execution Task	Ein einzelner Schritt, welcher zur Umsetzung eines ARs ausgeführt wird. Das Objekt repräsentiert eine Vorlage, welche anschliessend in einem konkreten Projekt instantiiert wird. Der Execution Task ist eine abstrakte Klasse und durch verschiedene Execution Task Typen konkretisiert.
User	Ein Benutzer des Systems.
User Group	Die Gruppe/Liste aller Benutzer, welche Zugriff auf das System haben.
Search	Von einem Benutzer auf das AR-Werkzeug ausgeführte Suche zum Finden eines geeigneten ARs.
TagCloud	Ein Hilfsmittel, mit welchem ein Benutzer eine vereinfachte Suche (mit einem Klick auf ein Schlüsselwort (Smell)) auf dem AR-Werkzeug ausführen kann.
Community User	Eine dem System aussenstehende Person.
Registration	Repräsentiert den Vorgang eines Community-Users, sich im AR-Werkzeug als Benutzer zu registrieren.
Role	Ein abstraktes Objekt im System. Jedem Benutzer wird eine Rolle zugeteilt. Sie definiert die Berechtigungen eines Benutzers innerhalb des Systems.
AR Editor	Eine Konkretisierung einer Benutzerrolle. Erlaubt es, Inhalte im AR-Werkzeug anzupassen.
AR Applier	Eine Konkretisierung einer Benutzerrolle. Erlaubt es, nach Inhalten zu suchen und diese zu betrachten.
AR Administrator	Eine Konkretisierung einer Benutzerrolle. Beinhaltet alle Berechtigungen für das AR-Werkzeug.

Tabelle 3.5: Konzeptionelle Klassen

3.3. Architectural Refactoring Werkzeug

Der folgende Abschnitt beschreibt die Rollen und das Design des AR-Werkzeugs. Ausführliche Details sind in der Projektarbeit 2 [Bis15b] enthalten. Die Benutzerrollen sind für die Wissens- und Werkzeug-Validierung (siehe Kapitel 7 auf Seite 53) von Bedeutung und hier nochmals im Detail beschrieben.

3.3.1. Rollen

Jeder Benutzer des AR-Werkzeugs nimmt eine der drei vorhanden Benutzerrollen, AR-Editor, AR-Applier oder AR-Administrator, ein.

- **AR-Editor** Die Rolle AR-Editor beschreibt einen Spezialisten auf dem Gebiet der Softwarearchitektur. Jemand, der ein ausgeprägtes Wissen im Bereich der Softwarearchitektur hat, z. B. ein sehr erfahrener Softwarearchitekt aus einem Unternehmen oder auch jemand, welcher auf dem Forschungsgebiet der Softwarearchitektur tätig ist. Der AR-Editor teilt sein Expertenwissen, in dem er Architekturumbauvorschläge im AR-Werkzeug erfasst.
- **AR-Applier (User)** Ein AR-Applier ist eine Person, welche Hilfe beim Lösen eines Problems sucht. Dies kann ein Softwarearchitekt oder ein Softwareentwickler aus einem Unternehmen sein, welcher 1-2 Jahre Berufserfahrung mit sich bringt. Die Person sucht eine Lösung für ein Problem anhand eines Symptoms (Smell) und hofft, mit diesem einen Architekturumbau (AR) zu finden, welchen er auf seine Architektur anwenden kann.
- **AR-Administrator** Der AR-Administrator hat die Rechte, um alle Daten im AR-Werkzeug zu erstellen, lesen, verändern und löschen. Insbesondere ist es ihm erlaubt, Datenobjekte (z. B. ein AR oder ein Execution Task) physisch von der Datenbank zu löschen. Es ist eine technische Rolle, welche nicht von einem Projektmitarbeiter oder einem Softwarearchitekten belegt ist, sondern von einer Person, welche das Softwaresystem betreibt (Operations).

3.3.2. Anwendungsfälle

Dieser Abschnitt beschreibt die Anwendungsfälle und Aktivitäten der verschiedenen Rollen des AR-Werkzeugs. Abbildung 3.4 zeigt die Erfassung und Modellierung durch einen AR-Editor und die anschliessende Überprüfung durch einen zweiten AR-Editor.

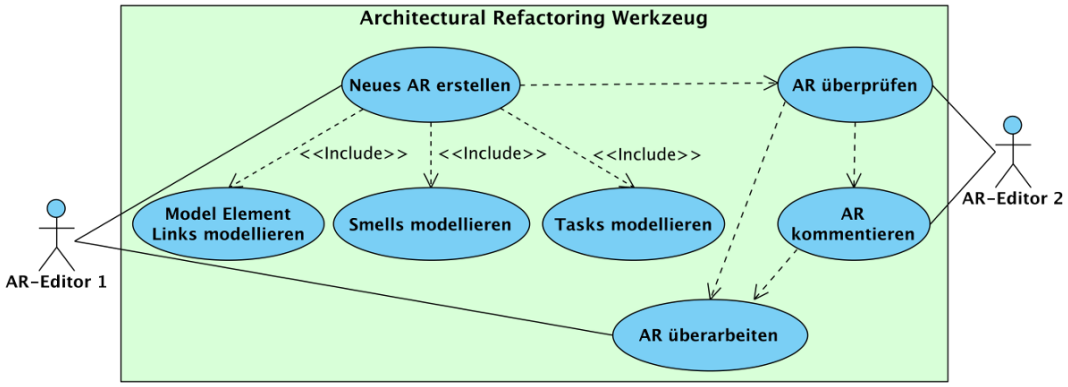


Abbildung 3.4: Anwendungsfall: AR erstellen und prüfen (AR-Editor)

Abbildung 3.5 zeigt, wie ein AR-Applier auf Grund eines Symptoms in einem Softwaresystem ein Problem erkennt und darauf eine Lösung mittels AR-Werkzeugs sucht. Dazu füllt der Benutzer das Smell Self-Assessment aus und wählt aus den vorgeschlagenen ARs eines oder mehrere aus, welche er umsetzen möchte. Die Umsetzung plant der AR-Applier in einem Projekt und die finale Implementation erfolgt im problemverursachenden Softwaresystem.

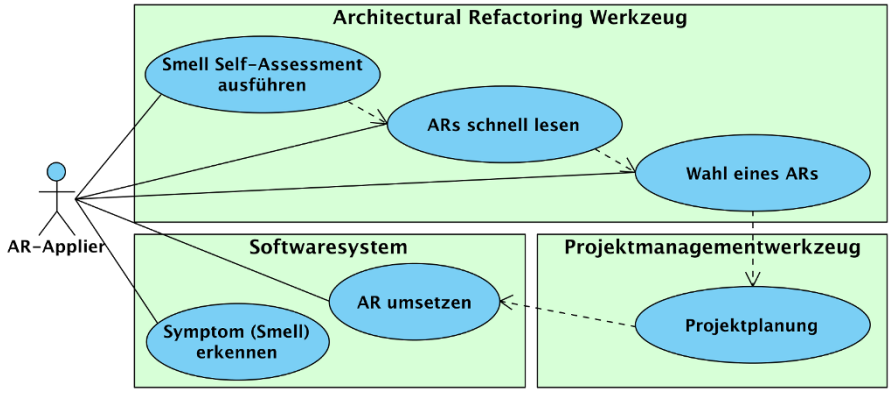


Abbildung 3.5: Anwendungsfall: AR suchen, finden und anwenden (AR-Applier)

Abbildung 3.6 stellt eine zweite Sicht auf den Anwendungsfall eines AR-Appliers dar. Das Aktivitätsdiagramm illustriert sequentiell die Schritte, um mit dem AR-Werkzeug ein AR zu finden und umzusetzen. Daneben zeigt das Diagramm vereinfacht die herkömmlichen Wege (Community kontaktieren und Internetsuche), um eine Lösung für ein Problem zu finden. Der Pfad über das AR-Werkzeug bietet die Vorteile einer zusammenfassenden Übersicht mit den nötigen Referenzen und Umsetzungsschritten.

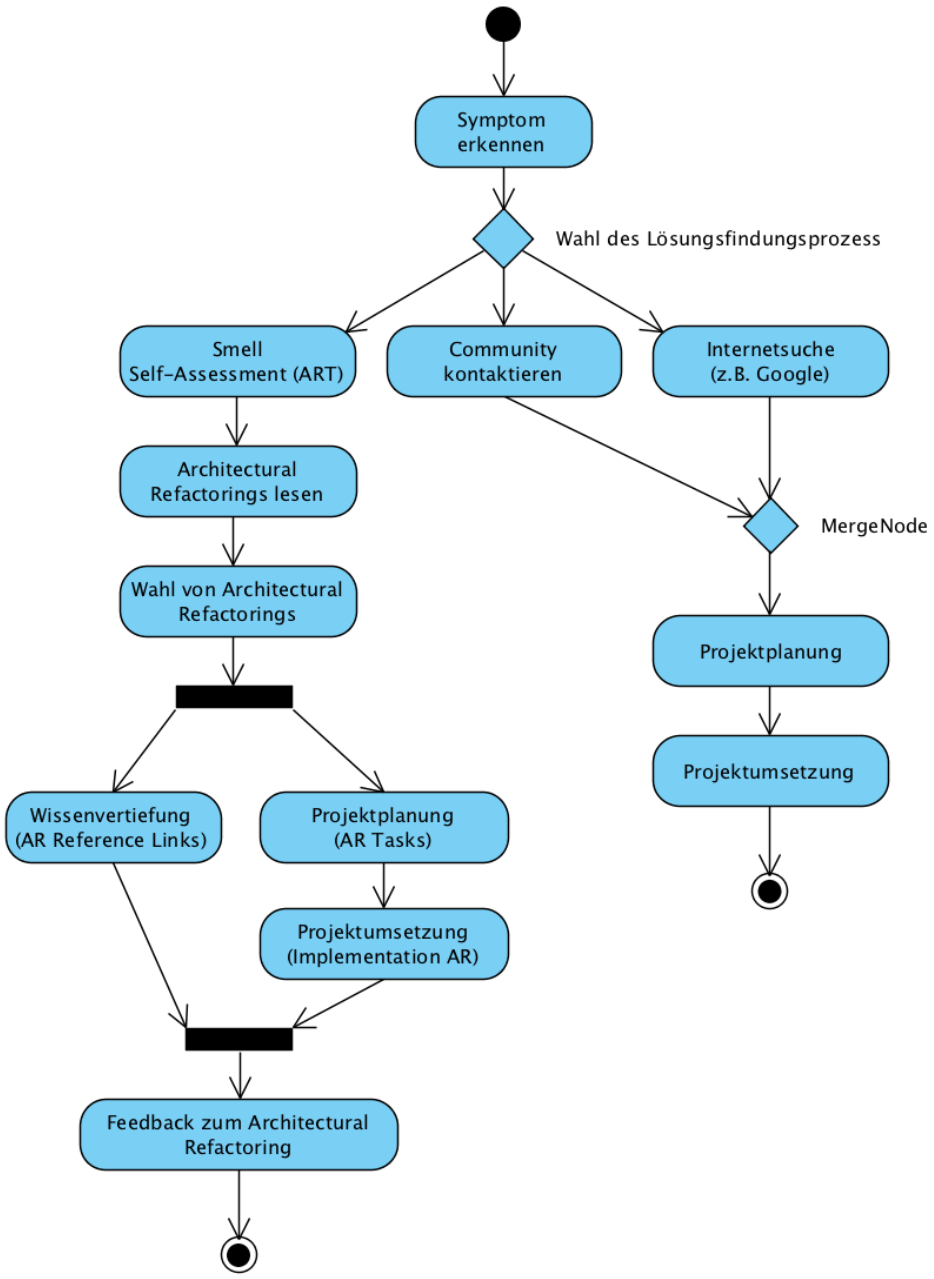


Abbildung 3.6: Aktivitäten: AR suchen, finden und anwenden (AR-Applier)

Abbildung 3.7 zeigt die Aufgaben des AR-Administrators. Es liegt in seiner Verantwortung, das AR-Werkzeug zu initialisieren und zu verwalten.

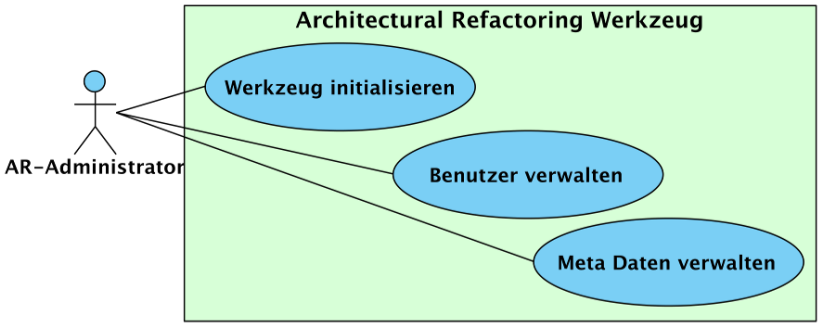


Abbildung 3.7: Anwendungsfall: AR-Werkzeug initialisieren und verwalten (AR-Admin)

3.3.3. Systemdesign und -architekur

Abbildung 3.8 zeigt die Systemarchitektur wie das AR-Werkzeug auf einer Virtual Machine (VM) der Hochschule für Technik Rapperswil (HSR) installiert ist. Die AR-Werkzeug Webapplikation besteht aus einem HTML-Javascript Frontend (verwendet Angular JS Framework von Google [Goo]) und einem Java Backend, welches das Web Applikations Framework Play [TZC] nutzt. Das Frontend ist eine Single-Page-Javascript Applikation. Das Backend stellt RESTful-Services zur Verfügung, über welche das Frontend Daten bezieht und manipuliert. Das Backend läuft mittels Webserver Frameworks Netty, welches Teil von Play ist, als eigener Serverprozess. Die Datenquelle für die ganze Applikation ist eine MySQL Datenbank.

Die Komponenten EEPPI (Task Template Management System) und ADRepo (DKS) sind an der HSR entwickelte Werkzeuge, zu welchen eine Schnittstelle im AR-Werkzeug vorbereitet ist. Für die Zukunft könnte ein Export von Execution Tasks ins EEPPI erfolgen oder aber Architekturentscheidungen aus dem ADRepo als ARs importiert werden.

Die Installation des AR-Werkzeugs erfolgt automatisiert über den Build Server Jenkins, welcher die Sourcen des AR-Werkzeugs aus einem Git-Repository bezieht.

Ein als Proxy (zur Port-Weiterleitung) konfigurierter Apache Webserver ermöglicht den Zugriff auf die Komponenten via eines Verzeichnispfades statt der Angabe eines HTTP-Ports. Z. B. http://sinv-56041.edu.hsr.ch/art-app/ für das AR-Werkzeug und http://sinv-56041.edu.hsr.ch/eeppi/ für das EEPPI.

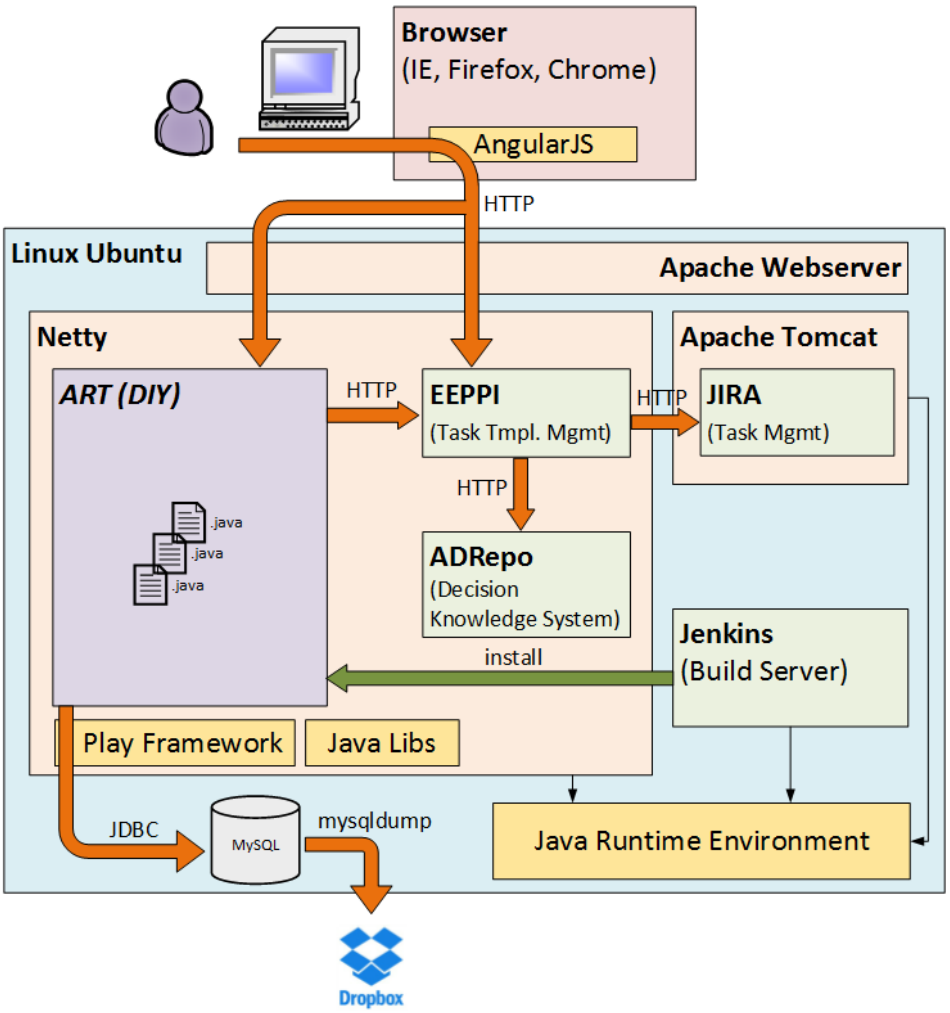


Abbildung 3.8: Kontext Diagramm AR-Werkzeug

Gegenüber der Vorversion hat sich im Kontext Diagramm ein Punkt geändert: Da das AR-Werkzeug während der Masterarbeit für Tests und Wissenserfassung in einem produktiven Modus genutzt wird, nimmt ein Batchjob jede Nacht eine automatische Sicherung der Datenbank vor (Datentransfer auf Dropbox). Diese Massnahme verhindert den Verlust von Änderungen, die älter als einen Tag sind. Ein Anwender kann den Batchjob auch manuell ausführen, um einzelne Arbeitsabschnitte zu sichern.

3.3.4. Anwendungsaufbau und -elemente

Abbildung 3.9 zeigt die Menüleiste mit allen Untermenüs des AR-Werkzeugs. Die nachfolgende Auflistung ist eine Übersicht der AR-Werkzeug-Webapplikation und beschreibt den Aufbau und die Funktionen der einzelnen Seiten.

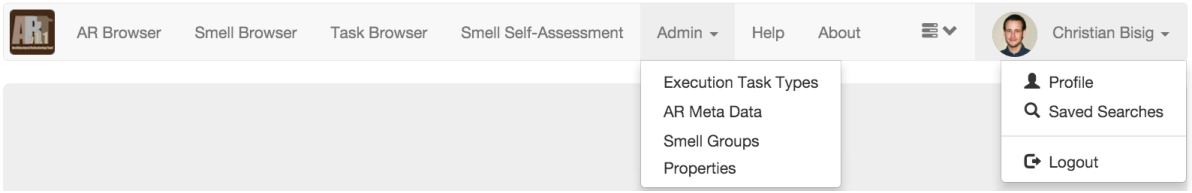


Abbildung 3.9: Architectural Refactoring Werkzeug Menüleiste

AR Browser

Der AR Browser zeigt eine tabellarische Auflistung der vorhandenen ARs (Abbildung 3.10). Für nicht eingeloggte Benutzer und Benutzer mit der Rolle AR-Applier sind nur die ARs mit dem Status “published” zu sehen. Mit einem Klick auf die Namen der ARs gelangt man auf die Detailansicht des ARs. Für einen AR-Editor oder -Administrator besteht die Möglichkeit, direkt in den Bearbeitungsmodus des ARs zu gelangen. Seitlich neben der Tabelle befindet sich die Smell-Cloud. Sie enthält die Namen der meistverlinkten Smells. Ein Klick auf einen Smell-Namen in der Cloud filtert die Tabelle der ARs entsprechend dem ausgewählten Smell. Oberhalb der Tabelle befindet sich der Button “Add AR”, welcher zum Erfassungsformular für ein neues AR führt.

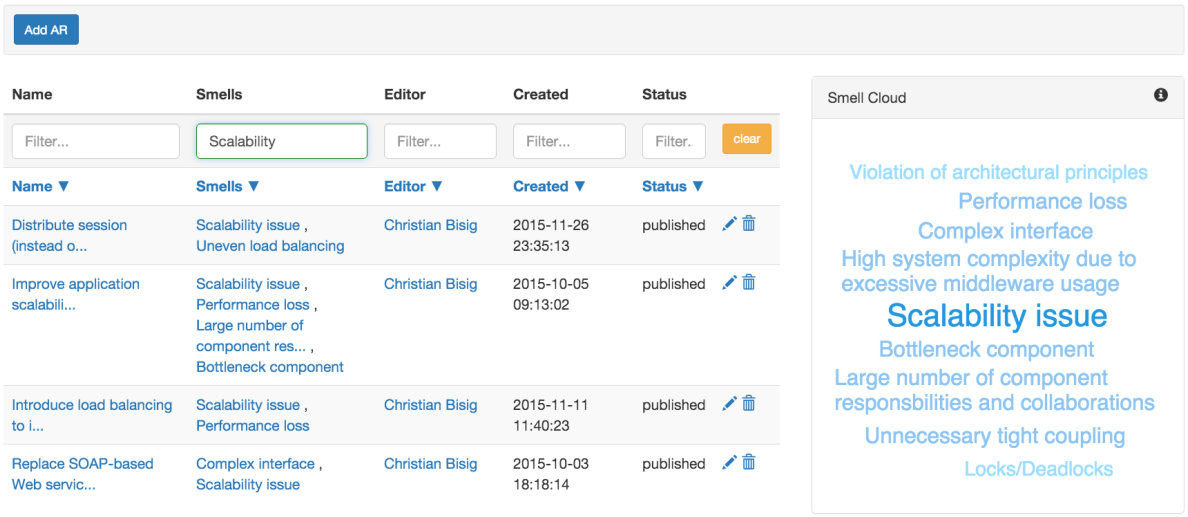


Abbildung 3.10: Screenshot AR Browser

- *Architectural Refactoring Form (Create)*
  Das Erfassungs-Formular für neue ARs bietet die Formularfelder um alle Attribute für ein AR zu setzen oder zu verlinken. Für die meisten Attribute (Context, Quality Attributes, Smells, Tasks etc.) bestehen Multiselektionslisten (Abbildung 3.11), um im AR-Werkzeug bereits vorhandene Werte auszuwählen und zu verlinken. Besteht ein gewünschter Wert noch nicht, kann direkt über einen Button ein Dialog zur Erfassung eines neuen Elements geöffnet werden, ohne das AR-Erfassungsformular zu verlassen.

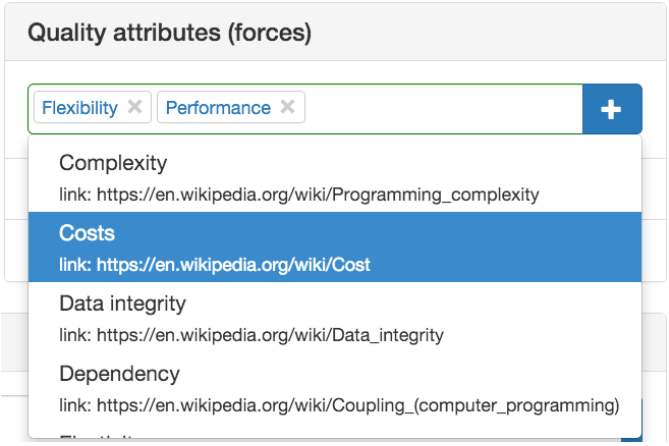


Abbildung 3.11: Screenshot Multiselektionsliste

- *Architectural Refactoring View*
  Die Detailansicht des ARs zeigt alle Versionen in Tabs an. Zudem besteht die Möglichkeit, das AR als PDF-Dokument zu exportieren. Ein AR-Editor kann in dieser Ansicht direkt in den Bearbeitungsmodus des ARs gelangen.

- *Architectural Refactoring Form (Update)*
  Der Bearbeitungsmodus besteht aus dem gleichen Formular wie das Erfassungsformular.

Smell Browser
Der Smell Browser zeigt eine tabellarische Auflistung aller im AR-Werkzeug vorhanden Smells (Abbildung 3.12). Wie im AR-Browser können nicht eingeloggte Benutzer oder AR-Applier nur die Smells mit Status “published” sehen. Mit einem Klick auf den Smell-Namen gelangt der Benutzer in die Detailansicht des Smells. Ein AR-Editor oder -Administrator kann in dieser Ansicht in den Bearbeitungsmodus des Smells gelangen.

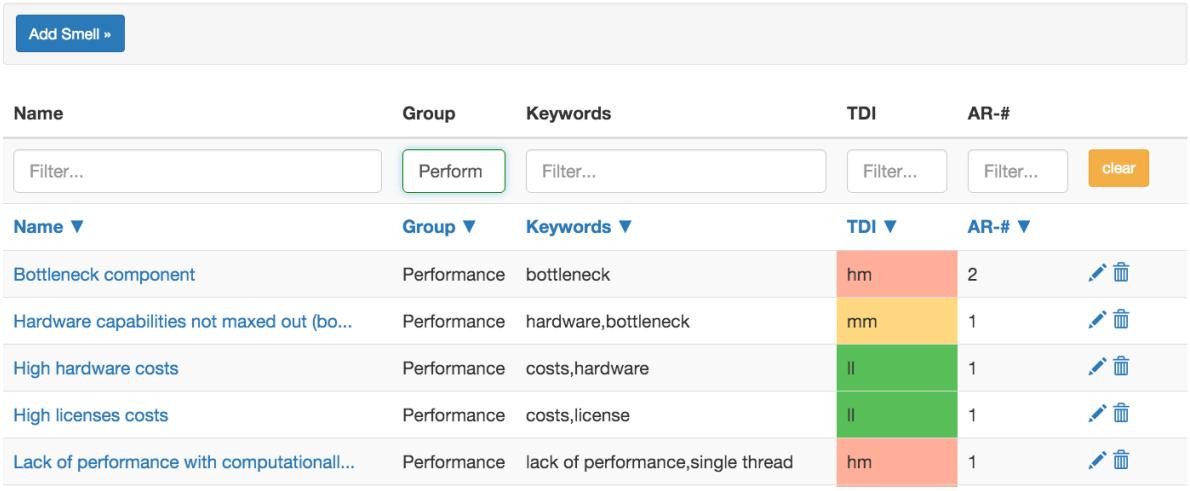


Abbildung 3.12: Screenshot Smell Browser

- *Smell View*
  In der Detailansicht des Smells kann dieser als PDF-Dokument exportiert werden. Ein AR-Editor oder -Administrator kann in den Bearbeitungsmodus des Smells gelangen.

- *Smell Dialog*
  Der Smell Dialog ist der Bearbeitungsmodus für den Smell und öffnet sich als Dialog Fenster.

Task Browser
Der Task Browser zeigt die tabellarische Auflistung der vorhandenen Execution Tasks. Ein AR-Editor oder -Administrator hat die Möglichkeit, in den Bearbeitungsmodus zu gelangen. Ein Klick auf den Namen des Tasks öffnet die Detailansicht.

- *Task View*
  In der Detailansicht des Tasks ist es möglich, diesen als PDF-Dokument zu exportieren. Ein AR-Editor oder -Administrator kann in den Bearbeitungsmodus des Tasks gelangen.

- *Task Dialog*
  Der Task Dialog zeigt das Formular für die Bearbeitung des gewählten Execution Tasks.

Smell Self-Assessment
Das Smell Self-Assessment ist ein Werkzeug, um eine geeignete Architekturumbaulösung für ein Problem in einem Softwaresystem zu finden (Abbildung 3.13). Es besteht aus einem Fragebogen, der alle Fragen, welche zu den verschiedenen Smells

erfasst sind, enthält. Die Fragen sind nach den Smell Gruppen geordnet. Es wird während dem Ausfüllen des Fragebogens zur Laufzeit angezeigt, welche ARs gefunden werden. Am Ende des Fragebogens gelangt der Benutzer mit einem Klick auf den Button “Show ARs” auf eine tabellarische Ansicht der gefundenen ARs. Nicht eingeloggte Benutzer können das Smell Self-Assessment ebenfalls verwenden. Diese und AR-Applier sehen aber nur Fragen von Smells, welche den Status “published”, haben und finden damit auch nur ARs mit dem selben Status.

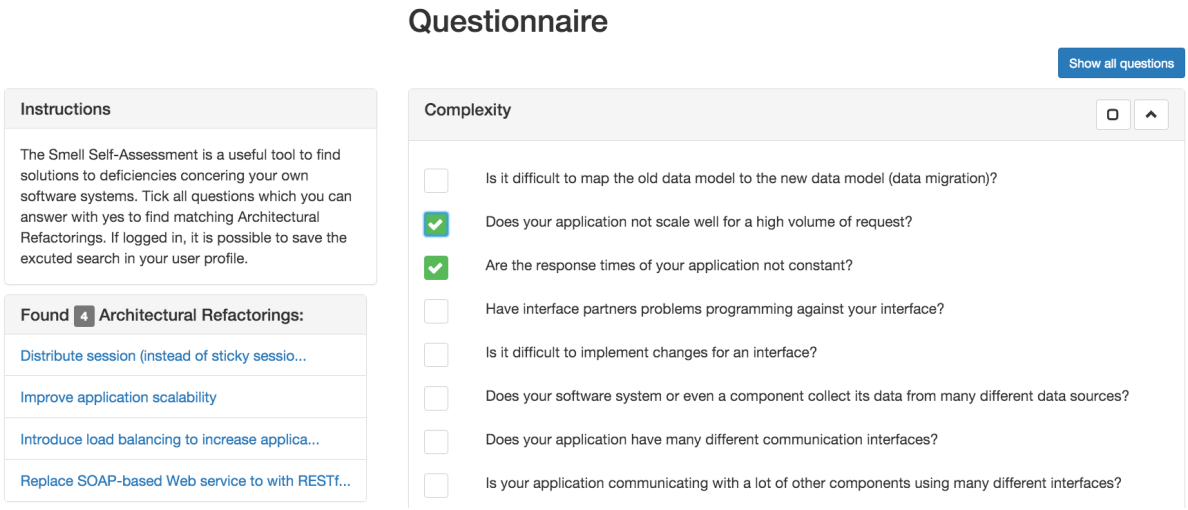


Abbildung 3.13: Screenshot Smell Self-Assessment

- *Architectural Refactorings Assessment Result*
  Zeigt eine tabellarische Übersicht der durch das Smell Self-Assessment gefundenen ARs. Die Abfrage, welche zu dieser Auflistung geführt hat, kann ein eingeloggter Benutzer, egal welcher Rolle, in seinem Profil unter einem beliebigen Namen speichern. Nicht eingeloggte Benutzer oder AR-Applier können in dieser Übersicht nur ARs mit dem Status “published” sehen.

- *Architectural Refactoring View*
  Aus der Resultats-Übersicht des Smell Self-Assessments kann der Benutzer in die Detailansicht eines ARs gelangen.

Admin

Der Link “Admin” enthält verschiedene Untermenüs, mit welchen ein AR-Editor oder -Administrator verschiedene sekundäre Datenelemente des AR-Werkzeugs bearbeiten kann. Für nicht eingeloggt Benutzer oder AR-Applier ist dieser Menüpunkt nicht sichtbar.

- *Execution Task Types Edit*
  Zeigt die hierarchische Auflistung der Execution Task Types. Es erlaubt neue Typen hinzuzufügen und bestehende zu bearbeiten oder zu ändern.

- *AR Meta Data*
  Unter dem Menüpunkt kann ein AR-Editor oder -Administrator alle Modell Element Links, welche in ARs verlinkt werden können, bearbeiten und neue hinzufügen.

- *Smell Groups*
  Unter dem Administrationsmenü Smell Groups kann ein AR-Editor oder -Administrator Gruppen für Smells hinzufügen oder verändern.

