Masterprojekt: Software-Engineering

Recherche zu Architekturanalyse-Tools

von
Dirk Markus Schablack
4schabla@informatik.uni-hamburg.de

Fachbereich Informatik
Universität Hamburg
Sommersemester 2015
Version vom 29. Mai 2015

Inhaltsverzeichnis

1 Einleitung

In dieser Arbeit werden Architekturanalyse-Tools vorgestellt, welche im Zuge des Masterprojekts gesucht und untersucht wurden. Gesucht wurden Tools, welche Informationen über ein bereits vorhandenes Softwaresystem liefern können, die eine Relevanz für die Softwarearchitektur haben. Das Ziel ist es zum Schluss ein oder mehr von diesen Tools zu empfehlen, welche im Masterprojekt eingesetzt werden sollten, um mit ihnen architekturrelevante Informationen von Projekten zu extrahieren und dann die Informationen für die Projekte in den Decision Buddy zu integrieren.

In dem Masterprojekt gab es noch weitere Arbeiten zu konkreten Architekturanalyse-Tools, welche die folgenden Tools untersuchten: Bauhaus, Dependometer, JDepend, Kieker Monitoring, Sonargraph und Sotoarc. Während diese Arbeiten ein oder zwei Tools relativ ausführlich behandeln, werden die Tools hier nur kurz vorgestellt, berichtet wie der Test der Tools verlaufen ist und für jedes wird ein Kriterienkatalog angefertigt. Der Kriterienkatalog ist grob nach dem Muster, welches in den anderen Gruppen erarbeitet wurde, wird jedoch, je nach Tool ein wenig angepasst. Das Original Muster für den Kriterienkatalog kann man in Anhang B finden. Die Tools, welche von den anderen Arbeiten behandelt werden, werden hier nicht weiter betrachtet. warum die 14 tools? how did he reach them?

Insgesamt ergab die Suche 14 Tools, von denen drei jedoch nicht getestet wurden. Diese drei sind Tools sind Lattix [8], Structure101 [12] und XRadar [13]. Lattix wurde nicht getestet, da die Lizenz- und Kostenlage unklar ist, und man nicht leicht und ohne Angabe sehr vieler Informationen eine Testversion erhalten kann. Structure101 wurde nicht getestet, da es laut der Beschreibung eher für Neuentwicklungen geeignet zu sein scheint. XRadar wurde nicht getestet, da die Beispiel-Analyse bereits daraufhin deuteten, dass kaum bis keine architekturrelevanten Informationen liefert.

Der Hauptteil der Arbeit ist in zwei Kapitel unterteilt. In Kapitel 2 werden die getesteten Tools vorgestellt, mit kurzem Ablauf des Test, dem Kriterienkatalog und einem kleinem Fazit. In Kapitel 3 werden die Tools vorgestellt, welche beim Test zu Problemen führten bzw. nicht testbar waren, mit einer kurzen Erklärung des Problems.
Anschließend wird im Fazit nochmal auf die Ergebnisse der Recherche und der Test zurückgeblickt. Abschließend werden im Anhang noch Informationen Testsystem, Ausgaben von Tools sowie eine Einführung zu Design Structure Matrizen und deren Bezug zur Softwarearchitektur gegeben.

2 Tools: getestet

2.1 Design Pattern Recognizer

2.1.1 Beschreibung

Design Pattern Recognizer ist ein Eclipse-plug-in, welches in Folge eines Diplomprojekts von von Lubomir Elko entwickelt wurde. Es soll bestimmte vorhandene und begonnene Design Pattern in Java-Projekten finden.[2]

2.1.2 Test

Getestet wurde der Design Pattern Recognizer mit der Mediathek. Es findet relativ viele Strategy und Facade Pattern. Allerdings scheint das Tool fast immer ein fertiges oder unfertiges Strategy Pattern anzuzeigen, wenn eine Klasse eine andere Klasse beerbt. Das Facade Pattern scheint hingegen scheint einfach dadurch bereits gefunden zu sein, wenn eine Klasse eine Variable eines anderen, im Projekt definierten, Typ hat.

2.1.3 Kriterienkatalog

Rahmenbedingungen	
Installation:	über eine Eclipse Update Site
Lizenzen:	frei
Kompatibilität:	benötigt vorhandenes Eclipse
Ergonomie	
Oberfläche:	eine eigene View in Eclipse
Bedienbarkeit:	sehr einfach
Support	
Dokumentation:	-
Tutorials:	-
Community:	-
Aktualität:	Arbeit an diesem Tool wurde beendet

[2]

2.1.4 Fazit

Aufgrund der Ungenauigkeit des Tools und der nicht vorhandenen Export-Funktion sollte man von einer Verwendung im Masterprojekt absehen.

2.2 Enterprise Architect

2.2.1 Beschreibung

Enterprise Architect ist ein Softwaremodellierungstool von Sparx Systems. Es bietet dabei eine Vielzahl von Funktionalitäten für für den gesamten Lebenszyklus eines Systems. Die Funktionalitäten reichen vom Modellieren der Anforderungen über dem Simulieren des Business-Ablaufs, dem Generieren von Code bis zur Simulation des Endsystems.[3]
Interessant für das Masterprojekt ist die versprochene Unterstützung beim Reverse Engineering.[3]

2.2.2 Test

In diesem Test wurde die 30-Tage-Testversion von Enterprise Architect verwendet, und das als Testprojekt diente die Mediathek. Der Import einzelner Dateien ist recht einfach, jedoch ist die Möglichkeit

ein ganzes Projekt zu importieren nicht so leicht zu finden.
Nach dem Import des Projekts werden automatisch eine Reihe von UML-Klassendiagrammen erzeugt. Ein Klassendiagramm für das gesamte System, und weitere für jedes Java-Package, welches Java-Klassen enthält. Hierbei kommt es allerdings zu Problemen, denn bei manchen Klassendiagrammen für die einzelnen Packages fehlen ein Teil der Klassen, wenige sind sogar gänzlich ohne Inhalt.
Ab diesem Zeitpunkt muss die weitere Bearbeitung für das Reverse Engineering manuell erfolgen, weitere Möglichkeiten zum Extrahieren von Informationen (insbesondere von architekturrelevanten Informationen) gibt es leider nicht.

