Zimmermann, Frank

Working Paper

Nutzung und Erfolg modellgetriebener Softwareentwicklung

Arbeitspapiere der Nordakademie, No. 2009-10

Provided in Cooperation with:
Nordakademie - Hochschule der Wirtschaft, Elmshorn

Suggested Citation: Zimmermann, Frank (2009) : Nutzung und Erfolg modellgetriebener Softwareentwicklung, Arbeitspapiere der Nordakademie, No. 2009-10, Nordakademie - Hochschule der Wirtschaft, Elmshorn

This Version is available at:
https://hdl.handle.net/10419/38597

Arbeitspapiere der Nordakademie

ISSN 1860-0360

Nr. 2009-10

Nutzung und Erfolg modellgetriebener Softwareentwicklung

Prof. Dr. Frank Zimmermann

Dezember 2009

Eine elektronische Version dieses Arbeitspapiers ist verfügbar unter:
http://www.nordakademie.de/arbeitspapier.html

Köllner Chaussee 11
25337 Elmshorn
http://www.nordakademie.de

Nutzung und Erfolg Modellgetriebener Softwareentwicklung

Frank Zimmermann

2. Dezember 2009

Zusammenfassung

Modellgetriebene Softwareentwicklung MDSD mit Open Source Werkzeugen ist eine neue Entwicklung im Softwareengineering. Erfahrene IT Spezialisten erinnern sich an die CASE Euphorie der 90er Jahre. Die Argumentation der Verfechter beider Techniken ist ähnlich. Es stellt sich die Frage, welche Lehren aus der CASE Welle gezogen werden können und ob MDSD ein ähnliches Schicksal erwartet wie CASE.

1 Problemstellung

Mit dem Aufkommen der ersten leistungsstarken PCs Anfang der 90er Jahre sollte sich die kommerzielle Entwicklung von betrieblichen Informationssystemen revolutionieren. Bis dahin hat man Programme in den Programmiersprachen Cobol, PL1 oder Assembler geschrieben. Als Ausführungsumgebung wurden in den meisten Fällen die Transaktionsmonitore CICS und IMS/DC verwendet und als Datenbanken IMS/DB oder VSAM. Diese Umgebungen stellten für den Programmier eine Herausforderung dar, die ihn von der eigentlichen Umsetzung der geschäftlichen Anforderungen abhielt. Lange Projektlaufzeiten und geschäftlich häufig unbrauchbare Anwendungen waren die Folge. Durch die neuen technischen Möglichkeiten der Desktop Rechner sollte die Situation verbessert werden. Unter dem Stichwort CASE (Computer Aided Software Engineering) wurden Systeme angeboten , die deutliche Vorteile versprachen:

- Grafische Oberflächen ermöglichten die Darstellung visuell ansprechender Modelle auf hohem Abstraktionsgrad.
- Programmgeneratoren ermöglichten die Generierung von technisch anspruchsvollem, aber geschäftlich irrelevantem CICS bzw. IMS/DC Code. Ferner konnte durch den Einsatz von Generatoren der Umstieg von hierarchischen zu relationalen Datenbanken leichter geschafft werden, da die Anwendungsprogrammierer von dem Erlernen der neuen Datenbankanfragesprache SQL befreit werden konnten.
- Zentrale Repositories sollten die Unternehmenssoftwarearchitektur in Form von Unternehmensmodellen verwalten.
- Methodisches Vorgehen nach dem klassischen Wasserfallmodell sollte den Projekterfolg sichern. Die Verwendung des zentralen Repositories sollte von der Informationsstrategischen Planung (heute unter dem Begriff Multiprojektmanagement bekannt) bis hin zur Umsetzung der Systeme in den Programmen für redundanzfreie und nachverfolgbare Vernetzung der geschäftsrelevanten Informationen sorgen.
- Unterstützung bei der Anwendung von Software Entwicklungsmethoden.