- *Properties*
  Unter dem Administrationsmenü Properties kann ein AR-Editor oder -Administrator verschiedene Eigenschaften des AR-Werkzeugs anpassen.

Help
Die Hilfeseite beinhaltet einen Download-Link für das Benutzerhandbuch zum AR-Werkzeug sowie eine Erklärung zum Konzept von Architectural Refactoring und seinen Objekten sowie Beispielen.

About
Auf der About-Seite sind Angaben zum Urheberrecht und der Software Lizenz, unter welcher das AR-Werkzeug veröffentlicht ist, zu finden.

User
Wenn ein Benutzer eingeloggt ist, zeigt das AR-Werkzeug in der linken oberen Ecke der Menüleiste statt des Login-Links, den Benutzername mit Benutzerbild an, unter welchem zwei Sub-Menüs vorhanden sind.

- *Edit User Profile*
  Seite, auf welcher ein Benutzer sein Profil bearbeiten kann.

- *Saved User Searches*
  Diese Seite zeigt eine Auflistung aller von einem Benutzer gespeicherten Suchabfragen aus dem Smell Self-Assessment. In dieser Übersicht kann der Benutzer die Suchabfragen erneut ausführen, ohne den Fragebogen nochmals ausfüllen zu müssen. Oder aber er kann nicht mehr gebrauchte Suchabfragen löschen. Diese Option steht allen registrierten Benutzern zur Verfügung.

4. Modellierungsprinzipien für Architectural Refactoring und Werkzeug

Dieses Kapitel erarbeitet ausführliche Modellierungsprinzipien für das Architectural Refactoring Werkzeug. Das Domain Modell wird durch die Modellierungsprinzipien erweitert, indem sie einen detaillierten Rahmen für den Inhalt der Architectural Refactorings und dessen Umfang bilden. Den AR-Editoren dienen die Modellierungsprinzipien als Leitfaden und Best Practices bei der Erfassung von ARs. Sie beschreiben die Attribute, welche in den Objekten des AR-Werkzeugs verwendet werden. Zuerst folgt eine technische Feld-Definition für die Attribute und anschliessend beschreibt das Kapitel formal, wie Wissensinhalte in die Objekte und deren Attribute zu erfassen sind. Die Modellierungsprinzipien sind massgebend für das Erfassen der Wissensinhalte (ARs) (Kapitel 5 auf Seite 34) und in der anschliessenden Wissens-Validierung (Kapitel 7 auf Seite 53). Obwohl die Modellierungsprinzipien vor der Validierung und den Benutzertests erstellt wurden, reflektieren sie Rückmeldungen, Erfahrungen und Verbesserungen, die aus diesen entstanden sind. Die Grössenangaben (Anzahl Wörter und Zeichen) basieren daher auf Erfahrungswerten, die sich während dem Arbeiten mit dem AR-Werkzeug bewährt haben. Die Grössen finden einen optimalen Kompromiss zwischen kurzer Länge und ausführlichem Informationsgehalt.

4.1. Abstrakte Attribut-/Feld-Typen

Im Folgenden sind die abstrakten Feldtypen aufgelistet, welche danach bei den konkreten Feldern und Attributen der AR-Werkzeug-Objekte Verwendung finden (z. B. in Webformularen). Der Name entspricht dem Erscheinungsbild im HTML-Webformular. Der Datentyp entspricht der Definition in der Datenbank (bzw. im Java Datenmodell). Die Beschreibung beinhaltet Angaben zur Länge, zum Format, zur Verwendungsform sowie zur formalen Erfassung des Inhalts.

Name	**Datentyp**
Textfield	varchar(255)
Wird für eine einzeilige Beschreibung, z. B. einen Namen oder einen Titel für ein Datenobjekt, verwendet. Die Standardlänge ist 255 Zeichen. Im Normalfall ist eine Länge von maximal 30-70 Zeichen einzuhalten (max. 10-15 Wörter).	
Dropdown	varchar(255)
Ist ein aus einer vorgegebenen Liste auszuwählender Textwert. Die Standardlänge in der Datenbank ist 255 Zeichen. Im Normalfall haben die Auswahlwerte eine Länge von maximal 20-25 Zeichen einzuhalten (max. 5 Wörter).	
Textarea (plain-Text)	longtext / varchar(5000)
Ist eine lange Textbeschreibung. Die Erfassung erfolgt in einem Standard HTML-Textarea Feld. Die Standardlänge in der Datenbank ist 5000 Zeichen. Im Normalfall ist eine maximale Textlänge von 1000 Zeichen einzuhalten (max. 200 Wörter).	
Textarea (rich-Text)	longtext / varchar(10000)
Ist eine lange Rich-Textbeschreibung. Die Erfassung erfolgt in einem WYSIWYG-Texteditor Feld, welches Formatieranweisungen in HTML-Kodierung umsetzt. Die Standardlänge in der Datenbank ist 10000 Zeichen. Im Normalfall ist eine maximale Textlänge von 2000-3000 Zeichen einzuhalten (max. 300-500 Wörter).	
Multiselectionlist (Mehrfachauswahlliste)	-
Ist eine Multiselektionsliste, die es erlaubt, mehrere Werte aus einer vorgegebenen Liste auszuwählen. In der Datenbank handelt es sich hierbei um die Herstellung von n:n Verbindungen. Im Normalfall sind maximal 5 Werte auszuwählen.	
Radio-Button (Mehrfachoptionsfeld)	-
Ist ein aus vorgegebenen Punkten auszuwählender Wert. Die Repräsentation in der Datenbank kann ein String- oder Zahlenwert sein. Im Normalfall wird im Web-GUI eine Wortbeschreibung für den tatsächlich in der Datenbank abgelegten Wert angezeigt. Die Beschreibung hat eine maximale Länge von 20-25 Zeichen (max. 5 Wörter).	
Checkbox (Auswahlkasten)	-
Ist eine Mehrfachauswahl aus vorgegebenen Punkten. Die Repräsentation in der Datenbank kann unterschiedlich sein (String- oder Zahlenwert). Im Normalfall wird im Web-GUI eine Wortbeschreibung für den tatsächlich in der Datenbank abgelegten Wert angezeigt. Die Beschreibung hat eine maximale Länge von 20-25 Zeichen (max. 5 Wörter).	
Datetime (Datum & Zeit)	datetime
Ist ein Datum mit Zeitangabe. Die Auswahl erfolgt mittels Texteingabe oder mittels eines Date-Time-Pickers (Einblendung eines Kalenders, in welchem Datum und Zeit auswählbar ist).	

Tabelle 4.1: Model Entitäten

4.2. Felddefinitionen ART

Dieser Abschnitt enthält die Definitionen der im AR-Werkzeug verwendeten Objekte und deren Attributen. Die Attribute sind alle von einem abstrakten Feldtyp (Abschnitt 4.1 auf Seite 26) abgeleitet. Die Beschreibungen der Attribute sind alle in Englisch gehalten und an einen Applikationsbenutzer adressiert. Dies ermöglicht es anschliessend, die Texte direkt in die AR-Werkzeug-Applikation zu übernehmen (z. B. als Benutzerhinweise in Form von Tool-Tipps im Erfassungs-Formular).

Es wird erwähnt, wie lange ein Name / eine Beschreibung sein soll, wie viele Elemente in einer Multiselektionsliste auszuwählen sind etc. Diese Angaben dienen zur Qualitätssicherung der Wissensinhalte und sollten von AR-Editoren eingehalten werden.

In den folgenden Tabellen ist ersichtlich, dass die meisten Felder zur Erfassung der AR-Objekte nicht zwingend sind. Grund dafür ist, dass ein Benutzer eine Erst-Erfassung und -Speicherung möglichst schnell und unkompliziert vornehmen kann und erst im Nachhinein die Details ausarbeitet.

4.2.1. Felddefinitionen für die AR-Erfassung/Editierung

Die Tabelle 4.2 beschreibt die Anforderungen an die Attribute eines ARs.

Feldname	**Feldtyp**	**Zwingend**
Name	**Textfield**	**Ja**
Maximum 50-60 characters which describe the AR consise. Start with a strong and meaningful verb like improve, convert, distribute, etc.		
Status	**Dropdown**	**Ja**
Select a status • *draft*: Work by the initial AR-Editor is still in progress or is in progress again after a review. • *in Review*: The initial AR-Editor is requesting a review for his work performed. • *accepted*: A second AR-Editor accepts the content of the Architectural Refactoring. • *published*: The originator of the Architectural Refactoring publishes the Architectural Refactoring after it has been accepted by another AR-Editor. • *deprecated*: Is a logical delete of the Architectural Refactoring. E.g. if it has been replaced by another AR (not ARVersion) or is no longer relevant. • *(unspecified)*: Only to be used for an unforeseen state of the Architectural Refactoring.		

Context	**Multiselectionlist**	**Nein**
Select/Create 2-3 elements. At least one viewpoint and a refinement level.		
Quality Attributes	**Multiselectionlist**	**Nein**
Select/Create 2-5 quality attributes, which are addressed by the Architectural Refactoring.		
Smells	**Multiselectionlist**	**Nein**
Select/Create 1-3 Smells. The Smells are the symptoms of the problem which the Architectural Refactoring is aimed at to solve.		
Description	**Textarea (rich-Text)**	**Nein**
Attach high importance to the description. It has to contain three sections in total around 2000 characters (ca. 300 words). • **Initial position** 2-4 pragraphs from which each should consist of 3-5 sentences. The first paragraph is an abstract description of the problem. The problem description has to address the before selected quality attributes and state what the bad influence of the problem is. A second paragraph contains a concrete example. • **Revised Design** 2-4 paragraphs with each 3-5 sentences. Explain the solution of the problem stated in the initial position. Describe the changes which have to be performed to implement the Architectural Refactoring and how it is ensured that the problem does not occure anymore. Use 1-2 illustrating images and explain them in the text. • **Pitfalls to avoid** Is a bullet list with 2-4 bullets. Each bullet addresses a difficulty in the revised design and identifies technical risks. It has to be written as a best practice rule or a recommended action which has to be taken. It has to be formulated in imperative. E.g. “Make sure that ...”		
Architectural decision(s)	**Multiselectionlist**	**Nein**
Select/Create 1-3 architectural decision(s) which might have been made formerly and now have to be revised during the implementation of the Architectural Refactoring.		
Affected components	**Multiselectionlist**	**Nein**
Select/Create 1-3 components which will have to be adapted with the implementation of the Architectural Refactorings.		

Execution tasks	**Multiselectionlist**	**Nein**
Select/Create 6-10 Execution Tasks. The task must not be generic. They have to be concrete and specific. Use at least three Execution Tasks of the Execution-Task-Type “Design” and at least one of the types “Documentation”, “Development”/“Integration Task”, “Test”.		
Reference (links)	**Multiselectionlist**	**Nein**
Select/Create 3-5 references. Can be a book, an article, a report or an internet reference where additional information for the Architectural Refactoring can be found.		

Tabelle 4.2: Model Entitäten AR

4.2.2. Felddefinitionen für die Smell-Erfassung/Editierung

Tabelle 4.3 beschreibt die Anforderungen an die Attribute eines Smells.

Feldname	**Feldtyp**	**Zwingend**
Name	**Textfield**	**Ja**
Maximum 50-60 characters which describe the Smell concise. Start by using an adjective or a noun. E.g. “High system complexity ..”		
Status	**Dropdown**	**Ja**
Select a status • *draft*: Work by the initial AR-Editor is still in progress or is in progress again after a review. • *in Review*: The initial AR-Editor is requesting a review for his work performed. • *accepted*: A second AR-Editor accepts the content of the Smell. • *published*: The originator of the Smell publishes the Smell after it has been accepted by another AR-Editor. • *deprecated*: Is a logical delete of the Smell. E.g. if it has been replaced by another Smell or is no longer relevant. • *(unspecified)*: Only to be used for an unforseen state of the Smell.		
Keywords	**Textfield**	**Nein**
Enter 2-3 keywords separated by commas with total maximum length of 60 characters.		
Group	**Dropdown**	**Ja**
Select one Smell group.		

Questions	**Textfield (mehrfach)**	**Nein**
Create 1-3 questions. Each question has to be answerable with yes or no. A question has to be factual and not rhetorical or a leading question. The questions should not be longer than 100 characters.		
Technical Debt Risk Index	**Radio-Button**	**Nein**
Select a Technical Debt Risk Index in the 3x3 matrix where the upper right value states a high probability of occurrence with a significant (high) impact and the lower left value states a low probability of occurence with a low impact.		
Description	**Textarea (rich-Text)**	**Nein**
At least one sentence which can be answered with yes or no. E.g. “It is the case ...”		

Tabelle 4.3: Model Entitäten Smell

Abbildung 4.1 zeigt die oben beschriebene Selektionsmatrix für den “Technical Debt Risk Index” wie diese im AR-Werkzeug implementiert ist. Die Spalten und Zeilen sind mit l: low, m: mid und h: high gekennzeichnet.

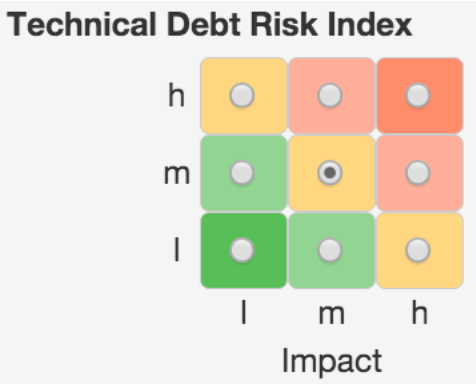


Abbildung 4.1: Technical Debt Risk Index Matrix

4.2.3. Felddefinitionen für die Task-Erfassung/Editierung

Tabelle 4.4 beschreibt die Anforderungen an die Attribute eines Ausführungsschrittes.

Feldname	**Feldtyp**	**Zwingend**
Name	**Textfield**	**Ja**
Maximum 100-150 characters which describe the task. The task must not be generic. It has to be concrete and specific. Start the name with a meaningful verb, e.g. “Organise”, “Implement”, “Change”, etc.		

Assignee	**Textfield**	**Nein**
Maximum 50 characters. Write the name of a project role to which the task has to be assigned. E.g. “Software architect”, “Project Manager”, “Test Manager”, etc.		
Type	**Dropdown**	**Nein**
Select an execution task type. Choose the most precise one in the list of hierarchically ordered types which means rather a child element than a parent.		
Description	**Textarea (rich-Text)**	**Nein**
Around 30-100 words. Give detailed information about the execution of the task, e.g. sub-steps or references to sources which give additional information.		
Priority	**Dropdown**	**Nein**
Select one of the given priorities low, medium, high to emphasize or de-emphasize the importance of a task execution.		
Due Date	**Datetime**	**Nein**
This is very project specific and optional to fill out.		
Estimated Duration	**Textfield**	**Nein**
A rough estimation of the time it will take to execute the task. Use either the unit man-hour (mh), man-day(md) or man-month(mm). E.g. 6mm		

Tabelle 4.4: Model Entitäten Execution Task

4.2.4. Felddefinitionen für den ExecutionTaskType-Tree

Tabelle 4.5 beschreibt die Anforderungen an die Attribute eines Ausführungsschritt-Types.

Feldname	**Feldtyp**	**Zwingend**
Name	**Textfield**	**Ja**
Has to consist out of 3 words maximum, from which the last word has to be “Task”.		
Description	**Textarea (plain-Text)**	**Nein**
Maximum 100 characters. Give a short explanation or an example.		

Tabelle 4.5: Model Entitäten Execution Task Type

4.2.5. Felddefinitionen für die AR-Meta Data Erfassung

Tabelle 4.6 beschreibt die Anforderungen an die Attribute eines Meta-Daten Elements. Es gibt fünf verschiedene Arten von AR-Meta Daten bzw. Modell Element Links (Abschnitt 3.1.4 auf Seite 11).

Feldname	**Feldtyp**	**Zwingend**
Name	**Textfield**	**Ja**
Choose a short title maximum 50-60 characters depending on the type of Meta-Data Element. If it is a book title give the name and in square brackets the names of the authors.		
Reference (URI)	**Textfield**	**Nein**
Has to be an Uniform Resource Identifier (URI) which leads to additional information or the broached ressource itself.		

Tabelle 4.6: Model Entitäten AR-Meta Data

4.2.6. Felddefinitionen für die Smell Group Erfassung

Tabelle 4.6 beschreibt die Anforderungen an die Attribute einer Smell Gruppe.

Feldname	**Feldtyp**	**Zwingend**
Name	**Textfield**	**Ja**
One word with a maximum of 20-30 characters. Only add main quality attributes under which Smells can be categorized.		
Description	**Textarea (plain-Text)**	**Nein**
Short description of maximum 50-60 characters or a link to an explaining source (e.g. a link to a Wikipedia page).		

Tabelle 4.7: Model Entitäten Smell Group

5. Architectural Refactoring Modellierung

Dieses Kapitel beschreibt das Vorgehen bei der Wissensschaffung für das Architectural Refactoring Werkzeug und welche Wissens-Elemente erarbeitet und zusammengestellt wurden. Die Wissensschaffung ist ein wichtiger Teil der Masterarbeit und bildet die Grundlage für die Prüfung des Konzepts und des Werkzeugs. Das initiale Wissen fördert zudem das Weiterbestehen des Konzepts und des Werkzeugs.

Insgesamt entstanden 20 Architectural Refactorings (ARs) mit einer durchschnittlichen Beschreibungslänge von 2245 Zeichen (349 Wörtern). Die minimale Länge einer Beschreibung beträgt 1966 Zeichen (301 Wörter) und die maximale Länge beträgt 2662 Zeichen (417 Wörter). Diese ARs verwenden insgesamt 33 Smells, 101 Tasks und verlinken 145 Modell Elemente (Kontexte, Qualitätsattribute, Entscheidungen, Komponenten und Referenzen).

5.1. Vorgehen

Für den Teil der AR-Modellierung nimmt der Autor dieser Arbeit die Rolle eines AR-Editors ein. Der AR-Editor ist ein Architekturexperte, welcher eine wissensschaffende Funktion einnimmt. Er ist eine Person mit ausgeprägtem Wissen im Bereich der Softwarearchitektur, der im Unternehmens- oder Forschungsumfeld arbeitet. Ziel ist dabei, das AR-Werkzeug mit initialem Architekturumbauwissen aus der Anwendungs- und Integrationsarchitektur zu befüllen. Das Wissen befolgt die in Kapitel 4 auf Seite 26 definierten Modellierungsprinzipien. Bei der Befüllung des AR-Werkzeugs wurden nicht nur die ARs erfasst, sondern auch alle referenzierbaren Objekte (Smells, Tasks und Metadaten-Felder) erarbeitet und somit das AR-Werkzeug in einen initialen Zustand versetzt. Das Hauptaugenmerk liegt auf den Beschreibungen der ARs. Die Smell- und Task-Beschreibungen hingegen sind kurz gehalten. Detailbeschreibungen der Smells sind für den Projektverlauf niedriger priorisiert und jene der Ausführungsschritte sind zukünftig für ein Task-Management-Werkzeug vorgesehen.

Um die Rolle des AR-Editors einnehmen, entsprechende ARs finden und dokumentieren zu können, greift der Autor auf Fachliteratur und sein eigenes Wissen zurück und befolgt Beschreibungen von Applikationseigenschaften wie dem IDEAL Prinzip [Feh+14].

Das IDEAL Prinzip beschreibt die folgenden fünf Eigenschaften, welche hauptsächlich von Cloud-Applikationen gefordert sind:

- **I**solated state
  Isolated state bemüht sich darum, eine Applikation möglichst ohne Status zu halten (stateless). Mit Ausnahme vom Status der Interaktionen eines Benutzers (session-state) sowie den Daten, welche eine Webapplikation handhabt (application-state).

- **D**istribution
  Applikationen sind in verschiedene Komponenten zu unterteilen, was den Vorteil bringt, dass diese einfacher auf die vorhanden Ressourcen eines Softwaresystems verteilbar ist.

- **E**lasticity
  Beschreibt das dynamische Skalieren einer (Cloud-)Applikation. Dies kann man mit horizontalem skalieren (scale out) erreichen, was soviel bedeutet wie die Leistung zu steigern durch Hinzufügen von Ressourcen in einen Verbund (z. B. einen neuen Server in einen bestehenden Cluster aufzunehmen) anstatt dem vertikalen skalieren, was bedeutet, einen bestehenden Server durch einen leistungsstärkeren zu ersetzen (scale up). Die Applikation muss dazu unabhängig auf verschiedenen Ressourcen (Servern) instantiierbar sein.

- **A**utomated management
  Automated Management ist die Fähigkeit eines Systems, automatisch zur Laufzeit die Ressourcen für eine Applikation umzuverteilen. Z. B. im Falle, dass eine Applikation mehr Ressourcen benötigt. Auf Grund erhöhter Last, wird die Applikation automatisch auf einer neuen Ressource instantiiert und anschliessend bei Nichtgebrauch wieder entfernt. Die Verrechnung der Ressourcen-Verwendung wird ebenfalls dynamisch vorgenommen. Dies ist eine Cloud-spezifische Eigenschaft.

- **L**oose coupling
  Lose Kopplung beschreibt die Eigenschaft, möglichst wenige Abhängigkeiten zwischen verschiedenen Komponenten zu schaffen. Dies bringt den Vorteil, einzelne Teile eines Systems leichter ersetzen, verteilen oder sogar entfernen zu können.

5.2. Erfassen von Metadaten im AR-Werkzeug

Das Erfassen eines Grundstocks an Metadaten-Feldern (z. B. Context, Quality attributes etc.) erfolgte im gleichen Schritt wie das Kreieren der ARs. Die Modellierungsprinzipien dienten dabei als Leitfaden. Die Attribute “Context” und “Quality attributes” verfügen jeweils über einen Link, der zu einer Begriffserklärung oder vertieftem Wissen führt. So sind folgende Werte zustande gekommen:

5.2.1. Context (viewpoint, refinement level)

Enthält die Werte der “4+1 Viewpoints” von Kruchten [Kru95], Logical-, Process-, Development-, Physical-View und Scenarios. Die sieben Viewpoints von Rozanski und Woods [RW13], Context-, Functional-, Information-, Concurrency-, Development-, Deployment- und Operational-View, sowie die Architectural Decision Levels [Bab+09] Executive-, Conceptual-, Technology- und Vendor-Asset-Level.

5.2.2. Quality attributes (forces)

Durch die Erfassung der ARs sind sukzessive die folgenden 19 Einträge von Qualitätsattributen im AR-Werkzeug entstanden. Die Qualitätsattribute orientieren sich an dem ISO-Standard für “Systems and software Quality Requirements” [Sta11]: Complexity, Costs, Data integrity, Dependency, Elasticity, Flexibility, Functionality, Maintainability, Performance, Portability, Probability, Productivity, Responsibility, Robustness, Scalability, Security, Simplicity, Standardization und Testability.

5.2.3. Architectural decision(s) to be revisited

Für das Attribut “zu überdenkende Architekturentscheidungen” sind 31 Werte erfasst. Die Attribute bestehen aus Entscheidungen zu Komponenten, Protokollen, Frameworks oder Architekturmustern. Beispielwerte aus dieser Liste sind: “Approach to threading (concurrency management)”, “Choice of application frameworks”, “Choice of Protocol” oder “Choice of tier architecture”.

5.2.4. Affected components and connectors (if modelled explicitly)

Die Liste der betroffenen Komponenten ist während dem Erfassen der ARs fortlaufend gewachsen und resultierte in den folgenden zehn Einträgen: Application software (Business software), Hardware (server), Hardware load balancer, Interfaces, Middleware, Operating system, Other applications, Quality assurance of service, Scheduler und Unit of work.

5.2.5. References (links)

Die Bearbeitung und das Erstellen der ARs hat über 60 Referenzen hervorgebracht. Die Attribute enthalten Verweise auf einschlägige Softwarearchitektur-Bücher, Artikel und Webseiten verschiedener Experten. Ebenfalls sind Links zu den im AR erwähnten Applikationen und Frameworks vorhanden oder Links auf Webressourcen, welche Begriffe erklären oder den Einstieg zu weiterführendem Wissen bilden.

5.3. Architecural Refactorings

Dieser Abschnitt enthält die Auflistung der erarbeiten Architectural Refactorings und eine deutsche Kurzfassung des Inhalts. Die Komplettfassungen sind im Anhang (Abschnitt A.1 auf Seite 79) in englischer Sprache und in Form und Layout wie die PDF-Generierungs-Funktion des AR-Werkzeugs diese exportiert hat. Einige der erfassten ARs basieren auf Erfahrungen aus meiner beruflichen Tätigkeit als Softwareentwickler vor dem Masterstudium, andere stützen sich auf theoretische Wissensaneignung durch Fachliteratur und Artikel. Es handelt sich bei den erarbeiteten ARs um initiales Wissen ohne Anspruch auf Vollständigkeit oder allgemeine Gültigkeit.

Die 20 Architectural Refactorings sind nach den adressierten Haupt-Qualitätsattributen und deren Häufigkeit sortiert:

1. Scalability: 8 ARs
2. Maintainability: 3 ARs
3. Performance: 3 ARs
4. Complexity: 2 ARs
5. Dependency: 2 ARs
6. Flexibility: 1 ARs
7. Functionality: 1 ARs

Jedes AR ist in den folgenden Kurzbeschreibungen mit einem persönlichen Erfahrungswert versehen. Der Erfahrungswert bezeichnet die Wissensgrundlage, auf welcher das AR basiert. Die Erfahrungswerte sind mit ...

... *theoretisch*: basierend auf Wissensaneignung mittels Literatur

... *praktisch*: Erfahrungen aus beruflicher Tätigkeit oder praktischer Umsetzung

... *theoretisch-praktisch*: Erfahrungen aus beruflicher Tätigkeit sowie erweiterter Wissensaneignung durch Literatur

gekennzeichnet.

5.3.1. Hinzufügen einer Hardware-Ebene, um die Skalierbarkeit zu erhöhen

Englischer Titel: “Add tier to improve independent scalability”
Qualitätsattribut: Scalability
Erfahrungswert: theoretisch

Mittels dieses ARs erreicht man die höhere Skalierbarkeit eines Systems, in dem eine Komponente des Systems auf einen separaten Tier (Server) auslagert wird. Der Vorteil dabei ist, dass Komponenten unabhängig voneinander skalierbar sind, z. B. das Separieren der Business Logik und der Datenhaltung.

5.3.2. Eine Single-Thread-Applikation Multithreading-fähig machen, um die Skalierbarkeit zu erhöhen

Englischer Titel: "Convert application from single- to multi-threading to improve scalability"
Qualitätsattribut: Scalability
Erfahrungswert: praktisch

Das AR beschreibt das Vorgehen, um die Hardware-Ressourcen eines Systems besser auszunutzen, indem man Arbeitsschritte aufteilt und diese anschliessend parallel verarbeitet. Dabei ist zu beachten, dass die Grösse der Arbeitsschritte sowie die Anzahl gleichzeitig verarbeitender Prozesse optimal auf das System abgestimmt sind.

5.3.3. Verteilte Session statt Sticky-Session verwenden, um die Skalierbarkeit zu erhöhen

Englischer Titel: "Distribute session (instead of sticky sessions) to increase scalability"
Qualitätsattribut: Scalability
Erfahrungswert: theoretisch

Vor allem bei skalierbaren Webapplikationen, welche auf verschiedene Systeme (Server-Cluster, Cloud) verteilt sind, ist es wichtig, dass der Benutzerstatus nicht fix auf einem Server hinterlegt ist. Alle Server des verteilten Systems müssen auf den Benutzerstatus Zugriff haben, damit eine Benutzersitzung von einem Load-Balancer während der laufenden Sitzung auf einen anderen Server verschoben werden kann. Dies ermöglicht eine dynamische und dauerhaft ausgeglichene Lastverteilung. Dazu ist es jedoch nötig, den Status der Benutzersitzung für alle Server zugreifbar zu machen (auf einem separaten System).

5.3.4. Verwenden von Caching, um die Skalierbarkeit eines Systems zu erhöhen

Englischer Titel: "Introduce caching to improve application scalability"
Qualitätsattribut: Scalability
Erfahrungswert: theoretisch

Dieses AR enthält Vorschläge zur Optimierung der Skalierbarkeit eines Systems, indem Caching-Mechanismen eingesetzt werden, um die Anzahl der I/O-Operationen auf langsamen Datenspeichern zu reduzieren. Der Datenspeicher soll nur kontaktiert werden, sofern die abgefragten Daten sich geändert haben. Dazu empfiehlt das AR einen Proxy-Server zu verwenden, welcher ganze Anfrageresultate zwischenspeichert. Das AR beinhaltet zu beachtende Hinweise in Bezug auf die Skalierung und die Lastverteilung.

5.3.5. Verwenden von Load Balancing, um die Kapazität und Skalierbarkeit einer Applikation zu erhöhen

Englischer Titel: "Introduce load balancing to increase application capacity and scalability"
Qualitätsattribut: Scalability
Erfahrungswert: theoretisch-praktisch

Für verteilte Systeme ist eine Lastverteilung eine essentielle Funktion. Vor allem in Cloud-Systemen spricht man vom sogenannten "Elastic Load Balancing", welches nebst der Benutzer-Anfrageverteilung auch die automatische Erhöhung der Applikationskapazität beinhaltet. Die Applikation wird automatisch auf weiteren Systemen instantiiert, um mehr Anfragen bewältigen zu können. Für diesen Mechanismus müssen verschiedene Bedingungen erfüllt sein. Hingegen muss in einem klassischen Load-Balancing-Setup die Erhöhung der Kapazität in einem Server-Cluster manuell vorgenommen werden. Das AR geht auf die verschiedenen Methoden ein und erläutert Punkte, die zu beachten sind.

5.3.6. Umstellen von client-basierter auf datenbank-basierte Session, um Skalierbarkeit zu gewährleisten

Englischer Titel: "Migrate from client to database session state to provide scalability"
Qualitätsattribut: Scalability
Erfahrungswert: theoretisch-praktisch

Das AR beschreibt den Vorgang, einen Benutzerstatus statt beim Benutzer im Browser zu speichern, auf der Serverseite in einer Datenbank abzulegen. Eine client-basierte Session hat nur einen begrenzten Speicherplatz und es ist nötig, den Status des Benutzers ständig zwischen der Serverapplikation und dem Browser des Benutzer hin und her zu transferieren. Der Vorteil einer server-seitigen Session hingegen ist, dass diese nach einer verlorenen Verbindung oder einem Absturz noch vorhanden ist.

5.3.7. Eine Web-Applikation in eine Cloud-Umgebung migrieren, um Skalierbarkeit zu gewährleisten

Englischer Titel: "Migrate web-application into a cloud-environment to provide scalability"
Qualitätsattribut: Scalability
Erfahrungswert: theoretisch-praktisch

Das AR beschreibt die Migration einer Webapplikation auf eine Cloud Plattform als Massnahme, um die Skalier- und Verfügbarkeit zu steigern. Als Beispiel dient eine Java-Play-Applikation und die Cloud-Umgebung Heroku. Das Play-Framework bietet

die Möglichkeit, einfache und leichte Webserverprozesse zu erstellen und zu betreiben. Jedoch verlangen diese bei einem Ressourcen-Engpass ein manuelles Setup von Instanzen der Applikation, was weitere Komponenten wie einen Webserver-Proxy und Load-Balancer benötigt. Es gibt bereits viele Cloud-Umgebungen im Internet, welche diese Funktionalitäten und die Skalierbarkeit durch Konfiguration bereitstellen. Heroku ist eine Cloud-Plattform, welche “out-of-the-box” Play-Applikationen unterstützt (verfügt über eine Scala-Build-Umgebung). Das AR beschreibt die nötigen Schritte für die Cloud-Migration.

Dieses AR ist in Abschnitt A.2 auf Seite 120 als Beispiel und Anwendungstest anhand der AR-Werkzeug-Play-Applikation umgesetzt.

5.3.8. Verteilen von Daten, um die Skalierbarkeit zu erhöhen

Englischer Titel: “Scatter data to increase scalability”
Qualitätsattribut: Scalability
Erfahrungswert: theoretisch

Das AR empfiehlt die Verteilung von grossen Datenmengen auf verschiedene Knoten/Server. Über einen Hauptknoten sollen danach die Datenabfragen ausgeführt werden. Der Hauptknoten delegiert die Abfragen an die einzelnen Datenknoten und aggregiert anschliessend die Resultate. Dies folgt dem Architektur Prinzip “scatter and gather” [HW03]. Durch dieses Verfahren können grosse Datenmengen parallel analysiert oder bearbeitet werden. Das AR beschreibt, was zu beachten ist, und nennt ein konkretes Beispiel.