2.2.3 Kriterienkatalog

Rahmenbedingungen	
Installation:	Windows: Installationsdatei (.exe) ausführen Mac und Linux: es wird Wine / Crossover benötigt
Lizenzen:	proprietäre Lizenz, Kosten: $135 - $599
Kompatibilität:	Enterprise Architect, benötigt keine weitere Software, bietet aber viele Plug-ins für andere Software an (u.a. Eclipse und Visual Studio)
Ergonomie	
Oberfläche:	Aufbau ähnlich zu einer IDE
Bedienbarkeit:	man benötigt einige Zeit um die gewünschten Funktionen zu finden
Support	
Dokumentation:	sehr gut, auf der Webseite erhält man viel, vom Benutzerhandbuch über online-Hilfe bis Demonstrationsvideos
Tutorials:	sehr gut, es gibt schriftliche Tutorials zu vielen Themen und Bereichen, aber auch Webinars sind vorhanden
Community:	sehr gut, dass zugehörige Forum weist eine hohe Aktivität auf
Aktualität:	Aktuelle Version ist vom 12.02.2015

[3]

2.2.4 Fazit

Enterprise Architect ist ein Tool, dass mit Sicherheit den Entwicklungsprozess von Software gut Unterstützen kann. Auch beim Reverse Engineering kann es eine Hilfe sein. Wirklich architekturrelevante Informationen kann es von einem bereits bestehenden System aber nicht liefern. Damit ist es für das Masterprojekt nicht geeignet.

2.3 JArchitect

2.3.1 Beschreibung

JArchitect ist ein Tool von CoderGears, welches laut Hersteller dabei helfen soll ein großes Java-Projekt zu verwalten.[7] Insbesondere soll es auch dabei unterstützen eine vorhandene Architektur zu entdecken und untersuchen.[7]

2.3.2 Test

Getestet wurde die 14-Tage-Testversion von JArchitect angewendet auf die in A genannten Projekte. Dabei wurde der Eclipse-Workspace-Import verwendet. JArchitect bietet auch die Möglichkeit jar/war-Dateien zu untersuchen. Dies wurde allerdings nicht getestet.

Nach der Analyse kann man auswählen welche Ergebnisse man aktuell anschauen möchte, darunter den Abhängigkeitsgraphen, die Dependency Structure Matrix und Metriken. Das, für das Masterprojekt, interessante Ergebnis ist die Dependency Structure Matrix (DSM), siehe Abbildung 1. In der DSM gibt es zwei Arten von Einträgen:

blau Spalte benutzt Zeile, Zahl: Anzahl der teilnehmenden Elemente der Zeile

grün Zeile wird von Spalte benutzt, Zahl: Anzahl der teilnehmenden Elemente der Spalte

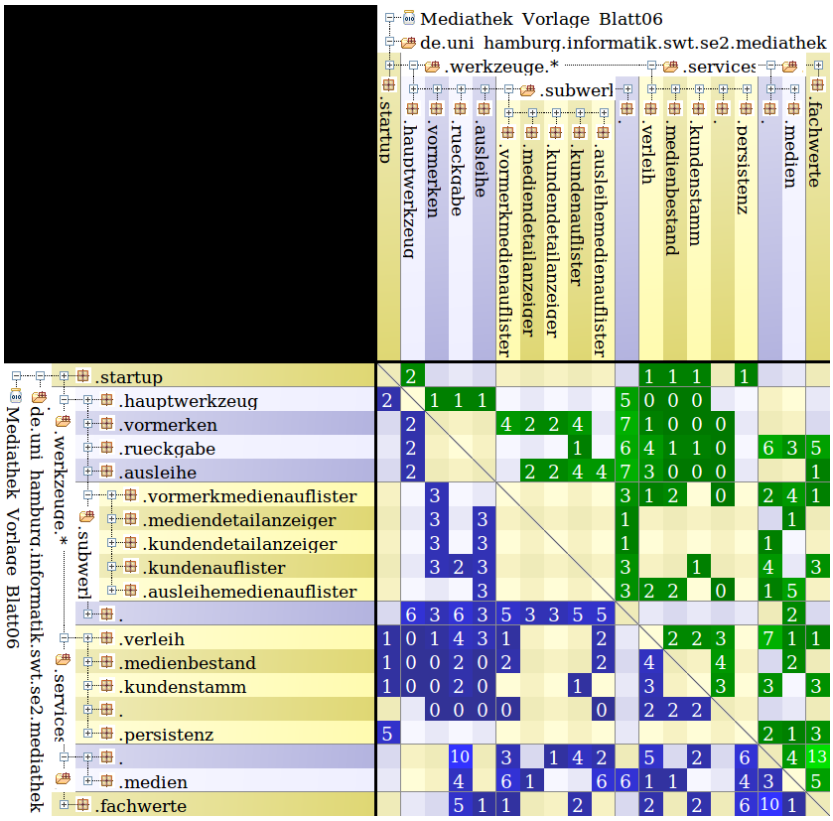

Abbildung 1: JArchitect: Export der Dependency Structure Matrix für die Mediathek

Für eine genauere Erklärung von Dependency Structure Matrizen und deren Relevanz siehe Anhang E.

Die DSM lässt sich als png exportieren. Auch für den html-Report wird eine DSM erzeugt, aber nur auf Projekt-Ebene, und ist damit nicht mehr sinnvoll einsetzbar.

2.3.3 Kriterienkatalog

Rahmenbedingungen	
Installation:	Nach den üblichen Arten der Betriebssysteme (Windows: exe, Mac: dmg, Linux tar.gz)
Lizenzen:	proprietäre Lizenz, Kosten: \$299 - \$649 Bedingungen für Open Source Lizenz: Verwendung für Open Source Projekt älter als 3 Monate, aktive Community. Für Details siehe: `http://www.jarchitect.com/JArchitectForOSS`
Hardwareanforderungen:	mindestens 1GB RAM
Kompatibilität:	Linux: benötigt Glibc > 2.13

Ergonomie	
Oberfläche:	ähnlich einer IDE
Bedienbarkeit:	Für die hier relevanten Funktionen: einfach
Support	
Dokumentation:	vorhanden, Qualität:
Tutorials:	nein
Community:	schwach
Aktualität:	Aktuelle Version 4.0.0 vom 11.11.2013
Daten	
Art:	Abhängigkeitsgraphen, Dependency Structure Matrix, Tree Map, Metriken
Formate:	.html (Report), .png (Grafiken)

[7]

2.3.4 Fazit

JArchitect ist sehr gut geeignet, wenn man einen Überblick über ein Java-Projekt haben, seine internen und externen Abhängigkeiten untersuchen oder es anhand von Regeln (ähnlich zu Checkstyle und Findbugs) analysieren möchte. Architekturrelevante Informationen sind hier im wesentlichen durch die Dependency Structure Matrix gegeben.

Aufgrund der an Bedingungen geknüpfte Open Source Lizenz und der fehlenden automatischen Generation einer DSM auf Package-Ebene ist JArchitect nur eingeschränkt für das Masterprojekt zu empfehlen.

2.4 Imagix 4D

2.4.1 Beschreibung

Imagix 4D ist ein Softwareanalysetool von Imagix Corp. Laut angaben des Herstellers soll Imagix an verschiedenen Stellen bei der Analyse von Software helfen, unter anderem bei dem Verstehen von fremden Code, bei der Change-Impact Analyse und beim Dokumentieren des Entwurfs.[6]
Mithilfe von Imagix soll man den Source-Code „auf jedem Level“[6] untersuchen können, von der Architektur-Ebene bis zu den Details von Funktionen und Variablen.[6]

2.4.2 Test

Getestet wurde die Windows-Testversion unter Wine. Die Testversion unterliegt der Beschränkung, Imagix 4D ausschließlich mit dem mitgelieferten Testprojekt verwenden zu dürfen / können. Die Bedienbarkeit ist einfach und intuitiv gestaltet. Jedoch stürzt das Programm bei bestimmten Darstellungen und sämtlichen Export-Versuchen ab. Dies ist aber vermutlich darauf zurückzuführen, dass es nicht um die Linux-native Version handelt.

Imagix stellt in der Tat viele anpassbare Diagramme zur Verfügung. Das für das Masterprojekt interessanteste Diagramm, ist das „Subsystem Architecture Diagramm“[6]. In der Abbildung 2 kann man das Subsystem Architecture Diagramm für das Testprojekt, mit Aufrufen auf Package/Klassen-Ebene, sehen. Dabei sind die rosa-roten Kästen die Klassen, die grünen repräsentieren Java-Packages.

Man kann die jeweiligen Kästen bis zur Funktionsebene ausklappen, bzw. auch wieder einklappen. Imagix 4D ordnet die Kästen dabei stets so an, dass nach Möglichkeit die Aufrufe einer Klasse nur von einer höheren Klasse erfolgen. Dieser Umstand macht es leicht eine (vorhandene) Schichtenarchitektur zu erkennen. Imagix bietet einem weiter auch an nur Aufrufe anzuzeigen, welche einer Schichtenarchitektur oder einer strikten Schichtenarchitektur widersprechen.

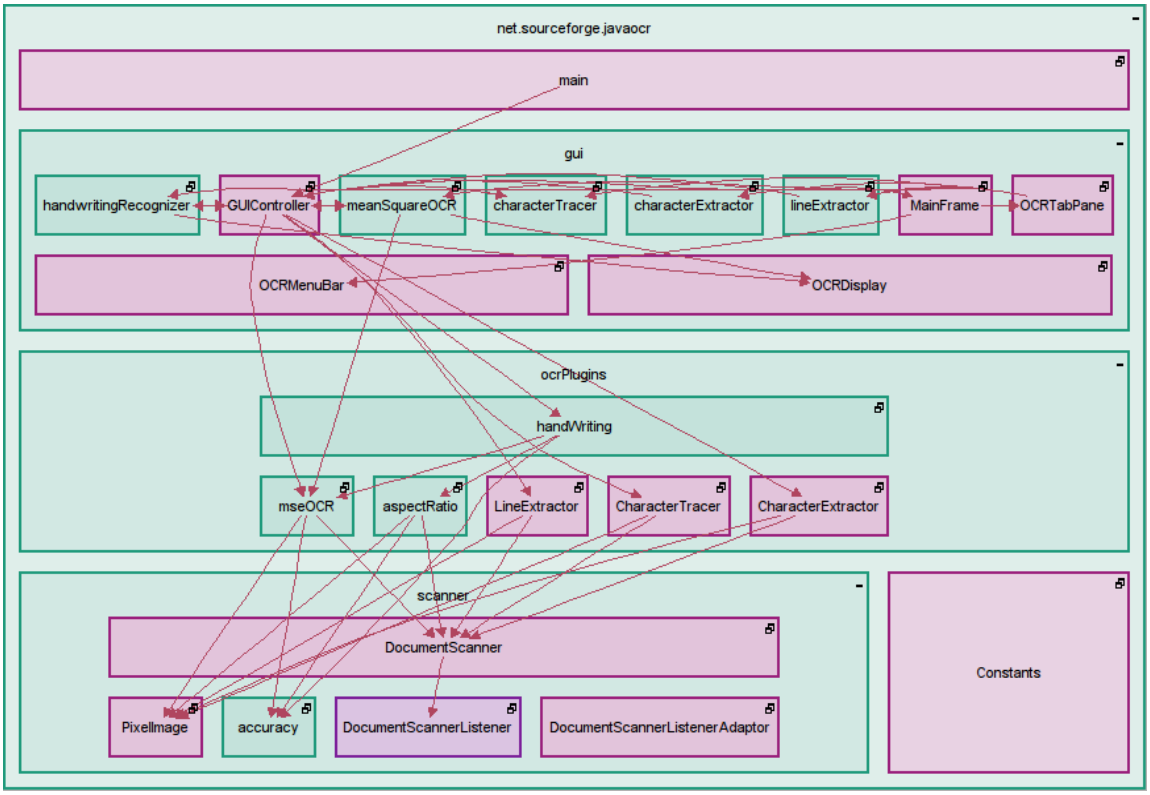

Abbildung 2: Imagix 4D: Screenshot des Subsystem Architecture Diagramms [6]

2.4.3 Kriterienkatalog

Rahmenbedingungen	
Installation:	Windows: Installationsdatei (.exe) ausführen Linux: tar-Archiv entpacken
Lizenzen:	proprietäre Lizenz, Kosten: unbekannt
Kompatibilität:	Es gibt Versionen für Windows, Linux, Solaris und HP-UX
Ergonomie	
Oberfläche:	sehr übersichtlich
Bedienbarkeit:	einfach, man kommt schnell zu den gewünschten Darstellungen / Funktionalitäten
Support	
Dokumentation:	gut
Tutorials:	-
Community:	-
Aktualität:	Aktuelle Version vom 28.03.2015
Daten	
Art:	Graphen auf Basis verschiedenster (objektorientierter) Eigenschaften (Abhängigkeiten, Funktionsaufrufe, Vererbung), textuelle Ausgabe über viele Metriken und Eigenschaften
Formate:	.png/.ps (Grafiken), ASCII/RTF/HTML (Text)

[6]

2.4.4 Fazit

Mit Imagix 4D kann man recht schnell und einfach eine Schichtenarchitektur erkennen. Andere Architekturmuster dürften aber deutlich schwerer mit Imagix zu erkennen sein, bzw. Imagix 4D kann einen dabei nicht unterstützen. In Anbetracht der Ungewissheit über die Lizenz und die auftretenden Kosten der Nutzung, sowie der geringen Verbreitung der Software, sollte man von der Nutzung im Masterprojekt eher absehen.

2.5 Pinot

2.5.1 Beschreibung

Pinot (Pattern INference and recOvery Tool) ist ein, von Nija Shi entwickeltes, Kommandozeilen-Programm welches viele der GoF Pattern erkennen kann.[9]

2.5.2 Test

Pinot wurde auf alle drei Testprojekte, welche in Anhang A genannt werden, angewendet. Die Ausgabeergebnisse sind in Anhang C zu finden. Weiter wurde zu Vereinfachung des Aufrufs und des Abspeicherns der Ergebnisse ein Bashskript verwendet, welches man in Anhang D findet. Aufgefallen ist dabei, dass in allen drei Projekten das Beobachtermuster verwendet wurde, Pinot es jedoch in keinem erkannt hat, obwohl es in der Liste der zu erkennenden Muster aufgeführt wird.

2.5.3 Kriterienkatalog

Rahmenbedingungen	
Installation:	Man muss den Source Code selbst kompilieren
Lizenzen:	IBM Public License
Kompatibilität:	Fähigkeit c++-Code zu kompilieren
Ergonomie	
Oberfläche:	Kommandozeilen-Programm
Support	
Dokumentation:	Installationshinweise + man-Seite
Tutorials:	nein
Community:	-
Aktualität:	Aktuelle Version vom 08.04.2006
Daten	
Art:	Liste mit gefundenen Design Pattern
Formate:	Kommandozeilenausgabe (Text)

[9]

2.5.4 Fazit

Es liefert einen Teil der verwendeten Design Pattern eines Projekts, scheint aber leider Probleme bei manchen Pattern zu haben. Auch wenn Design Pattern nicht die wichtigsten architekturrelevanten Informationen darstellen und kann man die Verwendung von Pinot im Masterprojekt durchaus in Betracht ziehen, da es leicht zu verwenden ist und die Ausgabe leicht zu verarbeiten ist.

2.6 SoMoX

2.6.1 Beschreibung

SoMoX (Software Model Extractor) ist ein Reverse Engineering Tool, welches an dem Karlsruher Institut für Technologie entwickelt wird und als Eclipse-plug-in zur Verfügung steht. Mithilfe von SoMoX soll man Komponenten, zusammengesetzte Strukturen und Konnektoren aus dem Source Code extrahieren können.[11]

2.6.2 Test

Getestet wurde SoMoX mit der Mediathek und den Standardeinstellungen. Die Ergebnisse teilen sich in verschiedene Bereiche auf. In der XMI-Datei befindet sich eine große Menge an statische Informationen der Art: wo und wie viele Klassendeklarationen / Switchcases / LineComments / Package-Zugriffe gibt es?
Weiter kann man sich ein Klassendiagramm erzeugen lassen, welches jedoch für die Mediathek bereits sehr schlecht organisiert ist, denn ein Großteil der Klassen ist einfach nur aufgereiht und nicht mit den anderen verbunden, andere Klassen sind im Diagramm sehr weit entfernt, haben dafür aber eine «references»-Verbindung zu anderen Klassen. Die Export-Funktion funktioniert leider nicht gut, und führt mit jedem Versuch dazu, dass der Xorg-Server rund 500MB des Arbeitsspeicher mehr verlangt und nicht wieder frei gibt. Dies ist allerdings wohl ein Problem des Xorg-Servers.
Der interessanteste Teil sind aber die extrahierten Komponenten. Hier gibt SoMoX zwei Möglichkeiten an. Die eine besteht darin, dass jede Klasse eine eigene Komponente ist. Die andere Komponentenaufteilung gibt an, dass alle Klassen zusammen eine Komponente bilden und der innere Enum VerleihEreignis in der Klasse Verleihprotokollierer eine zweite Komponente bildet.

2.6.3 Kriterienkatalog

Rahmenbedingungen	
Installation:	über Eclipse Update Site
Lizenzen:	frei
Kompatibilität:	benötigt vorhandene Eclipse-Installation
Support	
Dokumentation:	schwach
Tutorials:	nein
Community:	-
Aktualität:	Aktuelle Version vom 10.05.2015
Daten	
Art:	viele statische Informationen
Formate:	XMI-Format (XML Metadata Interchange)

[11]

2.6.4 Fazit

SoMoX liefert sehr viele statische Informationen, welche allerdings nicht wirklich architekturrelevant sind. Architekturrelevant wären allerdings gut zusammengesetzte Komponenten, im Test wurden solche allerdings nicht erzeugt. Um besser zusammengesetzte Komponenten zu erhalten wird man die Standardeinstellungen ändern müssen. Da man dann allerdings wohl für jedes Projekt erneut eine spezifische Einstellung finden muss, um vernünftige Komponenten zu erhalten, ist SoMoX weniger geeignet für das Masterprojekt.

2.7 SonarQube

2.7.1 Beschreibung

SonarQube ist laut Hersteller eine „Plattform zum Verwalten von Code-Qualität“ [14]. Es besteht aus einer Web-Applikation, einer austauschbaren Datenbank und dem Analyse-Tool SonarQube Runner. SonarQube Runner dient ausschließlich zur Analyse von Projekten, dessen Ergebnisse dann in SonarQube verwaltet und dargestellt werden. [14]

Hauptgrund für die Untersuchung dieses Tools ist, dass es als Open Source erhältlich ist und eine Design Structure Matrix erstellen kann.

2.7.2 Test

Getestet wurde SonarQube mit der mitgelieferten Datenbank und mit SonarQube Runner für die Analyse der Mediathek, sowie das Eclipse-plug-in. Weiter wurde SonarQube nur bezüglich der DSM getestet.

Zunächst wurde SonarQube gestartet und ein neues Projekt für die Mediathek angelegt. Als erstes wurde die Mediathek mit dem Eclipse-plug-in untersucht. Dies führt zu vielen Regelverletzung wie Verstöße gegen Namens Konvention. Diese werden als Warnungen in Eclipse angezeigt. Es war jedoch nicht möglich die Analysedaten in SonarQube zu integrieren und eine DSM erzeugt das Eclipse-plug-in selber nicht. Die in der Dokumentation beschriebene Möglichkeit das Eclipse-plug-in mit SonarQube zu verbinden ist nicht (in der Version 3.4.0.20140404-0949-RE) vorhanden.

Als nächstes wurde die Analyse mittels SonarQube Runner getestet. SonarQube Runner ist ein Kommandozeilenprogramm. Für die Projekt-Einstellungen für SonarQube Runner wurden die Standardeinstellungen (enthalten in dem Testprojekt von SonarQube) um die Zeile `sonar.binaries=bin` erweitert und dann explizit an SonarQube Runner beim Aufruf mittels `-D project.settings=sonar-project.properties` übergeben, da der Runner sonst nicht ganze Projekt untersucht und SonarQube keine DSM anzeigt.

Die so erzeugte DSM (siehe Abbildung 3) lässt sich nicht durch SonarQube exportieren, aber man kann die gesamte Seite im Browser als html speichern, welche die DSM enthält.

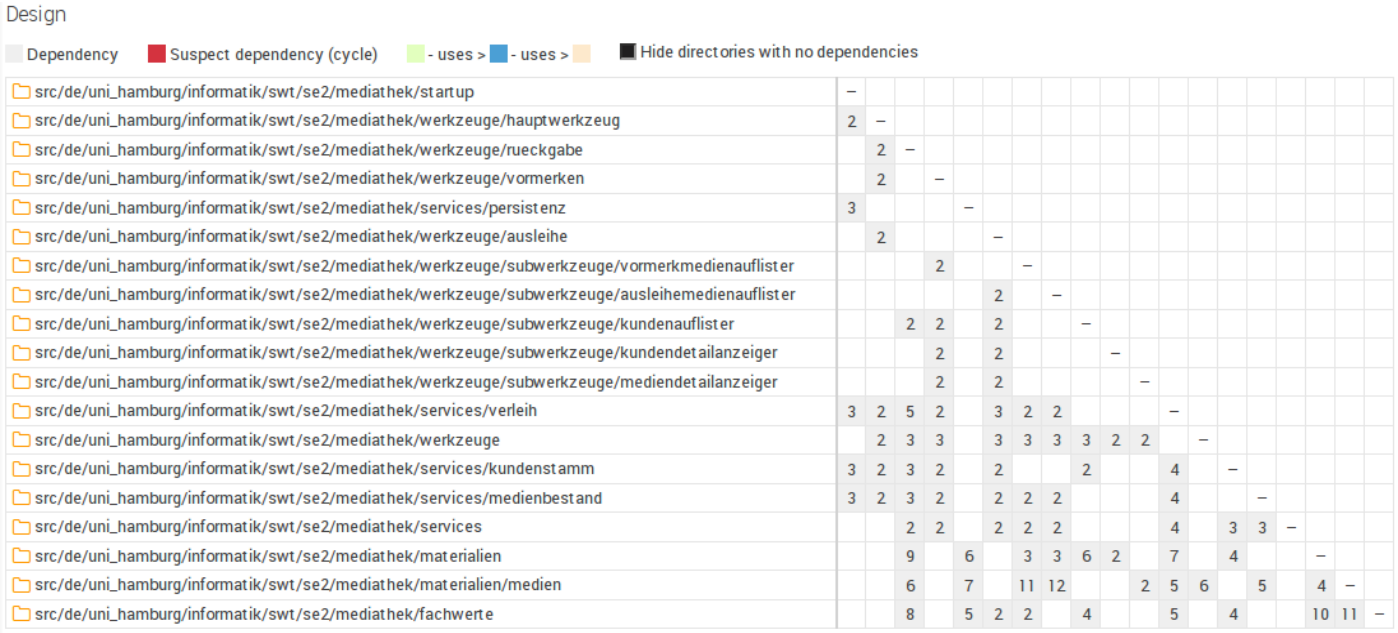

Abbildung 3: SonarQube: Screenshot der Design Structure Matrix für die Mediathek

2.7.3 Kriterienkatalog

Rahmenbedingungen	
Installation:	Entpacken einer zip-Datei
Lizenzen:	Open Source Lizenz
Kompatibilität:	benötigt für die Analyse weitere Tools: SonarQube Runner, Ant, Maven oder Gradle, es gibt viele zusätzliche Plug-Ins
Ergonomie	
Oberfläche:	Web-Applikation, sehr übersichtlich
Bedienbarkeit:	Web-Oberfläche: einfache Navigation Analyse: (SonarQube Runner) Analyseprogramm für Konsole, benötigt einmalig genaue Konfiguration (Dokumentation hierfür ist unvollständig)
Support	
Dokumentation:	gut
Tutorials:	„Getting Started“ ist Teil der Dokumentation
Community:	sehr aktiv
Aktualität:	Aktuelle Version 5.1 vom 02.04.2015
Daten	
Art:	Metriken, Regeln (Code Convention), Design Structure Matrix
Formate:	keine Export-Funktion, Daten lassen sich mit dem Browser als .html speichern Kann Daten in einer Datenbank speichern (MySQL, PostgreSQL, Oracle, Microsoft SQL Server)

[14]

2.7.4 Fazit

SonarQube wurde, wie bereits erwähnt, nur bezüglich der DSM untersucht. Diese kann SonarQube erzeugen, aber nicht selber exportieren. Da SonarQube jedoch das einzige Tool der Untersuchten ist, welches definitiv als Open Source zur Verfügung steht, sollte dieses Tool im Masterprojekt verwendet werden, wenn eine Design Structure Matrix verwendet werden soll.

3 Tools: mit Problemen

3.1 ePAD

3.1.1 Beschreibung

ePAD ist, wie der Design Pattern Recognizer 2.1, ein Eclipse-plug-in welches Design Pattern auffinden können soll.[4]

3.1.2 Problemursache

Leider erscheint die ePAD-Perspektive in Eclipse nach Installation des Tools nicht. Es kann somit nicht weiter getestet werden.

3.2 Fujaba

3.2.1 Beschreibung

Die Fujaba (From UML to Java and back again) Tool Suite ist ein Open Source Tool welches and der Universität Paderborn entwickelt wurde. Es soll Entwickler bei der modellbasierten Softwareentwicklung und Reengineering unterstützen.[5]

3.2.2 Problemursache

Fujaba selber sollte gar nicht für das Masterprojekt untersucht werden, jedoch setzen die beiden folgenden Programme (Reclipse und Archimetrix) auf Fujaba auf, bzw. gehöre mit zu dem Fujaba Projekt.
Leider scheint Fujaba jedoch nicht mehr verfügbar zu sein, denn die benötigten Installationsdateien (sowohl für die Standalone-Version als auch das Eclipse-plug-in) sind nicht mehr vorhanden.

3.3 Reclipse

3.3.1 Beschreibung

Reclipse ist ein Eclipse-plug-in welches nach bestimmten Pattern in Source Code sucht und baut auf Fujaba auf. Es sucht sowohl statisch als auch dynamisch nach vorher spezifizierten Pattern. Diese vorher spezifizierten Pattern muss man selber erstellen.[10]

3.3.2 Problemursache

Leider war es nicht möglich Pattern für die Suche zu erstellen. Das Reclipse Handbuch [16] deutet daraufhin, dass man diese Pattern mittels Fujaba erstellen bzw. bearbeiten muss. Fujaba ist aber wie oben erwähnt nicht mehr erhältlich.

3.4 Archimetrix

3.4.1 Beschreibung

Archimetrix ist ein Tool, welches iterativ komponentenorientierte Architekturen extrahieren und bearbeiten kann. Es ist wie Reclipse ein Eclipse-plug-in.[1]

3.4.2 Problemursache

Bei dem Versuch die möglichen Komponenten der Mediathek zu extrahieren stürzt Archimetrix mit folgender Fehlermeldung ab:

```
An internal error occurred during: "Component Relevance Analysis".
org.eclipse.gmt.modisco.java.emf.impl.EnumDeclarationImpl cannot be cast to
org.eclipse.gmt.modisco.java.ClassDeclaration
```

Dies ist kein Fehler von Archimetrix selbst sondern des benötigten Eclipse-plug-in MoDisco. Dennoch führt es dazu, dass Archimetrix (wenigstens bis zur Behebung dieses Fehlers) unbenutzbar ist.

4 Fazit

In dieser Arbeit wurden 11 Tools zur Architekturanalyse untersucht, wobei viele von diesen Tools nicht die gewünschten oder erhofften Funktionen bzw. Informationen lieferten. Bei einigen Tools (Design Pattern Recognizer, SoMoX) konnten die erzeugten Ergebnisse nicht überzeugen, bei einem anderen Tool (Enterprise Architect) fehlen die Funktionalitäten zum Extrahieren von architekturrelevanten Informationen und wiederum andere Tools (ePAD, Fujaba, Reclipse, Archimetrix) ließen sich gar nicht erst testen, da sie nicht einsatzfähig sind.

Es gab jedoch auch Tools die relevante Informationen liefern konnten. Mit Pinot haben wir ein Tool, welches einige Design Pattern in Java-Code entdecken kann. Auch wenn Design Pattern selbst nur bedingt architekturrelevant sind, ist deren Vorhandensein in einem Projekt durchaus eine wertvolle Information, welche in dem Decision Buddy angezeigt werden kann.

Von den untersuchten Tools sind diejenigen besonders interessant für das Masterprojekt, welche eine Design Structure Matrix erstellen können, insbesondere sind hier JArchitect und SonarQube zu nennen. Anhand dieser Matrix kann man in einigen Fällen Rückschlüsse auf die Architektur und vorhandene Komponenten gewinnen, aber in jeden Fall erhält man damit Einblicke in die interne Struktur und hat einen Überblick über die Abhängigkeiten in einem System. Beide Tools haben jedoch Nachteile. JArchitect steht unter Umständen weder kostenfrei noch günstig zur Verfügung und bei SonarQube erhält man die Dependency Structure Matrix nur über Umwege in einem Format, welches sich weiter verarbeiten lässt.

Abschließend muss ich sagen, dass die Recherche nach Architekturanalyse-Tools nicht so viele positive Ergebnisse hervorgebracht hat, wie ich mir vor der Recherche vorgestellt habe.

Abbildungsverzeichnis

Literatur- und Quellenverzeichnis

[1] Archimetrix. `https://code.google.com/p/archimetrix/`, 14.05.2015.

[2] Design Pattern Recognizer. `http://lubes.yweb.sk/projects/dprecognizer/`, 14.05.2015.

[3] Enterprise Architect. `http://www.sparxsystems.com/products/ea/`, 14.05.2015.

[4] ePAD. `http://www.sesa.dmi.unisa.it/ePAD/`, 14.05.2015.

[5] The Fujaba Project. `http://www.fujaba.de/no_cache/home.html`, 14.05.2015.

[6] Imagix 4D. `http://www.imagix.com/index.html`, 14.05.2015.

[7] JArchitect. `http://jarchitect.com/`, 14.05.2015.

[8] Lattix. `http://lattix.com/`, 14.05.2015.

[9] pinot. `http://web.cs.ucdavis.edu/~shini/research/pinot/`, 14.05.2015.

[10] Reclipse. `https://code.google.com/p/reclipse-emf/`, 14.05.2015.

[11] SoMoX. `https://sdqweb.ipd.kit.edu/wiki/SoMoX`, 14.05.2015.

[12] structure101. `http://structure101.com/`, 14.05.2015.

[13] XRadar. `http://xradar.sourceforge.net/`, 14.05.2015.

[14] SonarQube. `http://www.sonarqube.org/`, 25.05.2015.

[15] Neeraj Sangal. Mapping Software Architecture with a Dependency Structure Matrix (DSM). `https://phillyjug.files.wordpress.com/2013/03/dsm-slides-pdf.pdf`, 29.04.2008. Abrufdatum: 28.05.2015.

[16] Lothar Wendehals. Reclipse Handbuch. `http://www.reclipse.org/reclipse/handbuch.html`, 14.05.2015.

A Verwendetes Testsystem

Betriebssystem:	Kubuntu 14.04, Architektur: amd64
Java:	OpenJDK 6
	OpenJDK 7
	Oracle Java SE 8
Eclipse:	Eclipse IDE for Java Developers, Luna (4.4.2)
	Eclipse Modelling Tools, Mars (4.5.0M7)
Wine:	Version 1.7.41
Make:	GNU Make 3.81
Testprojekte:	SWT Mediathek, Version: Vorlage zu Blatt 6
	SWT Kinoticketverkauf, Version: Vorlage zu Blatt 9
	Der dunkle Lord, entstanden im Bachelorprojekt: Softwareentwicklungsprojekt

B Kriterienkatalog

Rahmenbedingungen	
Installation:	Wie ist das Programm zu installieren?
Lizenzen:	Unter welcher Lizenz steht das Programm zur Verfügung?
	Welche Kosten treten damit auf?
Hardwareanforderungen:	Welche Anforderungen stellt das Programm an die Hardware?
Kompatibilität:	Setzt das Programm ein bestimmtes Betriebssystem oder andere Software voraus?
	Gibt es weitere Kompabilitätseinschränkungen?
Sprachoptionen:	In welchen Sprachen ist das Programm verfügbar?
Ergonomie	
Oberfläche:	Wie ist die Oberfläche des Programms gestaltet?
Navigation:	Wie leicht lassen sich die gewünschten Funktionen auffinden?
Anpassungsmöglichkeiten:	Kann man die Oberfläche individuell anpassen?
Bedienbarkeit:	Wie ist die Bedienbarkeit des Programms im Allgemeinen?
Support	
Dokumentation:	Wie gut und ausführlich ist das Programm dokumentiert?
Tutorials:	Gibt es Tutorials für das Programm?
Community:	Wie groß und aktiv ist die Community des Programms?
Aktualität:	Wie aktuell ist das Programm?
	Wird noch an dem Programm gearbeitet?
Daten	
Graphische Aufbereitung:	Wie werden die Daten dargestellt?
	Werden die Daten graphisch aufbereitet?
Menge:	Wie groß ist der Umfang der Daten?
Art:	Welcher Art sind die Daten?
	Welche Informationen liefern sie?
Formate:	In welchen Dateiformaten kann das Programm die Daten zur Verfügung stellen?

C Pinot Ergebnisse

C.1 Mediathek

```
--------- Original GoF Patterns ----------


Strategy Pattern.
AusleiheMedienFormatierer is the Context class.
Medium is the Strategy interface.
Concrete Strategy classes:  AbstractMedium CD DVD KonsolenVideospiel PCVideospiel
Delegation through _medium of type Medium
File Location: /home/markus/workspaces/eclipse/Mediathek_Vorlage_Blatt06/src/de/uni_hamburg/informatik/
    swt/se2/mediathek/werkzeuge/subwerkzeuge/ausleihemedienauflister/AusleiheMedienFormatierer.java,
  /home/markus/workspaces/eclipse/Mediathek_Vorlage_Blatt06/src/de/uni_hamburg/informatik/swt/se2/
    mediathek/materialien/medien/Medium.java

Strategy Pattern.
VormerkMedienFormatierer is the Context class.
Medium is the Strategy interface.
Concrete Strategy classes:  AbstractMedium CD DVD KonsolenVideospiel PCVideospiel
Delegation through _medium of type Medium
File Location: /home/markus/workspaces/eclipse/Mediathek_Vorlage_Blatt06/src/de/uni_hamburg/informatik/
    swt/se2/mediathek/werkzeuge/subwerkzeuge/vormerkmedienauflister/VormerkMedienFormatierer.java,
  /home/markus/workspaces/eclipse/Mediathek_Vorlage_Blatt06/src/de/uni_hamburg/informatik/swt/se2/
    mediathek/materialien/medien/Medium.java

Facade Pattern.
VormerkMedienFormatierer is a facade class.
Hidden types: Medium Kunde
Facade access types: VormerkMedienFormatierer
File Location: /home/markus/workspaces/eclipse/Mediathek_Vorlage_Blatt06/src/de/uni_hamburg/informatik/
    swt/se2/mediathek/werkzeuge/subwerkzeuge/vormerkmedienauflister/VormerkMedienFormatierer.java


----------------------------------------

Pattern Instance Statistics:

Creational Patterns
==============================
Abstract Factory            0
Factory Method              0
Singleton                   0
------------------------------
Structural Patterns
==============================
Adapter                     0
Bridge                      0
Composite                   0
Decorator                   0
Facade                      1
Flyweight                   0
Proxy                       0
------------------------------
Behavioral Patterns
==============================
Chain of Responsibility     0
Mediator                    0
Observer                    0
State                       0
```

```
Strategy                    2
Template Method             0
Visitor                     0
-----------------------------

Number of classes processed: 7
Number of files processed: 85
Size of DelegationTable: 17
Size of concrete class nodes: 3
Size of undirected invocation edges: 6

nMediatorFacadeDual/nMediator = 0/0
nImmutable/nFlyweight = 0/0
nFlyweightGoFVersion = 0
```

C.2 Kinoticketverkauf

```
--------- Original GoF Patterns ----------



----------------------------------------

Pattern Instance Statistics:

Creational Patterns
==============================
Abstract Factory            0
Factory Method              0
Singleton                   0
------------------------------
Structural Patterns
==============================
Adapter                     0
Bridge                      0
Composite                   0
Decorator                   0
Facade                      0
Flyweight                   0
Proxy                       0
------------------------------
Behavioral Patterns
==============================
Chain of Responsibility     0
Mediator                    0
Observer                    0
State                       0
Strategy                    0
Template Method             0
Visitor                     0
------------------------------

Number of classes processed: 2
Number of files processed: 36
Size of DelegationTable: 0
Size of concrete class nodes: 0
Size of undirected invocation edges: 0

nMediatorFacadeDual/nMediator = 0/0
nImmutable/nFlyweight = 0/0
```

```
nFlyweightGoFVersion = 0
```

C.3 Der dunkle Lord

```
--------- Original GoF Patterns ----------


Strategy Pattern.
Konflikt is the Context class.
SpielObjekt is the Strategy interface.
Concrete Strategy classes:  BenutzbaresObjekt CraftingGegenstand Questgegenstand Schluessel NPC
Delegation through _taeter of type SpielObjekt
File Location: /home/markus/workspaces/eclipse/Zuul_Ausgangssystem/src/de/uni_hamburg/informatik/
    sep/zuul/Konflikt.java,
  /home/markus/workspaces/eclipse/Zuul_Ausgangssystem/src/de/uni_hamburg/informatik/sep/zuul/
    spielobjekte/SpielObjekt.java

Strategy Pattern.
Konflikt is the Context class.
SpielObjekt is the Strategy interface.
Concrete Strategy classes:  BenutzbaresObjekt CraftingGegenstand Questgegenstand Schluessel NPC
Delegation through _opfer of type SpielObjekt
File Location: /home/markus/workspaces/eclipse/Zuul_Ausgangssystem/src/de/uni_hamburg/informatik/
    sep/zuul/Konflikt.java,
  /home/markus/workspaces/eclipse/Zuul_Ausgangssystem/src/de/uni_hamburg/informatik/sep/zuul/
    spielobjekte/SpielObjekt.java


----------------------------------------

Pattern Instance Statistics:

Creational Patterns
==============================
Abstract Factory            0
Factory Method              0
Singleton                   0
------------------------------
Structural Patterns
==============================
Adapter                     0
Bridge                      0
Composite                   0
Decorator                   0
Facade                      0
Flyweight                   0
Proxy                       0
------------------------------
Behavioral Patterns
==============================
Chain of Responsibility     0
Mediator                    0
Observer                    0
State                       0
Strategy                    2
Template Method             0
Visitor                     0
------------------------------

Number of classes processed: 11
```

```
Number of files processed: 116
Size of DelegationTable: 21
Size of concrete class nodes: 5
Size of undirected invocation edges: 1

nMediatorFacadeDual/nMediator = 0/0
nImmutable/nFlyweight = 0/0
nFlyweightGoFVersion = 0
```

D Pinot Aufruf-Skript

Für den Aufruf von Pinot wurde ein Bashskript verwendet um den Aufruf zu vereinfachen. Es durchsucht den mittels `-p` angegebenen Pfad rekursiv nach `.java`-Dateien und übergibt diese an Pinot. Durch Angabe des Parameters `-v` werden die untersuchten Dateien auf der Konsole angezeigt. Abschließend werden die Ergebnisse in einer Textdatei gespeichert.

Hinweis: Mit \ werden Zeilenumbrüche signalisiert, welche im Originalskript nicht vorhanden sind, hier jedoch zur besseren Darstellung benötigt werden. Weiter handelt sich hierbei speziell für das Testsystem (siehe Anhang A) geschriebenes Skript und bedarf auf anderen Systemen unter Umständen weitere Anpassungen.

```
#!/bin/bash
projectpath=""
projectname=""
verbose=false
TEMP=‘getopt --long -o "p:v" "$@"‘
eval set -- "$TEMP"
while true ; do
    case "$1" in
        -p )
            projectpath=$2
            projectnametmp=${projectpath%/}
            projectname=${projectnametmp##*/}
            unset projectnametmp
            shift 2;;
        -v )
            verbose=true
            shift ;;
        --) shift ; break ;;
        *)
            break;;
    esac
done
if [ "$projectpath" = "" ] ; then
  echo "Project path is NOT set"
  return
fi
if [[ $CLASSPATH != *"rt.jar"* ]] ; then
  export CLASSPATH=${CLASSPATH}:/usr/lib/jvm/java-7-openjdk-amd64/jre/lib/\
  :/usr/lib/jvm/java-7-openjdk-amd64/jre/lib/rt.jar\
  :/usr/lib/jvm/java-7-openjdk-amd64/jre/bin/
fi
locate ${projectpath}**.java > $HOME/tmp/${projectname}-files.list
if [ "$verbose" = true ] ; then
  echo "$(<$HOME/tmp/${projectname}-files.list)"
fi
pinot @$HOME/tmp/${projectname}-files.list 2>&1 | tee "pinot-ergebnis-${projectname}.txt"
rm $HOME/tmp/${projectname}-files.list
```

E Design Structure Matrix

Bei einigen Tools wurden Design Structure Matrizen als architekturrelevante Informationen angegeben. Dieses Kapitel dient dazu, kurz zu erklären was eine Design Structure Matrix (DSM, oder auch: Dependency Structure Matrix) ist, und warum sie architekturrelevant ist. Es wurde anhand eines Folienvortrags eines Lattix-Mitarbeiters erarbeitet [15].

Eine Design Structure Matrix stellt Abhängigkeiten zwischen Elementen dar. Die Elemente können dabei beliebige (objektorientierte) Einheiten sein, wie Methoden, Klassen oder Packages. Die zu untersuchenden Elemente werden dann in die erste Spalte und in die erste Zeile eingetragen. Während es theoretisch möglich ist, in der Spalte und der Zeile unterschiedliche Elemente bzw. Elemente unterschiedlicher Hierarchie-Ebene (Klassen und Methoden) einzutragen, ist es, um architekturrelevante Informationen abzulesen, sinnvoll dieselben Elemente in derselben Reihenfolge in die Spalte und die Zeile einzutragen. In der einfachsten Form einer DSM werden nun diejenigen Zellen markiert, in welcher das Element der Spalte abhängig von dem Element der Zeile ist. Ein Beispiel einer DSM befindet sich in Abbildung 4.

	A	B	C	D	E
A					
B	×			×	
C		×			
D	×		×		
E			×	×	

Abbildung 4: Dependency Structure Matrix 1

Mithilfe einer DSM kann man nun architekturrelevante Informationen ablesen, genauer: man kann erkennen, ob die Architektur einer (strikten) Schichtenarchitektur entspricht und es ist unter Umständen möglich Komponenten zu erkennen.

Eine Schichtenarchitektur stellt sich in einer DSM so dar, dass es eine Reihenfolge der Elemente (Klassen oder Packages) gibt, so dass in der Matrix alle Einträge unterhalb der Diagonalen sind (siehe Abbildung 5).

	A	B	C	D	E
A					
B	×				
C		×			
D	×		×		
E			×	×	

Abbildung 5: Dependency Structure Matrix 2: Schichtenarchitektur

Für strikte Schichtenarchitekturen gilt die stärkere Bedingung, dass sich alle Einträge der DSM auf der Subdiagonalen befinden (siehe Abbildung 6).

	A	B	C	D	E
A					
B	×				
C		×			
D			×		
E				×	

Abbildung 6: Dependency Structure Matrix 3: strikte Schichtenarchitektur

Das Erkennen von Komponenten in der DSM beruht auf den Design-Prinzipien „hohe Kohäsion“ und „geringe Kopplung“. Unter der Annahme das die Komponenten auf diesen Prinzipien beruhen, sind die Komponenten in der DSM gerade diese Elemente, die viele Einträge untereinander haben, und wenige Einträge zu anderen Elementen. Ein Beispiel befindet sich in Abbildung 7, in welcher die möglichen Komponenten blau eingerahmt wurden.

Bisher haben wir hier lediglich die einfachste Form einer DSM betrachtet. Man kann aber statt dem einfachen Markieren einer Zelle auch eine Zahl eintragen, welche die Abhängigkeit gewichtet. Mögliche

	A	B	C	D	E	F	G	H	I	J	K	L
A												
B	X											
C		X										
D	X		X									
E	X	X	X	X								
F		X										
G									X			
H					X		X					X
I	X							X		X		
J							X	X	X			
K						X	X			X		
L								X			X	

Abbildung 7: Dependency Structure Matrix 4: Komponenten

Gewichte sind z.B. Anzahl der Abhängigkeiten bzw. Anzahl der abhängigkeitsverursachenden Subelemente. Ein Beispiel für eine solche gewichtete DSM haben wir bereits in Kapitel 2.3 in Abbildung 1 gesehen. Die Bedingungen für die architekturrelevanten Informationen können bei gewichteten DSM, je nach Aufbau, variieren. Für die DSM von JArchitect müssen, für eine Schichtenarchitektur, alle blauen Einträge unterhalb und alle grünen Einträge oberhalb der Diagonalen liegen. Abschließend wird in Abbildung 8 noch eine DSM von JArchitect gegeben, welche nicht einer Schichtenarchitektur entspricht. Hier werden Elemente einer zyklischen Abhängigkeit in schwarz eingetragen, und der Gesamtzyklus extra eingerahmt.

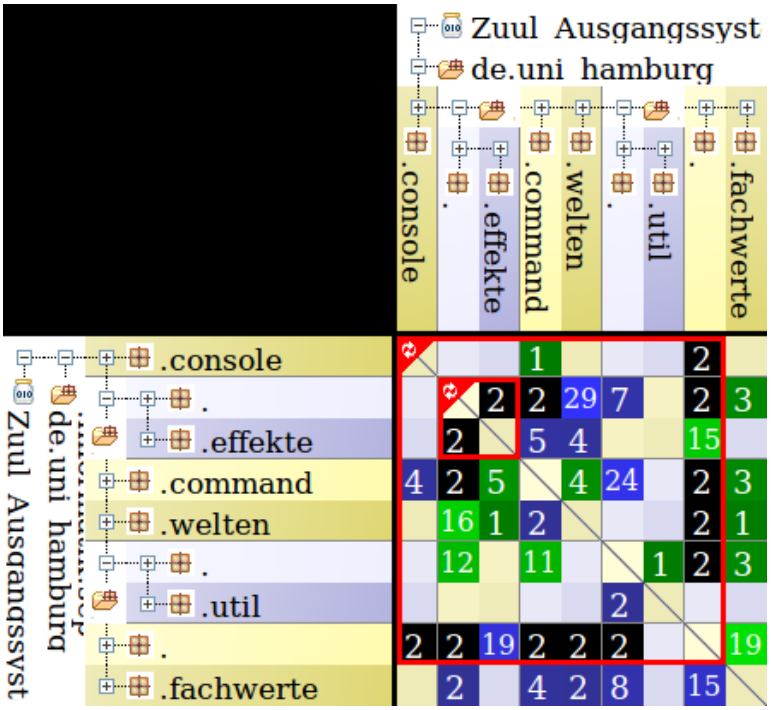

Abbildung 8: JArchitect: Export der Dependency Structure Matrix für „Der dunkle Lord“