Unter dem Projektnamen AD/Cycle[1] versuchte die IBM 1990, damals der marktbeherrschende Hersteller kommerzieller Soft- und Hardwaresysteme, eine Infrastruktur zu schaffen, die es verschiedenen Anbietern ermöglichen sollte, innerhalb des Rahmens Produkte zu entwickeln und zu vermarkten. Damit sollte der Best of Breed Ansatz unterstützt werden. Jeder Anwender sollte die seinen Anforderungen entsprechend beste Werkzeugkombination einsetzen. Standardisierungsgremien wie das National

[1] vgl. Mercurio u. a. (1990)

Institute of Standards and Technology (NIST) und die European Computer Manufacturers Association (ECMA), versuchten mit einem Standard feste/proprietäre Metamodelle einzelner Werkzeuge über definierte Import/Export Schnittstellen miteinander zu verbinden[2]. Dadurch konnten unterschiedliche Werkzeuge miteinander kombiniert und eine skalierbare Lösung für unternehmensweite Anwendung konfiguriert werden.

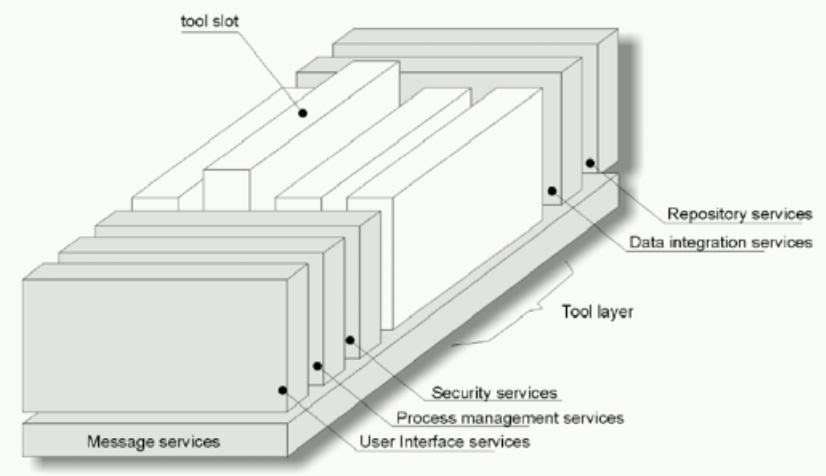

Abbildung 1: CASE Framework nach NIST und ECMA

Hersteller von CASE Werkzeugen versprachen

1. dramatische Verbesserungen in der Entwicklerproduktivität.
2. deutliche Verbesserungen in der Dokumentation der Software und dadurch deutlich reduzierte Wartungszeiten.
3. deutlich mehr Fokussierung auf geschäftliche Aufgabenstellungen.

Diese Versprechungen wurden durch empirische Studien angeheizt. So berichten Norman und Nunamaker (1989) von deutlich wahrgenommenen Vorteilen und Baker und Kaufmann (1991) sprechen von einer Verbesserung der Größenordnung der Produktivität.

Leider haben sich die zu Beginn der Case Hype durchaus nachvollziehbaren Vorteile nicht eingestellt. Kemerer (1992) berichtete schon 1992 von Untersuchungen die belegten, dass 70% der angeschafften CASE Tools nie , weitere 25% nur einmal und nur 5% wiederholt genutzt wurden.

2 Modellgetriebene Entwicklung

Die Modellgetriebene Softwareentwicklung (engl. Model Driven Software Development kurz MDSD) bezeichnet ein Vorgehensmodell für die Entwicklung von Software, bei dem nicht direkt der ausführbare Programmcode erstellt wird, sondern zunächst auf die Problemstellung angepasste domänenspezifische Modelle[3] erstellt werden. Domänenspezifisch bedeutet in diesem Zusammenhang, dass der Problembereich so stark eingeschränkt wird, dass sehr wenige aber sehr spezielle Konzepte übrigbleiben, um das Problem zu beschreiben. Diese domänenspezifischen Modelle werden in einer domänenspezifischen Sprache (Domain Specific Language DSL) formuliert. Designentscheidungen für die Verwendung von DSLs in Softwareprojekten werden von Mernik u. a. (2005) beschrieben. Prinzipiell unterscheidet man architekturzentrierte und businesszentrierte Problemdomänen. In architekturzentrierten Domänen werden Elemente der technischen Architektur modelliert. Sie kommen bevorzugt zum Einsatz, wenn komplexe Frameworks oder Frameworkstacks eingesetzt werden, deren Benutzung tiefgehende Kenntnisse des Zusammenspiel des/der Frameworks voraussetzen. Bei businesszentrierten Domänen sind die Modellelemente dem fachlichen Problembereich entlehnt. Sie kommen dagegen