5.3.9. Verwenden von Dependency Injection, um die Wart- und Testbarkeit zu erhöhen

Englischer Titel: “Introduce dependency injection to improve maintainability and testability”
Qualitätsattribut: Maintainability
Erfahrungswert: praktisch

Softwaresysteme und Applikationen sind bisweilen schwer testbar, sofern nicht bereits bei der initialen Entwicklung auf einen sauberen und Test-unterstützenden Quellcode geachtet wird. Z. B. Wird das Testen durch unflexible Schnittstellenanbindungen, insbesondere in einem isolierten Umfeld (Entwicklungssystem), in welchem nicht alle Schnittstellenapplikationen verfügbar sind, erschwert. Die Simulation von Schnittstellenapplikationen ist aber gerade für erste Modultests wichtig. Dieses AR empfiehlt und erklärt die Verwendung eines Dependency-Injection-Frameworks, welches es ermöglicht, das Verhalten einer Applikation zur Laufzeit zu beeinflussen. Dies z. B. mittels Objekten, welche Schnittstellen simulieren oder ersetzen. Diese Möglichkeit

vereinfacht zudem das Verändern der Applikation für neue Anforderungen und erhöht die Flexibilität.

5.3.10. Teilen von Komponenten, um enge Bindungen zu reduzieren

Englischer Titel: "Split components to reduce overly tight coupling"
Qualitätsattribut: Maintainability
Erfahrungswert: theoretisch-praktisch

Das AR rät, Funktionen in einer Komponente, welche nur begrenzt einen Zusammenhang haben, in separate Komponenten aufzuteilen. Eine unnötig enge Kopplung erschwert die Wart- und Veränderbarkeit. Dies widerspricht dem Grundbrinzip "high cohesion, low coupling", welches besagt, dass ein Programm nur Funktionen mit hohem Zusammenhang enthält und die Bindung unter Komponenten möglichst lose ist. Das AR erklärt die Schritte, um eine Aufteilung in mehrere Komponenten vorzunehmen und empfiehlt als Methode das Verwenden von CRC-Karten.

5.3.11. Aufrüsten von Dritthersteller-Software, um die Wartbarkeit zu wahren

Englischer Titel: "Upgrade of third-party software to preserve maintainability"
Qualitätsattribut: Maintainability
Erfahrungswert: praktisch

Das AR empfiehlt regelmässige Version-Upgrades von verwendeten Dritthersteller-Produkten, um die Wartbarkeit des Softwaresystems zu wahren. Support für spezifische Versionen einer Dritthersteller-Software ist oft zeitlich limitiert. Das heisst, der Hersteller liefert Fehlerkorrekturen nur für einen begrenzten Zeitraum. Versäumt man es daher, regelmässig die Version zu erneuern, entstehen grosse Risiken für ein Softwaresystem. Das AR bespricht Punkte, welche bei einer Migration zu einer neuen Version zu beachten sind und erklärt die nötigen Schritte.

5.3.12. Verwenden von Column-Stores, um die Leistung der analytischen Datenverarbeitung zu verbessern

Englischer Titel: "Introduce column-store to improve the performance of analytical data processing"
Qualitätsattribut: Performance
Erfahrungswert: theoretisch-praktisch

Business Prozesse automatisieren ist das Ziel jedes Softwaresystems. Diese Automatisierung basiert in der Regel auf transaktionalen Verarbeitungen, in welchen Da-

ten selektiert, erstellt, verändert oder gelöscht werden. Z. B. ein Kundeverwaltungssystem. Geht es aber darum, die automatisierten Prozess zu verbessern und mit den richtigen Daten, zum richtigen Zeitpunkt auszuführen, ist dazu eine analytische Datenverarbeitung nötig. Mit dieser Analyse soll die nötige "Business Intelligence" errungen werden, um die Prozesse zu verbessern. Dieses AR empfiehlt sogenannte spalten-basierte Datenbanktabellen zu verwenden, welche für die Zwecke der analytischen Datenverarbeitung eine deutlich bessere Performance bringen. Aggregationsabfragen und Daten-Scans, welche im analytischen Processing eine wesentliche Rolle spielen, werden durch die Datenpartitionierung der spalten-basierten Tabellen begünstigt.

5.3.13. Verwenden einer in-Memory-Datenbank, um die Leistung für hohe Transaktionsvolumen zu verbessern

Englischer Titel: "Introduce in-memory database to improve high volume transaction performance"
Qualitätsattribut: Performance
Erfahrungswert: theoretisch

Systeme, welche eine grosse Anzahl von Transaktionen (Insert/Update/Deletes) verarbeiten müssen, sind oft durch festplatten-basierten Speicher eingeschränkt. Neue, moderne in-Memory-Datenbanksysteme können eine Abhilfe sein. Diese unterliegen nicht den gleichen physikalischen Grenzen wie die festplatten-basierten Speicher. Das AR beschreibt die Vor- und Nachteile einer in-Memory-Datenbank und die Schritte zur Migration.

5.3.14. Von einer SPARC basierten auf eine x86 basierte Serverarchitektur wechseln, um die Leistung zu erhöhen

Englischer Titel: "Switch server architecture from SPARC to x86 to optimize the performance"
Qualitätsattribut: Performance
Erfahrungswert: praktisch

SPARC und x86 sind zwei verschiedene Prozessorarchitekturen und nicht gleichermassen für jede Anwendung zweckmässig. Ein SPARC-System ist ein hoch-skalierbares System, welches ein hohes Mass an Concurrency erreichen kann. Ein x86-System bietet oft nicht die gleiche Parallelität, dafür höhere Taktfrequenzen und schnellere Verarbeitung pro Prozess. Vor allem für verteilte Systeme sind Linux-basierte x86 Server auf Grund ihres verhältnismässig günstigen Preises (commodity hardware) beliebt. Durch die Anordnung in Server-Clustern oder Clouds entstehen leistungsfähige Systeme. Das AR erklärt, wann ein Umstieg von SPARC-Systemen sinnvoll ist und was zu beachten ist.

5.3.15. Batch-Applikation aus einem Applikationsserver extrahieren, um die Komplexität zu reduzieren

Englischer Titel: “Extract batch application from application server to reduce complexity”
Qualitätsattribut: Complexity
Erfahrungswert: praktisch

Oft verwendet ein System unnötigerweise einen Applikationsserver. Das heisst, die Funktionsvielfalt eines Applikationsservers wird nur in geringem Masse verwendet. Dies verursacht unnötige Komplexität und Abhängigkeit. Das AR zeigt Alternativen zu einem Applikationsserver und listet die Schritte auf, um eine Extraktion durchzuführen.

5.3.16. Ersetzen eines SOAP-basierten Webservices durch eine RESTFul HTTP Ressource, um die Komplexität zu reduzieren

Englischer Titel: “Replace SOAP-based Web service with RESTful HTTP resources to reduce complexity”
Qualitätsattribut: Complexity
Erfahrungswert: theoretisch

Simple Object Access Protocol (SOAP)-basierte Webservices bieten einen hohen Standard und Stabilität. Deren Komplexität durch Beschreibungssprachen, Protokoll und Schemas sind aber oft nicht zweckmässig. Eine Alternative dazu sind die immer populärer werdenden REST-Schnittstellen. Diese verwenden Standard HTTP als Protokoll und die vier Operationen Create Read Update Delete (CRUD) zum Beziehen und Manipulieren von Objekten. Das AR beschreibt die Unterschiede und zu beachtenden Punkte beim Umstieg von SOAP- zu REST-Servicen.

5.3.17. Zusammenführen und Vereinfachen von Dritthersteller-Produkten, um Abhängigkeiten zu reduzieren

Englischer Titel: “Consolidate and simplify product usage to reduce dependencies”
Qualitätsattribut: Dependency
Erfahrungswert: theoretisch-praktisch

Dieses AR spricht die Themen von falscher Tool-Evaluation und falscher Softwaresystem-Evolution an. Dies kann zu ungewollten Abhängigkeiten und Komplexität in einem System führen. Ein Dritthersteller-Produkt aus einem bestehenden System zu lösen ist schwierig. Das AR empfiehlt daher, eine klare Strategie für die Evolution eines Softwaresystems zu entwickeln und die Werkzeugauswahl anforderungs-basiert und mit klarem Vorgehen vorzunehmen.

5.3.18. Zentralisieren von Datenzugriffen, um Abhängigkeiten zu reduzieren

Englischer Titel: “Consolidate data access to reduce dependencies”
Qualitätsattribut: Dependency
Erfahrungswert: theoretisch

Das AR beschreibt, wie verschiedene Datenzugriff-Implementierungen und Datenquellen in eine eigene Komponente auszulagern sind. Es wird eine neue Komponente geschaffen, welche den alleinigen Zugriff auf eine gemeinsame Datenquelle implementiert. Die neue Komponente bietet danach anderen Komponenten über eine Standardschnittstelle Zugang zu den Daten.

5.3.19. Zentralisieren der Nachrichtenvermittlung und -transformation, um die Flexibilität zu erhöhen

Englischer Titel: “Centralize message routing and transformation to increase flexibility”
Qualitätsattribut: Flexibility
Erfahrungswert: theoretisch-praktisch

Das AR behandelt die Platzierung einer zentralen Messaging-Komponente. Anstelle von individuellen Schnittstellen zwischen Komponenten sollen diese über eine standardisierte Schnittstelle Informationen austauschen (z. B. Websphere MQ). Die Implementierung des ARs fördert eine lose Kopplung zwischen den Komponenten. Komponenten können einfacher ersetzt, verändert oder entfernt werden. Die Art der Kommunikation (z. B. synchron oder asynchron) kann konfigurativ geändert werden. Zudem übernimmt die zentrale Komponente die Transformation (z. B. Encoding) der Meldungen.

5.3.20. Einführen einer Online-Dokumenten-Formatier-Engine, um eine Druckfunktionalität zur Verfügung zu stellen

Englischer Titel: “Introduce online document formatting engine to provide printing functionality”
Qualitätsattribut: Functionality
Erfahrungswert: praktisch

In Webapplikationen werden oft Informationen dargestellt, welche einem Benutzer als PDF-Dokument oder Ausdruck von Nutzen sind. Das AR beschreibt das Vorgehen, wie mittels einer XML-XSL-Transformation und anschliessendem XSL-FO-Rendering in Echtzeit und mit aktuellen Daten ein PDF-Dokument generiert werden kann.

6. Validierungkriterien AR-Werkzeug und AR-Wissen

Dieses Kapitel enthält die Definition für die Konzept-, Wissens- und Werkzeug-Validierungen, welche im Kapitel 7 auf Seite 53 ausgeführt ist. Die Wissens-Validierung beschreibt, wie die Bewertung der Wissensinhalte (Architectural Refactorings) mittels den erarbeiteten Modellierungsprinzipien (siehe Kapitel 4 auf Seite 26) vorzunehmen ist. Die Konzept-Validierung definiert einen Kriterienkatalog basierend auf Anforderungen zur Beurteilung von Forschungsergebnissen [Zim09]. Die Werkzeug-Validierung besteht aus einem Anwendungstest (Funktionalität und die Benutzbarkeit) mit zwei Szenarien, zu welchem ein Fragenkatalog nach ISO-Norm 9421 “Ergonomics of human-system interaction” [Sta06] definiert ist.

6.1. Konzept-Validierung: Kriterien für das Architectural Refactoring Konzept

Wie bereits in der Einleitung erwähnt ist das Konzept von Architectural Refactoring ein Architektur-Designverfahren, eine Methode zur Architektur-Wissensverwaltung sowie im Ansatz ein Entwicklungs- bzw. Umsetzungsverfahren. Wobei das Architectural Refactoring Werkzeug die entsprechende Werkzeugunterstützung bietet.

In diesem Abschnitt sind die Validierungskriterien, basierend auf Anforderungen für die Beurteilung von Forschungsergebnissen [Zim09], definiert. Für die Kriterien ist in dieser Arbeit eine Bewertungsskala erstellt worden, welche die Einstufung mittels Sternen (★) vornimmt. Die effektive Bewertung des Designkonzepts wird in Abschnitt 7.1 auf Seite 53 ausgeführt. Wobei pro Kriterium 1-3 Sterne zu vergeben sind.

Die Bedeutung der Sterne ist wie folgt:

★ : Anforderung schlecht oder gar nicht erfüllt.

★★ : Anforderung gut erfüllt.

★★★ : Anforderung sehr gut erfüllt.

Die folgenden Auflistungen beschreiben für jede Anforderung die Kriterien, welche für das Erreichen der drei Sterne zu erfüllen sind bzw. wann nur ein Stern erteilt wird.

6.1.1. Anforderungen für Softwarearchitektur-Designverfahren

Die Anforderungen für Softwarearchitektur-Designverfahren zielen auf verschiedene Rollen, welche am Software-Entwicklungsprozess teilnehmen, sowie auf den ganzen Prozess. Das Architekturdesign befasst sich hauptsächlich mit der Rolle des Softwarearchitekten sowie der Phase Design im Entwicklungsprozess. Die Anforderungen werden in Kapitel 7 auf Seite 53 verwendet, um das Umsetzen von ARs mit Hilfe des AR-Werkzeugs als Softwarearchitektur-Designverfahren zu bewerten.

D-1 Architektur Viewpoints	★	★★★
Verfeinerung von Allzweck-Methoden: Mehrere Architektur-Viewpoints bereitstellen, um komplexe Designprobleme zu bearbeiten und eine Arbeitsteilung vornehmen zu können.	Einteilung in Viewpoints ist nur schlecht möglich.	Viewpoints und andere Verfeinerungen können einfach zugeordnet werden.
D-2 Qualitätsattribut getrieben	★	★★★
Getrieben durch Qualitätsattribute und Stakeholder-Ziele. Qualitätsattribute sind der Schlüssel zum Erfolg, aber oft miteinander konkurrierend.	Qualitätsattribute sind schlecht zuweisbar.	Qualitätsattribute sind Hauptobjekte des Verfahrens und einfach zuweisbar.
D-3 Top-Down-Verfeinerung	★	★★★
Unterstützung für die Auflösung von komplexen Designproblemen (Architekturanalyse). Top-Down Verfeinerung von Designproblemen.	Es besteht keine abstrahierte Einstiegsebene für die Analyse des Designproblems.	Die Analyse des Designproblems kann über eine abstrahierte Ebene begonnen und anschliessend weiterführend vertieft werden.
D-4 Bottom-Up-Aufbau	★	★★★
Möglichkeit, über die Analyse von bereits gelösten Teil-Designproblemen das Gesamtdesign zu erarbeiten (Architektursynthese). Bottom-up zusammensetzen für ein Gesamtdesign, Prototyp oder den vollen Umfang.	Bottom-Up zusammenfügen ist schlecht möglich	Es ist leicht möglich, verschiedene Problemlösungen zu kombinieren, um ein übergeordnetes Ziel zu erreichen.

D-5 Beziehungen zwischen Designproblemen	★	★★★
Definiert Beziehungen zwischen verschiedenen Designproblemen und setzt diese im Verfahrensdesign ein, um die Konsistenz zu wahren und irrelevante Verfahrenselemente entfernen zu können.	Das Verknüpfen von Designproblemen ist nicht möglich.	Designprobleme können einfach verknüpft und Beziehungen leicht nachvollzogen werden.
D-6 To-Do-Liste	★	★★★
Eine verwaltbare To-Do-Liste zur Verfügung stellen. Backlog Konzept.	Es besteht keine Möglichkeit, Ausführungsschritte zu erfassen.	Es besteht die Möglichkeit, Ausführungsschritte zur Lösung eines Designproblems zu verwalten.
D-7 Bewertbarkeit, Feedback	★	★★★
Unterstützung für Architektur Bewertung, Feedback-Runden und Rückverfolgung. Für ständige Verbesserung des Designs.	Es besteht keine Interaktionsmöglichkeit, um Feedback oder Bewertungen entgegen zu nehmen.	Es besteht eine komfortable Lösung zur Entgegennahme von Feedback und Bewertung um eine iterative Verbesserung des Designproblems zu erreichen.

Tabelle 6.1: Anforderungen für Softwarearchitektur-Designverfahren

6.1.2. Anforderungen für Architektur-Wissensverwaltung

Die Anforderungen für Architektur-Wissensverwaltung dienen zur Bewertung des Umgangs mit Wissen. Das heisst: Erfassen von neuem Wissen, Adaptieren oder Delegieren von Wissen. Die Anforderungen werden im Kapitel 7 auf Seite 53 verwendet, um das Architekturdesign des AR-Werkzeugs mittels einer Selbstbeurteilung zu bewerten.

W-1 Wissen erlangen	★	★★★
Erlangen des nötigen Wissens. Von Drittpersonen, z. B. einer firmeninternen Community.	Wissen erfassen ist nur eingeschränkt möglich.	Wissen erfassen ist allgemein möglich.
W-2 Wissen anwenden	★	★★★
Anwenden von identifiziertem Wissen. Auswählen von zu verwendendem Wissen, entsprechend dem Verwendungszweck eines Projektes (löschen, verändern, hinzufügen).	Es ist schwierig, geeignetes Wissen für den angestrebten Verwendungszweck zu finden.	Es ist einfach, für den Verwendungszweck nützliches Wissen zu finden.
W-3 Delegieren von Entscheidungen	★	★★★
Delegieren von Architekturdesignarbeiten an andere Architekten oder Entwickler.	Es ist nicht oder nur schwer möglich, Entscheidungen oder Arbeiten zu delegieren.	Es ist einfach möglich, Entscheidungen oder Arbeiten zu delegieren.
W-4 Community	★	★★★
Einbeziehen einer Community. Einbeziehen von weiteren Personen, um zusätzliche Kompetenz für Architekturdesignarbeiten zu erlangen.	Es besteht keine oder nur eine geringe Möglichkeit zur Kollaboration.	Kollaborationsfunktionen sind ein zentrales Element und ermöglichen einfachen Einbezug von weiteren Personen.
W-5 Dokumentieren von Entscheidungen	★	★★★
Entscheidungen forcieren durch die Erstellung von Artefakten (Dokumente, Templates, Protokollen).	Es gibt keine oder nur eine eingeschränkte Möglichkeit, Entscheidungen zu forcieren.	Es ist möglich, das Fällen von Entscheidungen zu dokumentieren und forcieren.
W-6 Einfluss von Wissen	★	★★★
An anderen Modellen ausrichten. Entscheidungen sollen in Designmodelle, Entwicklung, Artefakte (Source-Code, Konfigrationsdateien) und Testfälle einfliessen.	Das Wissen ist nur schwer in anderen Objekten wiederzuverwenden.	Das Wissen kann einfach in anderen Objekten wiederverwendet werden.

W-7 Teilen von Wissen	★	★★★
Teilen von gewonnenem Architekturwissen mit Drittparteien (Communities, etc.).	Die Verbreitung des errungenen Wissens ist mühsam.	Es ist einfach, das errungene Wissen zu Verbreiten.

Tabelle 6.2: Anforderungen für Architektur-Wissensverwaltung

6.1.3. Allgemeine Anforderungen von Software-Entwicklungsverfahren

Die allgemeinen Anforderungen werden in Kapitel 7 auf Seite 53 verwendet, um das AR-Model im AR-Werkzeug mittels einer Selbstbeurteilung zu bewerten. Das AR-Werkzeug ist im erweiterten Sinne als ein Software-Entwicklungswerkzeug zu betrachtet und ein AR als Teil vom AR-Werkzeug als Software-Entwicklungsverfahren.

E-1 Verfahrensanalyse	★	★★★
Verfahrensanalyse = Prozess + Notation + unterstützende Techniken und Inhalt. Verdeutlicht Verfahrensumfang. Technik und Inhalt garantieren die Wiederverwendbarkeit.	Es besteht kein klarer Verfahrensumfang. Die Technik und der Inhalt erschweren die Wiederverwendbarkeit.	Das Verfahren ist klar ersichtlich. Die Technik und der Inhalt ermöglichen eine einfache Wiederverwendbarkeit.
E-2 Beschreibungsformat	★	★★★
Ein Standard Beschreibungsformat steht zur Verfügung. Es erlaubt, unterstützende Werkzeuge zu entwickeln, z. B. für den Wissensaustausch (Kollaboration).	Es ist kein beschreibendes Modell vorhanden.	Ein ausführliches Beschreibungsmodell ist vorhanden, welches für unterstützende Werkzeuge oder Funktionen verwendet werden kann.

E-3 Allgemeine Anwendbarkeit	★	★★★
Es ist allgemein anwendbar (stehen Vorlagen und Beispiele zur Verfügung). Verfahrenselemente müssen in einen Software-Entwicklungszyklus passen (und dabei genug detailliert und konkret sein).	Das Verfahren ist schlecht allgemein anwendbar.	Das Verfahren ist allgemein Anwendbar und es stehen Vorlagen oder Beispiele zur Verfügung.
E-4 Anforderungs-Design Verbindung	★	★★★
Es besteht eine Verbindung zwischen der Anforderungs-Erarbeitung und der Designarbeit. Anforderungen sind im Design nachvollziehbar und sorgen dafür, dass das Design harmonisch ist.	Es besteht keine ersichtliche Verbindung zwischen Anforderung und Design.	Die Verbindungen zwischen Anforderungen und Design sind einfach nachzuvollziehen.
E-5 Projektverwaltungsmethoden	★	★★★
Bietet einen Link zu Projektverwaltungsmethoden. Unterstützt das Erteilen von Verantwortlichkeiten an Softwareentwickler oder Softwarearchitekten in Industrieprojekten.	Es besteht keine Möglichkeit, Verantwortlichkeiten zu delegieren.	Es ist möglich, Verantwortlichkeiten einfach zu delegieren.
E-6 Erweiterbarkeit	★	★★★
Einfache Inhaltsentwicklung und Erweiterbarkeit. Macht die Verfahren anpassbar und somit allgemein anwendbar.	Wissensinhalte erweitern ist schwierig.	Es besteht eine einfache Möglichkeit, die Wissensinhalte weiter zu entwickeln.
E-7 Verständlichkeit	★	★★★
Die Inhalte sind verwendbar und verständlich. Inhalt muss einfach verständlich und veränderbar sein. Ein Benutzer muss irrelevante Teil entfernen oder fehlende Teile hinzufügen können.	Inhalte sind schwer verständlich und umständlich anzupassen.	Die Wissensinhalte sind klar verständlich und einfach anzupassen.

Tabelle 6.3: Allgemeine Anforderungen von Software-Entwicklungsverfahren

6.2. Wissens-Validierung: Kriterien für die Wissensinhalte

Dieser Abschnitt enthält eine kurze Erklärung zur Ausführung der Wissens-Validierung, da der anzuwendende Massstab die im separaten Kapitel 4 auf Seite 26 definierten Modellierungsprinzipien sind. Das Ziel der Wissens-Validierung ist es, den

Inhalt der erarbeiteten ARs in Kapitel 5 auf Seite 34 mittels den Modellierungsprinzipien zu überprüfen. Die Prinzipien wurden bereits bei der Erfassung der meisten ARs als Referenz verwendet. Der Prozess der Wissensschaffung sowie die Erschaffung der Modellierungsprinzipien wurden jedoch während der Masterarbeit parallel vollzogen, das heisst die Ausführung der Wissens-Validierung entspricht teilweise der Endkontrolle und Nachbesserung der Wissensinhalte. Die detaillierte Besprechung der Validierung für jedes AR würde den Rahmen dieser Arbeit übersteigen, daher behandelt Abschnitt 7.2 auf Seite 57 ein AR als Beispiel.

6.3. Werkzeug-Validierung: szenario-basierter Anwendungstest

Für den Anwendungstest (Usability Test) des AR-Werkzeugs ist ein szenario-basierter Ansatz gewählt. Dazu ist ein Dokument entstanden, welches zwei durchzuführende Szenarien beinhaltet. Ein Benutzer liest das jeweilige Szenario und spielt es gemäss den Anweisungen durch. Hat der Benutzer das Szenario ausgeführt, wird er angehalten, Fragen zum Inhalt und zur Benutzerfreundlichkeit des AR-Werkzeugs zu beantworten. Der Anwendungstest orientiert sich an der ISO-Norm 9421 “Ergonomics of human-system interaction” [Sta06] und einem darauf basierenden Beispiel-Fragenkatalog [Mic09]. Der komplette Anwendungstest ist im Anhang aufgeführt (siehe Abschnitt A.5 auf Seite 125).

Szenario 1 wurde vom Institutsmitarbeiter (IFS) Carmelo Schumacher an der Hochschule für Technik Rapperswil (HSR) durchgeführt. Szenario 2 wurde vom Autor selber in der Rolle als wissenserschaffende Person (AR-Editor), wie sie auch im Kapitel 5 auf Seite 34 beim AR Modelling eingenommen wurde, durchgeführt. Szenario 2 verlangt die Erfassung eines ARs, für welches spezifisches Wissen über Datenbanksysteme nötig ist. Dieses Wissen hat sich der Autor im Verlaufe seines Masterstudiums in einem Datenbank-Seminar angeeignet.

Die Fragen zum Inhalt in Szenario 1 fallen beim Szenario 2 weg. Die Fragen zur Usability sind hingegen für beide Szenarien zu beantworten.

6.3.1. Szenarien

In diesem Abschnitt sind die beiden Szenarien des Anwendungstests allgemein beschrieben. Die ausführlichen und spezifischen Szenarien sind in Abschnitt A.5 auf Seite 125 beschrieben.

Szenario 1 befasst sich mit dem Auffinden eines ARs. Es wird ein Problem in einem Softwaresystem beschrieben und verschiedene Symptome und Beeinträchtigungen (Smells), die in diesem System auftreten. Der Benutzer soll nun mittels des AR-Werkzeugs, im Speziellen des “Smell Self-Assessment”, eine geeignete Lösung (AR) für das Problem finden.

Szenario 2 befasst sich mit dem Erfassen einer Architekturumbauanleitung. Der Benutzer ist angehalten, sein Wissen über einen Architekturumbau im AR-Werkzeug zu erfassen. Dazu muss er ein komplettes AR mit allen seinen referenzierten Objekten (Smells, Execution Tasks, Meta-Daten Element) im AR-Werkzeug erstellen.

6.3.2. Gestaltungsgrundsätze für Softwaresysteme

Dieser Abschnitt enthält eine kurze Beschreibung der Gestaltungsgrundsätze für Softwaresysteme nach Vorgabe der ISO-Norm 9241 [Sta06], welche die Grundlage für den Fragenkatalog des Anwendungstests bilden. Der Fragenkatalog enthält entsprechende Fragen, zu jeder der folgend aufgeführten Kategorien:

Aufgabenangemessenheit ist gut erfüllt, sofern die Software über die nötige Funktionalität, Charakteristik und Technologie verfügt, damit der Benutzer angemessen seine Arbeitsaufgabe erledigen kann. Z. B. durch sinnvolle Standardwerte bei Eingabefeldern.

Selbstbeschreibungsfähigkeit ist gut erfüllt, sofern ein Dialog selbsterklärend ist und es für den Benutzer offensichtlich ist, an welcher Stelle im Dialog er sich befindet. Zudem, welche Schritte er im Dialog vornehmen muss, um die Arbeitsaufgabe angemessen erledigen zu können. Z. B. erklären, wann eine Eingabe erwartet wird oder was die nächsten Schritte sind.

Steuerbarkeit ist gut erfüllt, sofern der Benutzer den Dialogablauf in seiner Geschwindigkeit und Richtung beeinflussen kann. Z. B. wenn es möglich ist, getätigte Eingaben rückgängig zu machen.

Erwartungskonformität ist gut erfüllt, sofern der Dialog über die ganze Anwendung gleiche Funktionen und Bedienung gewährleistet. Der Benutzer verfügt so über eine grundsätzliche Vorhersehbarkeit des Dialoges. Z. B. unter Verwendung von anerkannten Konventionen (Ergonomie Elemente).

Fehlertoleranz ist gut erfüllt, sofern das Arbeitsziel trotz fehlerhaften Eingaben mit kleinem Aufwand seitens des Benutzers korrigiert werden kann. Z. B. automatische Erkennung von fehlerhaften Eingaben.

Individualisierbarkeit ist gut erfüllt, sofern der Benutzer die Darstellung in der Anwendung individualisieren und seinen Bedürfnissen anpassen kann. Z. B. ausblenden nicht benötigter Informationen oder Speichern von Suchergebnissen.

Lernförderlichkeit ist gut erfüllt, sofern der Benutzer aktiv von der Anwendung beim Erlernen der Nutzung unterstützt wird. Z. B. durch Tool-Tipp-Anwendungshinweisen oder Hilfeseiten.

7. Validierung von AR-Konzept, -Wissen und -Werkzeug

Das folgende Kapitel führt die Validierung des Architectural Refactoring Konzeptes, Wissens und Werkzeugs aus. Die Kriterien dazu sind im Vorkapitel (Kapitel 6 auf Seite 45) definiert.

7.1. Konzept-Validierung für Architectural Refactorings

Dieser Abschnitt befasst sich mit der Validierung des Konzeptes von Architectural Refactoring. Zum Konzept von Architectural Refactoring gehört ebenfalls die Werkzeugunterstützung (das Architectural Refactoring Werkzeug).

Entsprechend den definierten Bewertungskriterien in Abschnitt 6.1 auf Seite 45 ist die folgende Liste der Sterne-Bewertung, um die Erklärung wann eine verbessernde Massnahme zu ergreifen ist, erweitert:

- ★ : Anforderung schlecht erfüllt, daher *muss* eine Massnahme zu Korrektur definiert werden
- ★★ : Anforderung gut erfüllt, bei Bedarf *kann* eine Massnahme zu Verbesserung definiert werden.
- ★★★ : Anforderung sehr gut erfüllt, es *muss keine* weitere Massnahme getroffen werden.

7.1.1. Bewertung des Softwarearchitektur-Designverfahrens

Tabelle 7.1 zeigt die Bewertung des Softwarearchitektur-Designverfahrens basierend auf der Anforderungsdefinition in Abschnitt 6.1.1 auf Seite 46.

D-1 Architektur Viewpoints	**Bewertung:**	★★★
Begründung: Das Verfahren erlaubt es, Wissensinhalte durch Viewpoints und Refinement-Levels einzuteilen, in der Beschreibung darauf Bezug zu nehmen und Ausführungsschritte in verschiedene Typen zu unterteilen. Dies ermöglicht eine Arbeitsteilung für die Umsetzung des ARs.		

D-2 Qualitätsattribut getrieben	**Bewertung:**	★★★
Begründung: Qualitätsattribute sind ein zentrales Element eines ARs und einfach zuweisbar. Mit den Smells besteht zudem eine weitere Ebene, die Symptome beschreibt, welche Qualitätsattribute in einem Softwaresystem beeinträchtigen und eine Anforderung zur Verbesserung antreibt.		
D-3 Top-Down-Verfeinerung	**Bewertung:**	★★
Begründung: Das AR-Konzept ermöglicht einen zusammenfassenden Einstieg in ein Problem und bietet die nötigen Referenzen und Verknüpfungen, um das Problem im Detail zu studieren und zu lösen.		
D-4 Bottom-Up-Aufbau	**Bewertung:**	★★★
Begründung: Über das "Smell Self-Assessment" kann nach geeigneten, verwandten Problemlösungen gesucht werden, welche sich unter Umständen miteinander kombinieren lassen. Die Smell-Tag-Cloud im AR-Werkzeug erlaubt es ebenfalls, mehrere Problemlösungen zu finden, welche das gleiche Symptom (Smell) referenzieren.		
D-5 Beziehungen zwischen Design-Problemen	**Bewertung:**	★★
Begründung: Eine Beziehung zwischen Designproblemen ist durch verschiedene Arten möglich. Weisen diese z. B. die gleichen Problemsymptome auf, kann eine Verknüpfung auf einen gemeinsamen Smell erstellt werden. Die Kategorisierung nach Qualitätsattributen stellt ebenfalls eine Verbindung her. Und in der Beschreibung der ARs kann ein Benutzer direkte Verknüpfungen zu anderen ARs einfügen.		
D-6 To-Do-Liste	**Bewertung:**	★★
Begründung: Dem AR kann eine sortierbare Liste von Ausführungsschritten zugeordnet werden. Es stellt somit die To-Do-Liste zur Umsetzung der Problemlösung dar.		
D-7 Bewertbarkeit, Feedback	**Bewertung:**	★★
Begründung: Das Domain-Modell für das AR-Werkzeug beschreibt eine Status-Einteilung der einzelnen Wissensobjekte (ARs, Smells), welche einen Review- und Überarbeitungs-Prozess erlaubt. Des Weiteren existiert im AR-Werkzeug eine Kommentarfunktion, die Feedback oder eine Diskussion zum Wissensinhalt ermöglicht.		
Massnahme: Eine konkrete Bewertungsfunktion besteht für die ARs nicht. Als Massnahme wurde der Feature-Request 76 im Feature Report Abschnitt A.4 auf Seite 124 erfasst.		

