Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminararbeit Anwendungen 2

Julia Pliszka

Modernisierung von Legacy Anwendungen unter Verwendung von MDA-Konzepten

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Julia Pliszka

Modernisierung von Legacy Anwendungen unter Verwendung von MDA-Konzepten

Seminarausarbeitung im Rahmen der Veranstaltung Anwendungen 2
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Abgegeben am 28. Februar 2009

Inhaltsverzeichnis

1 Einführung

Das Themengebiet der Masterarbeit soll die automatisierte Modernisierung von Legacy Systemen sein. Für die Modernisierung wird die Vorgehensweise Migration verwendet. Zudem wird die Migration mit dem MDA-Konzept kombiniert, um die Automatisierung zu fördern.
Es existieren viele Migrationsklassifikationen; in dieser Ausarbeitung sowie in der Masterthesis wird jedoch auf die Funktionale Migration eingegangen. Unter der Funktionalen Migration wird verstanden: der Vorgang, fachliche Funktionalitäten eines Altsystems auf eine neue Plattform zu transformieren.
*„Model Driven Architecture (MDA) ist ein von der OMG initiierter Standard für die modellgetriebene Software-Entwicklung. Der MDA-Standard soll für die Interoperabilität zwischen verschiedenen Entwicklungswerkzeugen sorgen und Software-Modelle automatisch in den Quellcode überführen. Die Model Driven Architecture unterstützt die Erstellung eines Modells, das unabhängig von einer konkreten technischen Plattform ist „*Wikipedia [2009]. Um die fachliche und technische Trennung zu realisieren, basiert das MDA-Konzept auf den folgenden drei Abstraktionsebenen:

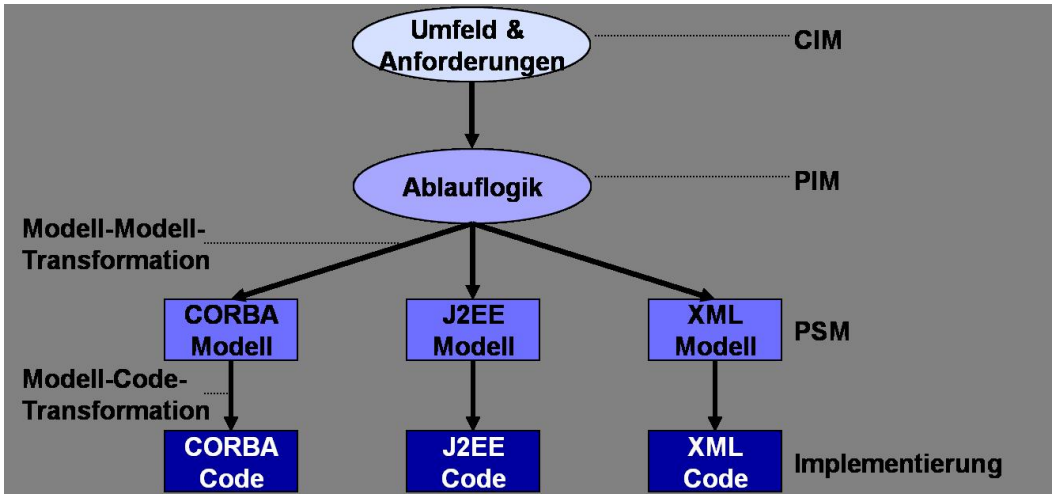

Abbildung 1.1: MDA Modellebenen

CIM - Computation Independent Model
Ist die höchste Abstraktionsebene und dient der Erfassung aller Anforderungen und stellt

das System aus Sicht der Geschäftsanwendungsfälle dar. Es ist somit unabhängig von der Implementierung des Systems. Das CIM entspricht weitgehend dem Analysemodell und ist in Fachkreisen auch unter dem Namen Domänenmodell oder Fachmodell bekannt.

PIM - Platform Independent Model
Die nächste Stufe wird über das PIM (Platform Independent Model) realisiert, das ohne konkrete Eigenschaften über die Plattform das Verhalten des Systems beschreibt. In den PIMs wird das fachliche Wissen (Fachlogik) des Softwaresystems/Anwendung, wie z. B. Geschäftsprozesse oder Fachverfahren, technologieunabhängig erfasst und modelliert. Es entspricht einem Entwurfsmodell ohne Detailinformation über die verwendete Plattform, implementierungsspezifische Details wie

- Betriebssystem,
- Programmiersprache,
- Datenverwaltungssystem
- etc.

werden ausgeblendet.

PSM - Platform Specific Model
Die letzte Stufe wird mit dem PSM (Platform Specific Model) erreicht,das konkrete Eigenschaften der Zielplattform beinhaltet. Die Implementierungstechnologien, d.h die technischen Aspekte bezogen auf eine konkrete Plattform, werden definiert. Hierbei handelt es sich um die letzte Modellierungsebene vor der Implementierung; diese enthält alle Informationen, die benötigt werden, um einen ausführbaren Code zu erzeugen. Siegel [2005] OMG [2008]

Wie bereits erwähnt, basiert das herkömmliche MDA-Konzept auf der Modellierung eines stabilen Modells, welches das Fundament für die automatisierte Quellcodegenerierung bildet. Die Masterthesis geht noch einen Schritt weiter, die Intention der Verfasserin ist es, nicht nur den Quellcode aus dem abstrakten Modell automatisch generieren zu lassen, sondern ebenso das abstrakte Modell aus dem Altsystem Quellcode.
Die automatisierte Modernisierung von Legacy Systemen umfasst im Groben die automatische Generierung des abstrakten Modells aus dem Quellcode des Altsystems sowie die Überführung des Zielsystems basierend auf dem abstrakten Modell. Dies lässt sich wie folgt zusammenfassen: Aus dem Altsystem Quellcode wird ein Platform Specific Model (PSM) und im Anschluss das Platform Independent Model (PIM) erzeugt. Daran anknüpfend kommt das standardisierte MDA-Konzept zum Einsatz, mit dem die Zielumgebung automatisiert erstellt werden kann.

Das hier vorgestellte Konzept der automatisierten Modernisierung von Legacy Systemen ist ein sehr großes Themengebiet und würde den Rahmen der Masterarbeit sprängen. Aus diesem Grund hat die Verfasserin das Themengebiet eingegrenzt. Der Schwerpunkt der Masterthesis ist die automatische Generierung des abstrakten Modells basierend aus dem Sourcecode des Altsystems. Dieser Forschungsansatz bringt viele Herausforderungen mit sich. In der Masterarbeit müssen wichtige Fragen beantwortet werden, wie, welche Arten von Altsystemen mit dieser Methodik überführt werden können. Welche Voraussetzungen muss das Altsystem in welchen Bereichen erfüllen? Die Voraussetzungen, der ein Altsystem Genüge leisten muss, wird mehrere Schichten des Altsystems betreffen. Die Debatte kann sich auf die architektonische Ebene, die Ebene der Standard Protokolle und Technologien, die Quellcodeebene und auf weitere Ebenen ausweiten.
Die Verfasserin möchte an dieser Stelle auf ihre Seminarausarbeitung verweisen, in der der Leser einen detaillierten Überblick über weitere Herausforderungen und Ziele der Masterarbeit erhält
Die Veranstaltung Anwendungen II soll sich schwerpunktmäßig auf ein Themengebiet der Masterarbeit beziehen. Die Ausarbeitung dieses Themenbereiches soll eine umfangreiche Analyse bereits existierender Forschungen, Projekte, Anwendungen, Methodiken usw. beinhalten. Aus diesem Grund hat sich die Verfasserin entschieden, auf Code Analysatoren einzugehen, denn die Analyse des Altcodes ist die Basis zur Überführung des Altsystems in ein abstraktes Modell und bildet somit einen sehr wichtigen Schwerpunkt der Masterarbeit.

