Inhaltsverzeichnis

1 Einleitung

Software war, ist und wird immer sehr komplex sein. Doch die Entwicklung muss termingerecht, vollständig und vor allem kostengünstig sein. Um diesen Anforderungen gerecht zu werden haben sich verschiedene Vorgehensmodelle entwickelt. Der Trend geht hin zu iterativen Vorgehensmodellen wie dem Spiralmodell oder agilen Entwicklungsmethoden. Die alternative zu den iterativen Vorgehensmodellen sind die Sequentiellen. Das wohl bekannteste sequentielle Vorgehensmodell ist das Wasserfallmodell. Beim Wasserfallmodell wird eine Sequenz von Aktivitäten festgelegt und abgearbeitet. Die Sequenz besteht im Allgemeinen aus den Phasen Anforderungsermittlung, Entwurf, Implementierung, Test, Betrieb und Wartung. MDA ist auch ein sequentielles Vorgehensmodell, umfasst im Groben aber nur die Phasen Entwurf und Implementierung. Ansonsten ähnelt MDA dem Wasserfallmodell sehr. Bei beiden Vorgehensmodellen sind die Phasen durch Zwischenergebnisse wie Dokumente oder Modelle gekoppelt. Während im Wasserfallmodell die Zwischenergebnisse als Informationsquelle für die nächsten Phasen genutzt wird, bilden diese bei MDA eine Basis zur automatisierten Weiterverarbeitung. MDA ist also kein vollständiges Vorgehensmodell, umfasst aber die wesentlichen Punkte für eine möglichst automatisierte Softwareentwicklung und kann in ein Vorgehen nach dem Wasserfallmodell integriert werden. Mittels automatisierter Verarbeitung von Zwischenergebnissen (Modellen) und plattformunabhängiger Programmierung wird bei MDA versucht den hohen Anforderungen an die Softwareentwicklung gerecht zu werden. Somit könnte der Trend zur agilen Softwareentwicklung abgeschwächt werden und sequentielle Vorgehen wie das Wasserfallmodell weiterhin eine weit verbreitete Methode bleiben.

2 Motivation für MDA

2.1 Softwareprojekte scheitern

Anders als in Bereichen wie der Automobilindustrie, wird Software trotz bekannter Fehler gekauft. Dabei können Fehler unterschiedliche Tragweite haben. Der Absturz der Ariane-Rakete zählt wohl zu einem der tragischsten Resultate von Softwarefehlern. [Gro14]

Der CHAOS Report der Standish Group befasst sich seit 1994 mit Softwareprojekten und deren Ausgang. Dafür werden alle zwei Jahre über 100.000 IT-Projekte analysiert und deren Ergebnisse einer von drei Kategorien zu-

geordnet. “successful“, “challenged“ und “failed“. Projekte in der Kategorie “successful“ wurden Termingerecht fertiggestellt und haben das Budget nicht überschritten. Zudem wird die Software eingesetzt und ist korrekt im Sinne der Spezifikation. Projekte welche korrekt und im Einsatz sind, aber länger in der Entwicklung gebraucht haben oder teurer waren als vereinbart, werden der Kategorie “challenged“ zugeordnet. Wurden Projekte erst gar nicht fertig gestellt oder werden nicht genutzt, dann zählen sie als “failed“.

Kategorie	1994	1998	2003	2009	2011
successful	16%	26%	34%	32%	34%
challenged	53%	46%	51%	44%	51%
failed	31%	18%	15%	24%	15%

Tabelle 1: Ergebnisse des CHAOS Reports

An den Ergebnissen des CHAOS Reports (vgl. Tabelle 1) wird ersichtlich, dass der Softwareentwicklungsprozess noch lange nicht ausgereift ist. Nachdem im ersten Jahrzehnt der Studie eine Verdoppelung der erfolgreichen Softwareprojekte (Kategorie 'successful) festgestellt wurde, stagniert das Ergebnis seitdem. [ZW05, S. 2 ff.]

2.2 Gründe für das Scheitern

Softwareprojekte werden schnell sehr Komplex und dadurch auch der Softwareentwicklungsprozess. Das ist nicht nur durch die Größe von Systemen bedingt. Auch schon kleinere Client-/Server-Systeme können abhängig von verschiedenen Plattformen, Frameworks und Librarys sein. Viele Arbeitsschritte bei der Programmierung müssen manuell getätigt werden, weil eine automatisierte Lösung fehlt und oft auch zu komplex wäre. Neuerungen in den verwendeten Technologien können dazu führen, dass das System überarbeitet werden muss. Bei großen Technologieänderungen muss das Personal womöglich noch geschult werden. Somit fließen Zeit und Geld in Teile des Systems, welche bereits entwickelt wurden. In [Wim05] werden dafür 4 Gründe genannt:

1. **Code besitzt eine zu niedrige Abstraktionsebene**
   Die meisten Softwareentwickler schreiben ihre Programme mit Hochsprachen. Diese abstrahieren zwar weitgehend von der Hardware und bieten Komfort in Verständlichkeit und technischen Erleichterungen (z.B. Schleifen), abstrahieren aber nicht von der Zielplattform (zur Verfügung stehende Technologien und Funktionen).

2. **Wiederverwendung basiert auf einer zu feinen Form**
   Da der entwickelte Code nicht plattformunabhängig ist, kann dieser auch nicht ohne weiteres auf anderen Plattformen wiederverwendet werden.
3. **Programmspezifikationen sind stark mit ihrer Infrastruktur verflochten**
   Die Infrastruktur von Plattformen besteht u.a. aus Domänen wie Programmiersprachen, Datenbankzugriffen und Betriebssystemzugriffen. Die Domänen bleiben aber länger erhalten als die konkret verwendeten Technologien. Wird eine Technologie nicht mehr unterstützt kann es sein, dass das Programm zu einem Großteil neu entworfen und implementiert werden muss.
4. **"Business to Software Mappings"werden nicht archiviert**
   Programmspezifikationen spannen einen Problemraum auf. Um diese zu implementieren muss der Problemraum in einen Lösungsraum abgebildet werden. Der Lösungsraum geht verloren, wenn nicht ausreichend dokumentiert wurde. Bei neuen Projekten mit ähnlicher Problemstellung wird dann erneut nach einer Lösung gesucht.

Für diese Probleme bietet die modellgetriebene Softwareentwicklung mit MDA Hilfsmittel und kann so zu einem qualitativ höherwertigerem Entwicklungsprozess und Ergebnis beitragen. [Wim05, S. 1 ff.]

3 MDA Konzepte

3.1 Grundkonzepte

MDA ist keine konkrete Implementation eines Tools, sondern ein abstrakter Lösungsvorschlag der OMG. Umsetzungen der Spezifikationen werden den Toolherstellern überlassen. Im Vordergrund steht eine gründliche Modellierung abstrakter Aspekte des Systems und das automatisierte überführen von abstrakten Modelltypen in spezifischere Modelltypen. Am Ende der Überführungskette entsteht dann der Quellcode (oder zumindest Quellcode-Skelette). MDA beschreibt somit eine Vorgehensweise zu effizienter Softwareentwicklung.

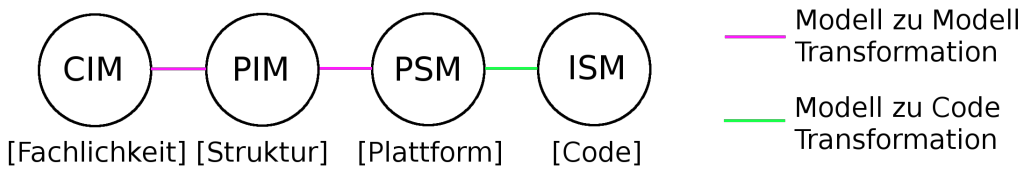

Abbildung 1: Modelle und Transformationen

In Abbildung 1 ist die bei MDA vorgesehene Überführungskette von Modellen visualisiert. Jeder Kreis repräsentiert einen der vier Modelltypen. Die Kanten zwischen den Modellen sind Transformationen. Diese überführen ein Modell in ein anderes Modell. Wie hier dargestellt, sind die Transformationen richtungsunabhängig. Die Überführungskette beginnt links in der Grafik (Abbildung 1) mit dem abstraktesten Modell CIM (Computation Independant Model), welches die in der Realität vorhandene Fachlichkeit modelliert. Das CIM wird in ein PIM (Platform Independant Model) überführt, welches eine plattformunabhängige Struktur des zu implementierenden Systems repräsentiert. Das PIM wird dann in ein PSM (Platform Specific Model) transformiert indem plattformspezifische Details einfließen. Aus dem PSM werden dann im letzten Schritt Code-Skelette generiert (ISM - Implementation Specific Model). Details zu Modelltypen und Transformationen folgen in den nächsten Kapiteln. Durch die Trennung von fachlicher und technischer Logik baut MDA nicht auf einer neuen Idee auf, sondern vereint bereits bekannte Methoden und Techniken zu einem standardisierten Vorgehen. Dabei kann MDA als eine Abstraktionsebene der Hochsprachen wie Java gesehen werden. Während diese noch Code für spezielle Plattformen erzeugen, hat MDA den Anspruch Systeme plattformunabhängig zu entwickeln.[ZW05, S. 60 ff.] Daraus ergeben sich folgende Vorteile:

- *„Das in den Modellen enthaltene Fachwissen kann über lange Zeiträume festgehalten werden.*
- *Bei einem technologiebedingten Plattformwechsel kann immer wieder auf dasselbe fachliche Modell zurückgegriffen bzw. kann dasselbe Modell auf verschiedenste Plattformen umgesetzt werden.*
- *Die Modellierung ist rein fachlich und somit entkoppelt von technologieabhängigen und werkzeugspezifischen Aspekten.*
- *Generatoren befolgen immer gleiche Regeln für die Abarbeitung gleichartiger Problemstellungen, so dass der produzierte Quelltext einfacher*

zu testen, pflegen und zu warten ist, da er durchgängig gleiche Bezeichnungen verwendet.

- *Standardaufgaben können durch ausreichende formale Modellierung automatisiert werden, so dass der entsprechende zugehörige Quelltext generiert werden kann.*

- *Alle fachlichen Änderungen werden im Modell vorgenommen, so dass das aufwändige Nachverfolgen einzelner Programmzeilen auf ein Minimum reduziert wird.“* [ZW05, S. 61 f.]

Als Hauptziele der MDA können die Übertragbarkeit, Interoperabilität und die Wiederverwendung identifiziert werden.[ZW05, S. 62]

3.2 Modelltypen

Die architektonische Trennung von Zuständigkeiten wird mit vier aufeinander aufbauenden Modelltypen erreicht. Es folgen Erklärungen zu den jeweiligen Modelltypen.

CIM - Computation Independant Model ist das abstrakteste der vier Modelltypen. Es werden Anforderungen an das zu modellierende System modelliert. In dem Modell wird die fachliche Umgebung spezifiziert, in welcher das System später eingebettet ist. So wird deutlich, welche Aufgaben das System später erfüllen muss. Der hauptsächliche Nutzen beim Erstellen eines CIM ist der Gewinn von Erkenntnissen über die fachliche Umgebung und Struktur seitens der Entwickler. Für eine automatisierte Weiterverarbeitung im Sinne von MDA ist ein CIM weniger wertvoll. Um das Verhalten zu modellieren eignen sich Use-Case Diagramme und Aktivitätsdiagramme. Wenn das erstellen eines CIM als erster Schritt in der modellgetriebenen Softwareentwicklung gesehen wird, dann unterscheidet sie sich bis jetzt noch nicht von der modellbasierten Softwareentwicklung.

PIM - Platform Independant Model modelliert Struktur und Verhalten des Systems. Anders als beim CIM wird ein System für einen Computer modelliert. Das heißt die Modelle beziehen sich nicht mehr nur auf die in der Realität vorfindbare Fachlichkeit sondern auch auf technische Aspekte wie z.B. Klassen und Abängigkeiten. Jedoch wird hier noch von konkrekten Plattformen abstrahiert. Zur Erstellung eines PIM reichen die in einem CIM modellierten Informationen in der Regel nicht aus. Folglich ist auch keine automatisierte Transformation von einem CIM zu einem PIM möglich. Bis hier unterscheidet sich die modellgetriebene Softwareentwicklung immer noch

nicht von einer gründlichen, modellbasierten Entwicklung.

PSM - Platform Specific Model ist durch das CIM und vor allem das PIM gegeben. Das PSM ist im Grunde eine Spezialisierung des PIM, indem zusätzlich plattformspezifische Details betrachtet werden. Plattformspezifische Details sind z.B. Programmiersprachen oder Kommunikationsprotokolle. Da alle benötigten Informationen in den bereits erstellten Modellen vorliegen, kann ein PSM automatisiert erstellt werden. Diese Transformation ist eines der Schlüsselkonzepte der MDA und ein entscheidender Schritt, welcher die modellbasierte von der modellgetriebenen Softwareentwicklung unterscheidet.

ISM - Implementation Specific Model ist der im letzten Schritt resultierende Quellcode des modellierten Systems. Die im PSM modellierte plattformabhängige Struktur wird automatisiert in Quellcode-Skelette transformiert. Die Informationen über funktionale Details fehlen im PSM. Deshalb kann kein fertiges System mit vollständigem Quellcode generiert werden. [ZW05, S. 63]

3.3 Transformation von Modellen

Modelltransformationen sind ein wichtiges Konzept in der MDA. Mit formalen Modellen als Grundlage, lassen sich Transformationen automatisiert durchführen. Erst dadurch ist eine plattformunabhängige Programmierung möglich. Modell zu Modell Transformationen werden als Mapping bezeichnet. Zu beachten ist, dass in diesem Unterkapitel keine Modell zu Code Transformationen betrachtet werden (diese folgen im nächsten Unterkapitel). Es gibt vier Arten von Mappings:

PIM → PIM Ein Mapping von einem PIM in ein anderes PIM stellt lediglich Veränderungen in Form von Verfeinerungen oder Verbesserungen dar. Diese Art von Transformation kann manuell wie auch gestützt von Werkzeugen durchgeführt werden.

PIM → PSM Diese Art von Transformation erweitert das PIM um plattformspezifische Details. Für dieses Mapping wird meist ein Werkzeug verwendet, welches den Vorgang automatisiert abarbeitet. Das verwendete Werkzeug muss wissen, für welche Plattform modelliert werden soll und wendet demnach die passenden Transformationsregeln an.

PSM → PSM Wie beim Mapping eines PIMs in ein anderes PIM, stellt

diese Transformation eine Veränderung in Form von Verfeinerungen oder Verbesserungen dar. Auch hier kann das Mapping von Werkzeugen gestützt sein.

PSM → PIM Mapping eines PSM auf ein PIM stellt eine Abstraktion von plattformspezifischen Details dar. Strukturen des Systems müssen analysiert und bearbeitet werden um ein vollständiges PIM zu erhalten. Eine solche Abstraktion ist sehr komplex und kann im allgemeinen nicht automatisiert ablaufen. [ZW05, S. 74 ff.]

3.4 Transformation vom Modell zum Code

Bei MDA findet die Transformation vom Modell zum Code im letzten Schritt statt, wenn vom PSM zum ISM transformiert wird. Zu dem Zeitpunkt hat das Modell den höchsten Grad an Spezifizierung erreicht. Dies bedeutet aber auch, dass wenig Spielraum bei der bevorstehenden Implementierung herrscht, sofern sich an die Modellvorgaben gehalten wird. Bei einer automatiserten Implementierung wie es hier der Fall ist, wird sich selbstverständlicherweise an die Modellvorgaben gehalten. Des weiteren sind die Modelle und der zu schreibende Code formal spezifiziert. Deshalb kann die Transformation vom Modell zum Code (vom PSM zum ISM) automatisiert vollzogen werden. [SV05, S. 25 f.]

3.5 UML als Modellierungssprache

MDA spezifiziert formale Modelle und braucht demnach auch eine formale Modellierungssprache. Dabei hat sich die OMG bei ihrem eigenen Standard, der UML bedient. Vorzüge der UML sind die Plattformunabhängigkeit, der deklarative Charakter und die Möglichkeit Modelle textuell und grafisch darstellen zu können. Die Formale Definition von MDA-Modellen wird durch UML-Profile erreicht. Damit werden die verwendbaren Modellelemente und deren Semantik festgelegt. [ZW05, S. 65 ff.]

4 Entwickeln mit MDA

4.1 Entwicklungsprozess

Im folgenden wird ein möglicher Softwareentwicklungsprozess mit MDA betrachtet und in Phasen eingeteilt. Wie auch bei Softwareentwicklung ohne MDA beginnt ein Projekt in der **Initailisierungsphase**. Es müssen z.B.

Mitarbeiter zugeteilt werden, ein Zeitplan entworfen und die zu verwendenden Technologien (z.B. einzusetzendes MDA Werkzeug) bestimmt werden. Es folgt eine erste Modellierung in der **Anforderungsanalyse**. Die Modellierung sollte auf einer abstrakten und berechnungsunabhängigen Ebene stattfinden. Hierbei entsteht das Computation Independant Model (CIM). Es folgen Iterationen von **Modellierung** und **Transformationen**. Damit ist MDA nun das Zentrum des Entwicklungsprozesses. Technische wie fachliche Bereiche der Anwendung müssen modelliert und in spezifische Modelle transformiert werden. Dabei sollten Modelle auf formale und inhaltliche Fehler überprüft werden, bevor sie mittels Transformation verfeinert werden. Die nach der letzten Transformation entstandenen Code-Skelette müssen dann noch in der Phase der **Programmierung** mit Funktionen gefüllt werden. Des weiteren sind Tests zu implementieren, da diese nicht komplett aus Modellen abgeleitet werden können. In der letzten Phase **Verteilung** und **Test** müssen die Systemkomponenten auf die vorhergesehenen Plattformen verteilt und konfiguriert werden. Anschließend muss das installierte System noch getestet werden. [Wim05, S. 37 ff.]

4.2 MDA Werkzeuge

Es existiert eine Vielzahl an MDA Werkzeugen. Einige sind kommerzielle Produkte, andere sind Open Source Projekte und frei verfügbar. Die Tools implementieren meist nicht die volle MDA Spezifikation, sondern beschränken sich auf Teilaspekte. Abhängig von ihrem Funktionsumfang, lassen sie sich in vier Kategorien einordnen, welche im folgenden näher betrachtet werden. [Wim05, S. 55]

4.2.1 Executable UML

Executable UML ist ein konkreter Vorschlag der OMG, wie MDA realisiert werden kann. Jedoch wird dabei nur eine Modell zu Code Transformation betrachtet, welche aus einem PIM (Platform Independant Model) Quellcode-Skelette generiert (ISM). Die PSM-Ebene wird nur als temporäres Modell bei der oben genannten Transformation betrachtet. Es kann also ein Softwaresystem plattformunabhängig mit UML modelliert werden und mit Hilfe eines Modell-Compilers daraus eine Implementation für eine bestimmte Plattform generiert werden.

Für die Modellierung des Softwaresystems bedient sich Executable UML nur einer Teilmenge der von UML bereitgestellten Notationselementen. Hauptsächlich sind dies Klassendiagrame, Zustandsdiagramme und Aktionen. Ne-

ben diesen Diagrammarten können auch noch andere verwendet werden, wie Anwendungsfalldiagramme oder Aktivitätsdiagramme. Da diese nicht transformiert werden können, dienen sie aber nur zur Dokumentation und für einen besseren Überblick über das modellierte System. Zur Beschreibung von Funktionen eines Objektes brauchen Executable UML Werkzeuge besondere Funktionen, welche nicht im UML Standard spezifiziert sind. Dafür kommt eine Action Semantic Language zum Einsatz. Tool-Hersteller liefern meistens ihre eigenen, proprietären Action Semantic Languages. In Tabelle 2 ist eine Übersicht von Executable UML Tools und der jeweils vewendeten Action Semantic Language. [Wim05, S. 55 ff.]

Toolhersteller	Produkt	Action Language
Kenedy Carter	iUML/ iCGG	Action Specific Language (ASL)
Kabira Technologies	Kabira Design Center	Kabira Action Semantics (Kabira AS)
Pathfinder Solution	PathMATE	PathMATE's Action Language (PAL)

Tabelle 2: Executable UML Werkzeuge [Wim05, S. 58]

4.2.2 Technologiespezifische Werkzeuge

Es gibt Werkzeuge, welche sich auf spezielle Technologien konzentrieren. Dies könnten z.B. Benutzeroberflächen für bestimmte Plattformen sein. Für diese Technologien kann dann eine sehr gute Unterstützung der Werkzeuge erwarten werden, jedoch geht dabei der so wertvolle Aspekt der plattformunabängigen Softwareentwicklung verloren. Besonders für die weit verbreiteten Plattformen JEE und .NET existieren solche fokusierten Werkzeuge. Tabelle 3 soll einen Überblick verschaffen. [Wim05, S. 59 f.]

Toolhersteller	Produkt	Unterstützte Plattform
Compuwarte	OptimalJ	J2EE
IBM	Rational XDE	J2EE, .NET
BITPlan	UML2PHP	PHP
Web Models	WebRatio	J2EE, .NET

Tabelle 3: Technologiespezifische Werkzeuge [Wim05, S. 60]

4.2.3 Eigentliche MDA Werkzeuge

Es gibt noch kein MDA Werkzeug, welches alle Konzepte von MDA zur Verfügung stellt. Vor allem Modell zu Modell Transformationen bereiten noch Schwierigkeiten. Es existieren jedoch Werkzeuge, welche sich nicht wie die Tools der vorigen zwei Kategorien auf wenige Teilaspekte beschränken, sondern bereits einen Großteil der MDA Konzepte umsetzen. Durch ausgereifte Modellierungseditoren können z.B. UML Profile definiert und Modell zu Modell Transformationen realisiert werden. Tabelle 4 zeigt eine (bescheidene) Übersicht von heute noch gepflegten Werkzeugen. [Wim05, S. 60 f.]

Toolhersteller	Produkt
SOFTEAM	Objecteering/UML
AndroMDA Team (open source)	AndroMDA

Tabelle 4: Eigentliche MDA Werkzeuge [Wim05, S. 61]

4.2.4 Codegeneratoren

Codegeneratoren sind MDA Werkzeuge, welche Modell zu Code Transformationen durchführen. Diese Transformationen sind nicht erweiterbar oder veränderbar. Meistens generieren diese nur Code für große Plattformen wie Java oder C#. Anders als bei Executable UML, kann aufgrund der Unveränderbarkeit der Transformationen nur Code für die modellierte Zielplattform erzeugt werden. Die generierten Codeskelette bestehen meist aus Klassen-, Attributen- und Methodendefinitionen. Diese Art von Werkzeug ist nur begrenzt zur Codegenerierung bzw. modellgetriebenen Softwareentwicklung einsetzbar. [Wim05, S. 61 f.]

5 Fazit

Mit der Idee von MDA lässt sich der Softwareentwicklungsprozess deutlich verbessern. Das Projekt wird auf verschiedenen Abstraktionsebenen dokumentiert und kann so, mit dem Fokus auf das wesentliche, optimal kommuniziert werden. Des weiteren ist der Aspekt der plattformunabhängigen Programmierung äußerst wertvoll hinsichtlich der Produktivität im Entwicklungsprozess. Leider ist MDA aber nur ein abstrakter Lösungsvorschlag. Das heißt es gibt von der OMG keine konkreten Werkzeuge, welche MDA implementieren. Andere Werkzeughersteller konzentrieren sich meistens nur

auf eine Teilmenge der MDA-Konzepte. Während einfache Codegeneratoren kaum Vorteile bringen, bieten Werkzeuge wie AndroMDA Modell-zu-Modell- und Modell-zu-Code-Transformationen. Auf Basis dieser mächtigen Konzepte kann dann die Struktur einer Anwendung plattformunabhängig erzeugt werden. Das Resultat ist viel generierter Code und eine Dokumentation des Softwaresystems auf verschiedensten Abstraktionsebenen und die Möglichkeit das System sowohl am Modell, wie auch im Code zu verändern. Das Einarbeiten in Werkzeuge erfordert immer einen gewissen Zeitaufwand. Bei so mächtigen MDA Werkzeugen wie die der Kategorie “Eigentliche MDA Werkzeuge“ (vgl. 4.2.2) kann aber im Laufe des Entwicklungsprozesses sehr viel Zeit gespart werden, wodurch jede Einarbeitungszeit für das Tool ausgeglichen wird. Das modellgetriebene Softwareentwickeln mit MDA bietet keinerlei Nachteile und sollte von jedem Programmierer ausprobiert werden.

6 Abbildungsverzeichnis

7 Tabellenverzeichnis

8 Literaturverzeichnis

[Gro14] Grotelüschen, Frank: *Der Absturz der Ariane 5*, August 2014. `http://www.deutschlandfunk.de/der-absturz-der-ariane-5.676.de.html?dram:article_id=25637`.

[SV05] Stahl, Thomas und Markus Völter: *Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management*. Dpunkt-Verlag, Köln, 1. Aufl. Auflage, 2005.

[Wim05] Wimmer, Manuel: *Model Driven Architecture in der Praxis - Evaluierung aktueller Entwicklungwerkzeuge und Fallstudie*. Diplomarbeit, TU Wien, 2005.

[ZW05] Zeppenfeld, Klaus und Regine Wolters: *Generative Software-Entwicklung mit der MDA -*. Spektrum Akademischer Verlag, Heidelberg, 2005. Aufl. Auflage, 2005.