Tabelle 7.1: Bewertung der Anforderungen für Softwarearchitektur-Designverfahren

7.1.2. Bewertung der Architektur-Wissensverwaltung

Die Tabelle 7.2 zeigt die Bewertung der Architektur-Wissensverwaltung basierend auf der Anforderungsdefinition in Abschnitt 6.1.2 auf Seite 47.

W-1 Wissen erlangen	**Bewertung:**	★★
Begründung: Die Wissenserfassung hat die Einschränkung, dass nur Personen mit der Benutzerrolle AR-Editor neue Wissenselemente erfassen können. Das Konzept legt fest, dass nur Architekturexperten ARs eintragen können.		
Massnahme: Eine Massnahme, welche aber eine Konzepterweiterung zur Folge hätte, ist es anderen Benutzern eine Funktion im AR-Werkzeug zur Verfügung zu stellen, mit welcher diese Vorschläge für Wissensinhalte anbringen können.		
W-2 Wissen anwenden	**Bewertung:**	★★★
Begründung: Die Wissensaneignung und Findung ist für alle Benutzer des AR-Werkzeugs möglich, auch für nicht registrierte Benutzer. Um das für den Verwendungszweck benötigte Wissen zu identifizieren, steht im AR-Werkzeug die Funktion “Smell Self-Assessment” zur Verfügung.		
W-3 Delegieren von Entscheidungen	**Bewertung:**	★★★
Begründung: Das Werkzeug erlaubt es, Entscheidungen oder Arbeiten über die Erstellung und Zuteilung von Ausführungsschritten an verschiedene Rollen zu delegieren.		
W-4 Community	**Bewertung:**	★
Begründung: Das AR-Werkzeug verfügt nur über eine eingeschränkte Kollaborationsfunktion (dem Kommentieren und Diskutieren von AR-Versionen). Für die Einbindung einer Community ist dies jedoch eine wichtige Funktion.		
Massnahme: Ausgeprägte Kommunikations-Funktionen sind mit hohem Entwicklungsaufwand verbunden, weshalb bis jetzt davon abgesehen wurde und auch für die Prüfung der Konzept- und Werkzeug-Tauglichkeit nicht relevant war. Für eine Weiterentwicklung des AR-Werkzeugs ist dies jedoch in Betracht zu ziehen.		
W-5 Dokumentieren von Entscheidungen	**Bewertung:**	★★
Begründung: Entscheidungsschritte bei der Umsetzung eines ARs können via Execution-Tasks forciert werden. Die vorhanden Execution-Task-Types wie z. B. “Decision Task”, “Documentation Task” oder “Leadership Task” unterstützen verschiedene Entscheidungstypen.		
W-6 Einfluss von Wissen	**Bewertung:**	★★
Begründung: Wie in anderen Schritten angedeutet sind die Execution Tasks die Schnittstelle zu einer projektorientierten Abwicklung der AR-Umsetzung. Der Einfluss des AR-Wissens kann daher mit den Ausführungsschritten gesteuert werden. Z. B. mit einem “Documentation Task”, welcher beschreibt, dass das Softwarearchitektur Dokument mit den Neuerung des Refactorings aktualisiert werden muss.		

W-7 Teilen von Wissen	Bewertung:	★★
Begründung: Das AR-Werkzeug verfügt über die Möglichkeit, ARs, Smells und Execution Tasks als PDF-Dokument zu exportieren und so zu verbreiten. Zudem sind alle publizierten Objekte ohne Login einfach über einen Direkt-Link abruf- und teilbar.		
Massnahme: Eine weitere Massnahme ist im Feature-Request 56 im Feature Report Abschnitt A.4 auf Seite 124 festgehalten. Sie beschreibt den Export der Wissensinhalte im Jekyll Format. Jekyll [Pre] ist ein aufkommendes Werkzeug zur Generierung von statischen Webseiten basierend auf einfachem Textformat.		

Tabelle 7.2: Bewertung der Anforderungen für Architektur-Wissensverwaltung

7.1.3. Bewertung des Software-Entwicklungsverfahrens

Tabelle 7.3 zeigt die Bewertung des Software-Entwicklungsverfahrens basierend auf der Anforderungsdefinition in Abschnitt 6.1.3 auf Seite 49.

E-1 Verfahrensanalyse	Bewertung:	★★★
Begründung: Durch die Definition von Prozess-/Ausführungsschritten für ein Architectural Refactoring, einer klaren Vorgabe zur Notation durch die bestehenden Modellierungsprinzipien und das Werkzeug als unterstützende Technik ergibt sich ein konkretes wiederverwendbares Entwicklungsverfahren.		
E-2 Beschreibungsformat	**Bewertung:**	★★★
Begründung: Es bestehen konkrete Modellierungsprinzipien und ein Konzept, welche das Beschreibungsformat definieren und es erlauben ergänzende Werkzeuge zu erstellen.		
E-3 Allgemeine Anwendbarkeit	**Bewertung:**	★★
Begründung: Es werden gängige Verfahrenselemente wie Viewpoints, Qualitätsattribute oder Prozess-/Execution-Tasks verwendet, welche auch in gängigen Software-Entwicklungsverfahren eingesetzt werden. Die Modellierungsprinzipien enthalten ansatzweise Beispiele, aber es ist darauf zu achten, dass vor allem in der Werkzeugunterstützung (dem AR-Werkzeug) Beispiele zur Erfassung vorhanden sind.		
Massnahme: Im AR-Werkzeug eine Hilfeseite mit Beispielen und Vorlagen erstellen. Dazu wurde der Feature-Request 74 im Feature Report Abschnitt A.4 auf Seite 124 erfasst.		
E-4 Anforderungs-Design Verbindung	**Bewertung:**	★★
Begründung: Im übertragenen Sinn sind bei einem AR die Qualitätsattribute die Treiber für die Anforderungen und stellen so als Teil des “Quality-Driven Developments” eine Verbindung zwischen Anforderung und Design her.		

E-5 Projektverwaltungsmethoden	**Bewertung:**	★★
Begründung: Execution-Tasks können an eine Projektrolle delegiert werden, wodurch die Ausführungsschritte in ein Projektverwaltungswerkzeug übertragbar sind. Der werkzeugunterstützte Export von Execution-Tasks ist im AR-Werkzeug vorgesehen aber noch nicht implementiert.		
Massnahme: Der Feature-Request 75 im Feature Report Abschnitt A.4 auf Seite 124 adressiert die Implementation einer Exportfunktion in ein Projektverwaltungswerkzeug.		
E-6 Erweiterbarkeit	**Bewertung:**	★★★
Begründung: Bereits im ersten Domain-Modell des AR-Werkzeugs wurde eine Versionierung der Wissensinhalte (ARs) vorgesehen und im AR-Werkzeug implementiert. Es ermöglicht die einfache und nachvollziehbare Inhaltsentwicklung und Erweiterung.		
E-7 Verständlichkeit	**Bewertung:**	★★★
Begründung: Das Konzept von Architectural Refactorings und die dazugehörigen Modellierungsprinzipien definieren die enthalten Objekte detailliert und machen die Inhalte einfach verständlich und verwendbar sowie leicht anpassbar.		

Tabelle 7.3: Bewertung der allgemeinen Anforderungen von Software-Entwicklungsverfahren

7.2. Wissens-Validierung für Architectural Refactorings

Dieser Abschnitt enthält eine Beispiel-Validierung eines ARs mittels der im Kapitel 4 auf Seite 26 definierten Modellierungsprinzipien. In der nachfolgenden Tabelle 7.4 sind die Inhalte der Attribute des ARs mit dem Namen “Split Components” kommentiert. Die verbesserte und finale Version dieses ARs ist im Anhang Abschnitt A.1 auf Seite 79 zu finden. Dieser Teil der Validierung widerspiegelt den Anwendungsfall “AR erstellen und prüfen” in der Rolle des “AR-Editors 2” (siehe Abschnitt 3.3.2 auf Seite 17).

<table>
<tr><th colspan="2">AR</th><th>Bewertung</th></tr>
<tr><td>Name
Split Components</td><td>Status
published</td><td>Dem Namen des ARs fehlt es an Prägnanz. Eine Ergänzung um den Teil “to reduce overly tight coupling”, stärkt die Aussage und das Ziel des ARs sowie den Bezug zur Beschreibung.</td></tr>
<tr><td>Context
• Logical View
• Conceptual Level</td><td>Quality attributes
• Maintainability</td><td>Die Kontextelemente reichen aus. Das Qualitätsattribut widerspiegelt die Beschreibung des ARs ungenügend. Die Qualitätsattribute Dependency und Responsibility ergänzen die Liste.</td></tr>
<tr><td colspan="2">Smells (refactoring drivers)
• Large number of component responsbilities and collaborations</td><td>Es fehlt ein Smell, welcher auf die enge Verknüpfung von unabhängigen Funktionen hinweist.</td></tr>
<tr><td colspan="2">Description
A component which has grown over a period of time, might have gained functionalities which are not necessarily related to each other. This creates unwanted dependencies and overly tight coupling. For example if a component contains a functionality which could be useful for another component, it might no be accessible.</td><td>Der Text beschreibt das Problem (Ausgangslage) ausreichend, aber es fehlen Hinweise zur Umsetzung (verbessertem Design) und zu vermeidenden Stolpersteinen.</td></tr>
<tr><td>Architectural decision(s) to be revisited
• Functional partitioning</td><td>Affected components and connectors
• Application software (Business software)</td><td>Die zu überdenkende Architekturentscheidung reicht aus. Bei den betroffenen Komponenten fehlen “andere Applikationen”.</td></tr>
<tr><td colspan="2">Execution tasks
• Create new component :: Systemmgmt. Task
• Move one or more responsibilities over to new component :: Architectural Decision Task
• Update all client collaborations of component to use new one :: Development Task</td><td>Es fehlt ein Dokumentations- und Testtask.</td></tr>
<tr><td colspan="2">References (links):
• OLAF’s Method Tailoring Guide
• Separation of concerns</td><td>Die Referenzliste ist noch mit mindestens einem Standardwerk zu ergänzen.</td></tr>
</table>

Tabelle 7.4: Architectural Refactoring Beispielvalidierung mit Modellierungsprinzipien

7.3. Werkzeug-Validierung: Architectural Refactoring Tool Anwendungstest

Die Werkzeug-Validierung besteht aus drei Teilen. Der erste Teil befasst sich mit einem Selbsttest; ein AR zu finden und dieses zu implementieren. Der zweite Teil beinhaltet einen Anwendungstest, ein AR auf Grund eines Szenarios selber zu erfassen (beschrieben in Abschnitt 6.3.1 auf Seite 51) und dem anschliessenden Ausfüllen eines Fragebogens (siehe Abschnitt A.5 auf Seite 125).

7.3.1. AR finden und implementieren als Selbsttest

Dieser Abschnitt befasst sich mit einer Beispiel-Implementation eines ARs ("Migrate web-application to a cloud-environement"). Die technische Ausführung ist im Anhang Abschnitt A.2 auf Seite 120 detailliert festgehalten. Das Finden eines ARs beschränkt sich auf den Vergleich, der verschiedenen Wege zur Suche einer Problemlösung (beschrieben im Anwendungsfall "AR suchen, finden und anwenden" Abschnitt 3.3.2 auf Seite 17). Die Aussagekraft des Suchens nach einem selbst erstellten AR wäre nicht geben. Der Anwendungsfall der Suche nach einem AR ist in Kapitel 8 auf Seite 65 durch eine Drittperson beurteilt worden. Tabelle 7.5 zeigt die im Anwendungsfall dargestellten Wege zur Findung einer Problemlösung und beschreibt Vor- und Nachteile der jeweiligen Optionen:

Option	**Vorteile**	**Nachteile**
Suche mittels AR-Werkzeugs	Das Smell Self-Assessment bietet einen einfachen Einstiegspunkt. Die ARs beinhalten in kompakter und strukturierter Weise eine Lösungsbeschreibung und Anhaltspunkte zu vertieftem Wissen. Es kann auf Erfahrungswerte in der Beschreibung oder den Kommentaren zurückgegriffen werden. Die Wissensinhalte sind auf den Umbau von Softwaresystemen ausgelegt.	Um das AR umzusetzen ist eine Vertiefung in weitere Literatur nötig. Ein AR ist keine vollumfängliche Beschreibung zur Umsetzung.
Community kontaktieren	Communities können verschiedene Erfahrungswerte und Meinungen zu einem Problem bieten.	Hilfestellung zur Umsetzung zu erhalten, funktioniert nur über den direkten Einbezug von Drittpersonen.

Internetsuche	Informationsvielfalt.	Unstrukturierte Informationsflut. Zeitaufwändige Suche und Filterung von relevanten und verlässlichen Informationen.

Tabelle 7.5: Vergleich von Suchen nach einer Problemlösung

Das AR “Migrate web-application to a cloud-environement” wurde gemäss der Aufgabenliste und der AR-Beschreibung umgesetzt. Aus der Umsetzung resultiert, dass der Quellcode des AR-Werkzeugs auf Github.com öffentlich verfügbar ist und die Applikation in der Cloud-Umgebung Heroku zur Verwendung bereit steht.

- Github URL: `https://github.com/bisigc/art`
- Heroku URL: `https://thawing-taiga-6031.herokuapp.com`

7.3.2. AR erfassen als Selbsttest

In diesem Abschnitt wurde das Szenario 2 des Anwendungstests (siehe Abschnitt A.5 auf Seite 125) ausgeführt und anschliessend der Fragekatalog über das AR-Werkzeug ausgefüllt. Auf Grund der beantworteten Fragen wurden im Anschluss, wenn nötig, entsprechende Massnahmen zur Verbesserung definiert.

1. Reagiert das Architectural Refactoring Werkzeug schnell auf Benutzereingaben?

	$--$	$-$	$-/+$	$+$	$++$	
langsam	☐	☐	☐	☐	☑	schnell

Kommentar:
Das ART enthält mittlerweile 20 ARs, 32 Smells und 90 Tasks und reagiert immer noch schnell.

2. Ist das Architectural Refactoring Werkzeug unkompliziert zu bedienen?

	$--$	$-$	$-/+$	$+$	$++$	
kompliziert	☐	☐	☐	☐	☑	unkompliziert

Kommentar:
Das ART verfolgt ein einheitliches und klares Design. Die Bedienelemente haben einen Wiedererkennungswert.

3. Erfordert das Architectural Refactoring Werkzeug überflüssige Eingaben?

	$--$	$-$	$-/+$	$+$	$++$	
überflüssige Eingaben nötig	☐	☐	☐	☐	☑	keine überflüssigen Eingaben nötig

Kommentar:

Ein neues Architectural Refactoring lässt sich mit nur einem Formular erfassen. Aus diesem Formular heraus können neue Attributwerte mittels Dialogen erstellt werden. Dieses Konzept ist übersichtlich und verlangt nur die nötigsten Eingaben.

4. Bietet das Architectural Refactoring Werkzeug die nötigen Funktionen, um eine Lösung für das beschriebene Szenario zu finden?

	$--$	$-$	$-/+$	$+$	$++$	
nötige Funktionen fehlen	☐	☐	☐	☑	☐	nötige Funktionen sind vorhanden

Kommentar:

Beim Arbeiten mit mehreren Objekten (AR, Smell, Task) gleichzeitig und dem Wechseln zwischen diesen Objekt ist das Auffinden des zuvor bearbeiteten Objektes schwierig oder umständlich (z. B. über den AR-, Smell- oder Task-Browser). Es wäre nützlich, eine Ansicht für die zuletzt verwendeten Objekte zu haben.

5. Bietet das Architectural Refactoring Werkzeug eine gute Übersicht über das Funktionsangebot?

	$--$	$-$	$-/+$	$+$	$++$	
schlechte Übersicht	☐	☐	☐	☑	☐	gute Übersicht

Kommentar:

Das Menü ist übersichtlich und einfach gehalten. Es bietet einen klaren Überblick über die Funktionen des Werkzeugs.

6. Verwendet das Architectural Refactoring Werkzeug gut verständliche Begriffe, Bezeichnungen, Abkürzungen oder Symbole in Masken und Menüs?

	$--$	$-$	$-/+$	$+$	$++$	
schlechte verständlich	☐	☐	☐	☑	☐	gut verständlich

Kommentar:

Das Architectural Refactoring Werkzeug verwendet einfache Begriffe und einheitliche Symbole.

7. Liefert das Architectural Refactoring Werkzeug in zureichendem Masse Informationen darüber, welche Eingaben zulässig oder nötig sind?

	$--$	$-$	$-/+$	$+$	$++$	
unzureichende Informationen	☐	☐	☑	☐	☐	zureichende Informationen

Kommentar:
Die Angabe über das Was und Wie etwas in einem Formular ausgefüllt werden muss, kann noch verbessert werden. Sinnvoll sind das Anzeigen von Muss-Feldern und Hinweise zum idealen Inhalt.

8. Erleichtert das Architectural Refactoring Werkzeug die Orientierung durch eine einheitliche Gestaltung.?

	$--$	$-$	$-/+$	$+$	$++$	
erschwert Orientierung	☐	☐	☐	☐	☑	erleichtert Orientierung

Kommentar:
Das Werkzeug ist einheitlich gestaltet und folgt über alle Seiten dem gleichen Aufbau.

9. Lässt sich das Architectural Refactoring Werkzeug durchgehend nach einem einheitlichen Prinzip bedienen?

	$--$	$-$	$-/+$	$+$	$++$	
uneinheitlich bedienbar	☐	☐	☐	☐	☑	einheitlich bedienbar

Kommentar:
Die Bedienelemente sind einheitlich und haben Wiedererkennungscharakter.

10. Erfordert das Architectural Refactoring Werkzeug viel Zeit zum Erlernen?

	$--$	$-$	$-/+$	$+$	$++$	
braucht viel Zeit	☐	☐	☐	☑	☐	braucht wenig Zeit

Kommentar:
Es erfordert nicht viel Zeit, um den Umgang mit dem Werkzeug zu erlernen. Eine Einstiegserklärung zu den Funktionen des Werkzeugs wäre jedoch von Vorteil.

11. Ist das Architectural Refactoring Werkzeug gut ohne Hilfe oder Handbuch erlernbar?

	$--$	$-$	$-/+$	$+$	$++$	
schlecht erlernbar	☐	☐	☐	☑	☐	gut erlernbar

Kommentar:
Der Funktionsumfang ist im Menü ersichtlich und man findet sich schnell zurecht. Eine Erklärung der Funktionen wäre hilfreich.

12. Ermöglicht das Architectural Refactoring Werkzeug einen leichten Wechsel zwischen einzelnen Menüs und Masken?

	−−	−	−/+	+	++	
keinen leichten Wechsel	☐	☐	☐	☐	☑	leichten Wechsel

Kommentar:
Der Wechsel zwischen den Menüs und Masken ist sehr einfach zu bewerkstelligen.

13. Liefert das Architectural Refactoring Werkzeug gut verständliche Fehlermeldungen?

	−−	−	−/+	+	++	
schlecht verständlich	☐	☐	☐	☐	☑	gut verständlich

Kommentar:
Die Fehlermeldungen sind einfach und verständlich formuliert und bieten ebenfalls technische Anhaltspunkte, z. B. eine “Constraint Violation”.

14. Lässt sich das Architectural Refactoring Werkzeug gut an eine persönliche, individuelle Art der Arbeitserledigung anpassen?

	−−	−	−/+	+	++	
lässt sich schlecht anpassen	☐	☐	☑	☐	☐	lässt sich gut anpassen

Kommentar:
Das Werkzeug bietet zwar die Möglichkeit, Suchabfragen aus dem “Smell Self-Assessment” im Profil zu speichern, andere Funktionen zur Individualisierung bestehen jedoch keine.

7.3.3. Interpretation und Massnahmen

Das AR-Werkzeug ist ein robustes Werkzeug mit solidem Funktionsumfang. Trotzdem ergeben sich aus dem Anwendungstest einige Punkte, durch welche sich die Verwendbarkeit der Applikation, vor allem beim Erfassen von neuen Inhalten, verbessern lässt. Die folgende Liste zeigt Verbesserungsvorschläge für das AR-Werkzeug, welche im Feature- und Bug-Report (siehe Abschnitt A.4 auf Seite 124) erfasst und in Kapitel 9 auf Seite 70 umgesetzt sind. Möglichkeiten zur Individualisierung sind hier nicht definiert, solche sind oft zeitaufwändig in der Implementation und bieten nur eine geringfügige Verbesserung der Kernfunktionen.

Massnahmen:

- **Last viewed items:** Die zuletzt besuchten Objekte (ARs, Smells, Tasks) in einer Liste im Werkzeug anzeigen.
- **Attribut Tool-Tipps:** Die Attributfelder in den Erfassungsformularen mit Popup-Hinweisen zur Erfassung versehen (beschreibende Definitionen aus den Modellierungsprinzipien).
- **Mussfelder:** Mussfelder in den Erfassungsformularen sichtbar kennzeichnen.
- **Task Typen bei Taskliste anzeigen:** Neben den Einträgen der Task-Liste, beim Erfassen oder Betrachten eines ARs, sollen die Task Execution Typen angezeigt werden. Dadurch ist eine schnellere Kontrolle der noch fehlenden Arbeitsschritte möglich.
- **Hilfeseite:** Um den Einstieg in das Werkzeug zu erleichtern, ist ein Benutzerhandbuch, nähere Beschreibungen von Objekten des AR-Werkzeugs und Beispiel-Einträge sinnvoll.
- **Smell Beschreibung verbergen:** In der AR-Detailübersicht sind die Smells sinnvollerweise vor der Beschreibung des ARs aufgelistet. Die Beschreibungen der Smells nehmen jedoch viel Platz ein. Daher sollen die Smell-Beschreibungen standardmässig verborgen sein und mittels eines Buttons eingeblendet werden können.
- **Dialoge nur mittels Klick auf Cancel oder ESC schliessen:** Es ist bis anhin möglich, einen Erfassungsdialog, z. B. für ein AR-Metadaten-Objekt, mittels Klick ausserhalb der Dialogbox zu schliessen. Dies geschieht oft unbeabsichtigt und führt zum Verlust der bereits erfassten Daten. Dies soll unterbunden werden. Eine Dialogbox soll nur durch einen Klick auf Cancel oder mit der ESC-Taste geschlossen werden können.

8. Diskussion Anwendungstests Drittperson

Dieses Kapitel enthält einen von Carmelo Schumacher (Mitarbeiter des Institute for Software (IFS) an der Hochschule für Technik Rapperswil (HSR)) durchgeführten Anwendungstest. Mein Studienbetreuer Prof. Dr. Olaf Zimmermann hat ebenfalls während verschiedenen Abschnitten der Masterarbeit Anwendungstests vorgenommen. Das Ziel der Anwendungstests durch Drittpersonen ist es, eine zusätzliche Sicht und mögliche Verbesserungsvorschläge für das Konzept und das Werkzeug zu erhalten. Alle Verbesserungsvorschläge und gefundenen Fehler sind im Feature- und Bug-Report (siehe Abschnitt A.4 auf Seite 124) festgehalten.

8.1. Szenario 1: Ein geeignetes Architectural Refactoring finden

Dieser Abschnitt enthält die Resultate und die Interpretation des Anwendungstest (Szenario 1), welcher vom IFS Mitarbeiter Carmelo Schumacher an der HSR durchgeführt wurde. Die Beschreibung des Szenarios und der komplette Fragebogen sind im Anhang Abschnitt A.5 auf Seite 125 ersichtlich.

8.2. Fragen & Resultate

8.2.1. Zum Inhalt

1. Sind die Fragen im Smell Self-Assessment verständlich?

	$--$	$-$	$-/+$	$+$	$++$	
schwer verständlich	☐	☐	☐	☐	☑	einfach verständlich

2. War es schwierig mittels des Smell Self-Assessment ein geeignetes Architectural Refactoring zu finden?

	$--$	$-$	$-/+$	$+$	$++$	
schwierig	☐	☐	☐	☐	☑	einfach

Kommentar:
Übersichtlich nur bei einer begrenzten Anzahl von Fragen. Skalierbarkeit bei grösserem Fragenkatalog evtl. nicht gegeben.

3. Ist das Architectural Refactoring verständlich formuliert?

	−−	−	−/+	+	++	
schwer verständlich	☐	☐	☐	☐	☑	einfach verständlich

Kommentar:
Struktur-logischer Aufbau finde ich gut. Schnelle Übersicht möglich.

4. Sind die zum Architectural Refactoring verlinkten Smells verständlich formuliert?

	−−	−	−/+	+	++	
schwer verständlich	☐	☐	☐	☑	☐	einfach verständlich

Kommentar:
Smells sind gut formuliert, aber manchmal etwas sehr generisch.

5. Machen die zum Architectural Refactoring verlinkten Tasks Sinn?

	−−	−	−/+	+	++	
nicht sinnvoll	☐	☐	☐	☑	☐	sinnvoll

Kommentar:
Wie bei Smells. Gut formuliert, aber auch hier manchmal etwas zu generisch.

8.2.2. Zur Benutzerfreundlichkeit

1. Reagiert das Architectural Refactoring Werkzeug schnell auf Benutzereingaben?

	−−	−	−/+	+	++	
langsam	☐	☐	☐	☐	☑	schnell

Kommentar:
Reaktionszeit sehr schnell. Aufgrund der noch sehr begrenzten Anzahl von Objekten aber noch nicht abschliessend testbar.

2. Ist das Architectural Refactoring Werkzeug unkompliziert zu bedienen?

	−−	−	−/+	+	++	
kompliziert	☐	☐	☐	☐	☑	unkompliziert

Kommentar:
Klare Struktur und Erwartungen stimmen mit Realität überein.

3. Erfordert das Architectural Refactoring Werkzeug überflüssige Eingaben?

	$--$	$-$	$-/+$	$+$	$++$	
überflüssige Eingaben nötig	☐	☐	☐	☐	☑	keine überflüssigen Eingaben nötig

Kommentar:
Zielgerichtet.

4. Bietet das Architectural Refactoring Werkzeug die nötigen Funktionen, um eine Lösung für das beschriebene Szenario zu finden?

	$--$	$-$	$-/+$	$+$	$++$	
nötige Funktionen fehlen	☐	☐	☑	☐	☐	nötige Funktionen sind vorhanden

Kommentar:
Sicher ein guter Startpunkt und gut für Übersicht. Tasks aber oftmals zu generisch.

5. Bietet das Architectural Refactoring Werkzeug eine gute Übersicht über das Funktionsangebot?

	$--$	$-$	$-/+$	$+$	$++$	
schlechte Übersicht	☐	☐	☐	☐	☑	gute Übersicht

6. Verwendet das Architectural Refactoring Werkzeug gut verständliche Begriffe, Bezeichnungen, Abkürzungen oder Symbole in Masken und Menüs?

	$--$	$-$	$-/+$	$+$	$++$	
schlecht verständlich	☐	☐	☐	☐	☑	gut verständlich

7. Liefert das Architectural Refactoring Werkzeug in zureichendem Masse Informationen darüber, welche Eingaben zulässig oder nötig sind?

	$--$	$-$	$-/+$	$+$	$++$	
unzureichende Informationen	☐	☐	☐	☐	☑	zureichende Informationen

8. Erleichtert das Architectural Refactoring Werkzeug die Orientierung durch eine einheitliche Gestaltung.?

	$--$	$-$	$-/+$	$+$	$++$	
erschwert Orientierung	☐	☐	☐	☐	☑	erleichtert Orientierung

9. Lässt sich das Architectural Refactoring Werkzeug durchgehend nach einem einheitlichen Prinzip bedienen?

	$--$	$-$	$-/+$	$+$	$++$	
uneinheitlich bedienbar	☐	☐	☐	☐	☑	einheitlich bedienbar

10. Erfordert das Architectural Refactoring Werkzeug viel Zeit zum Erlernen?

	−−	−	−/+	+	++	
braucht viel Zeit	☐	☐	☐	☐	☑	braucht wenig Zeit

11. Ist das Architectural Refactoring Werkzeug gut ohne Hilfe oder Handbuch erlernbar?

	−−	−	−/+	+	++	
schlecht erlernbar	☐	☐	☐	☐	☑	gut erlernbar

12. Ermöglicht das Architectural Refactoring Werkzeug einen leichten Wechsel zwischen einzelnen Menüs und Masken?

	−−	−	−/+	+	++	
kein leichter Wechsel	☐	☐	☐	☐	☑	leichter Wechsel

13. Liefert das Architectural Refactoring Werkzeug gut verständliche Fehlermeldungen?

	−−	−	−/+	+	++	
schlecht verständlich	☐	☐	☐	☐	☐	gut verständlich

Kommentar:
Keine Fehlermeldung bei der Benutzung.

14. Lässt sich das Architectural Refactoring Werkzeug gut an eine persönliche, individuelle Art der Arbeitserledigung anpassen?

	−−	−	−/+	+	++	
lässt sich schlecht anpassen	☐	☐	☐	☐	☐	lässt sich gut anpassen

Kommentar:
Kann ich nicht gut beurteilen. Nebst responsivem Verhalten sehe ich keine Möglichkeit zum Customizen.

Anmerkungen:
Das Tool wirkt klar und übersichtlich. Das strukturlogische Konzept ist schnell begreifbar. Bietet einen guten Anhaltspunkt. Da jedoch der Umfang der Inhalte noch beschränkt ist und das Usability-Szenario etwas vorhersehbar gewählt ist, ist die Usability-Aussage evtl. etwas verzerrt.

8.3. Interpretation

Die Inhalte wie auch die Struktur und die Funktionalitäten schneiden bei der Bewertung durch Carmelo Schumacher gut ab. Carmelo Schumacher als Drittperson konnte das Architectural Refactoring Werkzeug ohne Probleme verwenden. Das im Anwendungstest beschrieben Szenario führte schnell zum gewünschten Resultat, die Bedienung des Werkzeugs erfuhr dabei einen wertvollen Test.

Folgende Punkte habe ich auf Grund der Bewertung genauer analysiert und Massnahmen bestimmt:

- **Zu generische Task-Beschreibungen:** Dies ist ein wertvoller Kritikpunkt. In den Modellierungsprinzipien für die Architectural Refactorings habe ich explizit den Hinweis angebracht, möglichst spezifische Arbeitsschritte zu erfassen. Diesen Kritikpunkt habe ich auch bei der Überarbeitung der ARs berücksichtigt. Die Tasks sollten spezifischer sein, um das Umsetzungsvorgehen genauer zu beschreiben. Ausnahmen sind für Test oder Dokumentationstasks erlaubt (z. B. “Ausführen eines Lasttests” oder “Anpassen des Architekturdokuments”).

- **Zu generische Smells:** Dies ist beabsichtigt. Die Smells dürfen nicht zu spezifisch sein, dies fördert die Mehrfachverknüpfungen zu verschiedenen ARs und bezweckt die Auflistung von alternativen Lösungen z. B. beim Ausfüllen eines “Smell Self-Assessment”. Dadurch kann ein Softwarearchitekt aus verschiedenen Optionen, eine oder mehrere, wählen.