[2] vgl. Hefernan (1990)

[3] Bézivin (2005) beschreibt eindrucksvoll den Nutzen von Modellen in der Softwareentwicklung

zum Einsatz, wenn das Expertenwissen gesammelt und dokumentiert werden soll. Es kann passieren, dass die Problemdomäne so komplex ist, dass mehr als ein Modell zur Beschreibung benötigt wird. Insbesondere bei dem MDA [4] Konzept der OMG werden zum Beispiel computation-independent, platform-independent und platform-specific Modelle, die deutlich in ihrem Detaillierungsgrad zunehmen, unterschieden. Das Besondere an MDSD ist, dass diese Modelle nicht nur zur Dokumentation genutzt werden, sondern über hinterlegte Vorlagen, sogenannte Templates, ausführbarer Code generiert [5]. wird. Im allgemeinen kann jedoch nicht die gesamte Applikation aus dem Modell generiert werden. Statt dessen versucht man, den Teil der Anwendung, der sich schematisch wiederholt, zu generieren und durch manuell geschriebenen Code zu ergänzen.

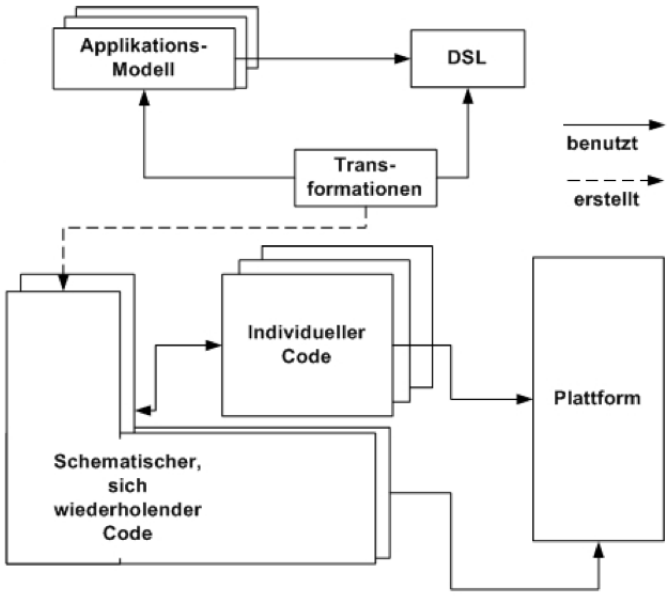

Abbildung 2: Modellgetriebene Entwicklung Stahl u. a. (2006)

Dadurch sind die Modelle fester Bestandteil der Systementwicklung. Insbesondere wird auch die Wartung immer im Modell vorgenommen. Danach ist eine erneute Generierung erforderlich, die manuelle Erweiterungen und Änderungen am Code nicht zerstören darf. Besonders wichtig ist es deshalb, ein Konzept für die Erhaltung der manuellen Änderungen zu haben, um einmal getane Arbeit nicht zu verlieren.

Allgemein anerkannte Ziele [6] von MDSD sind:

1. Beschleunigung des Entwicklungsprozesses durch Erhöhung der Abstraktion und Vermeidung von Cut&Paste von schematischem Code.

2. Erhöhung der Qualität des Codes durch automatisierte Transformationen, die 100% architekturkonformen Code erzeugen

3. Zentrale Fehlerbehebung durch zentrale Wartung innerhalb der Templates.

4. Steigerung der Wiederverwendung durch Bildung von Produktionsstraßen.

5. Verringerung der Komplexität der Softwareentwicklung durch Fokussierung auf wenige wesentliche Aspekte.

6. Bündelung des Expertenwissens in lebenden Modellen als Grundlage eines aktiven Wissensmanagements.

[4] vgl. Miller und Mukerji (2003)
[5] vgl. Stahl u. a. (2006)
[6] Weismann (2006)

