# Model-Driven Software Migration: A Methodology

Christian Wagner

# Model-Driven Software Migration: A Methodology

## Reengineering, Recovery and Modernization of Legacy Systems

Foreword by Dr.-Ing. Hans-Georg Pagendarm

Springer Vieweg

Dr.-Ing. Christian Wagner
Potsdam, Germany

Also PhD thesis „Modellgetriebene Software-Migration“ at the University of Potsdam, Chair of Service and Software Engineering

ISBN 978-3-658-05269-0 ISBN 978-3-658-05270-6 (eBook)
DOI 10.1007/978-3-658-05270-6

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Library of Congress Control Number: 2014933421

Springer Vieweg


Printed on acid-free paper

Springer Vieweg is a brand of Springer DE.
Springer DE is part of Springer Science+Business Media.
www.springer-vieweg.de

# Foreword

Model-driven software development offers the method of choice when it comes to manage complex software production projects. However, these concepts face some obstacles when applied to maintenance of existing software systems. In order to ally such modern methods it is frequently assumed that re-coding cannot be circumvent.

Christian Wagner demonstrates on a real-life example how existing software may be imported into a modern software development suite via application of automatic processes. Thus with modest effort a legacy code turns into a maintainable and expandable code-base. While re-programming would create a risk of introducing new bugs, the automatic conversion of legacy code removes obstacles for further maintenance and development of the code and at the same time conserves the know-how and quality contained within a well-tested proven code. The automatic conversion turn out to be by far more efficient than re-programming. Efficiency coincides with improved reliability of the software implementation process.

The concept of model-driven-software-maintenance which is demonstrated here, is very convincing and therefore hopefully will be widely adopted in the near future. Latest when facing a task which requires the integration of a variety of codes, originating from various frameworks into one single software system, there is probably no way around the methods of model-driven-software-maintenance.

Dr.-Ing. Hans-Georg Pagendarm

# Acknowledgement

Primarily I want to thank my girlfriend Dunja and my parents for their support that I have experienced in all ups and downs during this work. Incidentally, I could still learn a lot about language.

From my colleagues, I would especially like to express my thanks to Henning Bordihn. Not only for his instructions and constructive criticism, but also for his sympathetic ear, which he has always had for me. Additionally Henning strongly aroused my interest for formal descriptions. I could learn from him a lot about teaching and about the structure and organization of lectures. This knowledge will remain with me all my life.

Special thanks to Sven Jörges because he is simply a friend – and of course, for the excellent proofreading work, the regular discussions on modeling theory (I have finally managed the way from the meta-level to a concrete instance of this work), current research literature and common sporting activities, whether on rocks or anywhere on a specific coordinate.

Georg Jung has a large share in planning and structuring of this thesis. He showed me how many pages two people can bring to paper in a long day. Moreover I would like to thank Julia Rehder for here excellent English vocabulary and proofreading skills.

In addition, I would like to thank Hans-Georg Pagendarm who has encouraged me in my job at the DNW and my thesis. Without him I would never have thought about the possibility.

Dr.-Ing. Christian Wagner

# Abstract

Software has become part of our everyday daily life. Since the early days of software development in the 50s an innumerable amount of software is developed. In many cases, such systems are still active today which was even not anticipated by the developers. The estimated 220 million lines of code, that are written in a language which was born in 1959, are an outstanding example.

Of course all of these software systems need to be maintained and adapted to new environments. The software evolution reflects the longest phase in the software life cycle. It begins with the delivery of the application to the customer and ends with the exchange of the old system. Modern development technologies can help to minimize the problems that inevitably arise in the context of software evolution. The aim of this research is to investigate the impact and applicability of model-driven techniques in the field of software evolution.

This includes the design of a process model, the development and application of tools and methods as well as the study of several concrete use cases. The improvement in the areas of application understanding, reengineering and migration of software are addressed. The focus is to support the synchronization between the program code and the related artifacts (usually models) which is naturally lost in classical, code-centric software maintenance. Therefore, the software development and maintenance must move towards a model-centered thinking. The synchronization is ensured by a code generation step based on the model level.

The resulting method supports this approach and consists of five phases: transformation of the source code into models, model analysis, abstraction by model transformation, splitting and migration of the existing system and code generation.