- **Werkzeug-Funktionalität:** Aus dem Test geht hervor, dass das AR-Werkzeug über einen soliden Grundfunktionsumfang verfügt, aber noch ausbaufähig ist. Kollaborationsfunktionen und unterstützende Eigenschaften sind anzustrebende Punkte für ein erfolgreiches Weiterbestehen des Werkzeugs. Eine nützliche Funktion ist eine Liste der zuletzt betrachteten Objekte (AR, Smell, Task). Dieser Punkt ist auch im Test zur AR-Erfassung festgehalten (siehe Abschnitt 7.3.3 auf Seite 64).

9. ART-Verbesserungen und -Erweiterungen

Dieses Kapitel beschreibt die wesentlichen Änderungen und Verbesserungen, welche während der Masterarbeit am Architectural Refactoring Werkzeug vorgenommen wurden. Die Änderungen basieren alle auf Erfahrungen und Anforderungen, welche durch die spezifischen Anwendungstests und während der Benutzung zur Wissenserfassung entstanden sind. Bereits vor der Masterarbeit stellte das AR-Werkzeug ein solides Werkzeug dar, mit den in diesem Kapitel umgesetzten Änderungen konnte der Wert des Werkzeug, aber nochmals gesteigert werden.

Insgesamt wurden 36 neue Funktionen implementiert und 19 Fehler behoben, sowie die Qualität des Frontend-Quellcodes durch eine umfangreiche Massnahme erhöht. Die folgenden Abschnitte beschreiben die wichtigsten vorgenommenen Änderungen.

9.1. Funktionale Verbesserungen

Dieser Abschnitt beschreibt neue Funktionen, welche in den ursprünglichen Anforderungen [Bis15b] nicht definiert sind und während der Masterarbeit umgesetzt wurden.

9.1.1. Export von ARs, Smells und Tasks als PDF

Eine Exportfunktion für die Objekte des AR-Werkzeugs war vorgesehen, aber in einem Rohdatenformat (z. B. JavaScript Object Notation (JSON)). Mit der Implementation einer PDF-Exportfunktion ist eine Erweiterung entstanden, welche den formatierten Ausdruck der Wissensinhalte ermöglicht. Die PDF-Generierung basiert auf einer XML-Datenaufbereitung (XSL-Transformation) mit anschliessendem Rendering-Prozess (XSL-FO-Formatierung). Abbildung 9.1 zeigt den PDF-Rendering-Prozess. Das Vorgehen, um ein bestehendes System mit einer PDF-Rendering-Engine zu erweitern, ist als Architectural Refactoring im Abschnitt A.1 auf Seite 79 festgehalten.

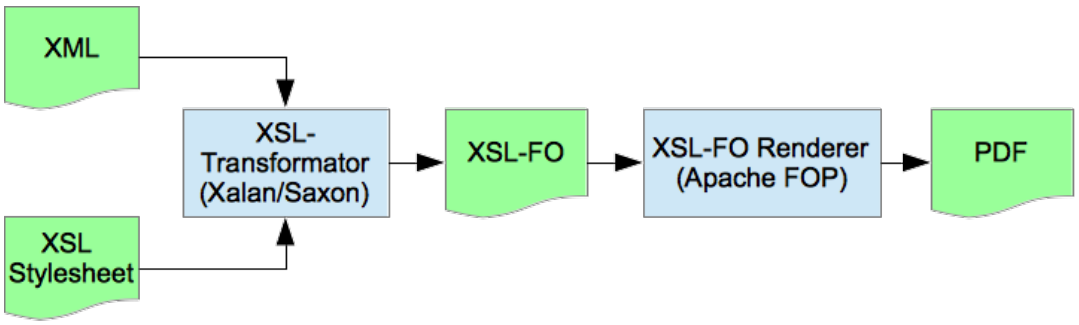


Abbildung 9.1: PDF Rendering Prozess mit XSL-T und XSL-FO

9.1.2. Abkürzungen verwenden im Rich-Texteditor

Für das verwendete Rich-Texteditor-Modul “textAngular” wurde eine Erweiterung entwickelt, welche es erlaubt, Abkürzungen in den Beschreibungstexten zu verwenden. Für verwendete Akronyme kann so die komplette Bedeutung hinterlegt und bei Bedarf eingeblendet werden.

9.1.3. Sortieren von verlinkten Modell-Elementen

Im AR-Erfassungsformular besteht nun die Möglichkeit, die Reihenfolge der verlinkten Smells, Tasks und Modell-Elemente pro Kategorie (Context, Quality attributes, ...), für das entsprechende AR festzulegen.

9.2. Usability Verbesserungen

Im AR-Werkzeug sind verschiedene Änderungen realisiert, welche die Bedienbarkeit steigern. Dieser Abschnitt beschreibt die wesentlichen Änderungen.

9.2.1. Verbesserung der Darstellung und Anordnung von Tabellen und Übersichten

Um die Übersichtlichkeit und Bedienbarkeit im AR-Werkzeug positiv zu beeinflussen, wurden verschiedene Änderungen an der Darstellung im Werkzeug vorgenommen.

Beispiele sind:

- Pop-Beschreibungen im AR-Browser
  Die Detailbeschreibung der ARs zeigt der AR-Browser an, sobald der Benutzer den Mauszeiger über den Namen des ARs hält.

- Tabellen-Attribute angepasst
  In allen Tabellen des Frontends (AR-, Smell- und Task-Browser) wurde die Spaltenbreiten, Anzahl der Attribute und deren Beschriftungen zu Gunsten der Leserlichkeit optimiert.
- AR-Detailansicht überarbeitet
  Die Anordnung der Attribute sowie deren Darstellung wurden auf Grund von Erfahrungswerten überarbeitet. Z. B. Die Smell-Beschreibung ist standardmässig ausgeblendet, kann aber mit einem Klick sichtbar gemacht werden (siehe Abbildung 9.2).

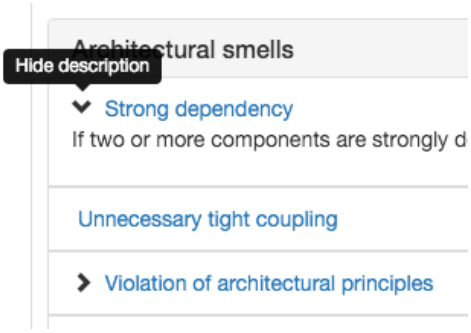


Abbildung 9.2: Zeigen/Verstecken der Smell-Beschreibung in der AR-Ansicht

9.2.2. Hilfeseite

Aus beiden Anwendungstests während der Masterarbeit resultierte die Erkenntnis, dass eine zusätzliche Benutzerorientierung mit Beschreibungen und Beispielen angebracht wäre. Als Massnahme wurde das AR-Werkzeug mit einer Hilfeseite ergänzt, welche das Benutzerhandbuch als Download, Begriffserklärungen und Beispiele beinhaltet.

9.2.3. Popup-Hilfestellung bei der Erfassung von Objekten

Um die Orientierung und Hilfestellung für den Benutzer zu erhöhen, wurde die Anweisungen zur Erfassung von Wissenselementen aus den Modellierungsprinzipien als Popup-Hinweise in die verschiedenen Erfassungsformulare integriert (siehe Abbildung 9.3). Die angezeigten Texte können über das Menü “Properties” administriert werden.

Abbildung 9.3: Popup-Hinweis zum Feldinhalt

9.2.4. Verbesserung des Smell Self-Assessments

Eine Echtzeitauflistung von gefundenen ARs erlaubt es, das Suchresultat des Smell Self-Assessment zu kontrollieren, bevor der Benutzer zur Gesamtübersicht gelangt.

9.2.5. Liste zuletzt angesehener Objekte

Beim Modellieren der ARs stellte sich hauptsächlich ein Problem heraus; die Navigation beim aktiven Editieren zwischen den verschiedenen involvierten Wissensobjekten und Attributen (Modell-Elementen) ist sehr umständlich. Der Zurück-Button des Browsers ist zu wenig transparent. Aus diesem Grund wurde entschieden eine Liste der zuletzt besuchten Objekte im Frontend zu implementieren. Ein Button in der Menüleiste öffnet eine Dropdown-Menü mit den Verknüpfungen der zuletzt verwendeten ARs, Smells oder Tasks (siehe Abbildung 9.4).

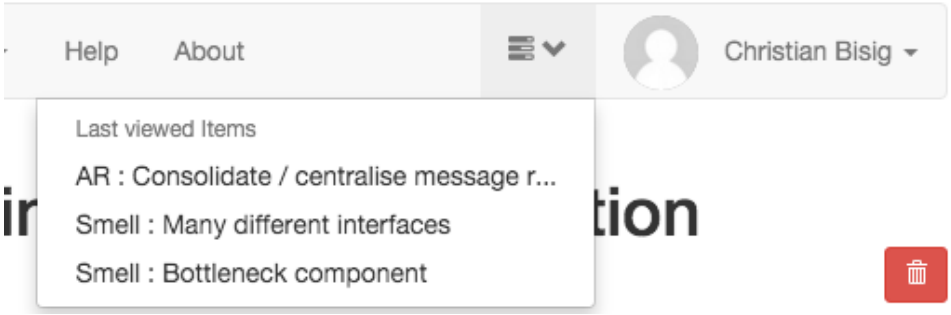


Abbildung 9.4: Menü der zuletzt besuchten Items im ART

9.3. Verbesserung der Robustheit

Hauptsächlich in der Anfangsphase der Masterarbeit wurden einige Softwarefehler korrigiert, welche das Arbeiten mit dem Werkzeug noch beeinträchtigten. Die frühe Behebung der Fehler ermöglichte ein stabiles Arbeiten mit dem Werkzeug während der Wissenserfassung und der Anwendungstests für die Konzept- und Wissensvalidierung.

Beispiele solcher Fehlerbehebungen sind:

- Das Auslösen eines Smell Self-Assessments ohne eine selektierte Frage verursachte einen Fehler.
- Die Filterfelder bei der AR-Browser-Tabelle funktionierten nicht alle korrekt.
- Die Löschung eines kompletten ARs war nicht fehlerfrei möglich.

9.3.1. Datenbank-Backup zu Dropbox

Um grossen Datenverlust während dem Arbeiten mit dem AR-Werkzeug zu vermeiden, wurde auf der Virtual Machine (VM), auf welcher das AR-Werkzeug läuft, ein nächtlicher Cronjob eingerichtet. Dieser erzeugt mit dem MySQL-Dump-Werkzeug einen kompletten Abzug der Datenbank und lädt das Datenfile anschliessend auf den Filehosting-Dienst Dropbox.

9.4. Code-Qualität und Wartbarkeit

Die Verbesserung der Quellcode-Qualität und Wartbarkeit des AR-Werkzeugs beinhaltet einen wesentlichen und umfangreichen Schritt. Dieser Schritt integriert ein Frontend-Build-Prozess unter der Nutzung von verschiedenen Werkzeugen (Grunt, Bower, NPM und Yeoman). Diese Werkzeuge ermöglichen unter anderem:

- Die optimale Vorbereitung der Frontend-Applikation für die Ausbreitung, z. B. mittels Komprimierung aller Javascript-Bibliotheken (Task Runner: Grunt).
- Die automatische Integration von Open-Source-Frontend-Bibliotheken aus dem Internet (Packet-Manager: Bower und NPM).
- Generierung von Frontend-Artefakten als Vorlage. Z. B. eine Frontend-Controller-Klasse (Scaffolding Werkzeug: Yeoman).

Details zur Integration des Frontend-Build-Prozess sind im Abschnitt A.3 auf Seite 122 zu finden.

Schlussfolgerung

Ich habe mich drei Semester mit dem Thema Architectural Refactoring befasst. Für mich war dieser Begriff neu und auch für die Welt der Softwarearchitektur ist dieser noch beinahe unbekannt. Die Projektarbeiten 1 und 2 erstellten in den ersten beiden Semestern das Konzept und unterstützendes Werkzeug für Architectural Refactorings. Die abschliessende Masterarbeit beweist nun das Potential von Architectural Refactoring, sich als ein neues Paradigma in der Softwarearchitektur zu etablieren. Das werkzeugunterstützte Konzept von ARs erfüllt gemäss der durchgeführten Analyse die Anforderungen für ein Architektur-Designverfahren und eine Wissensverwaltung. Dies bildet das Fundament für die zukünftige Verwendung und Erweiterung durch Softwarearchitekten und Fachexperten.

Die Masterarbeit schafft mit verschiedenen ARs aus dem Bereich der Applikations- und Integrationsarchitektur eine initiale Wissensgrundlage. Die Modellierungsprinzipien für die Wissenobjekte setzen dafür und für alle weiteren ARs einen konkreten Rahmen. Dieses Wissen steht nun offen zur Diskussion, zur Verbesserung und Erweiterung durch Anwender des Architectural Refactoring Werkzeugs.

Das AR-Werkzeug hat basierend auf den verschiedenen Anwendungstests, den Erfahrungen aus der Wissensschaffung und Validierungen, zahlreiche Verbesserungen erfahren. Im Bereich der Bedienbarkeit sind ergänzende und vereinfachende Funktionen hinzugekommen. Ein Beispiel ist die Anzeige für zuletzt verwendete Objekte. Das AR-Werkzeug ist unter der OSS Lizenz “Apache 2” veröffentlicht (Github: `https://github.com/bisigc/art`) und lauffähig verfügbar auf dem Cloud-Service Heroku (`https://thawing-taiga-6031.herokuapp.com`). Für die Weiterentwicklung des Werkzeugs empfehle ich den Ausbau der Kollaborations-Funktionen. Der Gedankenaustausch unter Editoren und Anwendern der ARs ist entscheidend für das Weiterbestehen des Konzeptes und AR-Werkzeugs.

In dieser Arbeit war für mich vor allem der Teil der Wissensentwicklung und Sammlung in Form von ARs sehr spannend. Er erlaubte mir eine Reflexion meiner bisherigen praktischen und theoretischen Erfahrungen sowie die Vertiefung in neue Architekturbereiche. ARs stellten sich dabei auf Grund ihrer Attribute als ideales und einfach anwendbares Mittel zur Strukturierung dieser Erfahrungen heraus.

Das Masterstudium hat mich der Domäne der Softwarearchitektur und meinem Ziel, eines Tages ein erfahrener Softwarearchitekt zu sein, ein grosses Stück näher gebracht. Ich freue mich nun auf neue Herausforderungen, bei denen ich das Gelernte anwenden und Erfahrungen für viele neue Architectural Refactorings sammeln kann.

Literaturverzeichnis

[Bab+09] Ali Babar u. a. *Software Architecture Knowledge Management*. Springer, 2009. ISBN: 9783642023743.

[Bet+12] D. Betts u. a. *Moving Applications to the Cloud, 3rd Edition*. 2012. URL: https://msdn.microsoft.com/en-us/library/ff728592.aspx.

[Bis15a] Christian Bisig. *Architectural Refactoring Werkzeug - Fachliches Konzept und Machbarkeitsstudie*. Apr. 2015. URL: http://www.hsr.ch/uploads/tx_icscrm/bisig.pdf.

[Bis15b] Christian Bisig. *Ein Werkzeug für Architectural Refactoring - Design und Implementierung*. Sep. 2015. URL: http://www.hsr.ch/uploads/tx_icscrm/bissig.pdf.

[DMG07] Paul Duvall, Steve Matyas und Andrew Glover. *Continuous Integration - Improving Software Quality and Reducing Risk*. Addison Wesley, 2007. ISBN: 9780321336385.

[Fai10] George Fairbanks. *Just Enough Software Architecture, A Risk-Driven Approach*. Marshall & Brainerd, 2010. ISBN: 9780984618101.

[Feh+14] C. Fehling u. a. *Cloud Computing Patterns*. Springer, 2014. ISBN: 9783709115671.

[Fow+02] M. Fowler u. a. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002. ISBN: 0321127420.

[Fow03] Martin Fowler. *TechnicalDebt*. Okt. 2003. URL: http://martinfowler.com/bliki/TechnicalDebt.html.

[Fow13] Martin Fowler. *UML Distilled, A Brief Guide to the Standard Object Modeling Language, Third Edition*. Addison-Wesley, 2013. ISBN: 0321193687.

[Fow99] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. ISBN: 9780201485677.

[Gam+94] Erich Gamma u. a. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN: 0201633612.

[Goo] Google. *AngularJS*. URL: https://angularjs.org/.

[HS09] Peter Hruschka und Gernot Starke. *Softwarearchitekten, Die Zehnkämpfer der IT*. Apr. 2009. URL: http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2009/04/hruschka_starke_OS_04_09.pdf.

[HW03] Gregor Hohpe und Bobby Woolf. *Enterprise Integration Patterns*. Addison Wesley, 2003. ISBN: 9780321200686.

[JC14] P. Jamshidi und X. Liu C. Pahl S. Chinenyeze. „Cloud Migration Patterns: A Multi-Cloud Service Architecture Perspective“. In: *ICSOC 2014 Workshops* (2014), S. 6–19. URL: http://link.springer.com/chapter/10.1007/978-3-319-22885-3_2.

[Kas05] Mark Kasunic. *Designing an Effective Survey*. Sep. 2005. URL: http://www.sei.cmu.edu/reports/05hb004.pdf.

[Kru95] Phillippe Kruchten. „Architectural Blueprints - The 4+1 View Model of Software Architecture“. In: *IEEE Software* 12.6 (Nov. 1995), S. 45–50. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759.

[Lar08] Craig Larman. *Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition*. Prentice Hall PTR, 2008. ISBN: 0131489062.

[Lie+08] D. Liebhart u. a. *Integration Architecture Blueprint, Leitfaden zur Konstruktion von Integrationslösungen*. Carl Hanser Verlag München, 2008. ISBN: 9783446417045.

[Lie09] Daniel Liebhart. *System und Softwarearchitektur - Grundlagen*. Zürcher Fachhochschule. Modul Systemarchitektur. Sep. 2009.

[Mic09] Jochen Prümper und Micael Anft. *Beurteilung von Software auf Grundlage der Internationalen Ergonomie-Norm DIN EN ISO 9241-110*. 2009. URL: http://people.f3.htw-berlin.de/Professoren/Pruemper/instrumente/ISONORM%209241-110-L.pdf.

[Pre] Tom Preston-Werner. *Jekyll*. URL: http://jekyllrb.com/.

[PW92] Dewayne E. Perry und Alexander L. Wolf. „Foundations of the Study of Software Architecture“. In: *ACM SIGSOFT Software Engineering Notes* 17.4 (Okt. 1992), S. 40–52. URL: http://dl.acm.org/citation.cfm?id=141884.

[RW05] Nick Rozanski und Eóin Woods. *So Now Im A Software Architect. What Do I Actually Do?* Okt. 2005. URL: http://www.informit.com/articles/article.aspx?p=417090.

[RW13] Nick Rozanski und Eoin Woods. *Software Systems Architecture, Second Edition*. Addison-Wesley, 2013. ISBN: 9780321718334.

[Sha03] Mary Shaw. „Writing Good Software Engineering Research Papers“. In: *ICSE 03 Proceedings of the 25th International Conference on Software Engineering* (2003), S. 726–736. URL: http://dl.acm.org/citation.cfm?id=776816.776925.

[SSS15] Girish Suryanarayana, Ganesh Samarthyam und Tushar Sharma. *Refactoring for Software Design Smells Third Edition, Managing Technical Debt*. Morgan-Kaufmann, 2015. ISBN: 9780128013977.

[Sta06] International Organization for Standardization. *Ergonomics of human-system interaction – Part 110: Dialogue principles*. 2006. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=38009.

[Sta08] Michael Stal. *Software Architecture Refactoring, OOPSLA 2007 Tutorial 1*. 2008. URL: http://www.sigs.de/download/oop_08/Stal%20Mi3-4.pdf.

[Sta11] International Organization for Standardization. *Systems and software Quality Requirements and Evaluation*. 2011. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733.

[TZC] Typesafe, Zenexity und Community. *Play Framework*. URL: https://www.playframework.com/.

[Var10] Jinesh Varia. *Migrating your Existing Applications to the AWS Cloud*. Okt. 2010. URL: https://s3.amazonaws.com/awsmedia/CloudMigration-main.pdf.

[Var11] Jinesh Varia. *Architecting for the Cloud: Best Practices*. Jan. 2011. URL: https://s3.amazonaws.com/awsmedia/AWS_Cloud_Best_Practices.pdf.

[Wie14] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Addison Wesley, 2014. ISBN: 9783662438398.

[Zim09] Prof. Dr. Olaf Zimmermann. *An Architectural Decision Modeling Framework for Service-Oriented Architecture Design*. 2009. URL: http://elib.uni-stuttgart.de/opus/volltexte/2010/5228/pdf/SOAD_ArchitecturalDecisionModeling_PhDThesisOlafZimmermann.pdf.

[Zim14] Prof. Dr. Olaf Zimmermann. *Architectural Refactoring - Agile Umsetzung von Modernisierungsentscheidungen, OOP2014 - Vortrag*. Feb. 2014. URL: http://www.ifs.hsr.ch/fileadmin/user_upload/customers/ifs.hsr.ch/Home/projekte/ARC-ArchRefAUME4OOPv102p.pdf.

[Zim15a] Prof. Dr. Olaf Zimmermann. „Architectural Refactoring: A Task-Centric View on Software Evolution“. In: *IEEE Software* 32.2 (März 2015), S. 26–29. URL: http://doi.ieeecomputersociety.org/10.1109/MS.2015.37.

[Zim15b] Prof. Dr. Olaf Zimmermann. *Architectural Refactoring for the Cloud: a Decision-Centric View on Cloud Migration*. Sep. 2015. URL: http://www.ifs.hsr.ch/fileadmin/user_upload/customers/ifs.hsr.ch/Home/projekte/ARC-SummerSoCSubmission2015v11.pdf.

A. Anhang

A.1. Architectural Refactorings

Dieser Abschnitt enthält die in dieser Arbeit erstellten Architectural Refactorings, wie sie mittels der PDF-Generierungs-Funktion aus dem Architectural Refactoring Werkzeug exportiert wurden. Die Sprache der Architectural Refacotrings ist Englisch, da alle Inhalte des AR-Werkzeug in Englisch gehalten sind. Die Detailbeschreibungen der Smells und der Tasks sind zudem hier nicht ersichtlich, können aber im AR-Werkzeug separat exportiert werden. Die Architectural Refactorings folgen der gleichen Sortierung wie jener im Abschnitt 5.3 auf Seite 37.

Add tier to improve independent scalability

Created	Modified	Status
11/26/2015 at 21:39:22	11/26/2015 at 21:39:22	published

Context (viewpoint, refinement level)

- Logical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Scalability https://en.wikipedia.org/wiki/Scalability
- Performance https://en.wikipedia.org/wiki/Computer_performance
- Flexibility https://en.wikipedia.org/wiki/Flexibility_(engineering)

Smells

Name / Description / Questions	Group	TDI
Bottleneck component	Performance	hh

If multiple components or system parts are running on the same hardware or even on separate hardware, one part can impair the rest of the system.

- Does a component or a part of your software system impair the whole system?

Description

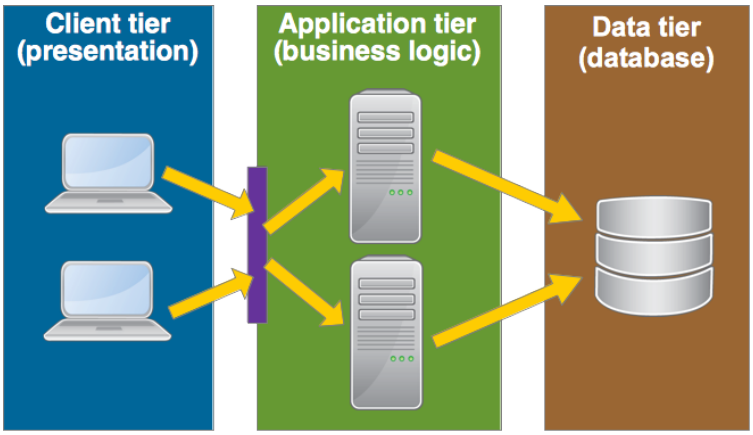
Initial position

A software system with multiple parts (components) running on the same server respectively on the same tier, lacks on flexibility. Which means, it is possible, that one part (component) of the system scales well while another does not (on the same hardware). E.g. a system where the business logic is placed on the same tier (server) as the database. If the application executes many and large data requests, the hardware requirements between the business logic layer and the database may be unequal. If the database does not perform well on the hardware, the requesting application will have slow response times. Since the database is on the same hardware, it is not possible to scale it separately.

Revised design

The goal of this architectural refactoring is to extract parts (components) of a software system and place it on a separat tier. The benefit of this will be the possibility to scale the extracted component separately. For this it is necessary to put a load balancer between the existing and the newly created tier (See AR: https://thawing-taiga-6031.herokuapp.com/#/ar/19).

Additionally, the newly freed resources on the existing tier, can be used by the remaining components on the system (e.g. create a second instance of business application).



The figure shows a classical three-tier-architecture where the data storage, the business logic and the presentation logic are separated on different physical hardware. Smaller applications often have a two-tier-architecture where the data and the business logic share the same server. This might become a problem as soon as traffic and data volume raise in a system.

Pitfalls to avoid

- Do not create too many tiers - otherwise your maintenance costs and systems management effort will increase; the latency for application requests will rise too.
- Make sure that you don't forget to secure the connections between the existing and the new tier(s). (Depending on your security requirements.)

Architectural decision(s) to be revisited

- Choice of tier architecture

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Middleware
- Hardware (server)

Tasks

- Estimate future maintenance and support costs :: Analysis Task
- Assess security impact (security architecture review) :: Analysis Task
- Design system architecture (component to tier distribution) :: Design Task
- Introduce load balancer to scale newly added tiers :: Integration Task
- Extract component from existing tier :: Integration Task
- Change connectivity configuration :: Integration Task
- Execute load test :: Testing Task
- Update software architecture document (SAD) :: Documentation Task

References (links):

- Patterns of Enterprise Application Architecture [Martin Fowler]
  http://martinfowler.com/books/eaa.html
- Load balancing [Wikipedia]
  https://en.wikipedia.org/wiki/Load_balancing_(computing)
- Scalability [Wikipedia]
  https://en.wikipedia.org/wiki/Scalability
- Microsoft Application Architecture Guide, 2nd Edition
  https://msdn.microsoft.com/en-us/library/ff650706.aspx

Convert application from single- to multi-threading to improve scalability

Created	Modified	Status
12/02/2015 at 15:49:00	12/06/2015 at 23:56:01	published

Context (viewpoint, refinement level)

- Process View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Deployment View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Technology Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Performance https://en.wikipedia.org/wiki/Computer_performance
- Scalability https://en.wikipedia.org/wiki/Scalability
- Data integrity https://en.wikipedia.org/wiki/Data_integrity

Smells

Name / Description / Questions	Group	TDI
Poor application performance	Performance	hm
The performance of your application has decreased or does not measure up to your expectations. • Are the processing times of an application to high (for one unit of work)?		
Slow response times for concurrent requests (latency)	Performance	hm
An application or a whole software system is not able to fulfill multiple requests in a desired timeframe or is not able to process them in parallel. • Is your software system not able to process multiple requests in a proper amount of time? • Is your software system not able to process multiple requests in parallel?		
Hardware capabilities not maxed out (bottleneck)	Performance	mm
You have a strong hardware system (e.g. multicore system) but your application is still running slow and the capacity of your hardware is not fully utilized. • Is your application to slow but the used hardware is not maxed out? • Is only one core of your multicore system maxed out by your application?		
Undesired queuing of incoming work requests	Performance	hm
There are coming in more work requests than your system can process (maybe only in peek times as well as permanently). • Is your application not able to catch up with the amount of work requests?		

Description

Initial position

If the performance of an application reaches its limit even though the capacity of the underlying hardware is not maxed out, the application processing might have to be changed. This can be the case when the amount of requests to the system has increased during the evolution of the application, but the application has only one thread to processes these requests. Another reason can be if a unit of work is very large and is processed by only one thread.

Revised design

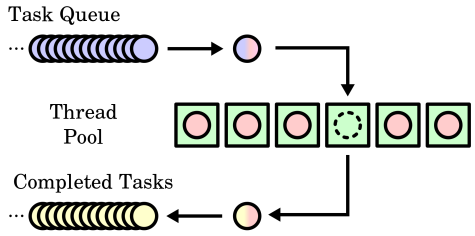
In order to be able to revise the design adequately, the following measurements have to be conducted and evaluated:

- Current size of the unit of work (e.g. the business transactions and updates to business objects caused by the incoming user requests)
- Concurrency level of your software
- Capable concurrency level of your hardware system

Having compiled these measurements, try to split your unit of work in as many equally sized pieces to increase the concurrency level of your software and meet the capable concurrency level of your hardware as close as possible.

If your application was single threaded you might need to implement a thread pool (use existing frameworks or application server mechanisms). A thread pool allows you to configure the level of concurrency. The amount of concurrent threads can be in- or decreased which makes it easy to execute several tests and find the optimal level of concurrency.

If you implement multithreading to an existing application it is necessary to re-evaluate existing libraries with respect to thread-safe property. You also have to reevaluate the rest of your source code.



(src: Wikipedia)

The figure shows how a thread pool works. There is a queue of tasks to be processed (violet dots), and there are 6 threads running (green squares). An idle thread picks up a task in the queue and starts the process. When the work is done, the thread returns to idle and can pick up the next task in the queue.

Pitfalls to avoid

- Make sure to implement your software best suitable for the underlying hardware. For example; different CPU architectures behave differently in concurrency control (e.g. SPARC vs. x86, see https://thawing-taiga-6031.herokuapp.com/#/ar/24).
- Make sure your existing software is already thread-safe or to make it thread-safe, to avoid locking and other unpredictable behavior (e.g. overwriting data in the memory).

Architectural decision(s) to be revisited

- Approach to threading (concurrency management)
- Approach to resource protection

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Unit of work
- Middleware

Tasks

- Assess scalability impact (concurrency level) :: Analysis Task
- Analyse current software for possible locking parts (deadlocks) :: Analysis Task
- Redesign your current system (soft- and hardware) :: Design Task
- Consider the use of parallel programming patterns (parallel computing) :: Decision Task
- Enhance application with concurrency features (e.g., implement multi-threading) :: Development Task
- Execute load test :: Testing Task
- Execute performance test (response times under normal circumstances) :: Testing Task
- Test concurrent accesses (try to break the system) :: Testing Task

References (links):

- Thread-Pool Pattern
  https://en.wikipedia.org/wiki/Thread_pool
- Concurrency Patterns
  https://en.wikipedia.org/wiki/Concurrency_pattern
- Java theory and practice: Thread pools and work queues [IBM, Brian Goetz]
  http://www.ibm.com/developerworks/library/j-jtp0730/
- Advanced Computer Architecture and Parallel Processing [Hesham El-Rewini,Mostafa Abd-El-Barr]

Distribute session (instead of sticky sessions) to increase scalability

Created	Modified	Status
11/26/2015 at 23:35:13	11/26/2015 at 23:35:13	published

Context (viewpoint, refinement level)

- Physical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Information View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Scalability https://en.wikipedia.org/wiki/Scalability
- Flexibility https://en.wikipedia.org/wiki/Flexibility_(engineering)
- Performance https://en.wikipedia.org/wiki/Computer_performance

Smells

Name / Description / Questions	Group	TDI
Scalability issue	Performance	hm
An application is not able to react on variable requests or work load and therefore has high latency during peak times. • Are the response times of your application not constant? • Does your application not scale well for a high volume of requests?		
Uneven load balancing	Performance	mm
A load balancer does not exist or is not working properly, which causes an imbalance of load to the available application instances. • Is the work load respectively the amount of requests not well balanced between existing application instances?		

Description

Initial position

To provide a highly scalable distributed application an efficient load balancing is necessary (See AR: https://thawing-taiga-6031.herokuapp.com/#/ar/19). One obstacle for dynamic load balancing is a sticky session. Which means the load balancer initially binds a user session to a specific application instance ("session stickyness" based on the IP-address or the SSL-session-id). For the whole duration of the users activity, the session is bound to one application instance and can not be switched to another instance (only by creating a new session). This can lead to an uneven load balance.

An example: 50 users are accessing the application and the load balancer is assigning them with a round-robin-approach to two application instances (25 each). After a while 20 people which were assigned to the instance two and 5 people which were assigned to the instance one have left the application. The result is an imbalance of the sessions. Reassigning them is not possible since the sessions are handled by individual application instances.