3 Vergleich der Ansätze

Sowohl MDSD als auch CASE haben das gemeinsame Ziel, eine Produktivitätssteigerung in der Softwareentwicklung zu erreichen. Das betrifft insbesondere die Produktivitätsfaktoren Entwicklungsdauer und Qualität. In einer vergleichenden Studie von Krogmann und Becker (2007) konnte die Entwicklungsdauer eines Softwareartefakts durch Einsatz von MDSD Techniken um den Faktor 9 reduziert werden. Dabei soll sich die Codequalität auch noch verbessert haben. Dies entspricht etwa den Ergebnissen, die von Baker und Kaufmann (1991) berichtet wurden. Bei beiden Studien ist aber nicht mit eingerechnet, welcher Aufwand in der Erstellung der MDSD Lösung / des CASE Werkzeugs steckt. Es werden also nur die Grenzkosten miteinander verglichen. Deshalb kann diese Studie nicht als Beispielrechnung für die Praxis verwendet werden. Gerade bei dem MDSD Ansatz ist es nicht realistisch, die Entwicklung der MDSD Umgebung außer Acht zu lassen. Verfechter von MDSD fordern dagegen, dass die Gestaltung und Entwicklung der Entwicklungsumgebung mit denselben Qualitätsansprüchen betrieben wird wie die mit MDSD entwickelte Software[7].

Der Anspruch, den CASE Softwareentwicklungsmethoden zu unterstützen, geht deutlich über den Anspruch von MDSD hinaus. Die meisten Veröffentlichungen im Zusammenhang MDSD und Entwicklungsmethodik beziehen sich auf die Frage, ob sich MDSD mit agilen Methoden kombinieren lassen. Goerigk und Stahl (2009) präsentieren als Vorgehensmodell für MDSD Projekte nicht mehr als die parallele Erstellung eines Referenzmodells.

Aufgrund der großen Ähnlichkeit der beiden Techniken stellt sich die Frage, aus welchem Grund man MDSD ein anderes Schicksal prophezeihen kann als CASE. Iivari (1996) untersucht Gründe für die schlechte Akzeptanz von CASE. Er stellt ein Vorhersagemodell für die Nutzung und Erfolg von Case Werkzeugen aufgrund von Regressionsanalysen auf. Dabei sind die untersuchten Einflussgrößen (unabhängigen Variablen):

1. Beteiligung der Anwender an der Werkzeugauswahl
2. Unterstützung durch das Management
3. Training der Anwender
4. Realitätsnähe der Erwartungen
5. (Wahrgenommene) Komplexität der Anwendung der Werkzeuge
6. (Wahrgenommene) Kompatibilität zu Anwendergewohnheiten und Unternehmensprozeduren
7. (Wahrgenommener) Vorteil für die Anwender
8. Freiwilligkeit der Nutzung
9. (Wahrgenommene) Produktivitätsvorteile

Eines der wesentlichen Ergebnisse der Untersuchung war die besonders hohe Signifikanz des wahrgenommenen Vorteils der Anwender. Sie konnte durch eine schrittweise Regressionsanalyse belegt werden. Als zweites eher überraschendes Ergebnis wurde eine negative Korrelation zwischen CASE Nutzung und Freiwilligkeit der Nutzung gesehen. Der Autor schließt daraus, dass Anreize durch das Management erforderlich sind, um die Einführung von CASE zu ermöglichen. Und schließlich kann der Autor eine positive Korrelation von CASE Nutzung und CASE Erfolg nachweisen. Dabei waren die Auswirkungen auf die Qualität der Anwendungen größer als die Auswirkung auf die Entwicklungsdauer.

Fichman (1992) untersucht kritische Erfolgsfaktoren in einem IT Umfeld. Er beschreibt mit seinem IT Diffusion Framework, eine auf empirischen Daten basierende Methodik, in der Erfolgsfaktoren, die die Einführung einer Innovation betreffen, ermittelt werden können. Je mehr Erfolgsfaktoren eine Innovation hat, desto unwahrscheinlicher ist es, dass sich diese Innovation durchsetzt.