The first part – the transformation of the program code – includes the development and application of tools from the fields of compiler construction and program analysis. The aim is to convert the source code into a machine readable form. The result is a representation of a control flow graph (code-model), which is visualized graphically by means of a modeling tool. Model analyzes (second phase) improve the understanding of the application. These analyzes are based on the code-models and can be flexibly adapted to the specific project situation. This includes the creation of new ones as well as the integration of external tools. The transition to the model level occurs in the third phase: A model abstraction step is applied. Thereby information can be classified in the code-models and are abstracted into a new model. The abstraction works on the programming interfaces of the underlying libraries and is therefore called *API-based abstraction*.

The first three stages form the basis for the subsequent migration of the system (step four). The migration is the remodeling of the existing software as process model. The information obtained through the application understanding will guide this step. The developed process model is also partially associated with the functionality of the existing system. After completing this step fully executable source code is generated from the migrated models (step five).

# Contents


**1 Introduction** 1
- 1.1 Motivation: Why Software-Archeology? 1
- 1.2 Scientific Contribution 3
- 1.3 Outline of the Book 6

**2 Fundamentals** 9
- 2.1 Software Life Cycle 10
  - 2.1.1 Process Models: Classical 10
  - 2.1.2 Process Model: Continuous 13
  - 2.1.3 Laws in the Software Life Cycle 15
  - 2.1.4 Process Models: Research Approaches 17
- 2.2 Software Maintenance 19
  - 2.2.1 Operation/Maintenance Phase 19
  - 2.2.2 Definition of Software Maintenance 20
  - 2.2.3 Types of Maintenance 22
  - 2.2.4 Tasks of Software Maintenance 24
- 2.3 Legacy Systems 26
  - 2.3.1 Characteristics of Legacy Systems 26
  - 2.3.2 Criteria for the Evaluation of Legacy Systems 28
  - 2.3.3 Strategies for Coping with Legacy Systems 29
- 2.4 Software Reengineering 29
  - 2.4.1 Definitions of Software Reengineering 30
  - 2.4.2 Levels of Software Reengineering 35
  - 2.4.3 System and Application Understanding 37
  - 2.4.4 Excursus: Business Process Reengineering 39
- 2.5 Software Migration 39
  - 2.5.1 Definition of Software Migration 40
  - 2.5.2 Migration Methods 40
  - 2.5.3 Migration Strategies 42
- 2.6 The Concept of Modeling 43
  - 2.6.1 Software Development at an Abstract Level 45
  - 2.6.2 Model Definition 49

2.6.3 Modeling Languages . . . . . . . . . . . . . . . . . . . 51
2.6.4 Model Relationship . . . . . . . . . . . . . . . . . . . 55
2.6.5 Model Transformation . . . . . . . . . . . . . . . . . . 57
2.6.6 Models and Software Evolution . . . . . . . . . . . . . 59
2.7 Software Design Methodologies . . . . . . . . . . . . . . . . 60
2.7.1 Situational Method and Research Overview . . . . . . 61
2.7.2 Process Delivery Diagrams . . . . . . . . . . . . . . . 62
2.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

**3 Model-Driven Software Migration 67**
3.1 Problem Description . . . . . . . . . . . . . . . . . . . . . 67
3.2 The Model as Central Artifact . . . . . . . . . . . . . . . . 69
3.3 Continuous Software Development . . . . . . . . . . . . . . 72
3.3.1 Continuous Model-Driven Engineering . . . . . . . . . 73
3.3.2 eXtreme Model-Driven Design (XMDD) . . . . . . . . 76
3.4 Model-Driven Evolution . . . . . . . . . . . . . . . . . . . . 82
3.5 Model-Driven Reengineering . . . . . . . . . . . . . . . . . 84
3.5.1 Reverse-Engineering . . . . . . . . . . . . . . . . . . 85
3.5.2 Modeling . . . . . . . . . . . . . . . . . . . . . . . . 88
3.5.3 Model-Driven Analysis . . . . . . . . . . . . . . . . . 90
3.5.4 Code Generation . . . . . . . . . . . . . . . . . . . . 92
3.6 Model-Driven Migration . . . . . . . . . . . . . . . . . . . . 94
3.6.1 Model-Driven Abstraction . . . . . . . . . . . . . . . 96
3.6.2 Process Modeling . . . . . . . . . . . . . . . . . . . . 100
3.6.3 Migration and Separation of the System . . . . . . . . 101
3.6.4 Code Generation . . . . . . . . . . . . . . . . . . . . 103
3.7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . 105

**4 Related Work 107**
4.1 Analysis of Source Code . . . . . . . . . . . . . . . . . . . . 107
4.2 Description Languages . . . . . . . . . . . . . . . . . . . . . 116
4.3 Reverse and Reengineering Tools . . . . . . . . . . . . . . . 124
4.4 Modeling Tools . . . . . . . . . . . . . . . . . . . . . . . . . 129
4.5 Transformation Systems and Languages . . . . . . . . . . . 136
4.6 Projects in Software Migration . . . . . . . . . . . . . . . . 139
4.7 Analysis and Remodeling of GUIs . . . . . . . . . . . . . . . 142

**5 Case Study: DeAs 145**
5.1 Excursus: Operation of a Wind Tunnel . . . . . . . . . . . . 146
5.2 DeAs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 148
5.2.1 DeAs System Architecture . . . . . . . . . . . . . . . 148

5.2.2 DeAs Applications and Libraries . . . . . . . . . . . . . 151
5.2.3 Definition of the Measurement Program . . . . . . . . . 154
5.2.4 Selection of the System Components . . . . . . . . . . . 157
5.3 Top-Down Approach . . . . . . . . . . . . . . . . . . . . . . . 158
5.4 Model-driven Migration with DeAs . . . . . . . . . . . . . . . . 164
5.5 Model-Driven Reengineering . . . . . . . . . . . . . . . . . . . 165
5.5.1 Reverse Engineering . . . . . . . . . . . . . . . . . . . . 165
5.5.2 Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . 177
5.5.3 Model-Driven Analysis . . . . . . . . . . . . . . . . . . . 190
5.5.4 The Code-Model Generator . . . . . . . . . . . . . . . . 201
5.6 Model-Driven Migration . . . . . . . . . . . . . . . . . . . . . 215
5.6.1 API-based Abstraction . . . . . . . . . . . . . . . . . . . 216
5.6.2 Preparing to Migrate . . . . . . . . . . . . . . . . . . . . 224
5.6.3 Create Process Models and Migration of DeAs . . . . 228
5.6.4 Remodeling of the `DeAsManagers` . . . . . . . . . . . . 228
5.6.5 Remodeling of `An_an` . . . . . . . . . . . . . . . . . . . 233
5.6.6 Rules for Process Modeling and Migration . . . . . . . 236
5.7 Results of the Case Study DeAs . . . . . . . . . . . . . . . . . 239

**6 Further Applications 241**
6.1 `Textedit` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 241
6.2 Toolbar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 243
6.3 Drawing Program `xFig` . . . . . . . . . . . . . . . . . . . . . 244
6.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 247

**7 Conclusions 249**
7.1 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 249
7.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . 253
7.3 Result . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 257

**Bibliography 259**

**Appendix**

**A DTD of XML description 291**

**B Evaluation of the DeAs system 293**

**Index 297**

# List of Figures


2.1 Waterfall model by Royce . . . . . . . . . . . . . . . . . . . . . . 11
2.2 Spiral model by Boehm . . . . . . . . . . . . . . . . . . . . . . 12
2.3 Agile process model . . . . . . . . . . . . . . . . . . . . . . 14
2.4 Continuous Software Engineering . . . . . . . . . . . . . . . . . . 18
2.5 Maintenance curve from Seacord . . . . . . . . . . . . . . . . . . 20
2.6 Ratio of initial development to maintenance . . . . . . . . . . . 25
2.7 Conceptual model of software reengineering . . . . . . . . . . . 31
2.8 Horseshoe Model . . . . . . . . . . . . . . . . . . . . . . . . 37
2.9 The concepts of MDA . . . . . . . . . . . . . . . . . . . . . . 46
2.10 Modeling Spectrum . . . . . . . . . . . . . . . . . . . . . . . 48
2.11 Refinement of the system concept by Favre . . . . . . . . . . . 50
2.12 The semiotic triangle . . . . . . . . . . . . . . . . . . . . . . 52
2.13 Base relations in MDE . . . . . . . . . . . . . . . . . . . . . 56
2.14 Representation of an eXtended PDD . . . . . . . . . . . . . . . 64

3.1 Process of Software Reengineering . . . . . . . . . . . . . . . . 68
3.2 Model-driven software reengineering and migration . . . . . . 70
3.3 jABC – Coordination Layer (Steffen et al. [SMN+07]) . . . . 76
3.4 XPDD – Phases of the Model-Driven Software Evolution . . . 83
3.5 XPDD – Model-Driven Reengineering . . . . . . . . . . . . . 85
3.6 Tool chain in model-driven reengineering . . . . . . . . . . . . 85
3.7 Generalized tool chain in reverse engineering . . . . . . . . . 86
3.8 XPDD – Reverse engineering phase . . . . . . . . . . . . . . . 87
3.9 XPDD – Modeling phase . . . . . . . . . . . . . . . . . . . . 89
3.10 XPDD – Process of analysis . . . . . . . . . . . . . . . . . . . 91
3.11 XPDD – Code generation . . . . . . . . . . . . . . . . . . . . 93
3.12 XPDD – Overview of the Model-Driven Migration . . . . . . 95
3.13 XPDD – Model-driven abstraction . . . . . . . . . . . . . . . 99
3.14 XPDD – Code Generation . . . . . . . . . . . . . . . . . . . . 103

4.1 GXL Example by Winter et al. [WKR02] . . . . . . . . . . . . 117
4.2 Architecture of the ISME [AEK05] . . . . . . . . . . . . . . . . 121

4.3 jABC – User interface . . . . . . . . . . . . . . . . . . . . . . . 130
4.4 jABC – Modeling hierarchy . . . . . . . . . . . . . . . . . . . . 132

5.1 Transonic Wind Tunnel Göttingen (DNW-TWG) . . . . . . . 147
5.2 The monolithic core of the DeAs system . . . . . . . . . . . . 149
5.3 Properties of a wind tunnel model . . . . . . . . . . . . . . . . 155
5.4 Instrumentation of measurement values . . . . . . . . . . . . 156
5.5 Configuration of a measurement mroject . . . . . . . . . . . . 157
5.6 DeAsManager – user interface . . . . . . . . . . . . . . . . . . . 159
5.7 DeAsManager – process model (start of application) . . . . . 160
5.8 DeAsManager – process model (start of DeAs applications) . 162
5.9 Modeling – definition of a wind tunnel . . . . . . . . . . . . . 163
5.10 Overview of the TF_func object (created with Doxygen) . . 169
5.11 Overview of the For loop (created with Doxygen) . . . . . . 170
5.12 jABC – SIBs representing elements of the C/C++ grammar . 178
5.13 Java Application Building Center (jABC) – SIB attributes . . 179
5.14 jABC – Excerpt prepared a code-model . . . . . . . . . . . . 180
5.15 Linking of Code-Models . . . . . . . . . . . . . . . . . . . . . . 182
5.16 jABC – Structure of the developed jABC plug-in . . . . . . . 183
5.17 jABC – Configuration of the plug-in . . . . . . . . . . . . . . 184
5.18 jABC – Database view of the plug-in . . . . . . . . . . . . . . 185
5.19 Number of methods per XML description . . . . . . . . . . . 187
5.20 Number of nodes per method . . . . . . . . . . . . . . . . . . . 188
5.21 Process graphs for the calculation of LOC metric . . . . . . . 192
5.22 Coarse-grained process graph of McCabe calculation . . . . . 193
5.23 Fine-grained process graph of McCabe calculation . . . . . . 194
5.24 Results of McCabe metric . . . . . . . . . . . . . . . . . . . . . 195
5.25 Process model for the detection of dead code . . . . . . . . . 196
5.26 Process graphs for the determination of function calls . . . . 197
5.27 An_an – Overview of function calls . . . . . . . . . . . . . . . 198
5.28 Result of the BundleView software for DeAs libraries . . . . . 199
5.29 Dependencies at different levels using Graphviz . . . . . . . . 200
5.30 Files that are related to the main function (excerpt) . . . . . 201
5.31 Call hierarchy of the main function . . . . . . . . . . . . . . . 202
5.32 Initialization of the code generator . . . . . . . . . . . . . . . 203
5.33 Main model of the code generator . . . . . . . . . . . . . . . . 204
5.34 Code generator: transformation into a block structure . . . . 205
5.35 While loop at the end of a function . . . . . . . . . . . . . . . 208
5.36 Variations of a reunited control flow . . . . . . . . . . . . . . 209
5.37 Different loop types . . . . . . . . . . . . . . . . . . . . . . . . . 210
5.38 case-statement with five branches . . . . . . . . . . . . . . . . 211

5.39 Structure of the back-to-back testing . . . . . . . . . . . . . . . 214
5.40 Process model of a filter . . . . . . . . . . . . . . . . . . . . . . 220
5.41 XView element tree starting from xv_init . . . . . . . . . . 222
5.42 Main window . . . . . . . . . . . . . . . . . . . . . . . . . . . 223
5.43 Editing window . . . . . . . . . . . . . . . . . . . . . . . . . . 223
5.44 An_an – Structure of the graphical interface . . . . . . . . . 224
5.45 An_an – Linking of code-models with process sequences . . . 225
5.46 Ap_au – Structure of the graphical interface . . . . . . . . . 226
5.47 CORBA – Construction of the communication layer . . . . . 227
5.48 CORBA – C++ Server and Java Client . . . . . . . . . . . . 227
5.49 DeAsManager – Structure . . . . . . . . . . . . . . . . . . . 229
5.50 DeAsManager – Menu with code-models and system call . . 230
5.51 DeAsManager – main function with Benutzer object . . . 231
5.52 DeAsManager – migrated process model . . . . . . . . . . . 232
5.53 DeAsManager – GUI of ChooseApplication-SIB . . . . . . . . 233
5.54 An_an – migrated process model . . . . . . . . . . . . . . . . 234
5.55 An_an – GUI of ChooseActionLoadWindTunnels-SIB . . . . . 235
5.56 An_an – edit tunnel user interface . . . . . . . . . . . . . . . 235

6.1 Textedit – User interface . . . . . . . . . . . . . . . . . . . 242
6.2 Textedit – Structure of the graphical interface . . . . . . . 243
6.3 Toolbar – User interface . . . . . . . . . . . . . . . . . . . . 243
6.4 xFig – User interface . . . . . . . . . . . . . . . . . . . . . . 244
6.5 Number of methods per XML description . . . . . . . . . . . 245
6.6 Number of nodes per function . . . . . . . . . . . . . . . . . . 245
6.7 Result of McCabe metrics . . . . . . . . . . . . . . . . . . . . 246

# List of Tables


2.1 Percentage of maintenance in the software life cycle . . . . . . 16
2.2 Proportion of the types of maintenance . . . . . . . . . . . . . 24

4.1 Summary of evaluated tools [WMP09] . . . . . . . . . . . . . . 115

5.1 Evaluation criteria of the DeAs legacy system . . . . . . . . . . 151
5.2 DeAs – configuration of wind tunnel and test object . . . . . 151
5.3 DeAs – configuration of the measurement technique . . . . . 152
5.4 DeAs – define and execute a test project . . . . . . . . . . . . 153
5.5 DeAs – system libraries . . . . . . . . . . . . . . . . . . . . . . 154
5.6 Conversion of DeAs libraries into code-models . . . . . . . . . 189
5.7 Conversion of DeAs applications into code-models . . . . . . 190

B.1 DeAs – Evaluation of preprocessor . . . . . . . . . . . . . . . . 293
B.2 DeAs – Evaluation of Elsa-Parsers . . . . . . . . . . . . . . . . 294
B.3 DeAs – Evaluation of jABC-Import . . . . . . . . . . . . . . . . 295
B.4 DeAs – Evaluation of the relation of the code-models . . . . . 296

# List of Source Code Examples


4.1 GraphML-Example of a structure description . . . . . . . . . 118
4.2 GraphXML-Example . . . . . . . . . . . . . . . . . . . . . . . . 119
4.3 JavaML-Example from [Bad00, p. 6] . . . . . . . . . . . . . . . 119

5.1 `main`-Function of DeAs application . . . . . . . . . . . . . . . 160
5.2 DeAs application with the replaced `main`-function . . . . . . 161
5.3 Elsa – Extract from the AnST for statement b = 3; . . . . . . 167
5.4 Elsa – XML output . . . . . . . . . . . . . . . . . . . . . . . . . 171
5.5 `ccparse` – Program output . . . . . . . . . . . . . . . . . . . . 171
5.6 Extract from the annotated intermediate format . . . . . . . 172
5.7 Pseudo-code – Extract a source line from code . . . . . . . . 174
5.8 Node description and associated attributes . . . . . . . . . . . 175
5.9 XML edges with source and taget . . . . . . . . . . . . . . . . 176
5.10 Sloccount == 6 . . . . . . . . . . . . . . . . . . . . . . . . . . 191
5.11 Sloccount == 8 . . . . . . . . . . . . . . . . . . . . . . . . . . 191
5.12 Pseudo-code – Calculation of the closing brackets (part 1) . . 206
5.13 Pseudo-code – Calculation of the closing brackets (part 2) . . 207
5.14 Pseudo-code – Calculation of the closing brackets (part 3) . . 208
5.15 Pseudo-code – calculation of closing parenthesis (part 4) . . . 209
5.16 Original code . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210
5.17 Generated Code . . . . . . . . . . . . . . . . . . . . . . . . . . 210
5.18 Creation of `Panel`-Elements with various parameters . . . . 217
5.19 Initialization and start of the graphical interface . . . . . . . 218
5.20 Generating of a button with associated callback function . . . 218
5.21 Pseudo-code for finding callback functions . . . . . . . . . . . 219
5.22 IDL – Description of the class `Benutzer` (excerpt) . . . . . . 228

A.1 DTD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 291

# List of Abbreviations

| | |
|---|---|
| **ABC** | Application Building Center |
| **ADM** | Architecture Driven Modernization |
| **AnST** | Annotated Syntax Tree |
| **API** | Application Programming Interface |
| **ASG** | Abstract Semantic Graph |
| **AST** | Abstract Syntax Tree |
| **ASTM** | Abstract Syntax Tree Meta-Model |
| **BPR** | Business Process Reengineering |
| **BMBF** | Bundesministerium für Bildung und Forschung |
| **BNF** | Backus Naur Form |
| **CEP** | Complex Event Processing |
| **CFD** | Computational Fluid Dynamics |
| **CFG** | Control Flow Graph |
| **CIM** | Computation Independent Model |
| **CMDE** | Continuous Model-Driven Engineering |
| **CMM** | Capture Maturity Model |
| **CORBA** | Common Object Request Broker Architecture |
| **CSE** | Continuous Software Engineering |
| **CSV** | Comma-Separated Values |
| **DAA** | Data Access Allocator |
| **DeAs** | Datenerfassungs- und Anlagensteuerungssoftware |
| **DLR** | Deutsches Zentrum für Luft- und Raumfahrt |
| **DNW** | German-Dutch Wind Tunnels |
| **DNW-TWG** | Transonic Wind Tunnel Göttingen |
| **DTD** | Document Type Definition |
| **EBNF** | Extended Backus Naur Form |
| **EMF** | Eclipse Modeling Framework |
| **EPT** | Elsa Parse Tree |
| **EU** | European Union |

| | |
|---|---|
| **GCC** | GNU Compiler Collection |
| **GMF** | Graphical Modeling Framework |
| **GNU** | GNU is not UNIX |
| **GUI** | Graphical User Interface |
| **GXL** | Graph eXchange Language |
| **HPI** | Hasso Plattner Institut |
| **HTML** | Hypertext Markup Language |
| **IDL** | Interface Description Language |
| **IML** | Intermediate Language |
| **jABC** | Java Application Building Center |
| **JNI** | Java Native Interface |
| **KDM** | Knowledge Discovery Meta-Model |
| **KTS** | Kripke Transitionssystem |
| **LIF** | Lanuage Independent Format |
| **LOC** | Lines of Code |
| **LPC** | Lightweight Process Coordination |
| **MDA** | Model-Driven Architecture |
| **MDD** | Model-Driven Design |
| **MDE** | Model-Driven Engineering |
| **MDRE** | Model-Driven Reverse Engineering |
| **MOF** | Meta Object Facility |
| **NLR** | Nationaal Lucht- en Ruimtevaartlaboratorium |
| **OCL** | Object Constraint Language |
| **OMG** | Object Management Group |
| **ORB** | Object Request Broker |
| **OTA** | One Thing Approach |
| **PDD** | Process Deliverable Diagram |
| **PIM** | Platform Independent Model |
| **PSM** | Platform Specific Model |
| **PUB** | Platform Independent Description |
| **QVT** | Query View Transformation |
| **RCL** | Rigi Command Language |
| **RFG** | Resource Flow Graph |
| **RSF** | Rigi Standard Format |
| **RTE** | Round Trip Engineering |
| **RUP** | Rational Unified Process |

**SDM** Software Design Methodology
**SIB** Service Independent Building Block
**SLG** Service Logic Graph
**SOMA** Service-Oriented Modeling Architecture
**PLC** Programmable Logic Controller
**SUS** System under Study
**SVG** Scalable Vector Graphics
**TA** Tuple Attribute
**TU** Technische Universität
**TXL** Turing eXtended Language
**UML** Unified Modeling Language
**VCG** Visualization of Compiler Graph
**XMDD** eXtreme Model-Driven Design
**XMI** XML Metadata Interchange
**XML** Extensible Markup Language
**XPDD** eXtended Process Deliverable Diagram
**XSL** Extensible Stylesheet Language
**XSLT** Extensible Stylesheet Language Transformation