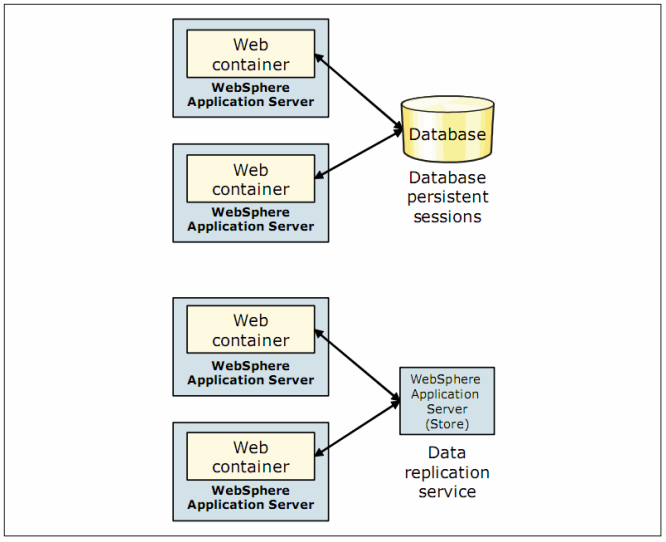
Revised design

The goal of this AR is to provide a distributed session. Which means, the same user session should be available for each application instances to which a user is routed even though the user has been rerouted (to another instance) during his stay in the application.

This behavior can be achieved with two approaches:

1. Database session (See AR: https://thawing-taiga-6031.herokuapp.com/#/ar/11)
   All the session attributes are stored in a database on a separate tier, which are accessible for all application instances.
2. Server session

This is depending on the used application server. The application has to provide a distributed server store. The configuration is individual to the application server.



(src: http://java.boot.by/ibm-317/images/c5s301.gif)

The figure shows two versions of using a distributed session store. In the upper part there two servers running the same web-application. The session is stored in a separate session store represented by the database. The lower part shows the same two servers which are using a third WebSphere application server that serves as a centralized session store.

Pitfalls to avoid

- Prevent to bloat the users sessions by adding to many attributes
- Make sure of high security standards. In case you generate the session id by yourself (not by the application server), certain security properties have to be met (see https://www.owasp.org/index.php/Session_Management_Cheat_Sheet#Session_ID_Properties).

Architectural decision(s) to be revisited

- Choice of session state management

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Middleware

Tasks

- Determine used session parameters :: Analysis Task
- Assess security impact (security architecture review) :: Analysis Task
- Evaluate session state store :: Decision Task
- Create system overview diagram with new session store :: Documentation Task
- Execute latency test :: Testing Task
- Integrate new session storage :: Integration Task

References (links):

- Patterns of Enterprise Application Architecture [Martin Fowler] http://martinfowler.com/books/eaa.html
- Load balancing [Wikipedia] https://en.wikipedia.org/wiki/Load_balancing_(computing)
- Cloud Computing Patterns [Fehling, Leymann, Retter, Schupeck, Arbitter]
- Open Web Application Security Project - Session_Management_Cheat_Sheet https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

Introduce caching to improve application scalability

Created	Modified	Status
10/05/2015 at 09:13:02	10/14/2015 at 07:20:39	published

Context (viewpoint, refinement level)

- Physical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Concurrency View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Scalability https://en.wikipedia.org/wiki/Scalability
- Performance https://en.wikipedia.org/wiki/Computer_performance

Smells

Name / Description / Questions	Group	TDI
Scalability issue	Performance	hm
An application is not able to react on variable requests or work load and therefore has high latency during peak times. • Are the response times of your application not constant? • Does your application not scale well for a high volume of requests?		
Performance loss	Performance	hm
General performance loss. Means not only at peak times. • Are the response times of your application constant but too high?		
Large number of component responsbilities and collaborations	Maintainability	mh
The component violates good OOAD and CBSE (Component-Based Software Engineering) practices, e.g. single responsibility (see video: https://www.youtube.com/watch?v=BD9VaTiLMpM). • Do any components expose multiple service responsibilities that are not closely related to each other? • Does a component violate the architectural principle of high cohesion/low coupling? • Does a component violate the "Single Responsibility" rule?		
Bottleneck component	Performance	hh
If multiple components or system parts are running on the same hardware or even on separate hardware, one part can impair the rest of the system. • Does a component or a part of your software system impair the whole system?		

Description

Initial position

It can have many different reasons if your application does not scale well. Most likely the reason for bad performance of an application is a specific bottleneck. Therefore it is inevitable to determine the bottlenecks before any actions can be taken. The most common bottleneck in a system is the access to data storages (hard-disk).

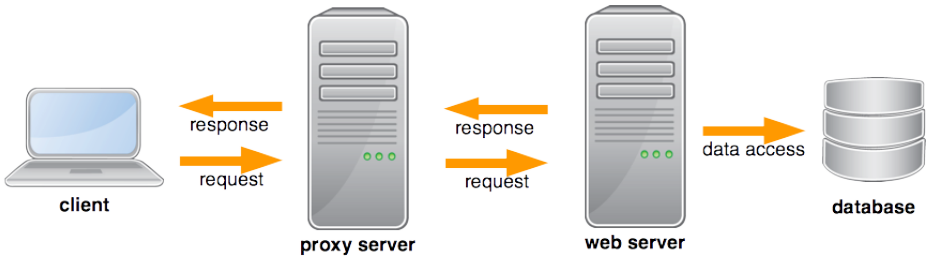
Different clients of an application often request the same data objects, without them being changed between the requests. Reading the same data from the hard-disk every time requires a lot of unnecessary I/O operations. Therefore a mechanism to reduce time-consuming hard-disk accesses should be put in place.

Revised design

The solution is to use caching mechanisms. A cache is a storage for query results of recurring requests. There are different caching mechanisms. Caching results from former requests in a memory-based cache is an effective way to spare capacity of a slower data storage. This measure increases the scalability of the application.

Caching can be applied on different levels in an system. For example on the data access layer where requested data objects are stored in a memory cache. Or on a service layer where the whole results of a service request is cached (e.g. a proxy server).

It might be necessary to use a distributed cache, which means the cached data has to be available to different servers or nodes. A common use of a distributed cache is in session management (see https://thawing-taiga-6031.herokuapp.com/#/ar/21).



The figure shows a client requesting data from a web server. In case the same request has been executed before, the intermediary proxy server is providing the cached result instead of the web server itself which would have to access the data storage.

Pitfalls to avoid

- Define a caching expiration time. Else the proxy will hold data to long which is no longer actual.
- Be aware, that the access logging is spread on the proxy server and the web server. Because a request will reach the web server only if the proxy does not provide a cached result.

Architectural decision(s) to be revisited

- Choice of middleware
- Choice of data source/store
- Choice of service interface
- Choice of messaging system

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Interfaces
- Middleware

Tasks

- Find bottlenecks in your system :: Analysis Task
- Define cache expiration times :: Decision Task
- Integrate proxy cache (server) :: Systemmanagement Task
- Implement new system setup :: Integration Task
- Choose scalability patterns to implement :: Architectural Decision Task
- Execute load test :: Testing Task
- Execute performance test (response times under normal circumstances) :: Testing Task
- Update software architecture document (SAD) :: Documentation Task

References (links):

- Cloud Computing Patterns [Fehling, Leymann, Retter, Schupeck, Arbitter]
- Scalability [Wikipedia]
  https://en.wikipedia.org/wiki/Scalability
- The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise [Abbott, Fisher]
- Performance and Scalability Patterns and Guidance [MSDN]
  https://msdn.microsoft.com/en-us/library/dn600224.aspx
- Address Scalability Bottlenecks with Distributed Caching [Iqbal Khan, MSDN Magazine]
  https://msdn.microsoft.com/en-us/magazine/ff714590.aspx

Introduce load balancing to increase application capacity and scalability

Created	Modified	Status
11/11/2015 at 11:40:23	11/11/2015 at 11:40:23	published

Context (viewpoint, refinement level)

- Physical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Deployment View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/

Quality attributes (forces)

- Scalability https://en.wikipedia.org/wiki/Scalability
- Performance https://en.wikipedia.org/wiki/Computer_performance

Smells

Name / Description / Questions	Group	TDI
Scalability issue	Performance	hm
An application is not able to react on variable requests or work load and therefore has high latency during peak times. • Are the response times of your application not constant? • Does your application not scale well for a high volume of requests?		
Performance loss	Performance	hm
General performance loss. Means not only at peak times. • Are the response times of your application constant but too high?		

Description

Initial position

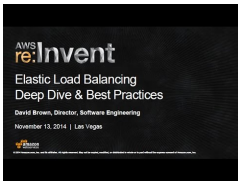
If the load on your application rises and the current system and application reaches its capacity limits, then it is either necessary to increase the capacity of the current application instance (scale up) or to put multiple instances of the same application in place (scale out). Scale out is the preferred way.

This will act like a parallelization for a non parallel application. It is possible to create multiple instances on the same system (plain, in VM (Virtual Machine)s or application containers), where each instance runs as an own process an the operating system in a multi core system can assign the process to the available cores. Or the application is instantiated on different servers. Either way, there is a mechanism necessary which distributes the client requests to the different instances automatically.

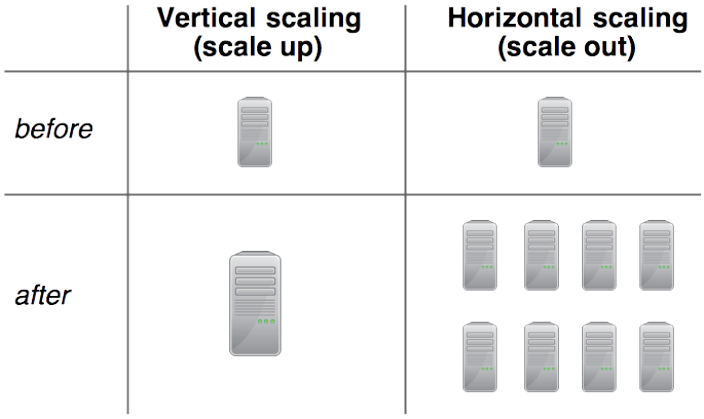
Revised design

A cloud based environment is able to automatically increase the capacity by creating new application instances and balance the load, depending on the current traffic on the application. This is referred to as an Elastic Load Balancer (see book Cloud Computing Patterns). In a non cloud based environment you have to take care of the instantiation. If you have an application server you have to create a cluster or add an instance to the cluster.

The load balancing itself can be done by a web-server or an application-server software, but for bigger systems it is advisable to use a so called Application Delivery Controller. An ADC is a separate hardware device which provides many other features besides load balancing. E.g. SSL, protection against DoS (Denial of Service)-Attacks, etc.



The video explains the Elastic Load Balancer in the Amazon Web Service Cloud.



The figure shows the difference of vertical and horizontal scaling. The horizontal scaling add new servers or virtual machines, for which a load balancer is essential. The vertical scaling replaces a small server with a stronger one.

Pitfalls to avoid

- Make sure to use session stickyness or distributed session (see https://thawing-taiga-6031.herokuapp.com/#/ar/21).
- Make sure to use a single source of data for a distributed application (if you instantiate your application multiple times). This can require a move of your data storage to a separate tier (see https://thawing-taiga-6031.herokuapp.com/#/ar/20).

Architectural decision(s) to be revisited

- Choice of hardware plattform

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Middleware
- Hardware load balancer

Tasks

- Evaluate session state management strategy :: Architectural Decision Task
- Evaluate load balancing strategy :: Architectural Decision Task
- Create new application instances :: Integration Task
- Execute load test :: Testing Task
- Execute failover test :: Testing Task
- Create deployment plan :: Documentation Task

References (links):

- Cloud Computing Patterns [Fehling, Leymann, Retter, Schupeck, Arbitter]
- Load balancing [Wikipedia] https://en.wikipedia.org/wiki/Load_balancing_(computing)
- The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise [Abbott, Fisher]
- Application delivery controller https://en.wikipedia.org/wiki/Application_delivery_controller

Migrate from client to database session state to provide scalability

Created	Modified	Status
09/29/2015 at 11:16:04	09/29/2015 at 11:16:04	published

Context (viewpoint, refinement level)

- Logical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Scalability https://en.wikipedia.org/wiki/Scalability
- Security https://en.wikipedia.org/wiki/Computer_security

Smells

Name / Description / Questions	Group	TDI
Large amount of session attributes	Performance	mh
Your application requires a lot of session attributes. The amount of attributes reaches the limit of the session store (e.g. a client side cookie (4093 Bytes)). • Are you reaching the limit of your session state storage?		
Sensitive session data	Security	hh
Your application stores sensitive data in the session, which can be a significant security issue. Depending on whether or not the session is hold (or its data is transferred) on the client side. • Does your session state contain sensitive session data?		

Description

Initial position

A Client Session State has some disadvantages which can become unhandy when an application runs in a cloud environment.

One major disadvantage for client session state is its limitation of space. For example a cookie can hold only 4093 Bytes. The limited data storage on the client side makes sense, since this forces the development to use either server (not recommended to use in a cloud environment, see https://thawing-taiga-6031.herokuapp.com/#/ar/12) or database session state.

Also the use of the client session state implies, that all session attributes are transferred back and forth between the client and the server even though not all of the attributes might be used (e.g. to display) on the client side.

Other advantages of the database session state are for example, that the session gets not lost after a connection loss between client and server or after either one of them crashes.

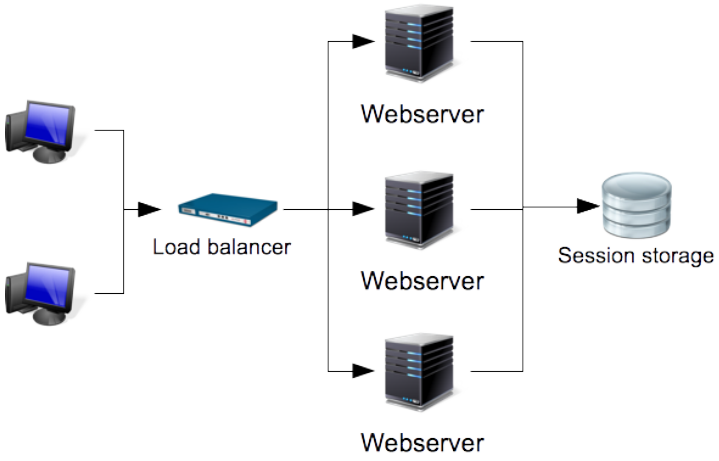
In addition database session state copes better with a clustered environment than a common server session state. Since the session is stored in the database layer, multiple web-servers can have access to it (session migration) and therefore it is not necessary to have session affinity (only one server which handles the session).

Remark: Remember, at least one attribute (user identifier / session id) has to be stored always on the client side.

Revised design

The business application (most probably only the backend) and the middleware are affected by the session state migration. Current middleware (e.g. RDBMS) can be used or new middleware (e.x. noSQL Key-Value-Store) has to be put in place.

The first step would be to determine all necessary session attributes. For a new application you are free in the design and choice of storage. There are different options to use as a session storage (database, in-memory cache, ...)



The figure shows that the session is stored in a separate layer. If it was stored on the web-server/application server itself, there would be the problem, that the application can loose the clients state if the load balancer reroutes the client to another server.

Pitfalls to avoid

- Make sure to implement a cleanup mechanism. When a user leaves without a proper logout, a session state entry is left behind in the database. Add an expiration timestamp field in the database and a processes which deletes expired entries.
- Avoid implementing an own session id generation algorithm. Use application server functionalities or follow strict security requirements to avoid session id prediction or hijacking.

Architectural decision(s) to be revisited

- Choice of session state management

Affected components and connectors (if modelled explicitly)

- Application software (Business software)

Tasks

- Determine used session parameters :: Analysis Task
- Evaluate session state store :: Decision Task
- Create attribute migration plan :: Design Task
- Integrate new session storage :: Integration Task
- Implement/configure session storage usage :: Development Task
- Execute load test :: Testing Task

References (links):

- Patterns of Enterprise Application Architecture [Martin Fowler]
  http://martinfowler.com/books/eaa.html
- Session management [Wikipedia]
  https://en.wikipedia.org/wiki/Session_(computer_science)
- Open Web Application Security Project - Session_Management_Cheat_Sheet
  https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

Migrate web-application into a cloud-environment to provide scalability

Created	Modified	Status
12/19/2015 at 17:09:49	12/19/2015 at 17:09:49	published

Context (viewpoint, refinement level)

- Physical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Deployment View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/

Quality attributes (forces)

- Scalability https://en.wikipedia.org/wiki/Scalability
- Elasticity https://en.wikipedia.org/wiki/Elasticity_(cloud_computing)
- Portability https://en.wikipedia.org/wiki/Porting

Smells

Name / Description / Questions	Group	TDI
Scalability issue	Performance	hm
An application is not able to react on variable requests or work load and therefore has high latency during peak times. • Are the response times of your application not constant? • Does your application not scale well for a high volume of requests?		
Standalone web-application does not scale well	Performance	hm
Your web application server process is using a lightweight server framework (e.g. Netty) which does not perform well due to the increase of traffic in your system. • Does your web-application run in a standalone web application process which does not scale well?		

Description

Initial position

Web application frameworks such as Play provide an easy way to create a web-application-server process. Those processes are very lightweight and perform well. There is no need to create a web application archive and deploy it to an application server or web container. But this server processes will reach their limits as soon as the traffic on your application rises.

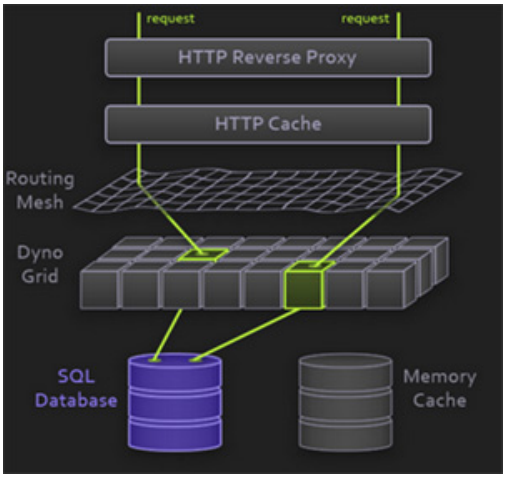
There is no direct and easy possibility to increase the capacity of the process. You would have to install the application multiple times in different directories. Start the application in each folder with different ports and then setup a web-server (e.g. Apache HTTP Server) which does the Port-Forwarding and the load balancing between the different processes.

Revised design

The much easier way to scale your application is to install it onto one of the many cloud services available in the internet. In a cloud environment you install your application once and then you can scale your application by configuration or automated (up to a defined limit). The elastic load balancer of a cloud environment will ensure that your application will be instantiated and scaled automatically. The pricing in a cloud environment will be done automatically as well, depending on your applications capacity needs.

The following describes an example using a Java Play application and the cloud service Heroku:

For Play-applications Heroku is one of the cloud environments which support the Play-framework (with a Scala build environment) out of the box. There are a few steps needed to deploy your play application to Heroku. The tasks of this Architectural Refactoring list the necessary steps to deploy a Play application onto Heroku Cloud Application Platform.



http://www.coloandcloud.com/wp-content/uploads/2012/05/heroku-infrastructure.jpg

The figure shows the architecture in Heroku. The so called "Dynos" are lightweight Linux containers which builds an instance of your application. To scale out your application you can add dynos. All the dynos have access to the same database.

Pitfalls to avoid

- Make sure your application is production ready (if needed). Which means use additional cloud modules for logging, alerting and redundancy.
- Make sure your application complies with the common cloud application patterns. E.g. use client- or distributed-session (database) which is accessible on every application instance.
- Be aware about the applied pricing models. The pricing is dynamic and depends on your applications capacity needs.
- Don't forget to inform you about the existing SLA (Service Level Agreement), to avoid service outtakes of your application (temporarily unused application may go into a sleep mode).

Architectural decision(s) to be revisited

- Choice of standalone web-application process

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Application configuration

Tasks

- Commit application sources into a git repository :: Development Task
- Add Heroku git-url as a remote repository :: Integration Task
- Create Heroku Cloud account :: Admin Task
- Create an application in the Heroku Cloud :: Admin Task
- Add Database Modul in Heroku :: Integration Task
- Commit your sources to Heroku :: Integration Task
- Document Design changes which were necessary for Cloud Migration :: Design Task

References (links):

- Getting Started with Scala and Play on Heroku
  https://devcenter.heroku.com/articles/getting-started-with-scala
- Cloud Computing Patterns [Fehling, Leymann, Retter, Schupeck, Arbitter]
- Play Framework
  https://www.playframework.com/

Scatter data to increase scalability

Created	Modified	Status
01/22/2016 at 10:23:04	01/22/2016 at 10:23:04	published

Context (viewpoint, refinement level)

- Physical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759

Quality attributes (forces)

- Scalability https://en.wikipedia.org/wiki/Scalability
- Robustness https://en.wikipedia.org/wiki/Robustness_(computer_science)

Smells

Name / Description / Questions	Group	TDI
Scalability issue	Performance	hm
An application is not able to react on variable requests or work load and therefore has high latency during peak times. • Are the response times of your application not constant? • Does your application not scale well for a high volume of requests?		
Bottleneck component	Performance	hh
If multiple components or system parts are running on the same hardware or even on separate hardware, one part can impair the rest of the system. • Does a component or a part of your software system impair the whole system?		

Description

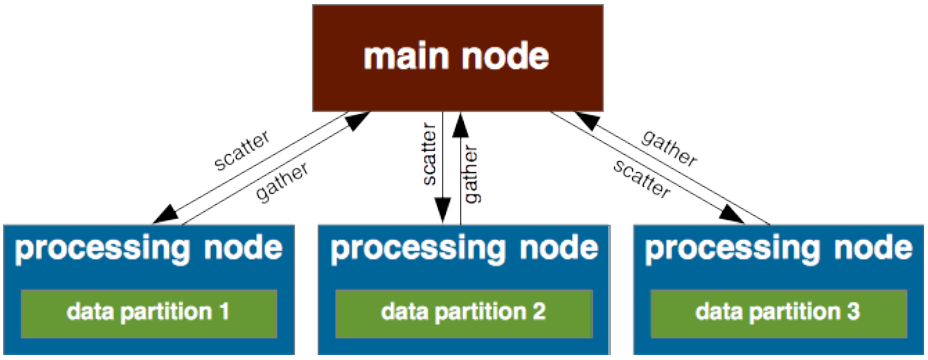
Initial position

Processing a large amount of data on a single server/node can have many disadvantages. If a software system accesses and manipulates data with only one process/thread, the processing might not perform well and can cause queueing for other requests. If a software system acts with multiple processes/threads, the concurrent accesses might obstruct themselves and the hard-disk becomes a bottleneck. A common solution for such contentions is to cache data in the memory, which can be accessed faster. The revised design proposes an extended solution which copes well with big data.

Revised design

The proposed solution is to distribute the data over more than one server/node. The data should be partitioned into distinct pieces and shard across the multiple nodes. A dedicated main node has to be put in place, which acts as an aggregator. The main node knows (or can request) the range of each data partition (which has to be sorted by an index) on the specific data node. A client can then execute a query via the main node, whereon the main node executes the query only on the concerned data nodes and aggregates the results of each node.

This behavior follows the pattern "scatter and gather" which is commonly used by modern NoSQL databases (e.g. MongoDB, MemSQL). The data storage ist spread over many different physical servers. This allows parallel and quick big data analysis. This conforms to the principle of horizontal scaling (scale out), by adding new physical servers (commodity hardware) to improve the performance of a system. A master node executes the requests on the other nodes. They calculate a local result which will then be aggregated by the master node.



The figure shows a main node which scatters the query to the different processing nodes. The processing nodes execute the query on their part of the data. The main node gathers the different results from the processing nodes and aggregates them.

Pitfalls to avoid

- Do not forget to think about resilience. A main node as well as the different data nodes should be kept redundant to avoid data loss in case of a break-down of one node. This is important to provide a robust system.
- Avoid to create too many nodes, to keep maintenance and operation costs low. Find an optimal balance between the amount of nodes, costs and performance (execute a load test).
- Make sure to choose the most suitable data attribute for the sharding/partitioning index. This depends heavily on the type of queries you execute on your system.

Architectural decision(s) to be revisited

- Choice of tier architecture
- Choice of data partitioning and layout
- Choice of data source/store

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Middleware
- Hardware (server)

Tasks

- Assess scalability impact (concurrency level) :: Analysis Task
- Choose approriate sharding key :: Architectural Decision Task
- Integrate new data storage :: Integration Task
- Execute integration test :: Testing Task
- Execute load test :: Testing Task
- Document service level agreement :: Documentation Task

References (links):

- Enterprise Integration Patterns [Hope, Woolf]
- Sharding Pattern [Microsoft]
  https://msdn.microsoft.com/en-us/library/dn589797.aspx
- Considerations for Selecting Shard Keys [MongoDB]
  https://docs.mongodb.org/manual/tutorial/choose-a-shard-key/

Introduce dependency injection to improve maintainability and testability

Created	Modified	Status
12/05/2015 at 18:22:31	12/07/2015 at 10:20:02	published

Context (viewpoint, refinement level)

- Development View [4+1, Kruchten / Rozanski, Woods] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Functional View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Maintainability https://en.wikipedia.org/wiki/Maintainability
- Testability https://en.wikipedia.org/wiki/Testability

Smells

Name / Description / Questions	Group	TDI
Difficulty to simulate interface (testing)	Maintainability	mm
It is hard to test a component due the difficulty to simulate an interface. • Do you have difficulties to simulate an interface while testing an application? • Is modul testing of an application difficult?		

Description

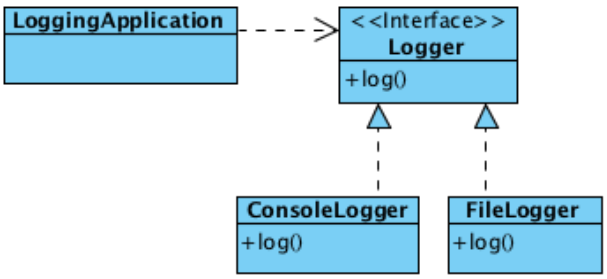
Initial position

If an application is difficult to maintain or test, it is often because its behavior is rigid. If you want to simulate an interface, you would most likely have to change the source code since the interface connection is not temporarily replaceable. In a unit test it is desirable to temporarily simulate an interface, for example to introduce static test data (in case the connection to the real interface component does not exist in a development/testing environment).

The consequences, if an application only uses concrete implementation classes instead of interfaces, is tight coupling. Which contradicts with the desire to have low coupling and high cohesion. (see https://de.wikipedia.org/wiki/GRASP).

Revised design

The solution, to make an application easier to test and easier to change its behavior, is to use a dependency injection framework. A dependency injection frameworks allows to change the behavior of an application at runtime. A configuration file or annotations tell the dependency injection framework where to inject which behavior. For this it is necessary to use interface classes and (multiple) classes with a concrete implementation of these interfaces.



The figure shows an example of an interface with two concrete implementations. The LoggingApplication is using the interface Logger. With a dependency injection framework it is now possible to configure which concrete implementation of Logger

should be instantiated at runtime at the position where the LoggerApplication is using the interface. This makes the behavior modifiable at runtime. E.g. in production you would like to log into a file, for which the FileLogger provides the necessary implementation, but in a development environment you would like to see your loggings directly in the console.

Pitfalls to avoid

- Make sure to avoid low cohesion. Do not create interfaces with combined functionalities, which means the interfaces should only contain functions which are strongly related (high cohesion).
- Make sure to use injection API (Application Programming Interface) where provided. E.g. in Java this is specified in the JSR (Java Specification Requests) 330. This allows to replace the underlying dependency injection framework without any change of the program code.

Architectural decision(s) to be revisited

- Choice of application frameworks

Affected components and connectors (if modelled explicitly)

- Application software (Business software)

Tasks

- Choose dependency injection framework :: Decision Task
- Implement interface class, concrete classes and configure dependency injection :: Development Task
- Implement unit testcases and mockup objects :: Development Task
- Rethink programming methods (e.g. test driven development) :: Leadership Task
- Execute integration test :: Testing Task
- Use UML to document the dependency injection coherence (class diagram) :: Documentation Task

References (links):

- Inversion of Control Containers and the Dependency Injection pattern [Martin Fowler]
  http://martinfowler.com/articles/injection.html
- Dependency injection [Wikipedia]
  https://en.wikipedia.org/wiki/Dependency_injection
- Mock Object [Wikipedia]
  https://en.wikipedia.org/wiki/Mock_object

Split components to reduce overly tight coupling

Created	Modified	Status
12/06/2015 at 17:07:09	12/07/2015 at 10:41:50	published

Context (viewpoint, refinement level)

- Logical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Maintainability https://en.wikipedia.org/wiki/Maintainability
- Dependency https://en.wikipedia.org/wiki/Coupling_(computer_programming)
- Responsibility https://en.wikipedia.org/wiki/Separation_of_concerns

Smells

Name / Description / Questions	Group	TDI
Large number of component responsbilities and collaborations	Maintainability	mh
The component violates good OOAD and CBSE (Component-Based Software Engineering) practices, e.g. single responsibility (see video: https://www.youtube.com/watch?v=BD9VaTiLMpM). • Do any components expose multiple service responsibilities that are not closely related to each other? • Does a component violate the architectural principle of high cohesion/low coupling? • Does a component violate the "Single Responsibility" rule?		
Unnecessary tight coupling	Dependency	mm
It is the case that a component has a strong dependency to one or many other component(s). This can cause the problem, that changes or extinctions in one component have an unnecessary effect to other components. • Does a change in a component imply a change in one or many other compnent(s)? • Does a problem in a component have effect in one or many other component(s)? • Do you have difficulties to replace an interface component? • Is one component in your system strongly dependent from an other?		

Description

Initial position

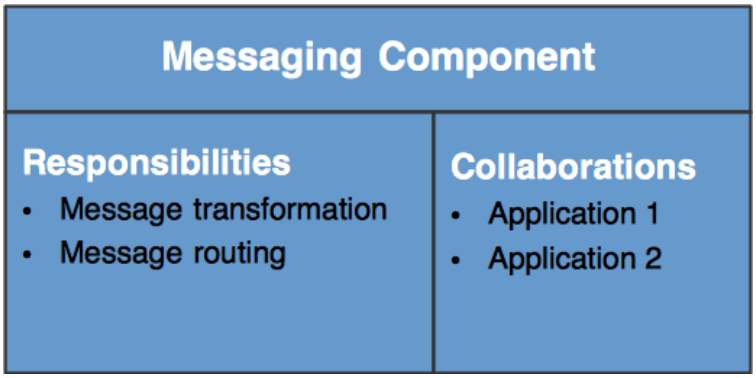
A component which has grown over a period of time, might have gained functionalities which are not necessarily related to each other. This creates unwanted dependencies and overly tight coupling. For example if a component contains a functionality which could be useful for another component, it might no be accessible. This leads to redundant implementations of functionalities which causes unnecessary costs. Also the replacement or change of such a compound component might be difficult (bad maintainability).

E.g. two components which interact with each other have both their own implementation of a message transformation functionality. This can lead to faulty communication and instability if changes are not done synchronously on both sides.

Revised design

The proposed solution is to hold a system/component/code review to reevaluate the togetherness of the functionalities. General functions which might be useful for other components should be extracted to a separate component with a standard interface. A constructive method to review an existing application is to use CRC (Class-responsibility-collaboration)-Cards. CRC-Cards allow to visualize the responsibilities and collaborations of single application parts and are a helpful tool of object-oriented analysis and design. The CRC-Card method should be executed in a workshop, including directly involved and project independent persons. Decide which separation of components makes sense and plan the implementation accordingly.

In the before mentioned example of the two communicating components (initial position). The solution would be to extract the message transformation into a separate component. For example by using a message router.



The figure shows an example of a CRC card which lists the responsibilities and collaborations of a specific program class.

Pitfalls to avoid

- Avoid to create too many components. This will increase the complexity of your system and rise your maintenance costs.
- Consider using third party libraries/products for common functionalities, instead of programming your own frameworks.