2 Codeanalyse

Wie bereits erwähnt, ist ein wichtiger Bestandteil der automatisierten Migration die Analyse des Altcodes. Es muss auf Codeebene analysiert werden, ob ein Altsystem entsprechende Voraussetzungen erfüllt, um automatisiert in eine Zielplattform überführt werden zu können. Des weiteren wird in den meisten Fällen ein Altsystem, welches über Jahre gewahrt wurde, Codeunschönheiten aufweisen, auch wenn das Altsystem auf den anderen Ebenen (wie z.B. der architektonischen, technologischen Ebene) hochqualifiziert ist. In solchen Fällen sollte der Code des Altsystems Mithilfe einer Codeanalyse bereinigt werden. Die Codebereinigung oder weitere Optimierungsverfahren werden auch in den anderen Phasen der automatisierten Migration eine wichtige Rolle spielen.
Zusammengefasst ist die Codeanalyse für die Masterarbeit in drei wichtigen Bereichen von großer Bedeutung:

- Um entscheiden zu können, ob das Altsystem alle benötigten Bedingungen auf Codeebene erfüllt. Welche Bedingungen das Altsystem genau erfüllen muss, wird in der Masterarbeit diskutiert.
- Zur Bereinigung des Altcodes bezüglich seiner Codemängel
- Zur Überführung des Altcodes in ein abstraktes Modell.

Die Codeanalyse wird bereits seit Jahrzehnten verwendet, und man findet sie in vielen Bereichen der IT wieder. Eine ausschlaggebende Rolle spielt die Codeanalyse in der Qualitätssicherung. Auch in diesem Zusammenhang wird die Codeanalyse aus Sicht der Qualitätssicherung betrachtet.

2.1 Klassifikation

Es existieren zahlreiche Klassifikationen im Bereich der Codeanalyse; nachfolgend werden die wichtigsten erläutert:

statistisch vs. dynamisch
Es existieren zwei Arten der Codeanalyse; die dynamische und die statische. Unter der

dynamischen Codeanalyse werden Testabläufe oder Testfälle verstanden, die in der Regel Testdaten benötigen. Zudem wird in den meisten Fällen ein lauffähiges System vorausgesetzt. Die bekanntesten dynamischen Codeanalysen sind die Unit-Tests, mit dem jeder Software-Entwickler sicherlich in seiner Arbeit konfrontiert wurde.
Für die statische Codeanalyse ist kein aktives System erforderlich, es genügt lediglich der blanke Quellcode für die Analyse.
Welches der beiden Verfahren die bessere Analyse erbringt, kann nicht beantwortet werden. Es ist abhängig von der Analyse, welches Verfahren eingesetzt wird.

manuell vs. Toolbasiert:
Eine Codeanalyse kann manuell von einem Menschen oder automatisiert durch unterstützende Tools durchgeführt werden. Beide Alternativen weisen Ihre Vor- und Nachteile auf. Welche der beiden Möglichkeiten am effizientesten ist, ist abhängig von der Problemstellung.
Analysen, die in einfachen Algorithmen abgebildet werden können, sollten automatisiert durchgeführt werden, da sie in diesem Fall sicherer und kostengünstiger sind. Die Monotonität der Analyse einfacher Regeln fordert die Kreativität des Analysten nur gering, und das Auftreten von Fehlern ist daher sehr wahrscheinlich. Zudem ist die Analyse trivialer Regeln angewendet auf umfangreiche Codes sehr anstrengend und kann ebenso zu Inkorrektheit führen.
Auf der anderen Seite gibt es viele Bereiche, in denen es nicht möglich ist, auf Tools zurück zu greifen. Analysen, die die menschliche Denkfähigkeit beanspruchen, sind in vielen Fällen auch nur von Menschen analysierbar, weil die Regeln nicht trivial auf Algorithmen abbildbar sind, oder zu viele Algorithmen konstruiert werden müssten. Eine manuell durchgeführte Analyse ist nicht unkompliziert, insbesondere bei komplexen Systemen. IEEE hat eine Vorgehensweise zur manuellen Codeanalyse standardisiert, welche in sechs Phasen gegliedert ist.

syntaktische vs. semantische
Unter der syntaktischen Codeanalyse wird die Anordnung oder Zusammenstellung des Programmcodes verstanden. Heutzutage beinhaltet jeder Compiler eine syntaktische Codeanalyse. Programmiert ein Software-Entwickler ein Java Programm in der Entwicklungsumgebung Eclipse, so wird z.B. jeder Methodenaufruf einer nicht vorhandenen Methode als Fehler markiert. Dies ist eine syntaktische Analyse, die heute nicht mehr wegzudenken ist. Auch die qualitätsunterstützenden Checkstyle-Tools basieren auf der syntaktischen Codeanalyse.
Bei der semantischen Codeanalyse geht es um die Bedeutung des Programmcodes. Es sei ein einfaches Beispiel zum besseren Verständnis gezeigt. Wagner [2008] [Hoffmann 2008, Kapitel 5] Bartholomew [2008]

```
if(true){
...
}else{
...
}
```

Die zweite Klammer ist ein unreadable code, da dieser Code niemals gelesen werden kann. Die semantische Codeanalyse ist unter anderem dafür zuständig, derartige Fehler aufzuspüren. Ebenso soll die semantische Analyse Code-Duplizierungen oder einen ähnlichen Code, der an mehreren Stellen auftritt, auffinden, um somit diese Schwachstellen zu identifizieren.

2.2 Bewertung

Alle Codeanalyse-Typen weisen jeweils Vor- und Nachteile auf. Es ist abhängig von der Problemstellung, welche der Analysetypen am effizientesten sind. Aus diesem Grund wird an dieser Stelle näher erläutert, aus welchem Grund die Codeanalyse für die Masterarbeit relevant ist. Wie oben bereits erwähnt, ist aus drei Gründen die Codeanalyse relevant. Es soll auf Basis der Codeebene analysiert werden können, ob ein Altsystem alle notwendigen Voraussetzungen für die Erstellung eines abstrakten Modells erfüllt. Des weiteren sollen Codeunreinheiten aufgespürt und beseitigt werden. Als letztes soll die Codeanalyse die Generierung des abstrakten Modells in einigen Bereichen unterstützen. Nachfolgend geht die Verfasserin auf die einzelnen Problemstellungen genauer ein.

Qualität des Altcodes
Damit ein Altsystem automatisiert in ein abstraktes Modell überführt werden kann, muss es sich um ein qualitativ hochwertiges System handeln. Das resultierende abstrakte Modell muss ebenso einen bestimmten Grad an Genauigkeit aufweisen, damit es später in eine beliebige Zielplattform transformiert werden kann. Die Debatte über die Voraussetzungen eines Altsystems ist nicht trivial. Hinzu kommt, dass auf verschiedenen Ebenen bzw. Schichten das Altsystem hochwertig sein muss. Diese Ausarbeitung beschränkt sich auf die Codeebene. Die Beantwortung der Fragestellung, was ist ein hochwertiger Altcode, der in ein abstraktes Modell überführt werden kann, wird in der Masterarbeit diskutiert. An dieser Stelle wird untersucht, welche Codeanalyse-Typen relevant für unsere Problemstellung sind. Zuerst sollen die statischen und dynamischen Analyse-Typen analysiert werden. Da bereits

ein Altsystem über mehrere Jahre im Einsatz ist uns sich bewährt hat, ist die dynamische Codeanalyse nicht von Interesse. Dagegen erweist sich die statische Analyse als großen Nutzen. Mit der statischen Analyse lassen sich Bedingungen, die das Altsystem erfüllen muss, überprüfen. Hierfür ist keine syntaktische sondern nur eine semantische Codeanalyse von Bedeutung. Syntaktische Fehler wären bereits bei der Kompilierung aufgefallen und sind irrelevant. Die semantische Codeanalyse ist in diesem Kontext von großem Interesse. Diese Analyse beinhaltet nicht nur das Finden von z.B. unreadable Codes, sondern bietet zudem die Möglichkeit, andere Bedingungen zu definieren, die einen überführbaren Altcode ausmachen. Für die Qualitätsanalyse können unterstützende Tools verwendet werden, aber auch diese werden an Ihre Grenzen stoßen. Die Qualitätsbewertung kann sich daher nicht nur auf Tools stützen, sondern es wird ebenso der menschliche Verstand benötigt. Es soll erwähnt werden, dass bei der Beurteilung des Altsystems auf den anderen Ebenen/Schichten ebenso nicht auf die manuelle Analyse verzichtet werden kann.

Bereinigung des Altcodes

Ein Altsystem, das migriert werden soll, ist in den meisten Fällen ein System, welches über Jahre betrieben und gewartet wurde. Sind einige Grundregeln bei der Wartung berücksichtigt worden, so kann es sich immer noch um ein qualitativ hochwertiges System handeln, welches sich in alen Schichten und Ebenen bemerkbar macht. Jedoch können einige Codeunreinheiten auftreten, wie z.B. ein ähnlicher duplizierter Quellcode. Bevor der Code in ein abstraktes Modell überführt wird, sollte er, soweit möglich, gesäubert werden. Hierfür ist eine Codeanalyse notwendig. Auch in diesem Kontext ist die Verwendung der statischen und nicht der dynamischen Analyse vorzuziehen. Die dynamische Analyse würde an dieser Stelle nicht sinnvoll sein, denn diese Codeunreinheiten können lediglich mit der statischen Analyse entdeckt und somit eliminiert werden. Ebenso ist es nicht sinnvoll, auf die syntaktische Analyse zurück zu greifen; die semantische Analyse ist hire angebracht, es sei denn, die Codebereinigung beinhaltet Normen wie beschränkte Längen von Klassennamen. In so einem Fall werden sowohl die syntaktische als auch die semantische Codeanalyse verwendet. Für eine derartige Codebereinigung wird in den meisten Fällen ein Tool verwendet, wobei nicht auszuschließen ist, dass in einigen Bereichen auf die manuelle Analyse zurück gegriffen werden muss.

Überführung des Altcodes

Bei der Überführung des Altcodes in ein abstraktes Modell wird ebenfalls auf die Codeanalyse zurückgegriffen. Die statische Analyse ist hier der sinnvollste Anwendungstyp. Da erst die Masterarbeit genau Aufschluss darüber geben wird, wie die Überführung vonstatten geht, können an dieser Stelle nur grob und mit äußerster Vorsicht Annahmen getroffen werden.
Höchstwahrscheinlich wird die Überführung des Altsystems in ein abstraktes Modell in mehreren Phasen durchgeführt. Die Herausforderung bei der Erstellung des abstrakten Mo-

dells besteht nicht nur darin, den Altcode eins zu eins zu überführen. Es müssen ebenfalls architektonische und andere Aspekte mit berücksichtigt werden.
Unabhängig davon kann derzeit die Aussage getroffen werden, dass die syntaktische Codeanalyse von großer Bedeutung ist. Der Altcode muss auf jeden Fall geparst und die Informationen in ein beliebigens Format compiliert werden. Für diese Tätigkeit ist eine syntaktische Codeanalyse von Vorteil. Auch die semantische Codeanalyse ist von großer Bedeutung. Mithilfe dieser können zum Beispiel Patterns im Altsystem gefunden werden.
Da das Masterthema die automatisiere Migration umfasst, wird versucht, bei der Überführung lediglich auf Tools zurück zugreifen. Jedoch wird an der einen oder anderen Stelle Hand angelegt werden müssen.

Zusammenfassend werden die Teilprobleme und die dafür notwendigen Codeanalyse-Typen tabellarisch dargestellt.

-	Qualität des Altcodes	Bereinigung des Altcodes	Überführung des Altcodes
dynamisch	nein	nein	?
statisch	ja	ja	ja
manuell	ja	ja/nein	ja/nein
automatisiert	ja	ja	ja
syntaktisch	nein	ja/nein	ja
semantisch	ja	ja	ja

Tabelle 2.1: Übersicht über die benötigten Codeanalyse-Typen

3 Automatisierte Migration Tools

Der Bereich der automatisierten Migration von Altsystemen ist in der IT noch sehr wenig erforscht worden; vielen ist noch Neuland. Einige Großunternehmen sind jedoch auf diesem Gebiet tätig. Diese Forschungen werden intern in den einzelnen Großunternehmen durchgeführt, und sind strengst vertraulich. IBM entwickelt derzeit ein Tool mit dem Namen ARC (Analysis Renovation Catalyst), welches basierend auf dem MDA-Konzept, neue Anwendungen aus einem bestehenden Quellcode erzeugt.

Da das Vorstellen des ARC-Tools und dessen unterstützter Methoden den Rahmen dieser Ausarbeitung sprengen würde, wird nur ein kleiner Teil von ARC ins Visier genommen. Hierbei handelt es sich um den Teil, der bereits in dieser Ausarbeitung als Schwerpunkt gewählt wurde, den Analysevorgang des bestehenden Quellcodes.

In Abbildung 3.1 wird dem Leser ein Gesamtüberlick über die ARC-Methode gegeben. Jedoch wird lediglich der erste Teil, die Analyse, kurz erläutert.

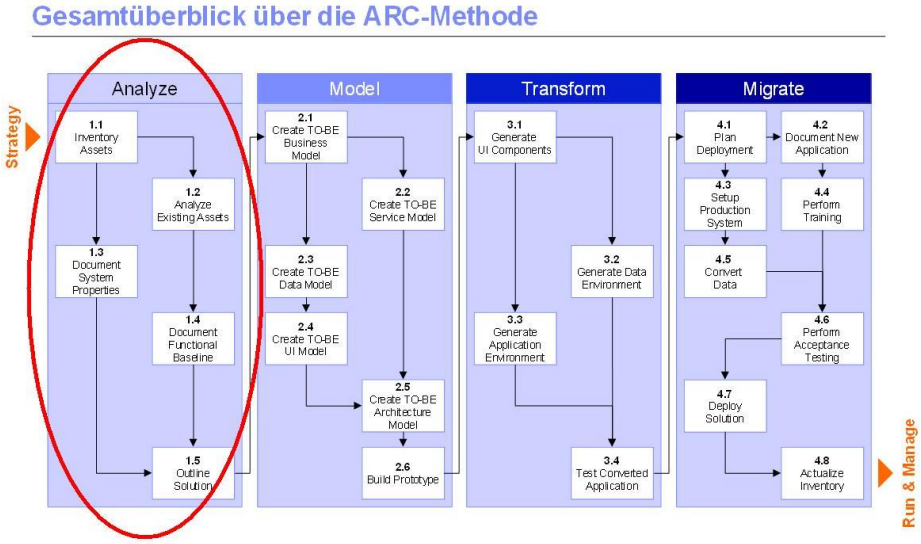

Abbildung 3.1: Gesamtüberblick über die ARC-Methode;Lojek u. Nogli [2008]

1.1 In dieser Aufgabe wird das bestehende System untersucht im Hinblick darauf, welche Vorgehensweise bei der Renovierung sinnvoll ist, untersucht; Renovierung, Migration oder Wiederverwendung.

1.2 In dieser Aufgabe wird das Inventory auf den Zustand des Codes untersucht, es werden die Hauptkomponenten der Anwendungen identifiziert und die Geschäftsregeln extrahiert. Es wird die Ausführungsreihenfolge identifiziert.

1.3 In dieser Aufgabe werden die Ausführungsmerkmale der Legacy-Anwendungen gesammelt um auf dieser Basis die nicht funktionalen Anforderungen an das migrierte System zu beschreiben.

1.4 In dieser Aufgabe werden die funktionale Ausgangslage für den Umbau erstellt, die gegenwärtigen Geschäftsprozesse dokumentiert und mit den vorhandenen Komponenten im Asset Inventory verbunden.

1.5 Das hauptsächliche Ziel dieser Aktivität besteht darin, eine Grundlage über die gesamte Komplexität und den Rahmen des Lösungsdesigns zu erstellen, das dominiert wird von der Anwendungs- und Infrastruktursicht, ausgehend von den gesammelten Daten in den Komponenten und operationalen Modellen. Es werden die architektonischen Komponenten skizziert und bei der weiteren Vorgehensweise werden diese bei der Transformation berücksichtigt. Lojek u. Nogli [2008]

4 Fazit

Das Themengebiet Modernisierung von Legacy Systemen unter Verwendung von MDA-Konzepten, oder auch automatisierte Migration von Altsystemen, unabhängig davon, unter welchen Namen man diesen Themenbereich abgrenzt, stellt ein sehr komplexes Gebiet dar. Nicht nur eine enorm hohe Problematik ist darin enthalten, zudem ist das Themengebiet noch nicht genügend erforscht. Besonders der Teilbereich, der durch die Masterarbeit abgegrenzt ist, befindet sich in einer frühen Erkundschaftungsphase. Die Überführung von einem Altsystem in ein abstraktes und vollständiges Modell ist hierbei die große Herausforderung, zumal das abstrakte Modell in so weit qualitativ hochwertig und genau sein muss, dass Mithilfe der MDA ein Zielsystem generierbar ist.
Diese Ausarbeitung bot den Lesern einen Gesamtüberblick über die Codeanalyse, welche in einigen Bereichen der Modellgenerieurng von großer Bedeutung ist. Zuerst wurde der Leser in die einzelnen Codeanalyse Klassifikationen und Typen eingeweiht. Anschließend wurden diejenigen Teilprobleme vorgestellt, bei der eine Codeanalyse zur effektiven Problemlösung beiträgt. Für jedes Problem wurde ebenso vorgestellt, welche Codeanalyse am effizientesten ist und für diese Herausforderung zum Einsatz kommen sollte.
Abschließend wurde auf das Tool ARC eingegangen, welches von der IBM entwickelt wird. ARC ist ein Model driven Architecture (MDA)-Tool für die Erstellung von neuen Anwendungen auf Basis bestehender Sourcecodes. Zusammengefasst stellt dieses Tool das Masterthema dar. Es wurde in dieser Ausarbeitung lediglich auf den Analyseteil des Tool eingegangen, der für diese Ausarbeitung den entscheiden Bereich anschneidet.

Literaturverzeichnis

Bartholomew 2008
BARTHOLOMEW, Redge: Evaluation of Static Source Code Analyzers for Avionics Software Development. ACM, 2008

Hoffmann 2008
HOFFMANN, Dirk: *Statische Code-Analyse*. Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–76322–2

Lojek u. Nogli 2008
LOJEK, Robert ; NOGLI, Achim: Kurzeinführung Analysis Renovation Catalyst (ARC) Analyse, Transformation und Weiterentwicklung von Altanwendungen. (Internes IBM Paper) Application Innovation Services IBM Global, 2008

OMG 2008
OMG (Hrsg.): *OMG Model Driven Architecture*. Version: 2008. `http://www.omg.org/mda/`, Abruf: 19. Februar 2008

Siegel 2005
SIEGEL, Jon: Why use the model driven architecture to design and build distributed applications? ACM, 2005

Wagner 2008
WAGNER, Artur: Codeanalyse - Einleitung und Einführung in das Thema Codeanalyse mit dem Schwerpunkt statische Codeanalyse. Version: 2008. `http://www.ba-horb.de/fileadmin/media/it/studienprojekte/seminararbeiten/SWE3-071204/Codeanalyse.pdf`. 2008

Wikipedia 2009
WIKIPEDIA (Hrsg.): Version: 2009. `http://de.wikipedia.org/wiki/Model_Driven_Architecture`, Abruf: 19. Februar 2009