Diese Erfolgsfaktoren sind:

[7] vgl. Herde (2009)

- Wahrgenommene Innovationseigenschaften
  - Relativer Vorteil: Besitzt die Innovation einen Nutzen bzw. einen Vorteil gegenüber bisherigen Ideen oder Techniken?
  - Kulturverträglichkeit: Ist die Innovation mit bestehenden Werten oder Erfahrungen in Einklang zu bringen?
  - Komplexität: Wie schwierig ist die Handhabung bzw. das Verstehen der Innovation?
  - Testbarkeit: Ist die Innovation im Kleinen prüfbar, kann sie verifiziert werden?
  - Überprüfbarkeit: Sind die Ergebnisse einer Innovation beobachtbar?
- Eigenschaften des Innovationseinführers
  - Innovatoren: unternehmerisch, gebildet, nutzt vielfältige Informationsquellen, risikofreudig
  - frühe Annehmer: Vorbilder, angesehen, gebildet
  - die frühe Mehrheit: reflektierend, viele informelle Kontakte
  - die späte Mehrheit: skeptisch, konservativ
  - die Nachzügler: Beharren auf Bewährtem, risikoscheu
- Informationsquellen und Kommunikationskanäle
- Change Agents und Meinungsführer
- Architekt, technischer Projektleiter
- Management

Die Kernaussage des IT Diffusion Frameworks ist, dass die für eine Innovation erforderlichen Erfolgsfaktoren aus einer Einschätzung der Innovation und ihren Eigenschaften und einer Bewertung der einführenden Organisationseinheit abgeleitet werden kann. Abbildung 3 zeigt ein Portfolio, aus dem die Erfolgsfaktoren abgelesen werden können.

Abbildung 3: Herleitung Erfolgsfaktoren nach Fichman (1992)

Zur Einordnung der Eigenschaften der Innovation wird einerseits die Wissensbarriere für die Nutzung der Innovation und andererseits die gegenseitige Nutzerabhängigkeit herangezogen. CASE Werkzeuge hatten aufgrund des Umstiegs vom Großrechner auf den PC und der vielfach komplett neuen

Entwicklungsmethoden, die über CASE eingeführt werden sollten, eine hohe Wissensbarriere. Dieses erkennt man sofort, wenn man den Anspruch eines Werkzeuges, das von der Planung bis zur Umsetzung durchgängig funktionieren soll, bedenkt. Welche Konsequenzen der Einsatz eines Konzepts in der Planung für die Anwendung konkret hat, ist nur für die Tool-Experten absehbar gewesen. Wurden mehrere Tools miteinander kombiniert, so war dieses im ersten Anlauf unmöglich.

Die DSLs des MDSD sollen aber gerade die Welt des Anwenders wiederspiegeln. Die Interpretation erfolgt nicht methodenabhängig sondern ist auf die spezielle Situation eines Unternehmens angepasst. Dadurch ist die Wissensbarriere bei MDSD ungleich niedriger. Allein die Anforderungen an den Softwarearchitekten, der die DSL entwerfen muss, steigen. Er muss sich sehr gut mit den Werkzeugen und den Methoden des Entwurfs von DSLs, wie sie von Mernik u. a. (2005) beschrieben, werden auskennen.

Die gegenseitige Nutzerabhängigkeit bei CASE ist sehr hoch. Zwar sind CASE Pilotprojekte durchführbar, der Nutzen eines solchen Werkzeugs kann jedoch erst realisiert werden, wenn eine gesamte Entwicklungsabteilung inklusive der Qualitätssicherung und des Konfigurationsmanagements auf das CASE Werkzeug umgestiegen sind.

Andererseits wird modellgetriebene Softwareentwicklung immer für die Produktivitätssteigerung in einem Projekt eingesetzt. Aufgrund eines des üblicherweise zu erstellenden Referenzmodells fügen sich MDSD Projekte ideal in ein Unternehmensumfeld ein. Daher ist es sehr gut möglich zunächst mit einem kleinem Teil von Nutzern, nämlich den Entwicklern zu beginnen.

Im Falle von CASE wurde immer eine unternehmensweite Einführung angestrebt. Erst dann macht die Nutzung eines Repositories Sinn. Nur im unternehmensweiten Einsatz kann eine Entwicklungsmethode Erfolg zeigen. Das ist bei MDSD völlig anders. Bietet MDSD in einem Projekt einen Produktivitätsvorteil, so wird es für genau dieses Projekt eingesetzt werden. Ist dieser Vorteil überzeugend, so wird sich die Technik im Unternehmen verbreiten. Die Einführung von MDSD folgt den Regeln der Evolution.

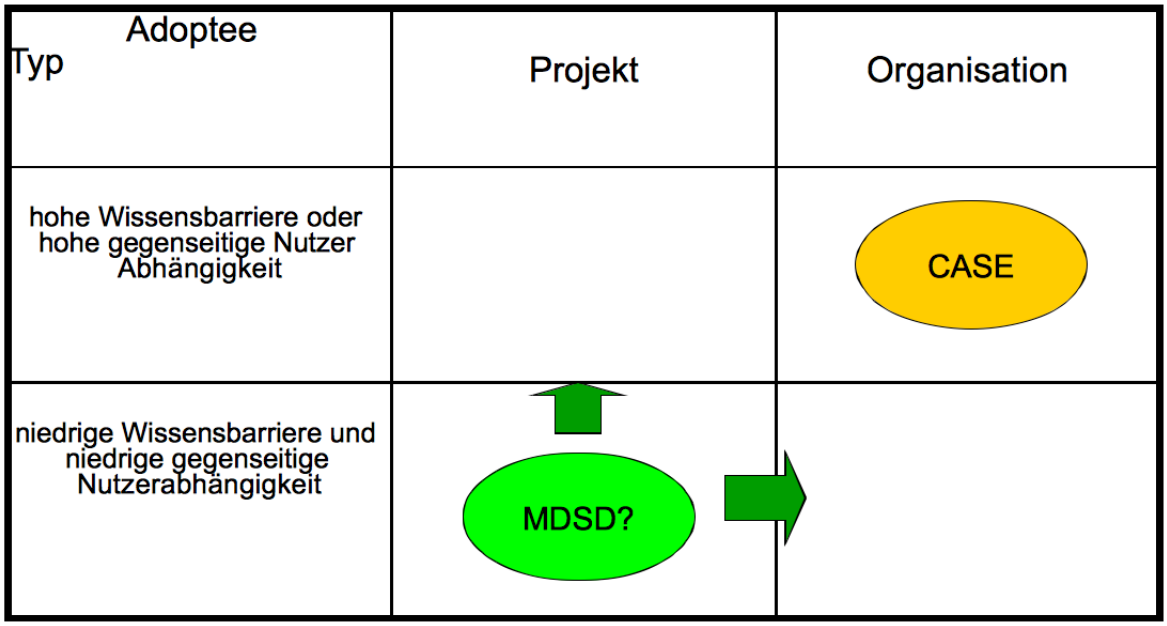

Abbildung 4: Einordnung von CASE und MDSD in das Portfolio nach Fichman (1992)

Abbildung 4 zeigt die Einordnung von CASE und MDSD in der Portfoliomatrix. CASE befindet sich in dem kritischen Bereich, der die meisten Erfolgsfaktoren aufweist. MDSD dagegen hat aufgrund des geringeren Anspruchs deutlich weniger Erfolgsfaktoren. Es ist deshalb davon auszugehen, dass MDSD durchaus nicht dasselbe spektakuläre Misserfolgsszenario aufweisen wird, wie es den CASE Anwendungen ergangen ist.

4 Zusammenfassung

CASE und MDSD sind verwandte Techniken, wobei der methodische Aspekt bei MDSD deutlich geringer ausgeprägt ist. Die Portfolioanalyse nach Fichman (1992) zeigt, dass wesentlich Unterschiede in der Reichweite der Auswirkungen von CASE und MDSD liegen, die nahelegen, dass eine Umsetzung von CASE deutlich mehr Risiken aufwirft als eine Einführung von MDSD. Allerdings kann im Erfolgsfall MDSD durchaus zu einem unternehmensweiten Einsatz kommen. Insofern kann man die Aussicht, dass MDSD überleben wird, optimistischer einschätzen als es bei CASE der Fall war. Die wesentlichen Erfolgsfaktoren können aus der Analyse von Iivari (1996) entnommen werden. Das sind einerseits der wahrgenommene Vorteil für den Anwender und andererseits die Ermutigung durch das Management, das den Weg zu einem evolutorisch wachsenden Einsatz von MDSD eröffnen muss .

Literatur

[Baker und Kaufmann 1991] Baker, R.D. ; Kaufmann, R.J.: Reuse and productivity in integrated computer aided software engineering. In: *MIS Quaterly* 15 (1991), Nr. 3, S. 375–401

[Bézivin 2005] Bézivin, Jean: On the Unification Power of Models. In: *Software and System Modeling (SoSym)* 4 (2005), Nr. 2, S. 171–188. – URL `http://www.sciences.univ-nantes.fr/lina/atl/www/papers/OnTheUnificationPowerOfModels.pdf`

[Fichman 1992] Fichman, Robet G.: Information Technology Diffusion: A Review of Empirical Research. In: Gross, J.I. (Hrsg.) ; Becker, J.I. (Hrsg.) ; Elam, J.J. (Hrsg.): *Proceedings of the third International Conference on Information Systems*, 1992, S. 195–206

[Goerigk und Stahl 2009] Goerigk, Wolfgang ; Stahl, Thomas: *10 Jahre Model DrivenSoftware Development*. 6 2009. – URL `http://se.informatik.uni-kiel.de/wp-content/uploads/2009/04/software-architektur-2009_goerigk.pdf`

[Hefernan 1990] Hefernan, Henry: NIST, ECMA join forces. In: *Software Magazine* 29 (1990). – URL `http://findarticles.com/p/articles/mi_m0SMG/is_n12_v10/ai_9536763/`

[Herde 2009] Herde, Hartmut: Kritische Fragen eines IT Managers zu Modellbasierter Softwareentwicklung. In: Schröder (Hrsg.) ; Zimmermann (Hrsg.): *Tagungsband zum ersten Elmshorner Wirtschaftsinformatiktag* Bd. 1 Nordakademie (Veranst.), Shaker Verlag, 10 2009, S. 97–105

[Iivari 1996] Iivari, Juhani: Why Case Tools are not used. In: *Communications of the ACM* 39 (1996), 10, Nr. 10, S. 94–103

[Kemerer 1992] Kemerer, C.F.: How the learning curve affects CASE tool adoption. In: *IEEE Software* 9 (1992), Nr. 3, S. 23–28

[Krogmann und Becker 2007] Krogmann, Klaus ; Becker, Steffen: A Case Study on Model-Driven and conventional software development. In: *Proc. Software Engineering 2007, Lecture Notes in Informatics (LNI)* 106 (2007), S. 169–176. – URL `http://www.sciences.univ-nantes.fr/lina/atl/www/papers/OnTheUnificationPowerOfModels.pdf`

[Mercurio u. a. 1990] Mercurio, V. J. ; Meyers, B. F. ; Nisbet, A. M. ; Radin, G.: AD/Cycle strategy and architecture. In: *IBM Systems JournaL* 29 (1990), Nr. 2, S. 170. – URL `http://www.research.ibm.com/journal/sj/292/ibmsj2902C.pdf`. – ISSN ISSN 0018-8670

[Mernik u. a. 2005] Mernik, Marjan ; Heering, Jan ; Sloane, Anthony M.: When and how to develop domain-specific languages. In: *ACM Comput. Surv.* 37 (2005), Nr. 4, S. 316–344. – URL `http://fparreiras/papers/WhenHowDevelopDSL.pdf`. – ISSN 0360-0300

[Miller und Mukerji 2003] Miller, J. ; Mukerji, J.: MDA Guide Version 1.0.1 / Object Management Group (OMG). 2003. – Forschungsbericht

[Norman und Nunamaker 1989] Norman, R.J. ; Nunamaker, J.F.: CASE produktivitiy perceptions of Software Engineering professionals. In: *Communications of the ACM* 32 (1989), Nr. 9, S. 1102–1108

[Stahl u. a. 2006] Stahl, Thomas ; Voelter, Markus ; Czarnecki, Krzysztof: *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006. – ISBN 0470025700

[Weismann 2006] Weismann, Benedikt: *Architekturzentrierte Modellgetriebene Softwareentwicklung - Fallbeispiel und Evaluierung*. Business Informatics Group der Technischen Universität Wien, TU Wien, Diplomarbeit, September 2006