Architectural decision(s) to be revisited

- Functional partitioning
  http://www.encyclopedia.com/doc/1O11-functionalpartitioning.html

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Other applications

Tasks

- Hold CRC-Card workshop :: Hold Meeting Task
- Create new component :: Systemmanagement Task
- Choose a standard interface for component interaction :: Architectural Decision Task
- Move one or more responsibilities over to new component :: Architectural Decision Task
- Update all client collaborations of component to use new one :: Development Task
- Update component diagram :: Modelling Task

References (links):

- OLAF's Method Tailoring Guide
  http://www.ifs.hsr.ch/Method-Selection-and-Tailoring.13195.0.html
- Separation of concerns [Wikipedia]
  https://en.wikipedia.org/wiki/Separation_of_concerns
- Tailoring Software Infrastructures [Christian Dörner]
- Architect's Toolset - CRC Cards [Michael Stal]
  http://stal.blogspot.ch/2006/12/architects-toolset-crc-cards.html

Upgrade of third-party software to preserve maintainability

Created	Modified	Status
12/04/2015 at 10:19:43	12/07/2015 at 09:34:09	published

Context (viewpoint, refinement level)

- Physical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Deployment View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Software Level
- Vendor asset level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Maintainability https://en.wikipedia.org/wiki/Maintainability
- Performance https://en.wikipedia.org/wiki/Computer_performance
- Costs https://en.wikipedia.org/wiki/Cost
- Portability https://en.wikipedia.org/wiki/Porting

Smells

Name / Description / Questions	Group	TDI
Software compatibility not given anymore	Maintainability	hh
It is the case that a component you use, is depending on old technology (e.g. interfaces, frameworks) which is not supported by newer components in your system anymore. • Is a component to old to be compatible with other newer components?		
Misbehaving third party software (buggy)	Performance	hh
A third party software is faulty and has to be replaced with a bugfix version. • Is a third party product which you use faulty?		

Description

Initial position

A software system which uses third party software should always kept up to date. Which means it is necessary to upgrade the third party product to newer releases on a regular basis. Especially for commercial products which are not open source and bugs can not be fixed by your own. Since a version of a third party software most likely has a limited support timeframe. E.g. 5 years. This means the third party vendor will stop providing bugfixes for versions older than a specified timeframe.

This is a big risk and can jeopardize a whole software system. Particularly for legacy systems which are often kept operational longer than they should and in a dependency to an other component. E.g. if a component, which has a connection to a legacy system, changes and causes a problem in the outdated third party software.

Further impacts could be missing performance due to an increase of activity from peripheral systems. Or missing portability, if for example the hardware strategy of a company changes (server platform).

Revised design

An upgrade of a third party software has to be carefully planed. The more versions were skipped the higher the costs and risks for the migration. Study the change logs and the current interface documentations in detail. Changes can be significant, e.g. a completely new interface (different protocol or data structure). A server platform or operating system change can require completely different binaries. Be aware that the testing effort can be as high as for a first integration of the product. Extensive regression test is inevitable (in modul and integration test).

Setup a migration road map and consider parallel operation of both the old and the new version for a defined period. Either migrate all processes/components at once and use the old version as a fall back, or migrate single processes/components step by step to the new version. Elaborate a fallback plan in case of a problem. After production installation get confirmations of all affected stakeholders.

Pitfalls to avoid

- Consider to request onsite support from the third party vendor during the production deployment.
- Do upgrades of software products in legacy systems only as a last solution. Always consider the removal or replacement of the component first.

Architectural decision(s) to be revisited

- Choice of thirdparty vendor
- Choice of third party product
- Choice of software version

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Middleware
- Interfaces

Tasks

- Create migration road map :: Documentation Task
- Create a fallback plan :: Documentation Task
- Assess security impact (security architecture review) :: Analysis Task
- Assess scalability impact (concurrency level) :: Analysis Task
- Assess compatibility of new software version/product :: Analysis Task
- Assess future readiness of new version/product :: Analysis Task
- Package new software version/product :: Integration Task
- Execute system regression tests :: Testing Task

References (links):

- Continuous Integration [Duvall, Matyas, Glove]
- Use and integration of third-party components in software development [Leena Arhippainen] http://www.vtt.fi/inf/pdf/publications/2003/P489.pdf
- Integration Testing [Wikipedia] https://en.wikipedia.org/wiki/Integration_testing

Introduce column-store to improve the performance of analytical data processing

Created	Modified	Status
12/31/2015 at 11:20:44	12/31/2015 at 11:20:44	published

Context (viewpoint, refinement level)

- Logical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Performance https://en.wikipedia.org/wiki/Computer_performance

Smells

Name / Description / Questions	Group	TDI
Slow database query response times	Performance	mh
It is the case that your database queries have slow response times. • Do database queries in your system have high latency?		
Slow analytical data processing	Performance	mh
It is the case that analytical data queries (aggregations and table scans) are performing badly. • Do you have an analytical data processing system which is performing badly? • Are aggregation and table scan queries performing badly?		

Description

Initial position

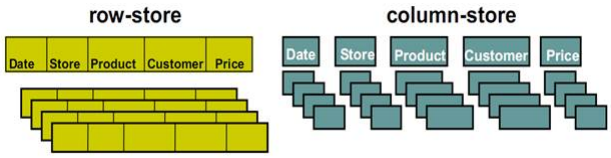
The first step of business process optimization is often based on business process automation which is again mostly based on transactional data processing. E.g. in a client data base; making an entry with new a client record, show or update client data. Which are all operations on single data records. The procedure of business process improvement through gaining business intelligence, becomes more and more important. Aggregating business intelligence is mostly based on analytical data processing of historical data (in contrary to the transactional data processing).

The online analytical data processing (OLAP) consists of aggregated queries and large table scans, which differs from single record queries and manipulations in online transaction processing (OLTP). To perform well in OLAP cases it is advised to use columnar oriented data stores.

Revised design

The solution to perform better in OLAP, this architectural refactoring recommends to use data stores which are capable of creating columnar tables. Columnar tables partition and arrange the record data significantly different than row tables. Instead of storing the records with all their attributes sequentially, they store the attributes of all records together by column. This makes aggregations in one column over all records much faster.

Most databases which support columnar stores, provide them via standard SQL. Therefore the only thing to do, is storing the data in a column-oriented table and respect the intended purpose of columnar tables when executing queries. Which means execute mainly aggregation queries (SQL functions: avg, sum, min, max, count) and table scans such as: SELECT price FROM products WHERE price <> '50';



(src: http://www.codeproject.com)

The figure shows how row-stores differ from column-stores. In a row-store all attributes of one record are stored together and the records are stored sequentially. In a column-store the attributes of all records are stored together by column.

Pitfalls to avoid

- Make sure to avoid executing a lot of small transactions (inserts/update/deletes). Data manipulation in columnar stores requires always multiple I/O operations per record, due to the partitioning layout (except when updating only one attribute of one record).
- Make sure to avoid complex join statements. When necessary, do joins only on key columns.

Architectural decision(s) to be revisited

- Choice of data source/store
- Choice of data partitioning and layout

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Middleware

Tasks

- Determine data processing purpose :: Analysis Task
- Choose columnar data store :: Decision Task
- Purchase new data store :: Commercial Task
- Integrate new data storage :: Integration Task
- Put data export process in place (move historical data to columnar stores) :: Admin Task
- Document data migration plan :: Documentation Task

References (links):

- Column-Stores vs. Row-Stores: How Different Are They Really? [Abadi, Madden, Hachem]
  http://db.csail.mit.edu/projects/cstore/abadi-sigmod08.pdf
- The Design and Implementation of Modern Column-Oriented Database Systems [Abadi, Boncz, Harizopoulos]
  http://db.csail.mit.edu/pubs/abadi-column-stores.pdf
- Online analytical processing [Wikipedia]
  https://en.wikipedia.org/wiki/Online_analytical_processing
- Business intelligence [Wikipedia]
  https://en.wikipedia.org/wiki/Business_intelligence
- VLDB 2009 Tutorial on Column-Stores [Daniel Abadi]
  http://de.slideshare.net/abadid/vldb-2009-tutorial-on-columnstores

Introduce in-memory database to improve high volume transaction performance

Created	Modified	Status
12/06/2015 at 14:58:51	12/07/2015 at 10:24:37	published

Context (viewpoint, refinement level)

- Physical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Deployment View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Performance https://en.wikipedia.org/wiki/Computer_performance
- Scalability https://en.wikipedia.org/wiki/Scalability

Smells

Name / Description / Questions	Group	TDI
Locks/Deadlocks	Performance	hh
Is your application being locked due to concurrent operations. E.g. concurrent data access or two processes trying to call a critical (synchronized) program section. • Are some threads or processes blocking other threads/processes? • Are concurrent database accesses blocking your application?		
Low transaction throughput	Performance	hm
It is the case that an application can not keep up with a high transaction volume. • Does your application process a lot of transactions and does not perform well?		

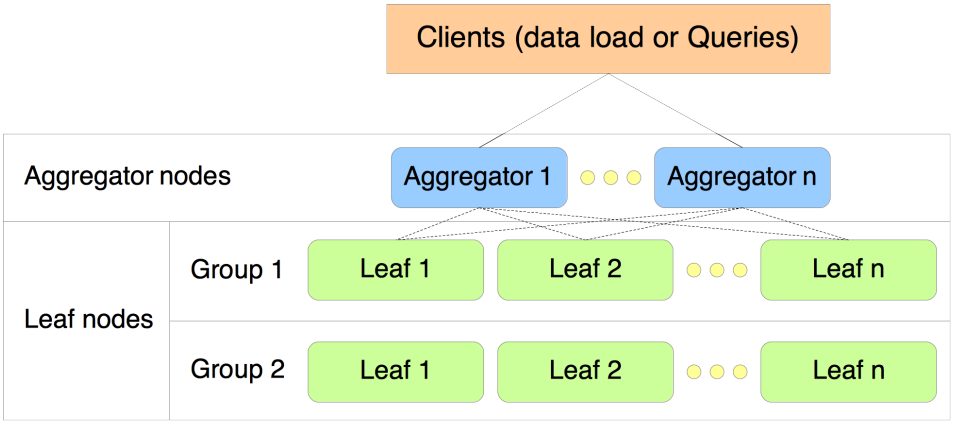
Description

Initial position

Software systems which have to process a lot of database transactions (OLTP (Online Transaction Processing)) often struggle with I/O being the bottleneck of the application. Real Time Ad Bidding is an example where a huge amount of transaction has to be processed. Real Time (Ad) Bidding are online marketplaces where advertising spots are auctioned in realtime. To compete with this amount of transaction a disk-based database might reach its limits. Modern solid state drives or flash based memory can reduce access latency, but can not compete with the access times of random access memory (RAM).

Revised design

The solution to be able to process a huge amount of transactions is to use a (distributed) in-memory database. The main memory has significantly lower response times than a hard-disk. It is advisable to keep a hybrid system which provides disk-based and memory-based storage. For example a compressed column-store to save historical data to the disk. Very common are databases which are distributed over many systems (commodity hardware). The data is scattered over different nodes and the query is aggregated on a main node. This provides resilience and distributed query performance on a large amount of data.



The figure shows the architecture of MemSQL, which is an example of a distributed in-memory database. The leaf nodes contain the distributed data. The aggregator nodes delegate the queries and collect the results from the leaf nodes. Arranging the leaf nodes in groups, provides resilience and increased performance.

Pitfalls to avoid

- Keep in mind, that some in-memory stores not always provide durability. Or the standard configuration does not enable it. This means that the data is stored only in the volatile memory and will be lost after a system shutdown.
- Avoid to scale up a system if the capacity of your in-memory database is reached. Do horizontal scaling (add new servers) and make sure the data is distributed across the systems. This provides resilience and eliminates a single point of failure.

Architectural decision(s) to be revisited

- Choice of data source/store

Affected components and connectors (if modelled explicitly)

- Middleware

Tasks

- Create attribute migration plan :: Design Task
- Evaluate in-memory database :: Decision Task
- Integrate new data storage :: Integration Task
- Execute load test :: Testing Task
- Execute failover test :: Testing Task
- Update software architecture document (SAD) :: Documentation Task

References (links):

- In-memory database [Wikipedia]
  https://en.wikipedia.org/wiki/In-memory_database
- Online Transaction Processing [Wikipedia]
  https://en.wikipedia.org/wiki/Online_transaction_processing

Switch server architecture from SPARC to x86 to optimize the performance

Created	Modified	Status
12/03/2015 at 09:27:07	12/07/2015 at 08:58:02	published

Context (viewpoint, refinement level)

- Deployment View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Product level

Quality attributes (forces)

- Performance https://en.wikipedia.org/wiki/Computer_performance
- Costs https://en.wikipedia.org/wiki/Cost
- Scalability https://en.wikipedia.org/wiki/Scalability

Smells

Name / Description / Questions	Group	TDI
Lack of performance with computationally intensive, single threaded applications	Performance	hm
You have large unit of works which need a lot of cpu power and are not parallelized (e.g. rendering processes).		
• Does your system have computationally intensive unit of works?		
High hardware costs	Performance	ll
The costs for your systems hardware are to high.		
• Are your hardware costs to high?		

Description

Initial position

A SPARC (Scalable Processor ARChitecture) system is highly scalable multicore system. In simple words, a SPARC system allows to run much more concurrent threads than an x86 system, but mostly at a lower clock rate per CPU core. Therefore choosing between a SPARC and a x86 system highly depends on your applications behavior and needs.

E.g. an application which has to render large printing streams for a high volume printer will run better on a x86 System. The higher clock rate processes the CPU intensive rendering faster, than a SPARC system. Splitting the input data in to many small pieces an rendering them concurrently, improves the performance on a SPARC system.

Revised design

This Architectural Refactoring describes the move from a SPARC to a x86 system. There is as well the option to optimize your application on a SPARC system, for example to re-portion your unit of work. Make the unit of work smaller but run more concurrent threads.

The switch of the processor architecture to x86 (commodity hardware) most likely involves a switch of the operating system. The most common operating system for SPARC is Solaris. On a x86 system a Linux distribution is most common. In fact it is possible to run the mentioned operating systems on either of the architectures, but it is not recommended since there is only little software support (middleware, third party products and libraries).

A switch of the operating system entails replacing all software versions so they are able to run on the new system. Therefore it is often necessary to involve a third party vendor of the used applications and middleware for support.

A switch of the system architecture might need a redesign of the applications threading behavior. Do different test with different level of concurrency to find what brings the best performance.

Pitfalls to avoid

- Avoid running too many threads on a x86 system. If the operating system is doing too many context switching (switching threads running on the CPU) the system will drop the performance significantly.
- Make sure to scale out if your system reaches its limits. Add new servers in a cluster which allows to process more tasks at the same time.
- Make sure of appropriate concurrency control. Which means to avoid accessing resources which need locking.

Architectural decision(s) to be revisited

- High parallelisation
- Choice of hardware plattform
- Choice of operating system

Affected components and connectors (if modelled explicitly)

- Hardware (server)
- Application software (Business software)
- Operating system

Tasks

- Assess security impact (security architecture review) :: Analysis Task
- Assess scalability impact (concurrency level) :: Analysis Task
- Procure new hardware :: Commercial Task
- Switch operating system (Solaris to Linux) :: Systemmanagement Task
- Upgrade / version switch middleware (x86 version instead of SPARC) :: Integration Task
- Port the application to the new hardware and operating system :: Development Task
- Interface changes (reconfiguration) :: Integration Task
- Execute system regression tests :: Testing Task

References (links):

- Solaris Application Programming [Darryl Gove]
- Concurrency control [Wikipedia]
  https://en.wikipedia.org/wiki/Concurrency_control
- x86 [Wikipedia]
  https://en.wikipedia.org/wiki/X86

Extract batch application from application server to reduce complexity

Created	Modified	Status
12/02/2015 at 11:24:00	12/06/2015 at 22:24:00	published

Context (viewpoint, refinement level)

- Deployment View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Software Level

Quality attributes (forces)

- Complexity https://en.wikipedia.org/wiki/Programming_complexity
- Dependency https://en.wikipedia.org/wiki/Coupling_(computer_programming)
- Costs https://en.wikipedia.org/wiki/Cost
- Portability https://en.wikipedia.org/wiki/Porting

Smells

Name / Description / Questions	Group	TDI
Unnecessary tight coupling	Dependency	mm
It is the case that a component has a strong dependency to one or many other component(s). This can cause the problem, that changes or extinctions in one component have an unnecessary effect to other components. • Does a change in a component imply a change in one or many other compnent(s)? • Does a problem in a component have effect in one or many other component(s)? • Do you have difficulties to replace an interface component? • Is one component in your system strongly dependent from an other?		
High licenses costs	Performance	ll
It is the case that you have too high costs for third party software in your system. • Does your software system have high costs for third party licenses?		
High system complexity due to excessive middleware usage	Complexity	mm
Your system uses many middleware components which makes it difficult to configure, maintain and operate. • Is your system difficult to maintain due to the use of many middleware components? • Is it difficult to determine the root production problem due to the use of many middleware components?		

Description

Initial position

Application servers provide a lot of functions and technologies for web applications as well as for batch applications. But there are applications which use only a small part of the possibilities of an application server. For these it is in many cases not necessary to maintain and operate an application server. A lot of the functions used in applications servers are provided by frameworks which are much more lightweight and easier to maintain. Using such a framework can reduce the complexity and costs of a software component. Furthermore the dependencies can be reduce and the portability can be improved. The application is not dependent on the functionality implementations of the application server anymore and the used frameworks can be exchanged easily.

Revised design

To extract an application from an application server, it is necessary to analyze the functionalities which are currently provided by the application server and decide how they should be replaced. For example, thread-pooling, service connection interfaces, messaging interfaces, load-balancing, transaction management, logging, authentication, security, etc.

Always consider existing frameworks to replace common functionalities, instead of implementing them yourself.

The benefit of this extraction is the reduction of the system complexity and maintainability effort. Creating a lightweight standalone application can save you system resources which formerly have been used by an application server. Application (re-) starts and stops can be significantly faster.

An example is a java batch application processing messages from a messaging queue. With the JMS (Java Message Service) API (Application programming interface) it is possible to handle the transactions on the message queue and there is no need to use an application server for its transaction handling.

Pitfalls to avoid

- Avoid to implement own frameworks (use existing OSS (Open Source Software) or proprietary software).
- Avoid to try to extract an application which requires a lot of the application servers functionality. This will increase the complexity of the application (a patchwork of frameworks).

Architectural decision(s) to be revisited

- Choice to use an application server
- Choice of thirdparty vendor

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Interfaces
- Middleware
- Scheduler

Tasks

- Assess security impact (security architecture review) :: Analysis Task
- Assess scalability impact due to the disposal of the application server :: Analysis Task
- Analyse the necessity of an own transaction management :: Analysis Task
- Analyse logging mechanism :: Analysis Task
- Compare current to new dependencies (with/without application server) :: Analysis Task
- Update interface specifications :: Design Task
- Implement standalone application :: Development Task
- Execute system regression tests :: Testing Task

References (links):

- Why Java Batch? [Kaushik Mukherjee]
  https://developer.ibm.com/wasdev/docs/java-batch/
- Batch processing [Wikipedia]
  https://en.wikipedia.org/wiki/Batch_processing
- Java application servers are dead! [Eberhard Wolff]
  https://jaxenter.com/java-application-servers-dead-112186.html

Replace SOAP-based Web service with RESTful HTTP resources to reduce complexity

Created	Modified	Status
10/03/2015 at 18:18:14	10/14/2015 at 06:09:01	published

Context (viewpoint, refinement level)

- Development View [4+1, Kruchten / Rozanski, Woods] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Technology Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Complexity https://en.wikipedia.org/wiki/Programming_complexity
- Simplicity https://en.wikipedia.org/wiki/KISS_principle
- Flexibility https://en.wikipedia.org/wiki/Flexibility_(engineering)
- Productivity https://en.wikipedia.org/wiki/Productivity

Smells

Name / Description / Questions	Group	TDI
Complex interface	Complexity	hm
Is it the case that a component has a complex interface to another component which makes changes difficult and makes it difficult for interface partners to program against it. • Is it difficult to implement changes for an interface? • Do interface partners have problems programming against your interface?		
Scalability issue	Performance	hm
An application is not able to react on variable requests or work load and therefore has high latency during peak times. • Are the response times of your application not constant? • Does your application not scale well for a high volume of requests?		

Description

Initial position

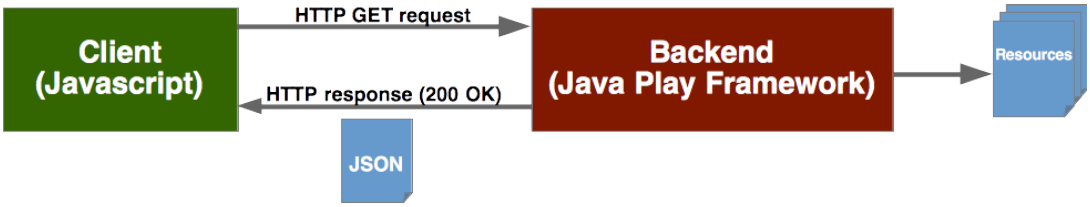
SOAP (Simple Object Access Protocol) based Webservices might often have a high complexity (use of schemas, RPC (Remote Procedure Call) configuration, use of WSDL (Web Services Description Language), UDDI (Description Discovery and Integration)) and often use XML. Whereby a typical REST (Representational State Transfer) Service interface provides the main CRUD operations on a specific resource using standard HTTP protocol and the lightweight JSON (JavaScript Object Notation) data representation (can have any other format (XML, text, binary, ...), but JSON becomes more and more popular). Therefore it is possible access REST services with simple HTTP requests which does not require specific tools (web browsers or WGET tool are sufficient).

Revised design

The main difference between SOAP and REST Services is speaking to an Object/Resource rather than functions. In a SOAP Service you normally provide arguments to functions which then act depending on this arguments. In REST you request, provide or change resources. See Details on Richardson Maturity Model described by Martin Fowler: http://martinfowler.com/articles/richardsonMaturityModel.html

The REST architecture complies well with modern Javascript based web applications, since they are able to call the backend services via simple HTTP requests. An important property of REST is that it is stateless. Which means every request should contain all the necessary information to interpret the message. There should be no additional state (on the server or client side) necessary to understand and process the message.

Since the preferred message format for REST services is JSON it complies very well with modern noSQL databases which can store and query JSON objects without conversion. JSON is a much more compact format than XML to which SOAP services are bound to.



The figure shows an example of a REST service request of a Javascript client to a Java Play Backend. The backend access data resources and provides them as a JSON format via the HTTP response.

Pitfalls to avoid

- Keep in mind that the XML schema validation is a useful tool to validate service input. For JSON schemas are not that common yet.

Architectural decision(s) to be revisited

- Choice of service interface
- Choice of Protocol

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Interfaces

Tasks

- Assess security impact (security architecture review) :: Analysis Task
- Evaluate REST web service framework :: Decision Task
- Change generation of data model representation (from XML to JSON) :: Development Task
- Update interface specifications :: Design Task
- Change interface applications :: Development Task
- Execute load test :: Testing Task

References (links):

- Cloud Computing Patterns [Fehling, Leymann, Retter, Schupeck, Arbitter]
- Richardson Maturity Model [Fowler]
  http://martinfowler.com/articles/richardsonMaturityModel.html
- Enterprise Integration Patterns [Hope, Woolf]
- RESTful Web Services vs. “Big” Web Services
  http://www8.cs.umu.se/kurser/5DV095/HT09/literature/restvsbig.pdf

Consolidate and simplify product usage to reduce dependencies

Created	Modified	Status
12/04/2015 at 11:53:03	12/07/2015 at 10:06:53	published

Context (viewpoint, refinement level)

- Deployment View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Software Level

Quality attributes (forces)

- Dependency https://en.wikipedia.org/wiki/Coupling_(computer_programming)
- Performance https://en.wikipedia.org/wiki/Computer_performance
- Complexity https://en.wikipedia.org/wiki/Programming_complexity
- Portability https://en.wikipedia.org/wiki/Porting
- Maintainability https://en.wikipedia.org/wiki/Maintainability

Smells

Name / Description / Questions	Group	TDI
High system complexity due to excessive middleware usage	Complexity	mm
Your system uses many middleware components which makes it difficult to configure, maintain and operate. • Is your system difficult to maintain due to the use of many middleware components? • Is it difficult to determine the root production problem due to the use of many middleware components?		
Third party vendor dependency	Dependency	hm
Is your system dependent on a third party product. • Are you bound to costly third party license costs? • Is your system strongly dependent on a third party product?		

Description

Initial position

A software system gains complexity the more functionality is added to it. Which is almost unavoidable the longer a system exists. Functionality is often added by introducing new libraries, middleware or other third party products. All this components increase the complexity and the dependency among the components. The more components a system contains the more complex it is to maintain.

A major problem is the integration of third party products based on intransparent decisions. A tool should always be considered and evaluated after the definition of the requirements. It is bad practice if the management dictates the usage of a tool or a tool dictates the requirements. Third party products often provide a large set of functionalities and features from which usually only a fraction is need.

Revised design

Replacing third party products or resign a middleware component is not easy. Often exist strong dependencies or there are contracts to fulfill. This Architectural Refactoring is not a general recommendation to replace third party products. But a suggestion to reevaluate them based on requirement needs. An alternative to a third party product or a middleware component can be a lightweight framework. Example: Extracting a web application from an application server and using a lightweight web framework.

A good step is to look through the company software catalog and search for consolidation possibilities. It is very important to have a software evolution strategy. This strategy should give the rules, how a system should grow to avoid becoming

more and more complex. It describes how the business value of a system can be kept or increased by preserving the maintainability. This also includes defining a regular version upgrade (see https://thawing-taiga-6031.herokuapp.com/#/ar/25) or a reevaluation of third party products.

Pitfalls to avoid

- Find the thin line between implementing own functions and reinventing the wheel. Implementing own frameworks or middleware functionalities is usually not advisable.
- Avoid creating requirements based on a tools capabilities. Define the requirements first and then evaluate a product accordingly.

Architectural decision(s) to be revisited

- Choice of service interface
- Choice of middleware
- Choice of third party product
- Software evolution strategy

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Middleware
- Interfaces
- Other applications
- Quality assurance of service

Tasks

- Assess security impact (security architecture review) :: Analysis Task
- Assess scalability impact (concurrency level) :: Analysis Task
- Assess the need for timeouts :: Measurement Task
- Update interface specifications :: Design Task
- Implement new service :: Development Task
- Execute integration test :: Testing Task
- Design a software evolution strategy :: Design Task

References (links):

- Quantitative Methods for Software Selection and Evaluation [Michael S. Bandor]
  http://www.sei.cmu.edu/reports/06tn026.pdf
- Systems and software Quality Requirements and Evaluation (SQuaRE) - ISO/IEC 25040
  http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=35765
- Java application servers are dead! [Eberhard Wolff]
  https://jaxenter.com/java-application-servers-dead-112186.html
- Play Framework
  https://www.playframework.com/
- Software Evolution und Reengineering [Martin Glinz, Harald Gall, University of Zurich]
  https://files.ifi.uzh.ch/rerg/amadeus/teaching/courses/software_engineering_ws0506/Kapitel_12_EvolReeng.pdf

Consolidate data access to reduce dependencies

Created	Modified	Status
11/04/2015 at 09:27:26	11/04/2015 at 09:27:26	published

Context (viewpoint, refinement level)

- Functional View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/

Quality attributes (forces)

- Dependency https://en.wikipedia.org/wiki/Coupling_(computer_programming)
- Responsibility https://en.wikipedia.org/wiki/Separation_of_concerns

Smells

Name / Description / Questions	Group	TDI
Different data access implementations	Maintainability	hm
In a steady growing software system, different components most probably have different data access implementations. • Do different components have different data access implementations? • Do you have difficulties replacing an underlying data storage? • Does the migration of a data store require changes in a lot of components?		
Many different data sources	Complexity	hm
A software system or a single component collects its data from many data sources. E.g. different databases. • Does your software system or even a component collect its data from many different data sources?		

Description

Initial position

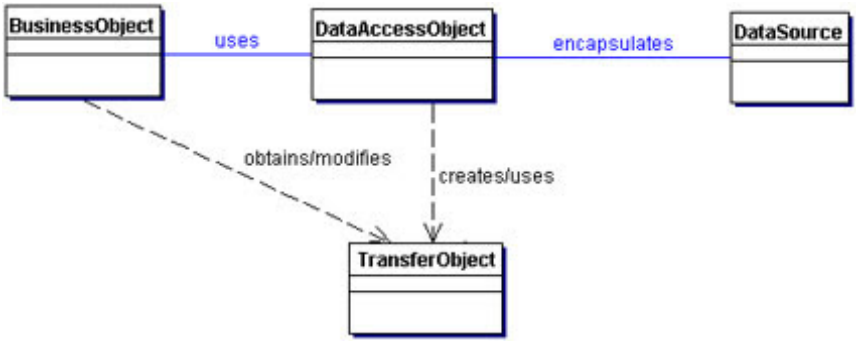
In a steady growing software system, different components might have different implementations of data access or even different underlying data stores. E.g. if other components have been integrated into an existing software system due to closings of other departments or due to acquisition of them.

If this is the case it might be necessary to consolidate the different implementations or even the data sources. The goal is to extract the implementations from the existing components and create a new component which handles the data access to the data sources. The new data access component should provide only a defined set of functionalities to the data requesting components. The requestors will not need to have any knowledge of the underlying data source, since they are all communicating through a standard interface and not directly with the data source (e.g. via JDBC (Java Database Connectivity)).

Revised design

The solution is to introduce a central data access component over which different components can request the necessary data. The data access component should have a standard interface. The data source behind it should be transparent and easily replaceable. Further details can be found in the book "Cloud Computing Patterns" (see References).

Such a component should provide generic functions for other components to access the data. E.g. the four CRUD (Create Read Update Delete) operations. A REST (Representational State Transfer) interface is an example which implements this type of pattern and allows a component in the system to request data objects via the HTTP (Hypertext Transfer Protocol) protocol using the four CRUD (Create, Read, Update and Delete) operations.



(src: http://www.oracle.com/)

The figure shows the Core J2EE pattern of Data Access Objects. The data access component itself should encapsulate the data access itself as well. This makes a replacement of the data source even in the data access component much easier. The Data Access Objects pattern is the most common one. A business object requests data via the data access object. The data access object connects to the data source and provides the data in a independent format (transfer object).

Pitfalls to avoid

- Make sure to avoid all tool dependencies in the data access component. The data source has to be transparent (should be replaceable without interface changes).
- Make sure to use standard data access patterns and frameworks (DAO (Data Access Object)s, JPA (Java Persistence API), etc.)
- Don't introduce multiple data access components.

Architectural decision(s) to be revisited

- Choice of data source/store
- Software system landscape

Affected components and connectors (if modelled explicitly)

- Middleware
- Application software (Business software)

Tasks

- Choose common data source :: Architectural Decision Task
- Update software architecture document (SAD) :: Documentation Task
- Create data access component :: Design Task
- Extract data acces code from existing components :: Development Task
- Write data access logic in data access component :: Development Task

References (links):

- Cloud Computing Patterns [Fehling, Leymann, Retter, Schupeck, Arbitter]
- Patterns of Enterprise Application Architecture [Martin Fowler]
  http://martinfowler.com/books/eaa.html
- Core J2EE Patterns - Data Access Object [Oracle]
  http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Centralize message routing and transformation to increase flexibility

Created	Modified	Status
11/11/2015 at 08:56:13	11/11/2015 at 08:56:13	published

Context (viewpoint, refinement level)

- Logical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Conceptual Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Flexibility https://en.wikipedia.org/wiki/Flexibility_(engineering)
- Robustness https://en.wikipedia.org/wiki/Robustness_(computer_science)
- Standardization https://en.wikipedia.org/wiki/Standardization

Smells

Name / Description / Questions	Group	TDI
Unnecessary tight coupling	Dependency	mm
It is the case that a component has a strong dependency to one or many other component(s). This can cause the problem, that changes or extinctions in one component have an unnecessary effect to other components. • Does a change in a component imply a change in one or many other compnent(s)? • Does a problem in a component have effect in one or many other component(s)? • Do you have difficulties to replace an interface component? • Is one component in your system strongly dependent from an other?		
Violation of architectural principles	Standards	mm
Loose coupling has the following autonomy dimensions: platform autonomy, reference autonomy, time autonomy and Format (see book: Cloud Computing Patterns, Fehling). • Is your implementation missing patterns that promote architectural principles such as seperation of concerns or loose coupling?		
Many communication channels	Complexity	mm
Is it the case that your application has many different interfaces to communicate with many other components. • Does your application have many different communication interfaces?		
Complex interface	Complexity	hm
Is it the case that a component has a complex interface to another component which makes changes difficult and makes it difficult for interface partners to program against it. • Is it difficult to implement changes for an interface? • Do interface partners have problems programming against your interface?		
Many different interfaces	Complexity	mm
It is the case that your application is communicating with a lot of other components using many different interfaces and maintaining them is complex. • Is your application communicating with a lot of other components using many different interfaces? • Do you have a problem with maintaining many different interfaces?		

Description

Initial position

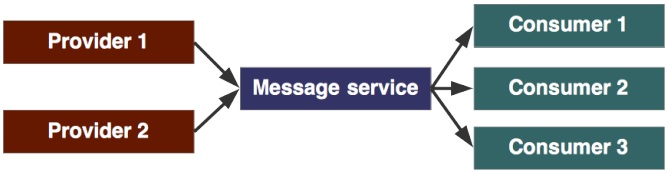
A software system which has many different communication interfaces to a lot of other components (on different hardware), contains a lot of complexity. Individual (non-standard) interfaces lead to unnecessary tight coupling (instead of loose coupling). It is difficult to change the interface or to replace one of the communication parties. Most probably there exists in-house developed middleware (-parts) or frameworks. This will make it hard for new employees to understand the software quickly and make changes.

The solution of this AR is to put a standard message broker in place (WebSphere MQ, Apache ActiveMQ, ...) and replace the non-standard communication interfaces. Direct dependencies should be broken and there should be a clear separation of concerns.

Revised design

After the implementation of the message broker, it should be easier to replace a communication party or to change the interface of two communication parties. The message routing and transformation (e.g. encoding) is configurable. With most standard products it is possible to choose the type of the communication (asynchronous, synchronous, as a service, message queue, ...).

It is easier to deploy messages to many different components. The different parties can connect from different type of platforms to the messaging service and either receive or send messages. The communication channels can be secured. To integrate a centralized messaging service, the current communication implementations have to be replaced by the standard messaging interface. This requires a change in all involved parties.



The figure shows a centralized messaging service over which different components communicate. The communication does not have to be one-way.

Pitfalls to avoid

- Introduction of custom made middleware or (DIY (Do It Yourself)) integration frameworks
  Avoid to implement your own message broker. There are commercial and open source third party products which provide you the necessary functions.
- Message broker does not provide enough resilience: can create a single point of failure.
- Centralized integration component (broker) is overloaded with rich business logic.

Architectural decision(s) to be revisited

- Choice of service interface
- Choice of message exchange pattern
- Choice of transport protocol
- Synchronous vs. asynchronous communication

Affected components and connectors (if modelled explicitly)

- Application software (Business software)
- Middleware

Tasks

- Evaluate performance impact by due to communication via broker :: Analysis Task
- Choose standardized serialization format for message transport :: Architectural Decision Task
- Integrate message broker :: Integration Task
- Update interface specifications :: Design Task
- Update software architecture document (SAD) :: Documentation Task
- Replace individual communication implementation with interface implementation to the new message broker :: Development Task

References (links):

- Cloud Computing Patterns [Fehling, Leymann, Retter, Schupeck, Arbitter]
- Exploring the Enterprise Service Bus [IBM, Greg Flurry]
  http://www.ibm.com/developerworks/library/ar-esbpat1/
- Message Oriented Middleware [Wikipedia]
  https://de.wikipedia.org/wiki/Message_Oriented_Middleware
- Enterprise Service Bus [Wikipedia]
  https://de.wikipedia.org/wiki/Enterprise_Service_Bus

Introduce online document formatting engine to provide printing functionality

Created	Modified	Status
12/07/2015 at 09:44:53	12/07/2015 at 11:35:27	published

Context (viewpoint, refinement level)

- Logical View [4+1, Kruchten] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759
- Functional View [Rozanski, Woods] http://www.viewpoints-and-perspectives.info/home/viewpoints/
- Technology Level [Architectural Decision Level] http://www.springer.com/us/book/9783642023736

Quality attributes (forces)

- Functionality https://en.wikipedia.org/wiki/Functional_requirement
- Productivity https://en.wikipedia.org/wiki/Productivity

Smells

Name / Description / Questions	Group	TDI
Missing online formatting functionality	Functionality	II
Is your web application missing a function to provide PDF documents with current information.		
• Would you like to provide your users PDF documents with current information?		

Description

Initial position

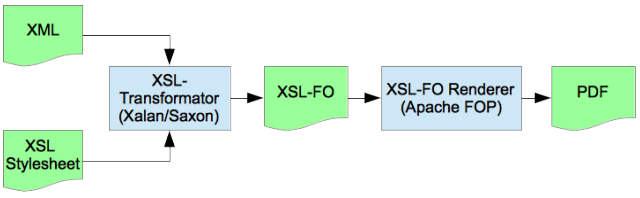
Web applications often have useful information which a user might like to print out. A browser printing function is unsatisfying since the printout might contain unnecessary HTML elements or the representation of the necessary data is not adequate. An online formatting engine can provide professional documents with current data and the flexibility of changing the document layout independent from the website presentation.

Revised design

This architectural refactoring describes a Java based online formatting engine using XSL-FO with an XSL-Transformation and Apache FOP. XSL-FO provides advanced layouting features for professional document generation.

Java provides an API (Application programming interface) for XML processing (JAXP). You can bind any XSLT-Library, which implements the standard JAXP interfaces, into the Java classpath and address its transformation functions via the standard interface. Such Libraries are for example the Apache XALAN (XSLT version 1) or Saxon (XSLT version 1 and 2). The input data can easily be provided as XML, for example by using the same JAXP API to serialize a Java Object to XML.

JSON (Javascript Object Notation) becomes more and more popular, but there is no stylesheeting language available yet. But there are libraries which can convert JSON to XML quickly and easily (E.g. http://www.json.org/).



The figure shows the sequence of generating a PDF document. The XML-Input data is transformed with a XSL-Stylesheet to a XSL-FO document. This XSL-FO document is then rendered with Apache FOP. There are other commercial products available, for example from Antenna House or Compart.

Pitfalls to avoid

- Make sure to limit the size of documents which are generated or else split the document. A rendering process can consume a lot of CPU resources depending on the elements used in the document and the size of the input data.
- Do not underestimate the implementation of a XSL-Stylesheet. Professional document layouting is a very specific task and can require specialized developers.

Architectural decision(s) to be revisited

- Choice of information presentation method
- Choice of application frameworks

Affected components and connectors (if modelled explicitly)

- Application software (Business software)

Tasks

- Choose XSL-Transformation library :: Decision Task
- Evaluate XSL-FO rendering engine :: Decision Task
- Purchase 3rd party software license :: Commercial Task
- Evaluate document layouting implementation (in-house/external) :: Leadership Task
- Implement XSL-Stylesheet :: Development Task
- Execute load test :: Testing Task

References (links):

- XSL-FO in der Praxis [Manuel Montero Pineda, Manfred Krüger]
  http://www.data2type.de/en/publications/xslfo-in-practice/
- The XSLT and XQuery Processor [Saxonica]
  http://saxon.sourceforge.net/
- Apache Xalan
  https://xalan.apache.org/

A.2. ART Migration in die Cloud (Heroku)

Dieser Abschnitt beschreibt die Umsetzung des ARs “Migrate web-application to cloud-environement”.

Hierfür wurde der Quellcode des AR-Werkzeug auf GitHub publiziert und die Applikation anschliessend auf der Cloud Environment Platform Heroku installiert.

- Github URL: `https://github.com/bisigc/art`
- Heroku URL: `https://thawing-taiga-6031.herokuapp.com`

Folgend werden die nötigen Schritte beschrieben:

1. Erstellen eines Github Accounts (`https://github.com/join`) → `https://github.com/bisigc`
2. Beim bestehenden Git-Repository, welches den Quellcode des AR-Werkzeug enthält, zusätzlich die Remote Adresse für das neue Github.com Repository hinzufügen und den Quellcode auf Github übermitteln (pushen):

Auflistung A.1: Source Code nach Github publizieren

```
git remote add github https://github.com/bisigc/art.git
git push -u github master
```

3. Erstellen eines Heroku accounts (`https://signup.heroku.com/`)
4. Herunterladen und installieren des Heroku Toolbelts (`https://toolbelt.heroku.com/`)
5. Erstellen eines Applikationscontainers auf Heroku mittels des Heroku Toolbelts:

Auflistung A.2: Heroku Applikation erstellen

```
heroku create
```

6. Installieren des Heroku Addons “ClearDB” als Datenbank (ClearDB ist eine MySQL Cloud Implementation):

Auflistung A.3: Heroku ClearDB(MySQL) erstellen

```
heroku addons:create cleardb:ignite
```

7. Migration der MySQL Daten aus der Hochschule für Technik Rapperswil (HSR)-Virtual Machine (VM) in die erstellte ClearDB Instanz auf Heroku. Der erste Schritt exportiert die Inhalte der Ursprungsdatenbank in eine Datei (mysldump), der zweite Schritt importiert den Inhalt in die Zieldatenbank.

Auflistung A.4: Migration der Datenbank auf Heroku

```
mysqldump -u art_user -h localhost -p<password> art > file_name.
    sql
mysql --host=us-cdbr-iron-east-03.cleardb.net --user=<user> --
    password=<password> --reconnect <dbname> < file_name.sql
```

8. Setzen der Datenbank Verbindungsattribute als Umgebungsvariablen in der Heroku Konfiguration:

Auflistung A.5: DB Verbindungsparameter als Variablen in Heroku setzen

```
heroku config:set MYDBURL=jdbc:mysql://us-cdbr-iron-east-03.
    cleardb.net/<dbname> MYDBUSER=<username> MYDBPASSWORD=<password
    >
```

9. Erstellen einer Datei namens “Procfile” im Wurzelverzeichnis des Git-Repositories mit untenstehendem Inhalt (das Procfile enthält den Befehl, welcher Heroku verwendet, um den Webanwendungsprozess zu starten):

Auflistung A.6: Heroku Processing File

```
web: target/universal/stage/bin/art-app -Dplay.server.http.port=${
    PORT} -Ddb.default.url=${MYDBURL} -Ddb.default.username=${
    MYDBUSER} -Ddb.default.password=${MYDBPASSWORD}
```

10. Beim bestehenden Git-Repository, welches den Quellcode des AR-Werkzeug enthält zusätzlich die Remote Adresse für das Heroku Repository hinzufügen:

Auflistung A.7: Remote Git-Adresse für Heroku hinzufügen

```
git remote add heroku https://git.heroku.com/thawing-taiga-6031.
    git
```

11. Pushen des Git-Repositories auf Heroku. Heroku erkennt auf Grund der Artefakte im Repository, dass es sich um eine Play-Framework-Applikation handelt und löst nach dem Push automatisch einen SBT-Build aus und startet danach die Applikation in der Cloud. Mit dem zweiten Befehl wird die Applikation in einem Internet Browser geöffnet:

Auflistung A.8: Applikation auf Heroku installieren

```
git push heroku master
heroku open
```

12. Die Installation der Play Applikation auf die Heroku Cloud Platform ist nun erfolgreich abgeschlossen.

A.3. Frontend Build Prozess Implementation

Dieser Abschnitt beschreibt die technische Vorgehensweise, um den Frontend Build Prozess in das bestehende Play Applikations Projekt zu integrieren. Ein Frontend Build Prozess erhöht die Performance einer Applikation durch die Komprimierung der Frontend Artefakte und fördert die Übersicht über den Quellcode durch eine klare Strukturierung der Artefakte.

Für den Frontend Build Prozess werden folgende Werkzeuge eingesetzt:

- **Grunt** (`http://gruntjs.com/`): Javascript Task Runner, zur Ausführung von konfigurierbaren Arbeitsschritten.
- **Yeoman** (`http://yeoman.io/`): Scaffolding Werkzeug, erzeugt Quellcodevorlagen für Frontend Artefakte.
- **NPM** (`https://www.npmjs.com/`): Packetmanager für Javascript, zur Installation von Frontend Bibliotheken und Werkzeugen.
- **Bower** (`http://bower.io/`): Packetmanager für Web-Frontend Bibliotheken

Folgend die Schritte zur Integration des Build Prozesses:

1. Hinzufügen der play-yeoman Plugin Abhänigkeit im Play Projekt (Datei project/plugins.sbt). Dieser Schritt generiert die Ordnerstruktur für die Frontend Artefakte und erstellt ebenfalls Konfigurationsdateien für den Task Runner Grunt sowie die Packet Manager NPM und Bower:

Auflistung A.9: Yeoman Plugin für Play Projekt hinzufügen

```
addSbtPlugin("com.tuplejump" % "sbt-yeoman" % "0.8.1")
```

2. Installation des Packetmanagers NPM (`https://www.npmjs.com/`).
3. Installation des Scaffolding Werkzeugs Yeoman inklusive der Generatoren für AngularJS Frontend Artefakte, sowie des Packetmanagers Bower und der Komponente Grunt, welche die Build Prozesse ausführt (Task Runner):

Auflistung A.10: Yeoman, Bower, Grunt Installation mittels NPM

```
sudo npm install -g yo
sudo npm install -g generator-angular
sudo npm install -g bower
sudo npm install -g grunt-cli
```

4. Aufsetzen der AngularJS Ordnerstruktur für die Frontend Artefakte, sowie wie das Generieren dieser Artefakte (vier Beispiele):

Auflistung A.11: Yeoman Artefaktgenerierung

```
yo angular
yo angular:controller TaskAddController
yo angular:factory TaskService
yo angular:filter toSafeHtml
yo angular:directive hilighter
```

5. Integration von Frontend Bibliotheken (Open Source) mittels dem Packetmanager Bower. Das Beispiel zeigt die Integration des Open Source WYSIWYG-Editors textAngular:

Auflistung A.12: Integration von Frontend Bibliotheken mittels Bower

```
bower install textAngular --save
```

6. Auslösen des Frontend Build Prozesses. Grunt führt verschiedene in einer Konfigurationsdatei (Gruntfile.js) festgehaltene Schritte aus. Die Schritte beinhalten das zusammenziehen und komprimieren aller Frontend Artefakte (Javascripts, Bilder, HTML-Views) und referenzierten Javascript Bibliotheken.

Auflistung A.13: Grunt Build Prozess auslösen

```
grunt build
```

A.4. ART Bug und Feature Report

Dieser Abschnitt enthält eine Liste aller in dieser Arbeit referenzierten Feature Requests, welche auf Grund von Anwendungstests und Validierungen entstanden sind. Die komplette Liste aller Fehler sowie neu beantragten Funktionen und Qualitätsverbesserungen für das AR-Werkzeug ist im Github Repository zu finden:
`https://github.com/bisigc/art`

Alle Punkte die in der kompletten Liste enthalten sind, entstanden auf Grund von Anwendungstests, der Wissenserschaffung im Werkzeug, sowie während der AR-Werkzeug- und Wissens-Validierung (siehe Kapitel 7 auf Seite 53). Die meisten Einträge der Liste sind während der Masterarbeit umgesetzt worden, doch bestehen noch einige nicht realisierte Punkte, deren Umsetzung bei einem Weiterbestehen der Applikation von Nutzen sind.

Nr.	Datum	Beschreibung	Prio	Status
56	27.10.15	Jekyll Integration. Export der Datenobjekte im Jekyll Format ermöglichen.	low	akzeptiert
68	03.11.15	Versionierung von Smells (analog ARs).	low	akzeptiert
71	03.11.15	Getrennte Diskussion für ARVersionen, eine für AR-Applier und eine für AR-Editoren.	low	akzeptiert
74	12.12.15	Erstellen einer Hilfe Seite mit Begriffs Erklärungen und Link zum Benutzerhandbuch.	mid	akzeptiert
75	12.12.15	Task Export ins EEPPI oder ein anderes Werkzeug ermöglichen.	low	akzeptiert
76	12.12.15	Bewertungsfunktion für Architectural Refactorings.	low	akzeptiert

Tabelle A.1: Bug und Feature Report

A.5. Szenariobasierter Anwendungstest komplett

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Ein Werkzeug für Architectural Refactoring

Usability Test

Autor: Christian Bisig
Version: 1.0

Inhaltsverzeichnis

1. Einleitung

Das folgende Dokument enthält einen kurzen Test, welcher zur Bewertung der Inhalte sowie der Benutzbarkeit des Architectural Refactoring Werkzeugs durch eine Drittperson dient.

Das Architectural Refactoring Werkzeug beinhaltet drei Rollen, eine Administrationsrolle (AR-Admin), eine Wissen erfassende Rolle (AR-Editor) und eine Wissen anwendende Rolle (AR-Applier). Der Usability Test beinhaltet zwei Szenarien, eines als AR-Applier und eines als AR-Editor. Jedes Szenario besteht aus einer Ausgangslage, dem Ziel das zu erreichen ist, die nötigen Ausführungsschritte dazu, sowie Fragen die nach der Ausführung zu beantworten sind. Als Hilfsmittel steht das Benutzerhandbuch des Architectural Refactoring Werkzeugs zur Verfügung.

Am Ende jedes Szenarios befindet sich ein Abschnitt “Hinweis“ in welchem jeweils eine Anmerkung zum erwarteten Ziel (der Lösung) beschrieben ist.

2. Szenario 1: Ein geeignetes Architectural Refactoring finden

2.1 Ausgangslage

Als Softwarearchitekt einer Firma in der Finanzbranche sind sie technisch verantwortlich für das Softwaresystem einer grösseren Abteilung. Vereinfacht gesagt handelt es sich um ein System das Transaktionsaufträge mittels einer Messaging Queue entgegennimmt, verarbeitet und weiterleitet. Eines Tages beklagt sich das Back-Office, welches die verarbeiteten Transaktionen in ihrem System überprüft, dass Transaktion immer öfters mit grosser Verspätung in ihrem System ersichtlich sind. Sie setzen sich mit dem Systemadministrator zusammen und beobachten den Server, auf welchem ihr Softwaresystem läuft, während der Zeit zu welcher die meisten Transaktionen verarbeitet werden.

Sie stellen fest, dass die Incoming-Messaging-Queue sich laufend füllt und die Abarbeitung der Transaktionen nur sehr langsam von statten geht. Der Server scheint aber bei weitem noch nicht vollständig ausgelastet. Arbeitsspeicher, CPU und Harddisk sind noch genug vorhanden. Sie stellen jedoch fest, dass von dem Multi-Core System, auf welchem die Software läuft, lediglich eine CPU komplett ausgelastet ist. Anschliessend konsultieren Sie mit einem Softwareentwickler den Quelltext der Applikation und stellen fest, dass die Software nicht fähig ist mehrere Anfragen gleichzeitig zu bearbeiten. Sie ziehen den Schluss, dass die Software nicht optimal auf das Hardware System abgestimmt ist.

2.2 Ziel

Suchen Sie im Architectural Refactoring Werkzeug nach einer geeigneten Lösung für das in der Ausgangslage beschrieben Problem. Das Ziel ist es ein geeignetes Architectural Refactoring zu finden, welches beschreibt wie die Software zu verändern ist, damit die Hardwareressourcen des Servers optimal genutzt werden können und somit auch der Durchsatz des Systems gesteigert wird.

2.3 Ausführungsschritte

1. Das Architectural Refactoring Werkzeug (ART) aufrufen:
   `http://sinv-56041.edu.hsr.ch/art-app/.`
2. Einloggen als AR-Applier; Usermame: applier@hsr.ch , Passwort: test
3. Starten eines “Smell Self-Assessment”:
   `http://sinv-56041.edu.hsr.ch/art-app/#/smellassess`
4. Ausfüllen des “Smell Self-Assessment” um ein geeignetes Architectural Refactoring für das in der Ausgangslage beschriebene Problem zu finden.
5. Durchlesen des gefundenen Architectural Refactorings.

2.4 Fragen

2.4.1 Zum Inhalt

1. Sind die Fragen im Smell Self-Assessment verständlich?

	$--$	$-$	$-/+$	$+$	$++$	
schwer verständlich	☐	☐	☐	☐	☐	einfach verständlich

Kommentar:

2. War es schwierig mittels des Smell Self-Assessment ein geeignetes Architectural Refactoring zu finden?

	$--$	$-$	$-/+$	$+$	$++$	
schwierig	☐	☐	☐	☐	☐	einfach

Kommentar:

3. Ist das Architectural Refactoring verständlich formuliert?

	$--$	$-$	$-/+$	$+$	$++$	
schwer verständlich	☐	☐	☐	☐	☐	einfach verständlich

Kommentar:

4. Sind die zum Architectural Refactoring verlinkten Smells verständlich formuliert?

	$--$	$-$	$-/+$	$+$	$++$	
schwer verständlich	☐	☐	☐	☐	☐	einfach verständlich

Kommentar:

5. Machen die zum Architectural Refactoring verlinkten Tasks Sinn?

	$--$	$-$	$-/+$	$+$	$++$	
nicht sinnvoll	☐	☐	☐	☐	☐	sinnvoll

Kommentar:

2.4.2 Zur Benutzerfreundlichkeit

1. Reagiert das Architectural Refactoring Werkzeug schnell auf Benutzereingaben?

	$--$	$-$	$-/+$	$+$	$++$	
langsam	☐	☐	☐	☐	☐	schnell

Kommentar:

2. Ist das Architectural Refactoring Werkzeug unkompliziert zu bedienen?

	$--$	$-$	$-/+$	$+$	$++$	
kompliziert	☐	☐	☐	☐	☐	unkompliziert

Kommentar:

3. Erfordert das Architectural Refactoring Werkzeug überflüssige Eingaben?

	$--$	$-$	$-/+$	$+$	$++$	
überflüssige Eingaben nötig	☐	☐	☐	☐	☐	keine überflüssigen Eingaben nötig

Kommentar:

4. Bietet das Architectural Refactoring Werkzeug die nötigen Funktionen, um eine Lösung für das beschriebene Szenario zu finden?

	$--$	$-$	$-/+$	$+$	$++$	
nötige Funktionen fehlen	☐	☐	☐	☐	☐	nötige Funktionen sind vorhanden

Kommentar:

5. Bietet das Architectural Refactoring Werkzeug eine gute Übersicht über das Funktionsangebot?

	$--$	$-$	$-/+$	$+$	$++$	
schlechte Übersicht	☐	☐	☐	☐	☐	gute Übersicht

Kommentar:

6. Verwendet das Architectural Refactoring Werkzeug gut verständliche Begriffe, Bezeichnungen, Abkürzungen oder Symbole in Masken und Menüs?

	$--$	$-$	$-/+$	$+$	$++$	
schlecht verständlich	☐	☐	☐	☐	☐	gut verständlich

Kommentar:

7. Liefert das Architectural Refactoring Werkzeug in zureichendem Masse Informationen darüber, welche Eingaben zulässig oder nötig sind?

	$--$	$-$	$-/+$	$+$	$++$	
unzureichende Informationen	☐	☐	☐	☐	☐	zureichende Informationen

Kommentar:

8. Erleichtert das Architectural Refactoring Werkzeug die Orientierung durch eine einheitliche Gestaltung.?

	$--$	$-$	$-/+$	$+$	$++$	
erschwert Orientierung	☐	☐	☐	☐	☐	erleichtert Orientierung

Kommentar:

9. Lässt sich das Architectural Refactoring Werkzeug durchgehend nach einem einheitlichen Prinzip bedienen?

	$--$	$-$	$-/+$	$+$	$++$	
uneinheitlich bedienbar	☐	☐	☐	☐	☐	einheitlich bedienbar

Kommentar:

10. Erfordert das Architectural Refactoring Werkzeug viel Zeit zum Erlernen?

	$--$	$-$	$-/+$	$+$	$++$	
braucht viel Zeit	☐	☐	☐	☐	☐	braucht wenig Zeit

Kommentar:

11. Ist das Architectural Refactoring Werkzeug gut ohne Hilfe oder Handbuch erlernbar?

	$--$	$-$	$-/+$	$+$	$++$	
schlecht erlernbar	☐	☐	☐	☐	☐	gut erlernbar

Kommentar:

12. Ermöglicht das Architectural Refactoring Werkzeug einen leichten Wechsel zwischen einzelnen Menüs und Masken?

	$--$	$-$	$-/+$	$+$	$++$	
kein leichter Wechsel	☐	☐	☐	☐	☐	leichter Wechsel

Kommentar:

13. Liefert das Architectural Refactoring Werkzeug gut verständliche Fehlermeldungen?

	$--$	$-$	$-/+$	$+$	$++$	
schlecht verständlich	☐	☐	☐	☐	☐	gut verständlich

Kommentar:

14. Lässt sich das Architectural Refactoring Werkzeug gut an eine persönliche, individuelle Art der Arbeitserledigung anpassen?

	−−	−	−/+	+	++	
lässt sich schlecht anpassen	☐	☐	☐	☐	☐	lässt sich gut anpassen

Kommentar:

Anmerkungen:

2.5 Hinweis

Auf Grund der Aufgabenstellung und des formulierten Ziels, sollte das Smell Self-Assessment das Architectural Refactoring mit dem Namen “Convert application from single- to multi-threading” gefunden haben.

3. Szenario 2: Ein Architectural Refactoring erfassen

3.1 Ausgangslage

Als Softwarearchitekturexperte auf dem Gebiet der Anwendungs- und Integrationsarchitektur haben sie sich auf Datenbanksysteme spezialisiert und arbeiten an der Hochschule für Technik Rapperswil. In einem Industrieprojekt mit einer Firma, welche im Affiliate-Marketing tätig ist und riesige Datenmengen zur Verbesserung der Werbemittel-Nutzung auswertet, werden sie als Experte für die Verbesserung des analytischen Software Systems beigezogen. Die Firma beklagt sich über schwindende Performance der nötigen Datenauswertungen und den enormen Speicherplatzgebrauch auf Grund der ständig wachsenden Datenmenge. Sie nehmen sich dem Problem an und stellen fest, dass die aggregierenden Datenabfragen in der analytischen Datenverarbeitung des Systems, auf dem bestehenden Relationalen Datenbanksystem mit zeilen-basierten Tabellen, sehr langsame Antwortzeiten hat. Zudem sind die Tabellen mit vielen redundanten Daten bestückt, in welchen aber nur ein Bruchteil der Einträge eindeutig sind.

Auf Grund ihrer Erfahrung im Datenbankbereich empfehlen sie der Firma den Datenspeicher auf spalten-basierte Tabellen umzustellen und auf flash-basierten Speicher zu setzen. Sie wissen, die physische Datenpartitionierung in spalten-basierten Tabellen ist prädestiniert für die Art der analytischen Datenabfrage, wie sie die Firma benötigt. Zudem sind die spalten-basierten Tabellen um ein vielfaches besser für Datenkompression geeignet bei gleichzeitiger Wahrung von schnellen Antwortzeiten.

Die Firma zeigt sich überzeugt und beschliesst den Wechsel des Datenspeichers vorzunehmen und bittet sie dabei das Projekt als Experte zu begleiten. Sie unterstützen die Firma bis zum Ende des Projektes, welches den erhofften Erfolg erzielte. Die Datenanalysen der Marketing Firma wurden deutlich beschleunigt und gleichzeitig konnten signifikante Speicherplatzeinsparungen erreicht werden.

3.2 Ziel

Sie als Softwarearchitekturexperte, welcher die Umsetzung des Architekturumbaus in der Firma empfohlen und begleitet haben, möchten nun das Wissen über den Umbau

festhalten und veröffentlichen. Sie haben sich entschieden das Wissen als Architectural Refactoring im AR-Werkzeug mit allen nötigen Details zu erfassen.

3.3 Ausführungsschritte

1. Das Architectural Refactoring Werkzeug (ART) aufrufen:
   `http://sinv-56041.edu.hsr.ch/art-app/`
2. Einloggen als AR-Editor; Usermame: editor@hsr.ch , Passwort: test
3. Öffnen des AR-Browsers:
   `http://sinv-56041.edu.hsr.ch/art-app/#/arbrowser`
4. Dort mittels des Buttons “Add AR“ das Erfassungsformular für ein Architectural Refactoring öffnen.
5. Den in der Ausgangslage beschriebenen Architekturumbau mit allen Objekten (Smells, Execution Tasks und Meta-Daten-Objekten) erfassen.

3.4 Fragen

3.4.1 Zur Benutzerfreundlichkeit

1. Reagiert das Architectural Refactoring Werkzeug schnell auf Benutzereingaben?

	$--$	$-$	$-/+$	$+$	$++$	
langsam	☐	☐	☐	☐	☐	schnell

Kommentar:

2. Ist das Architectural Refactoring Werkzeug unkompliziert zu bedienen?

	$--$	$-$	$-/+$	$+$	$++$	
kompliziert	☐	☐	☐	☐	☐	unkompliziert

Kommentar:

3. Erfordert das Architectural Refactoring Werkzeug überflüssige Eingaben?

	$--$	$-$	$-/+$	$+$	$++$	
überflüssige Eingaben nötig	☐	☐	☐	☐	☐	keine überflüssigen Eingaben nötig

Kommentar:

4. Bietet das Architectural Refactoring Werkzeug die nötigen Funktionen, um eine Lösung für das beschriebene Szenario zu finden?

	$--$	$-$	$-/+$	$+$	$++$	
nötige Funktionen fehlen	☐	☐	☐	☐	☐	nötige Funktionen sind vorhanden

Kommentar:

5. Bietet das Architectural Refactoring Werkzeug eine gute Übersicht über das Funktionsangebot?

	$--$	$-$	$-/+$	$+$	$++$	
schlechte Übersicht	☐	☐	☐	☐	☐	gute Übersicht

Kommentar:

6. Verwendet das Architectural Refactoring Werkzeug gut verständliche Begriffe, Bezeichnungen, Abkürzungen oder Symbole in Masken und Menüs?

	$--$	$-$	$-/+$	$+$	$++$	
schlecht verständlich	☐	☐	☐	☐	☐	gut verständlich

Kommentar:

7. Liefert das Architectural Refactoring Werkzeug in zureichendem Masse Informationen darüber, welche Eingaben zulässig oder nötig sind?

	$--$	$-$	$-/+$	$+$	$++$	
unzureichende Informationen	☐	☐	☐	☐	☐	zureichende Informationen

Kommentar:

8. Erleichtert das Architectural Refactoring Werkzeug die Orientierung durch eine einheitliche Gestaltung.?

	$--$	$-$	$-/+$	$+$	$++$	
erschwert Orientierung	☐	☐	☐	☐	☐	erleichtert Orientierung

Kommentar:

9. Lässt sich das Architectural Refactoring Werkzeug durchgehend nach einem einheitlichen Prinzip bedienen?

	$--$	$-$	$-/+$	$+$	$++$	
uneinheitlich bedienbar	☐	☐	☐	☐	☐	einheitlich bedienbar

Kommentar:

10. Erfordert das Architectural Refactoring Werkzeug viel Zeit zum Erlernen?

	$--$	$-$	$-/+$	$+$	$++$	
braucht viel Zeit	☐	☐	☐	☐	☐	braucht wenig Zeit

Kommentar:

11. Ist das Architectural Refactoring Werkzeug gut ohne Hilfe oder Handbuch erlernbar?

	$--$	$-$	$-/+$	$+$	$++$	
schlecht erlernbar	☐	☐	☐	☐	☐	gut erlernbar

Kommentar:

12. Ermöglicht das Architectural Refactoring Werkzeug einen leichten Wechsel zwischen einzelnen Menüs und Masken?

	$--$	$-$	$-/+$	$+$	$++$	
kein leichter Wechsel	☐	☐	☐	☐	☐	leichter Wechsel

Kommentar:

13. Liefert das Architectural Refactoring Werkzeug gut verständliche Fehlermeldungen?

	$--$	$-$	$-/+$	$+$	$++$	
schlecht verständlich	☐	☐	☐	☐	☐	gut verständlich

Kommentar:

14. Lässt sich das Architectural Refactoring Werkzeug gut an eine persönliche, individuelle Art der Arbeitserledigung anpassen?

	$--$	$-$	$-/+$	$+$	$++$	
lässt sich schlecht anpassen	☐	☐	☐	☐	☐	lässt sich gut anpassen

Kommentar:

Anmerkungen: