Requirements Engineering for Model Transformation Development

Yassipour Tehrani, Sobhan

Awarding institution: King's College London

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without proper acknowledgement.

END USER LICENCE AGREEMENT

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International licence. https://creativecommons.org/licenses/by-nc-nd/4.0/

You are free to:
• Share: to copy, distribute and transmit the work

Under the following conditions:
• Attribution: You must attribute the work in the manner specified by the author (but not in any way that suggests that they endorse you or your use of the work).
• Non Commercial: You may not use this work for commercial purposes.
• No Derivative Works - You may not alter, transform, or build upon this work.

Any of these conditions can be waived if you receive permission from the author. Your fair dealings and other rights are in no way affected by the above.

Take down policy

If you believe that this document breaches copyright please contact librarypure@kcl.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Requirements Engineering for Model Transformation Development

by

Sobhan Yassipour Tehrani

Submitted in partial fulfilment of the requirements for the Degree of Doctor of Philosophy in Computer Science

Department of Informatics
School of Natural and Mathematical Sciences
King’s College London

March 2018
Acknowledgements

I would like to thank everyone who provided me with support, guidance and encouragement throughout my PhD career. In particular, I would like to express my sincere gratitude and thanks to my supervisor, Dr Kevin Lano, for his help, time, advice, guidance and indispensable support for the duration of the course of this PhD in an encouraging manner. I cannot imagine to have been able to fulfil this PhD without his help and support. I would also like to thank my second supervisor, Dr Steffen Zschaler, for his helpful ideas and motivating suggestions.

Besides my supervisors, I would like to thank all the faculty, staff and students in the Department of Informatics at King’s College London for their assistance.

I thank my fellow PhD colleagues, Dr Vahid Towhidloo and Dr Ali Hosseini for their illuminating insights and support especially during the deadline periods, and for all the fun we have had in the last few years.

Last but not least, my sincere thanks goes to my dear parents and Miss Haydeh Taheri for their continued support, both practically and spiritually, not only during the writing of this thesis but in general for all they have done, enabling me to acquire a higher education at one of the most distinguished institutions of its kind and partaking in a PhD program. I hope that my advancements and achievements will be a reflection of my gratitude for all their efforts and sacrifices to this end.
Abstract

Model transformation (MT) is central to model driven engineering. It can be used for a range of purposes, including to improve the quality of models, to refactor models, to migrate or translate models from one representation to another, and to generate code or other artifacts from models. At present, the development of MT is mainly focused on the specification and implementation phases, whereas there is a lack of support in other phases including requirements, analysis, design and testing. In this thesis, we are only interested in the requirements phase of MT development, namely the initial phase of software development life-cycle where the software’s specifications are determined, for which at present there is no systematic requirement engineering (RE) process.

In this research study, we aim to systematically find out how MT is being developed. We are particularly interested in understanding how requirements for MT are being identified. A comprehensive systematic literature review together with an interview-based study have been applied in order to address these shortcomings. Moreover, this thesis addresses the lack of a guideline for a systematic RE process in MT by defining a systematic procedural RE process framework for MT development and it identifies criteria for selecting the most appropriate RE techniques.

This framework is evaluated and validated through its application on two substantial industrial cases. The first case is an example of model driven development applied to MT development. The second is a financial application involving risk evaluation of multiple financial investments.
Publications

Throughout this PhD career, more than 20 publications have been accepted and published of which some are directly related to this thesis and some are partially related.

Directly related publications:

  
  **Yassipour Tehrani, S.**, Lano, K., Zschaler, S.

  This publication is based on Chapter 3 and Chapter 4 of the thesis.


  **Yassipour Tehrani, S.**, Zschaler, S. & Lano, K.

  This publication is based on Chapter 3 of the thesis.

- Transformation from UML to C: A large-scale example of MDD for model transformation development. In *International Journal on Software and Systems Modeling 2017*

  Lano, K., **Yassipour Tehrani, S.** Alfraihi, H. A. A. & Kolahdouz-Rahimi, S

  This publication is based on Chapter 6 of the thesis.
• Model Transformation Applications from Requirements Engineering Perspective. In the 10th International Conference on Software Engineering Advances (Awarded Best Paper)
  Yassipour Tehrani, S. & Lano, K.
  This publication is based on Chapter 5 of the thesis.

  Yassipour Tehrani, S. & Lano, K.
  This publication is based on Chapter 5 and Chapter 6 of the thesis.

  Yassipour Tehrani, S. & Lano, K.
  This publication is based on Chapter 5 of the thesis.

• Improving the Application of Agile Model-based Development: Experiences from Case Studies. In the 10th International Conference on Software Engineering Advances.
  Lano, K., Yassipour Tehrani, S. & Alfraihi, H. A. A.
  This publication is based on Chapter 6 of the thesis.

• Translating OCL to ANSI C. In the 17th International Workshop in OCL and Textual Modeling.
  Lano, K., Yassipour Tehrani, S. & Kolahdouz-Rahimi, S
  This publication is based on Chapter 6 of the thesis.

• Precise Requirements Engineering for Model Transformations. In STAF 2014 Doctoral Symposium.
  Yassipour Tehrani, S. & Lano, K.
  This publication is based on Chapter 5 of the thesis.

• The significant role of Requirement Engineering in Model Transformation. In International Conference on New Trends in Information
and Communication Technologies.

**Yassipour Tehrani, S.** & Lano, K.
This publication is based on Chapter 5 of the thesis.

  Lano, K., Haughton, H., **Yassipour Tehrani, S.** & Alfraihi, H. A.
  This publication is based on Chapter 6 of the thesis.

**Partially related publications:**

  Lano, K., **Yassipour Tehrani, S.** & Kolahdouz-Rahimi, S.

  Lano, K., Kolahdouz-Rahimi S. & **Yassipour Tehrani, S.**

  Lano, K. **Yassipour Tehrani, S.**

- Solving the Class Responsibility Assignment Case with UML-RSDS. In *Proceeding of the the 9th Transformation Tool Contest, co-located with the 2016 Software Technologies: Applications and Foundations (STAF 2016).*
  Lano, K. **Yassipour Tehrani, S.** & Kolahdouz-Rahimi

- Patterns for Specifying Bidirectional Transformations in UML-RSDS. In *the 10th International Conference on Software Engineering Advances.*
  Lano, K., Alfraihi, H., **Yassipour Tehrani, S.** & Haughton, H.
• Experiences of Teaching Model-based Development. In *ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*.
Lano, K., *Yassipour Tehrani, S.* & Alfraihi, H. A. A.

Lano, K. & *Yassipour Tehrani, S.*

• Design Patterns for Model Transformations: Current Research and Future Directions. In *International Workshop on Patterns in Model Engineering 2015*.
Lano, K. & *Yassipour Tehrani, S.*

• Mapping FIXML to OO with Aspectual Code Generators. In *Proceedings of the 7th Transformation Tool Contest part of the Software Technologies: Applications and Foundations*.
Zschaler, S. & *Yassipour Tehrani, S.*

• Solving the TTC 2014 Movie Database Case with UML-RSDS. In *Proceedings of the 7th Transformation Tool Contest part of the Software Technologies: Applications and Foundations*.
Lano, K. & *Yassipour Tehrani, S.*

• Aspectual Code Generators for Easy Generation of FIXML to OO Mappings. In *TTC 2014 FIXML Case Solution*.
Zschaler, S. & *Yassipour Tehrani, S.*

• Case study: FIXML to Java, C# and C++. In *TTC 2014 FIXML Case Solution*.
Lano, K., *Yassipour Tehrani, S.* & Maroukian, K.
# Contents

## 1 Introduction

1.1 Overview ................................................. 2  
1.2 Motivation ............................................... 2  
1.3 Research Objectives ..................................... 3  
1.4 Overall Aims and Contributions ....................... 5  
1.5 Overall Thesis Structure ............................... 6  

## 2 Background on Software & Requirements Engineering and Model Transformation  

2.1 The Software Development Process ....................... 9  
  2.1.1 Software Requirements ............................ 9  
  2.1.2 Software Project Types ............................ 14  
2.2 Software Process Model .................................. 16  
2.3 Software Measurements and Metrics ................. 20  
  2.3.1 Software Quality Models ......................... 22  
  2.3.2 Goal-Questions-Metrics .......................... 26  
2.4 Requirements Engineering ............................... 27  
  2.4.1 Domain Analysis and Requirements Elicitation .. 33  
  2.4.2 Evaluation and Negotiation ..................... 39  
  2.4.3 Specification and Documentation ................. 42  
  2.4.4 Validation and Verification ..................... 46  
2.5 Model Driven Engineering ............................... 49  
  2.5.1 Model Driven Development in MDE ............... 51  
  2.5.2 Model Driven Architecture in MDE ............... 52
Contents

4.4 Comparison ............................................. 115
4.5 Threats to Validity ...................................... 118
4.6 Summary .................................................. 118

5 Requirements Engineering Activity for MT ............................................. 121
  5.1 Application of RE in MT .................................. 121
    5.1.1 Requirements Taxonomy ................................. 124
  5.2 RE Process Adaptation for MT ............................... 130
    5.2.1 Domain Analysis and Requirements Elicitation ....... 130
    5.2.2 Evaluation and Negotiation ............................. 132
    5.2.3 Specification and Documentation ....................... 134
    5.2.4 Validation and Verification ............................. 137
    5.2.5 Tool Support for RE in MT ............................ 140
  5.3 A Framework for Selecting Suitable RE Techniques ........ 141
    5.3.1 Technique Attribute .................................. 143
    5.3.2 Project Attribute ..................................... 150
    5.3.3 Organisational Attribute .............................. 156
  5.4 Application Framework Example ............................. 157
  5.5 Framework Implementation ................................ 167
  5.6 Summary .................................................. 169

6 Evaluation .................................................. 171
  6.1 Case study 1: UML to C Transformation .................... 171
    6.1.1 F1.1: Translation of Types ............................ 185
    6.1.2 F1.2: Translation of Class Diagrams ................... 185
    6.1.3 F1.3: Translation of OCL Expressions .................. 186
    6.1.4 Translation of Activities .............................. 190
    6.1.5 Translation of Use Cases .............................. 193
    6.1.6 Evaluation ............................................ 195
  6.2 Case Study 2: CDO Risk Estimation ......................... 201
    6.2.1 Evaluation ............................................ 214
  6.3 Framework Evaluation .................................... 216
  6.4 Summary .................................................. 217
## Contents

7  **Summary and Concluding Remarks**  221

7.1  **Introduction**  221

7.2  **Objectives of Research**  222

7.3  **Overview of Thesis**  222

7.4  **Limitations**  224

7.5  **Future Work**  225

7.5.1  Requirements Management in MT  225

7.5.2  Applying the Framework to Several Cases  226

7.5.3  Integration with transML  227

7.5.4  Further Extension of the Framework  227

7.6  **Concluding Remarks**  228

A  **SLR Tables**  246

B  **Interview Guide**  259

C  **Description of RE Techniques**  262

D  **Surveyed Papers**  266
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A taxonomy of non-functional requirements</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Spiral model of the software process</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Comparison table using impact estimation</td>
<td>44</td>
</tr>
<tr>
<td>2.4</td>
<td>McCall Quality Model</td>
<td>106</td>
</tr>
<tr>
<td>2.5</td>
<td>Boehm Quality Model</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>GQM model</td>
<td>101</td>
</tr>
<tr>
<td>2.7</td>
<td>Cost of late correction (Boehm)</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>Three dimensions of requirements engineering</td>
<td>143</td>
</tr>
<tr>
<td>2.9</td>
<td>Classification of statements in RE</td>
<td>97</td>
</tr>
<tr>
<td>2.10</td>
<td>Stakeholder onion model</td>
<td>98</td>
</tr>
<tr>
<td>2.11</td>
<td>NFR types</td>
<td>26</td>
</tr>
<tr>
<td>2.12</td>
<td>Requirements inspection and review process</td>
<td>48</td>
</tr>
<tr>
<td>2.13</td>
<td>Model-driven frameworks, adapted from</td>
<td>51</td>
</tr>
<tr>
<td>2.14</td>
<td>Transformation between different representations of a model</td>
<td>53</td>
</tr>
<tr>
<td>2.15</td>
<td>The general architecture of model transformation</td>
<td>55</td>
</tr>
<tr>
<td>2.16</td>
<td>General process of UML-RSDS</td>
<td>62</td>
</tr>
<tr>
<td>2.17</td>
<td>Overview of the ATL transformational approach</td>
<td>63</td>
</tr>
<tr>
<td>2.18</td>
<td>QVT architecture</td>
<td>65</td>
</tr>
<tr>
<td>2.19</td>
<td>Class diagram metamodel</td>
<td>75</td>
</tr>
<tr>
<td>2.20</td>
<td>Rule 1</td>
<td>75</td>
</tr>
<tr>
<td>2.21</td>
<td>Rule 2</td>
<td>75</td>
</tr>
<tr>
<td>2.22</td>
<td>Rule 3</td>
<td>75</td>
</tr>
<tr>
<td>2.23</td>
<td>Removing a many-to-many association</td>
<td>74</td>
</tr>
<tr>
<td>2.24</td>
<td>Replacing an inheritance by an association</td>
<td>74</td>
</tr>
</tbody>
</table>
List of Figures

2.25 Introducing a superclass . . . . . . . . . . . . . . . . . . 74

3.1 Onion model of stakeholder general relationships . . . . . 86
3.2 Adapted onion model of MT stakeholder relationships . . 88

4.1 Surveyed cases per year . . . . . . . . . . . . . . . . . . . 109
4.2 SLR MT stakeholders . . . . . . . . . . . . . . . . . . . . . 111
4.3 Transformation types . . . . . . . . . . . . . . . . . . . . . 112
4.4 RE technique used in MT cases . . . . . . . . . . . . . . . 113
4.5 SLR case RE rigour (x-axis) versus outcomes (y-axis) . . 114
4.6 MT projects scale . . . . . . . . . . . . . . . . . . . . . . . 116
4.7 Interview case RE rigour (x-axis) versus outcomes (y-axis) 117

5.1 A taxonomy of functional requirements . . . . . . . . . . . 125
5.2 A taxonomy of non-functional requirements for MT . . . . . 125
5.3 Functional requirements decomposition . . . . . . . . . . . 139
5.4 RE technique framework metamodel in UML-RSDS . . . . . 168

6.1 Functional requirements decomposition in SysML . . . . . . 181
6.2 C code generator architecture . . . . . . . . . . . . . . . . 184
6.3 UML-RSDS class diagram metamodel . . . . . . . . . . . . 188
6.4 CDO version 1 system specifications . . . . . . . . . . . . 209
6.5 CDO version 2 system specifications . . . . . . . . . . . . 209
6.6 CDO example . . . . . . . . . . . . . . . . . . . . . . . . . 212
6.7 CDO version 3 system specifications . . . . . . . . . . . . 214

7.1 Requirements management process . . . . . . . . . . . . . 226
# List of Tables

2.1 Project failure analysis (Standish Group (1995)) ................................ 11
2.2 Project success analysis (Standish Group (1995)) .......................... 11
2.3 Project success analysis (Standish Group (2015)) .......................... 12
2.4 Definition of non-functional requirements ..................................... 13
2.5 ISO/IEC 9126-1 quality characteristics [40] .................................. 23
2.6 General properties of transformations ........................................... 56
2.7 Comparison of transformation languages ...................................... 65
2.8 Model elements in UML [6] .......................................................... 71

3.1 Types of MT project ....................................................................... 85
3.2 Stakeholders of model transformation projects ................................. 87
3.3 Requirements engineering techniques in MT projects ................... 91
3.4 RE revision activity in MT projects .............................................. 92
3.5 RE techniques in MT projects ....................................................... 95
3.6 Outcomes of MT projects ............................................................ 99

4.1 Number of reviews ....................................................................... 110
4.2 Comparison of results ................................................................... 116

5.1 Transformation requirements catalogue ........................................ 128
5.2 Standard quality framework (ISO 9126) ......................................... 129
5.3 Requirements priority for different types of transformation ............ 134
5.4 RE technique attributes and classifications adapted and extended from [62] ............................................................... 145
## List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>Domain Analysis &amp; Requirements Elicitation technique attributes evaluation $V(a_x, t)$</td>
<td>146</td>
</tr>
<tr>
<td>5.6</td>
<td>Requirements Evaluation &amp; Negotiation technique attributes evaluation $V(a_x, t)$</td>
<td>147</td>
</tr>
<tr>
<td>5.7</td>
<td>Requirements Specification &amp; Documentation technique attributes evaluation $V(a_x, t)$</td>
<td>147</td>
</tr>
<tr>
<td>5.8</td>
<td>Requirements Validation &amp; Verification technique attributes evaluation $V(a_x, t)$</td>
<td>148</td>
</tr>
<tr>
<td>5.9</td>
<td>Project attributes weighting</td>
<td>154</td>
</tr>
<tr>
<td>5.10</td>
<td>Technique weight descriptor values</td>
<td>156</td>
</tr>
<tr>
<td>5.11</td>
<td>Attributes calculation of RE techniques</td>
<td>164</td>
</tr>
<tr>
<td>6.1</td>
<td>Domain Analysis &amp; Requirements Elicitation technique attributes evaluation</td>
<td>173</td>
</tr>
<tr>
<td>6.2</td>
<td>Domain Analysis &amp; Requirements Elicitation technique attributes evaluation for UML to C case</td>
<td>176</td>
</tr>
<tr>
<td>6.3</td>
<td>Technique attributes evaluation of the Evaluation &amp; Negotiation stage $V(a_x, t)$ for UML to C case</td>
<td>183</td>
</tr>
<tr>
<td>6.4</td>
<td>Technique attributes evaluation of the Specification &amp; Documentation stage $V(a_x, t)$ for UML to C case</td>
<td>183</td>
</tr>
<tr>
<td>6.5</td>
<td>Technique attributes evaluation of the Validation &amp; Verification stage $V(a_x, t)$ for UML to C case</td>
<td>184</td>
</tr>
<tr>
<td>6.6</td>
<td>Informal scenarios for types2C</td>
<td>186</td>
</tr>
<tr>
<td>6.7</td>
<td>Informal scenarios for the mapping of UML class diagrams to C</td>
<td>187</td>
</tr>
<tr>
<td>6.8</td>
<td>Mapping scenarios for Basic Expressions</td>
<td>189</td>
</tr>
<tr>
<td>6.9</td>
<td>Mapping scenarios for Logical Expressions</td>
<td>190</td>
</tr>
<tr>
<td>6.10</td>
<td>Mapping scenarios for Comparator Expressions</td>
<td>191</td>
</tr>
<tr>
<td>6.11</td>
<td>Mapping scenarios for Numeric Expressions</td>
<td>192</td>
</tr>
<tr>
<td>6.12</td>
<td>Scenarios for the mapping of Selection and Collection Expressions</td>
<td>192</td>
</tr>
<tr>
<td>6.13</td>
<td>Scenarios for the translation of Collection Operators (1)</td>
<td>193</td>
</tr>
<tr>
<td>6.14</td>
<td>Scenarios for the translation of Collection Operators (2)</td>
<td>194</td>
</tr>
<tr>
<td>Table</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>6.15</td>
<td>Scenarios for mapping of UML Activities to C Statements</td>
<td>194</td>
</tr>
<tr>
<td>6.16</td>
<td>Achievement of requirements</td>
<td>196</td>
</tr>
<tr>
<td>6.17</td>
<td>Overall development effort for C code generator</td>
<td>197</td>
</tr>
<tr>
<td>6.18</td>
<td>Generated C code versus Java code</td>
<td>197</td>
</tr>
<tr>
<td>6.19</td>
<td>Development effort for code generators (person months)</td>
<td>198</td>
</tr>
<tr>
<td>6.20</td>
<td>Software quality measures of C++ and C code generators</td>
<td>199</td>
</tr>
<tr>
<td>6.21</td>
<td>Software quality comparison</td>
<td>200</td>
</tr>
<tr>
<td>6.22</td>
<td>Technique attributes of the Domain Analysis &amp; Requirements Elicitation stage $v(a_x,t)$ for CDO case</td>
<td>203</td>
</tr>
<tr>
<td>6.23</td>
<td>Use cases for CDO risk analysis application</td>
<td>208</td>
</tr>
<tr>
<td>6.24</td>
<td>Execution times for CDO versions</td>
<td>214</td>
</tr>
<tr>
<td>6.25</td>
<td>CDO project comparison</td>
<td>215</td>
</tr>
<tr>
<td>6.26</td>
<td>Framework evaluation form</td>
<td>216</td>
</tr>
<tr>
<td>A.1</td>
<td>Stakeholder information (1)</td>
<td>247</td>
</tr>
<tr>
<td>A.2</td>
<td>Stakeholder information (2)</td>
<td>248</td>
</tr>
<tr>
<td>A.3</td>
<td>Stakeholder information (3)</td>
<td>249</td>
</tr>
<tr>
<td>A.4</td>
<td>MT requirements</td>
<td>250</td>
</tr>
<tr>
<td>A.5</td>
<td>MT project information (1)</td>
<td>251</td>
</tr>
<tr>
<td>A.6</td>
<td>MT project information (2)</td>
<td>252</td>
</tr>
<tr>
<td>A.7</td>
<td>Methodology information (1)</td>
<td>253</td>
</tr>
<tr>
<td>A.8</td>
<td>Methodology information (2)</td>
<td>254</td>
</tr>
<tr>
<td>A.9</td>
<td>Methodology information (3)</td>
<td>255</td>
</tr>
<tr>
<td>A.10</td>
<td>Methodology information (4)</td>
<td>256</td>
</tr>
<tr>
<td>A.11</td>
<td>SLR case outcomes (1)</td>
<td>257</td>
</tr>
<tr>
<td>A.12</td>
<td>SLR case outcomes (2)</td>
<td>258</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The increasing complexity and size of today’s software systems has resulted in raising the complexity and size of model transformations. Model Transformations are automated methods of modifying and creating models and are the central building blocks of Model Driven Engineering (MDE). Transformations are used widely in model-driven engineering and model-based development (MBD). Their uses include migration of models from one language to another, refactoring of models to improve quality, refinement of models from a specification to a design, or from design to implementation, code generation to generate program code from models, and bidirectional transformations to synchronise two different models and to maintain their consistency [29].

Although there have been different transformation tools and languages, most of them are focused on the specification and implementation phases. According to [40], most of the transformation languages proposed by model driven engineering (MDE), a software development methodology, are only focused towards the implementation phase and are not integrated in a unified engineering process. It could be said that, at the moment, the transformation process is performed in an ad-hoc manner; defining the problem and then directly beginning the implementation process.
1.1 Overview

At present, there is a need in model transformation to provide for the whole life-cycle of transformation development in a supportive way rather than the current practice of focusing mainly on the implementation phase. In the current practice whereby the main focus is on the implementation phase, it not only makes it difficult to design large scale transformations but their understandability and maintenance are also adversely affected in a similar manner. In the transformation development life-cycle, there are other phases including requirements, analysis, design and testing which need to be addressed as vigorously, if not more [46].

So far, little attention has been paid to the requirements engineering of model transformations. Requirements engineering is the process of identifying, analysing, documenting and validating the requirements of an application. It could be said that at present in many companies, the RE process is performed in a rule of thumb manner [129], meaning that the goal of defining the rules and requirements of a system is mainly to find an approximate solution(s) in the fastest possible procedure, whereas RE must be applied accurately and its outcome must be precise and reliable [27].

1.2 Motivation

The motivation behind this research is to introduce a specific RE process for model transformations development, since the model transformation field has not yet been considered from a systematic requirements engineering point of view. In this research project, I propose to investigate the most appropriate requirements engineering process for model transformation development. As previously mentioned, there is a lack of systematic engineering support in the model transformation field. In order to achieve any given goal using software, having a scheme in which its requirements have been identified is essential. However, getting the right requirements under the right assumptive environment is a necessary
precondition and often a quite challenging task for developing the right software [142]. It is a challenging task as there are a number of inherent difficulties in this process. The number of stakeholders may be numerous and they may be distributed, their goals may differ and conflict in some cases depending on their needs and perspectives, and their goals may not be defined explicitly which would lower the satisfaction level of these goals as they may be constrained by a variety of factors [113].

One of the appropriate techniques that will be used throughout this project is requirements elicitation, one of the most important stages in developing a software application. Requirements elicitation is essential for identifying requirements to achieve given goals in software. Having an appropriate understanding of the actual problem is equal to half of the solution. Therefore, in order to have a successful solution for a project, which meets its requirements, one needs to understand the different aspects of the project to define the specific requirements for each aspect. Requirements elicitation is about discovering software requirements and techniques by which engineers can collect them. It is essential for engineers to be able to identify and evaluate all potential alternative solutions regarding the software.

1.3 Research Objectives

One characteristic of current MT technology is that a lot of effort is focused on specification and implementation. There is a lack of research into the requirement engineering process in MT as well as in the selection of the most suitable RE techniques regarding a specific requirement. This gap has also been remarked by other researchers in the field with regards to the transML work [46]. This thesis research is not focused on the invention of a new RE process or technique for MT but it is rather focused on the development of an RE framework, which provides a systematic

transML is a family of modelling languages, which covers the whole life-cycle of the transformation development: requirements, analysis, design and testing. It can be used together with any transformation implementation language.
1.3. Research Objectives

process of applying the most suitable RE technique regarding different requirements during MT development. This would also allow the MT developer(s) to select and even customise the existing RE process and techniques according to their experience, organization policy and transformation properties. Thus, the overall objective of this research is to contribute to the earlier stages of model transformation development by introducing a professional requirements engineering process through an investigation of specific techniques for model transformation.

In short, the objectives of this thesis are:

- To carry out a literature survey on different industrial and academic transformation projects.
- To carry out an interview-based study on different industrial transformation projects.
- To define a requirements engineering process for MT.
- To define a taxonomy for functional and non-functional requirements for MT.
- To validate the choice of RE methods and techniques via two case studies.

The proposed framework is a formal model of requirements engineering activity for model transformations that provides a generalization of all known requirements engineering techniques. We are proposing this model, a framework for RE technique selection, which can be used during the requirements engineering phase in any given transformation development. So far, not much research has been dedicated to model transformations from a requirements engineering aspect. In our proposed model, unlike [55], not only an elicitation phase is included, but also all four stages of techniques proposed by [133] namely: Domain Analysis & Requirements Elicitation, Evaluation & Negotiation, Specification & Documentation, Validation & Verification.
1.4 Overall Aims and Contributions

The aim of this research is to contribute to the earlier stages of model transformation development by introducing a professional and systematic requirements engineering process through an investigation of specific techniques and methods for model transformation. By systematically comparing and evaluating the selected RE techniques and applying them to the case studies, a systematic RE process is proposed to provide a guideline and facilitate the RE process for MT developers. The main contributions of this thesis are as follows:

- Applying a semi-structural interview-based study with industrial MT experts and analysing real industrial MT projects (Chapter 3).
- Applying a systematic literature review survey on MT and its relationship with the RE process (Chapter 4).
- Defining a taxonomy for functional requirements for MT (Chapter 5).
- Defining a taxonomy for non-functional requirements for MT (Chapter 5).
- Proposing a novel methodology and framework for evaluating requirements engineering techniques for model transformation (Chapter 5).
- Applying the proposed methodology on different case studies to evaluate the outcome (Chapter 6).

Defining taxonomies and techniques will allow the requirement engineers to identify the right requirements for a given transformation. Moreover, once the requirements have been identified and categorized, engineers could refer to them during the Validation & Verification phase in order to make sure that nothing is omitted.
1.5. Overall Thesis Structure

Initially we will interview industrial transformation developers to ascertain the level of the requirements engineering that has been applied to their projects. We will also analyse and review some real industrial projects to evaluate the level of requirements engineering being used. Since the current problem in the model transformation field is that no engineering principles are being applied systematically, my approach in resolving this problem is to come up with a specific requirements engineering framework designed for model transformation development.

The process regarding this thesis research involved the following phases:

- Problem analysis and literature review
- Systematic literature survey
- Empirical investigation
- Design of the framework
- Case studies

1.5 Overall Thesis Structure

In order to achieve the overall aims and goals, this thesis is categorised into seven chapters. Chapter 1 is dedicated to the introduction of this research followed by Chapter 2 which provides a wide literature review on model transformation, requirements engineering and the relation of requirements engineering in model transformation development. In Chapter 3, we report on the results of an exploratory interview-based study with industry experts in real world model transformation projects. Chapter 4 provides a systematic literature review based on more than 160 case studies in the field in order to provide a better understanding of the research by analysing the related and existing works. Moreover, Chapter 5 proposes a new taxonomy for both functional and non-functional requirements in model transformations followed by a method for selecting suitable Requirements Engineering techniques with which the MT developers are able to select the most suitable technique for a specific
1.5. Overall Thesis Structure

requirement. Chapter 6 proposes two real case studies in order to evaluate the proposed framework. Chapter 7 summarises the outcomes of the research, and outlines the areas of possible future work.
Chapter 2

Background on Software & Requirements Engineering and Model Transformation

This chapter of the thesis gives a background on the software development process, software metrics and quality models in detail. It describes the current application of requirements engineering, its advantages, process models, methodologies and techniques. It also explains model transformation, its current application, its languages, its relationship with model driven engineering (MDE), its context and its various types.

2.1 The Software Development Process

In general, in software development, the development process is divided into different life-cycle phases namely: feasibility analysis, requirements analysis, specification, design, implementation, testing, and maintenance. The focus of this thesis is on the requirements analysis life-cycle.

2.1.1 Software Requirements

In general, requirements for any given software project are divided into functional and non-functional requirements. It is important to define
2.1. The Software Development Process

the functional and non-functional requirements before building the software and to make sure that the system or software will achieve the set objectives. Selić [129] looks at technical and non-technical aspects of requirements engineering and he believes that both have a significant influence in increasing developer productivity and product quality in industrial projects. Functional and especially non-functional requirements are sometimes neglected as the developers do not consider them as a component in their field of profession and consider them in a separation of concern manner [118]. In general, most companies mainly focus on the implementation phase more than other phases and try to consider every solution regarding a problem during this phase, which is similar to the man who only has a hammer and sees everything as a nail [129]. If during the requirements elicitation stage of the RE process the requirements are poorly specified, this will lead to wrong implementation or implementing something which is not needed and in some cases will result in the project’s failure.

According to the Standish Group International [52], a failure in a project means project cancellation or not meeting the main requirements of a project such as budgets, delivery time and objectives. In order for a project to be completed successfully, it has to meet its budget, delivery time and business objectives. The Standish Group report analyses the data of success and failure factors in different projects. The success and partial success rate of these projects were 29% and 50% respectively, whereas the failure rate was approximately 19% [52]. Tables 2.1, 2.2 and 2.3 show some of the important factors which had a role in the project’s success or failure.

Even by skimming through this data, we can notice the importance of the requirements engineering process, even though these numbers do not show details of the requirements since the required information was not available due to confidentiality. The highly significant role of the RE process is due to the fact that it consists of specific techniques applied in a certain sequence.
2.1. The Software Development Process

<table>
<thead>
<tr>
<th>Table 2.1. Project failure analysis (Standish Group (1995))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete requirements</td>
</tr>
<tr>
<td>Lack of user involvement</td>
</tr>
<tr>
<td>Lack of resources</td>
</tr>
<tr>
<td>Unrealistic expectations</td>
</tr>
<tr>
<td>Lack of executive support</td>
</tr>
<tr>
<td>Changing requirements/specifications</td>
</tr>
<tr>
<td>Lack of planning</td>
</tr>
<tr>
<td>Did not need it any longer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2.2. Project success analysis (Standish Group (1995))</th>
</tr>
</thead>
<tbody>
<tr>
<td>User involvement</td>
</tr>
<tr>
<td>Management support</td>
</tr>
<tr>
<td>Clear statement of requirements</td>
</tr>
<tr>
<td>Proper planning</td>
</tr>
<tr>
<td>Realistic expectations</td>
</tr>
<tr>
<td>Smaller milestones</td>
</tr>
<tr>
<td>Competent staff</td>
</tr>
<tr>
<td>Ownership</td>
</tr>
</tbody>
</table>

 Functional Requirements

Functional requirements refer to services which a software has to provide and how the system will respond to a particular input(s). Functional requirements contain the intended behaviour of a system and they are relevant to the *what* dimension (Figure 2.8). Such requirements describe the intended behaviour of the system explicitly. Functional requirements could be categorised into coarse-grained functionalities which must be supported by the system-to-be (the system to be developed)[143]. It is required for functional requirements to be satisfied in the system-to-be and it can be fully observed whether or not a functional requirement is satisfied.
2.1. The Software Development Process

| Executive management support | 20% |
| User involvement             | 15% |
| Optimization                 | 15% |
| Skilled resources            | 13% |
| Project management expertise | 12% |
| Agile process                | 10% |
| Clear business objectives    | 6%  |
| Emotional maturity           | 5%  |
| Execution                    | 3%  |
| Tool and infrastructure      | 1%  |

Non-functional Requirements

Non-functional requirements, an important factor in requirements engineering, could be regarded as softgoals. By softgoals it is meant that there is no clear-cut explanation regarding the goal’s achievement level. In general, it could be said that for softgoals, there is no complete satisfactory condition, thus the term ‘satisficing’ (partial degree of satisfaction) is applied. It could be said that a softgoal is satisfied once the goal has reached a certain level of achievement [95]. In this sense, non-functional requirements are not similar to functional requirements, whose satisfaction is fully required. There are several definitions regarding non-functional requirements among researchers, some of the most common of which are listed in Table 2.4.

Figure 2.1 illustrates a general classification of non-functional requirements. It represents the main criteria of non-functional requirements and is not exclusive to any specific case.

Advantage of Requirements Taxonomies

Taxonomizing the requirements according to their type not only would make it clearer to understand what the requirements refer to, but also by
# 2.1. The Software Development Process

## TABLE 2.4. Definition of non-functional requirements

<table>
<thead>
<tr>
<th>Source</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anton [5]</td>
<td>Non-functional requirements of a system describe the non-behavioural aspects of a system, capturing the properties and constraints under which a system must operate.</td>
</tr>
<tr>
<td>Kotonya et al. [80]</td>
<td>Non-functional requirements are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet.</td>
</tr>
<tr>
<td>Paech et al. [25]</td>
<td>“The term non-functional requirement is used to delineate requirements focusing on how good software does something as opposed to the functional requirements, which focus on what the software does.”</td>
</tr>
<tr>
<td>Landes [82]</td>
<td>“Putting it another way, non-functional requirements (NFRs) constitute the justifications of design decisions and constrain the way in which the required functionality may be realized.”</td>
</tr>
</tbody>
</table>

![Figure 2.1. A taxonomy of non-functional requirements](image)

having this type of distinction among them, will allow for a more semantic characterization of requirements. The following are some examples of possible distinctions:

- Requirements which describe *desired* behaviour (many functional requirements are of this kind)

- Requirements which describe *unacceptable* behaviour (many safety, security and accuracy requirements are of this kind)

- Requirements which describe *preferred* behaviour (many performance and ‘ility’ requirements are of this kind such as usability,
2.1. The Software Development Process

By having requirements taxonomies, confined and cross-cutting concerns could be differentiated. Confined concerns refer to functional requirements which focus on one particular point of functionality, whereas cross-cutting refers to non-functional requirements, meaning that the same requirements might contain more than one unit of functionality. For instance, in a library system, in order to be able to search for a book, user registration might be required. Having a requirement taxonomy is an aid in understanding what the requirements refer to and what category they belong to.

2.1.2 Software Project Types

Projects can be classified into different groups according to their characteristics. Depending on the type of project, a particular requirements engineering (RE) activity must be applied. An RE process model is an abstract definition of how to operate a group of activities. The term technique denotes how to perform a particular activity. The term method refers to identifying a guideline of how to perform a set of activities, mainly emphasising how a related set of techniques can be integrated.

Before commencing a project, a certain amount of preparation is required depending on the type of project. The type of project has a direct effect on the RE methods that need to be implemented. For instance requirements engineering for information systems differ from requirements engineering for embedded control systems or requirements engineering for generic services such as networking. According to the type of project, the necessary type of RE activities and techniques may differ. A description of some of the most well-known types of project is given below.
2.1. The Software Development Process

Brownfield vs Greenfield

A project can either be built from scratch or it can be built upon an already existing system which needs to be improved, integrated or extended. In a Greenfield project, the project is brand new, which will result in developers having to start from scratch and build the software from the beginning, whereas in Brownfield type of projects, a system already exists but it has to be further developed and improved. In this case, developers could work on the current system (system-as-is) and extend its functionalities [143].

Customer vs Market Driven

A project could be either a solution for a particular type of client in the market (customer-driven) or a solution which would cover the need of a large percentage of the market (market-driven). In customer-driven types of projects, the project is designed according to the needs of a specific type of client, whereas in market-driven projects, a larger scope of solution is considered covering more than just one particular type of client [143].

In-house vs Outsourced

A project could be assigned to a particular organization in order to carry out all the project’s life-cycle processes (in-house) or it could be assigned to different companies according to the different phases of the project (outsourced). In an in-house type of project, one team or company will carry out all the phases of the project, whereas in an outsourced project, usually once the requirements have been identified different teams from different companies will carry out the different phases such as design, implementation, testing, etc. [143].
2.2. Software Process Model

Single-product vs Product-line

The outcome of a project could have only one version which would satisfy the customer’s need or it could have different versions each of which would cover particular needs in a large organisation. “In a single-product project, a single product version is developed for the target customer(s). In a product-line project, a product family is developed to cover multiple variants” [143].

2.2 Software Process Model

So far, several development models have been introduced which can be applied in software development projects. However, time has revealed that they all contain flaws which have rendered them not fully efficient. The following are some of the most well-known processing models which have been applied for developing software projects:

- The Code-and-fix model
- The Stagewise model
- The Waterfall model
- The Evolutionary Development model
- The Spiral model
- Agile model

The Code-and-fix model [18] was one of the first models used in software development. The idea behind this model was to begin the development by doing some coding at first and then to come up with the requirements, design and testing units. One of the disadvantages of this model is allowing simultaneous bug fixing as the development carries on. By applying a number of fixes, the primary structure of the code would
2.2. Software Process Model

be affected negatively. Moreover, since the requirement phase is not carried out in a systematic manner, usually even when the software is fully developed, it would still differ from the user’s need which in turn would either result in redeveloping the software or rejecting it.

The Stagewise model [18] was designed in a way to develop software in successive stages such as ‘operational plan’, ‘operational specifications’, ‘coding specifications’, ‘coding’, ‘parameter testing’, ‘assembly testing’, ‘shakedown’, ‘system evaluation’ [18]. Similar to the Stagewise model, the Waterfall model is an augmented version of the Stagewise model. It supports ‘feedback loops’ between different stages in order to allow revisiting the earlier tasks and the redoing of any task mentioned in those feedbacks. Moreover, the Waterfall model provides an initial prototyping model which would allow for the evaluation of requirements, and design of the system [18]. Although the Waterfall model is more efficient than previous processing models, it still encountered difficulties. This model’s main shortcoming is that the requirements and design of the system have to be fully extracted and documented during the early stages of the development process, which does not allow for any kind of requirements and design modification throughout the remainder of the development life-cycle. This feature would reduce the quality of the delivered software especially in cases where the requirements have not been fully understood at the early stages or if the client(s) would like to modify or add any extra functionalities at a later stage.

The Evolutionary Development model [18] is another processing model that has been used in software development. One of the exclusive features of this model is its use in scenarios where the user(s) does not actually know what is explicitly needed. The model would provide the user with an initial realistic operational ground in a smaller scale compared to the final product. This would allow the client(s) to have a better understanding about the system and to better evaluate what is needed. Yet this model is not without flaws. One of the main shortcomings of this model is the lack of planning before software development. Also, most of the time there are “unrealistic assumptions that the user’s
2.2. Software Process Model

operational system will be flexible enough to accommodate unplanned evolution paths[18].

The Spiral model[18] is one of the most commonly used development models at the present time. It consists of four iterative stages including:

1. Determining objectives, alternatives and constraints
2. Identifying and resolving risks and evaluating alternatives
3. Developing iterations, deliverables and verifying correctness
4. Planning next iteration

In the Spiral model, a software project will go through these stages repeatedly. During the initial stage, requirements are gathered by applying relevant techniques (interview, background reading, group sessions, etc.). Then risks are identified and the level of risk of each requirement is assessed and alternative solutions are considered accordingly. During the next stage which is followed by testing, the client(s) would then be able to verify the proposed functionality. Once the client has been satisfied with the result, the next spiral is planned. Due to the iterative feature of this model, each stage can be revised, adapted or extended throughout the development life-cycle. It allows for having the requirements modified and/or added during late iteration, increasing the flexibility of the system, especially in cases where the client(s) comes up with new issues. The Spiral model particularly focuses on risk analysis. If there exists any type of risk, then a formulation of cost-effective solutions must be considered to resolve the risk(s). Depending on the type of risk, a different solution might be proposed in the form of ‘prototyping’, ‘simulation’, ‘benchmarking’, ‘questionnaire’, ‘analytic modelling’ or a combination of all these. Figure 2.2 illustrates a general framework of the Spiral model.

Agile model is centred around four values defined by the Agile manifesto [41]: individual and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan.
2.2. Software Process Model

Figure 2.2. Spiral model of the software process [18]

Agile development follows an iterative and incremental development in a highly collaborative style in order to produce the software with high quality in a cost and time effective manner. It emphasizes on delivering the smallest piece of software with functionality, as early as possible and throughout the development, even while the system is evolving, by adding more functionalities during the entire development process. This allows the project to adapt rapidly to potential changes. Agile highlights the importance of relationships and communications amongst the developers. The client’s collaboration is another important factor that must be considered throughout the whole development cycle. Both the development team and the client (or the client’s representative) should be well informed, competent and authorised to make potential adjustments that may emerge during the development process [1].

There are different existing methods in Agile, but for the sake of brevity, we will not go into more detail about these methods (for more
2.3 Software Measurements and Metrics

detail refer to [1] and will just list a sample of Agile methods as follows:

- Extreme programming
- Scrum
- Crystal family of methodologies
- Feature driven development

2.3 Software Measurements and Metrics

Software measurement and metrics are important factors in software engineering. In general, measurement refers to a particular attribute of a product such as size and quality. For instance, the number of lines of code (LOC) in a program is a measurement. Metric refers to the ratio of a measurement that a particular attribute contributes to the product. For instance, the number of LOC per developer hours would be considered as a metric.

“Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules” [39]. Measurement is a useful method by which developers are able to get some sense of whether or not a requirement is consistent, completed, satisfied or in general whether the requirement is ready to be released or not. From the beginning of software development, software measurement and estimation have been a cause of discussion amongst engineers. Software and system engineers need an appropriate method in order to measure the effectiveness of a given requirement.

Several number of software metrics have been developed since 1976. From all introduced software metrics, four have been the source of the majority of research conducted on software metrics. The first theories are defined by Halstead [50], Albrecht [2], DeMarco [33] and McCabe [104] known as cyclomatic complexity, a measure of the number of paths
2.3. Software Measurements and Metrics

Figure 2.3. Comparison table using impact estimation [44]

through a program. “The number of paths can be infinite if the program has a backward branch. Therefore, the cyclomatic measure is built on the number of basis paths through the program” [105]. Software development process has undergone a dynamic revolution during the past decades. This has resulted in evolving the software development methodologies in order to meet changing life cycle patterns which they have had as their objectives, emphasis on design and analysis [105].

Impact Estimation (IE) provides estimation tables, which allows developers to analyse any technical or organizational idea according to requirements and costs. “The intention of impact estimation is that it helps answer the question of how our design ideas impact all of a system’s critical performance attributes (such as usability and reliability) and all its resource budgets (such as financial cost and staff headcount) for implementation and operational running” [44]. Figure 2.3 provides an example regarding the purpose of the impact estimation that can be
2.3. Software Measurements and Metrics

used during the early stages of software engineering.

2.3.1 Software Quality Models

A large amount of research has been dedicated to create a feature catalogue to be used in software development. For instance, the International Organisation Standardization (ISO) and the International Electro technical Commission (IEC), have proposed a number of standards for software quality evaluation [40]. The following are a sample of quality models that are appropriate for model transformation as a measurement framework:

- **ISO/IEC Quality Model** [21]. It is a quality standard which applies to both quality models and metrics that has been used widely and well accepted. It defines a wide-ranging set of quality attributes by which software products can be evaluated. Moreover, it defines a guideline to measure the attributes’ quality. ISO/IEC quality model can be used to evaluate any type of software product. It includes two categories of attributes: *internal* and *external*. Internal attributes can be measured during the development process, whereas external attributes can be measured within the performance and testing process of the software product. Table 2.5 presents the six quality characteristics and sub-characteristics of ISO/IEC.

- **Dromey Quality Model** [35]. In this approach, the quality model differs depending on the attributes of particular products. Dromey developed his model quality framework to analyse software components. He defines a set of attributes that has direct relation with software component characteristics.

- **McCall Quality Model.** McCall introduced one of the first comprehensive quality models in [106]. The proposed framework is divided into two sections: *measurable* and *non-measurable*. Metrics are assigned to the measurable qualities in a subjective manner. Figure 2.4 represents the McCall quality model.
2.3. Software Measurements and Metrics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Sub-characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
</tr>
<tr>
<td></td>
<td>Accuracy</td>
</tr>
<tr>
<td></td>
<td>Interoperability</td>
</tr>
<tr>
<td></td>
<td>Security</td>
</tr>
<tr>
<td></td>
<td>Functionality compliance</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
</tr>
<tr>
<td></td>
<td>Recoverability</td>
</tr>
<tr>
<td></td>
<td>Reliability compliance</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
</tr>
<tr>
<td></td>
<td>Operability</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
</tr>
<tr>
<td></td>
<td>Usability compliance</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Resource utilization</td>
</tr>
<tr>
<td></td>
<td>Time behaviour</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Analysability</td>
</tr>
<tr>
<td></td>
<td>Changeability</td>
</tr>
<tr>
<td></td>
<td>Stability</td>
</tr>
<tr>
<td></td>
<td>Testability</td>
</tr>
<tr>
<td></td>
<td>Maintainability compliance</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
</tr>
<tr>
<td></td>
<td>Installability</td>
</tr>
<tr>
<td></td>
<td>Co-existence</td>
</tr>
<tr>
<td></td>
<td>Replaceability</td>
</tr>
<tr>
<td></td>
<td>Portability compliance</td>
</tr>
</tbody>
</table>

- **Boehm Quality Model** [19]. It is a well-defined framework which allows software quality characteristics to be analysed. In this framework, the initial quality characteristics are regarded as *general utility* which itself is composed of: *as-is utility, maintainability* and *portability*. Metrics are then generated in order to assess each individual character. Figure 2.5 represents the Boehm Quality...
Similar to any type of software product, model transformation also needs to be evaluated regarding its functional qualities such as: understandability, modifiability, usability, interoperability and etc. Identifying functional qualities, therefore, is an important task. A quality model could be used as a paradigm to define qualities. Dromey’s framework is very useful, however “it is not a trivial task to generate a framework
2.3. Software Measurements and Metrics

individually for each particular element in MDE. For instance, model transformation approaches have different styles and can be analysed from different perspectives. This results in having different frameworks for evaluation of each transformation” [74]. Based on [74] findings, McCall Quality Model metrics can only be measured subjectively and does not provide sufficient evidence for assessment of MT. According to Boehm Quality Model, it is possible to evaluate model transformation. The Boehm Quality Model supports mainly top level quality aspects which are more generic and inappropriate for measuring the quality of model transformation.

25

Figure 2.5. Boehm Quality Model [19]
2.3. Software Measurements and Metrics

2.3.2 Goal-Questions-Metrics

The Goal Question Metrics (GQM) [10] is a useful method for identifying measurements for model transformation. In any software development process, a measurement mechanism is essential in order to evaluate the system and its components. This would not only improve the overall quality of the system, but it could also be used as proof and enactment of the quality of the system. In general, in order for a measurement technique to be effective, it has to be focused on specific goals and be applied on all products, processes and resources throughout the entire developing process. Then the result should be defined according to the organization’s context and characterization.

In 1994, Basili [10] introduced GQM, a measurement approach, which has been used both in academia and industry ever since. In GQM, abstract goals are each characterised by several concrete objectives (questions), which are associated with measurable dimensions (metrics) that are grounded in reference values.

The GQM approach is "based upon the assumption that for an organization to measure in a purposeful way it must first specify the goals for itself and its projects, then it must trace those goals to the data that are intended to define those goals operationally, and finally provide a framework for interpreting the data with respect to the stated goals" [103].

In other words, GQM defines the measurement model on three levels: conceptual level (goals: abstract qualities that we wish the system to have), operational level (questions: concrete questions about some aspects of a goal), quantitative level (metrics: scaled unit of measurement for the responses).

Defining goals is a useful process as it allows to focus on what is important and to make the goal more specific while offering metrics that are relevant to these goals. Not only it defines complete relationships amongst goals and metrics, but also it discovers any missing goals or inconsistency between goals [24].
2.4. Requirements Engineering

In GQM, goals are written explicitly, allowing the focus to be diverted more effectively to what the important issues are. It is possible to have more than one goal to be achieved at the same time. Goals in GQM have no meaningful definition until they are associated with some qualitative measure. Questions would make it possible to approach the problem from a conceptual level through to an operational level. Therefore, firstly, goals are refined into quantifiable questions, where a question can be used for more than one goal. Then eventually, every question is associated with a set of metrics in order to identify the result in a qualitative manner. Note that the same metric may apply to different questions.

![Figure 2.6. GQM model](image)

2.4 Requirements Engineering

Software development has suffered from a lack of requirements engineering almost since the beginning of the industrialization of software development. Royce [58] states:

There are four kinds of problems that arise when one fails to do adequate requirements analysis: top-down design is impossible; testing is impossible; the user is frozen out; management is not in control. Although these problems are lumped under various headings to simplify discussion, they are actually all variations of one theme - poor management. Good project management of software procurements is impossible
2.4. Requirements Engineering

without some form of explicit (validated) and governing requirements.

According to [94], research shows that RE is not being applied properly in industry. For instance, a study in 2003 suggests that about 52% of nearly 2000 software/system developers in south-east Pennsylvania believed that their company did not do enough requirements engineering [112]. In a similar follow-up survey in 2008 by Marinelli [102], the circumstances remained still the same and again around 52% of the participants reported that their company did not perform an adequate amount of requirements engineering. Moreover, another more recent study of seven independent companies by [94] indicates that “existing requirements engineering methods are insufficient for handling requirements for complex embedded systems”.

The primary focus of the RE process is to provide guidelines regarding the order of different development phases such as: requirements, design, implementation, testing and validation. Its main aim is to provide guidance on how to apply the appropriate task throughout each development phase. For instance, “determining data, control, or ‘uses’ hierarchies; partitioning functions; allocating requirements and how to represent phase products (structure charts, stimulus-response threads, state transition diagrams)” [18].

According to the standard software development process [133], requirements engineering is the initial phase of the software development life-cycle where the software’s specifications are declared. During the requirements engineering process, all the requirements to be achieved by the software (functional and non-functional requirements), and the criteria for measuring the degree of their satisfaction, must be elicited and documented in the requirements specification.

According to Bell et al. [13], requirements for any type of system do not naturally arise, rather they have to be systematically engineered for this reason the term engineering is used. An important advantage of the RE process is that it not only saves costs, but also saves time
2.4. Requirements Engineering

It is not yet possible to claim that all organisations and companies apply the RE process as diligently as they should [27]. This could be due to several factors, the most common of which are: lack of time and/or budget. In some projects the RE phase is either neglected or performed incompetently in order to save time and budget. Paradoxically this is a false assumption which project managers often make [14]. As mentioned earlier, currently the RE process is performed in many companies in a rule of thumb manner [129]. Both money and time can be saved if errors and flaws are detected during the RE stage rather than later. According to Boehm [17], it costs approximately five times more to detect and resolve errors during design, ten times more during the implementation phase, 20 times more during the testing and up to 200 times more after delivering the system (Figure 2.7).

![Figure 2.7. Cost of late correction (Boehm)](image)

During the requirements engineering process, all the requirements to be achieved by the software (functional and non-functional requirements) and the criteria for measuring the degree of their satisfaction must be elicited and documented in the requirements specification. According to [119] “if the specification describes both hardware and software, it is called system requirements specification; if it describes only software, it is
2.4. Requirements Engineering

called software requirements specification”. The process of constructing a
requirements specification for a system is called requirements engineering.

“Requirements Engineering is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the context in which it will be used. Hence, RE acts as a bridge between the real-world needs of users, customers, and other constituencies affected by a software system and the capabilities and opportunities afforded by software-intensive technologies” [157].

In other words, RE is referred to as a set of activities as it is neither a single stage nor phase. It is concerned with identifying and communicating, which means that communication is as important as analysis. It identifies and communicates the purpose of a software-intensive system. Because quality means fitness for purpose, it is not really possible to say anything regarding the quality unless the actual purpose is understood. RE is also concerned with the context in which the software-intensive system will be used. Context is important since designers need to know how and when the system will be used. Therefore, RE is like a link between the actual needs (requirements are partly about what is needed) of users, customers and other constituencies affected by a software system, which means there is a need to identify all the stakeholders, not just users and customers; and the capabilities and opportunities afforded by software-intensive technologies, meaning that the requirements must be realistic and possible [113].

The highly significant role of RE activity is due to the fact that firstly, it declares the importance of the goals which were the reason for developers to develop such a software system. Secondly, it highlights precise specifications of the intended software to be built [113]. It could be said that it is the foundation of the development process which consists of specific techniques applied in a certain sequence.

The RE process is categorized into two groups of models according to their orientation: top-down orientation and bottom-up orientation. Top-down oriented methods such as DeMarco [32], start the process with an abstract description regarding the current and future circumstances. The
2.4. Requirements Engineering

abstract model will eventually result in a concrete form as the process progresses. On the other hand, bottom-up oriented methods such as Sommerville et al. [133] “start with observations about the real world (concentrate on the instance level) and build abstract descriptions from the observations as the process proceeds. Although it is obvious that both approaches bear unique advantages and are therefore essential for developing a specification, methods offering an integration of both approaches are still missing. Moreover, existing methods ignore the fact that RE is an iterative process in which the RE team learns about the current and/or future reality” [119].

In this thesis, we use the RE process model proposed by Sommerville et al. [133] and adapt it according to our specific needs. This process model is widely accepted by researchers and professional experts [143]. The following are the most important phases of RE (proposed by Sommerville et al.) which have to be applied:

- Domain Analysis & Requirements Elicitation
- Evaluation & Negotiation
- Specification & Documentation
- Validation & Verification

The first step in RE is to understand what problem should be solved and why such a problem needs to be solved. Then it has to be declared who is responsible for solving such a problem. In other words, there are three main dimensions that we must consider:

1. What dimension: what problem should be solved
2. Why dimension: why such a problem needs to be solved
3. Who dimension: who should be involved in the responsibility of solving the problem
2.4. Requirements Engineering

In general, it could be said that the primary role of requirements engineers is to inquire about the problem which usually results in considering two different versions of one system:

1. System-as-is: the existing system

2. System-to-be: the developed system as it should be by solving the problem

During the RE process, we have to collect as much information as we can. Having sufficient knowledge about the problem is extremely important; firstly to understand what the problem is, secondly to be able to find the appropriate solution(s). Throughout the RE process, requirements should be categorised into two types of statements: descriptive and prescriptive statements. Descriptive statements refer to those types of statements that only consider properties of the system and states, which properties are true about the system irrespective of the way the system behaves. On the other hand, prescriptive statements refer to those types of statements that consider the properties of the system according to how the system behaves and state what should be true about the system. The distinction between descriptive and prescriptive statements is an essential factor in RE. Descriptive statements are static and therefore cannot be modified, whereas prescriptive statements are more dynamic and the engineers have the flexibility to modify them. In Figure 2.9 the classification of RE statements is illustrated in a diagram.
2.4. Requirements Engineering

Figure 2.9. Classification of statements in RE [97]

2.4.1 Domain Analysis and Requirements Elicitation

The initial step in the RE process is the act of obtaining a great deal of knowledge regarding the domain of the current problem, the organization or company confronting the problem and the existing system that is facing the problem. Once the required knowledge or information has been acquired, a draft document could be provided which would help developers to:

- Understand the context of the actual problem
- Identify the stakeholder’s actual needs and requirements
- Find an alternative solution to fulfil stakeholder’s needs
- Understand the structure of the organization in which the system would be used (i.e. business objectives, policies, roles and responsibilities)

During the Domain Analysis & Requirements Elicitation stage, there are two main techniques that could be performed efficiently and systematically: artefact-driven technique and stakeholder-driven technique. The artefact-driven technique is the process of using artefacts which already exist such as a collection of data and documentation about the system. The following methods should be applied during the artefact-driven technique according to the type of project:
2.4. Requirements Engineering

- Background study
- Data collection
- Questionnaire
- Storyboard and scenario
- Mock-ups and prototypes for early feedback

Stakeholder-driven technique is another useful technique to be applied during the Domain Analysis & Requirements Elicitation phase. It is more focused on interacting with a specific type of stakeholder in order to gain relevant information about the required system, organization, main users, etc. The following methods are applied during the stakeholder-driven technique according to the type of project:

- Interview
- Observation and ethnographic studies
- Group session and brainstorming

In general, the term stakeholder can be defined as an individual or an organisation or group of people who is either affected by or has an effect on the outcome of a given project [122]. It is essential to fully identify all the stakeholders of the project as an initial step prior to any other action, because by missing an important group of stakeholders, there is a major risk of missing a whole set of requirements of the system. A good participation of stakeholders in the software development cycle not only would result in a better understanding of the actual problem, but also would help to build what is required according to the stakeholders’ needs. The onion model of project stakeholders has been used to describe different types of stakeholders and their relation to the system under development. In this model, stakeholders are categorized into three different types: Operational, Containing Business and Wider Environment. Figure [2.10] illustrates an onion model in which the stakeholders have been categorised according to their role and effect on the system.
2.4. Requirements Engineering

The Operational area includes those types of stakeholders which have a direct interaction with the system. The Containing Business area includes types of stakeholders that somehow benefit from the system and the Wider Environment area includes stakeholders which have an effect on or an interest in the system.

In the following section, some RE techniques have been selected based on their importance and relevance to this thesis which can be applied during the Domain Analysis & Requirements Elicitation stage.

**Interview**

Interviews can be considered as a preliminary requirements elicitation technique. In general, there are two types of interview: structured interview and unstructured interview. In a structured-interview, a set of pre-defined questions have been drafted according to the specific purpose of the interview. Whereas, in an unstructured-interview, the interview is based on an informal discussion with the stakeholder(s) about the current system and the stakeholder’s needs regarding the new software systems.

Regardless of being structured or unstructured, interview techniques are usually based on the following procedures:

---

Figure 2.10. Stakeholder onion model

---
2.4. Requirements Engineering

- Selecting a specific type of stakeholder according to the required information that is needed
- Organizing a session with the stakeholder(s) where questions are asked and recorded
- Writing a report about the interview results (interview transcripts)
- Submitting the outcome of the interview (report) to the stakeholder(s) for refinement and validation

Prototyping

Prototyping is a RE techniques for receiving early feedback during the Domain Analysis & Requirements Elicitation stage. Often, it is a difficult procedure for stakeholders to comprehend the project’s textual system descriptions since prior knowledge may be necessary. Therefore, a reduced sketch of the product is represented instead in order to give the stakeholder(s) some idea regarding the appearance and functionality of the future software system in the form of a prototype.

The primary target of prototyping approach is to identify a set of requirements. This type of prototyping is known as mock-up or throwaway prototyping (rapid prototyping). On the other hand, if the prototype is evolving and converting into the actual final product throughout the development process, then the term evolutionary prototyping is used in such cases [143].

A prototype is particularly helpful for the requirements that are unclear or hard to understand. In general, there are two kinds of prototype: functional prototype which demonstrates functional aspects of the software and user-interface prototype which demonstrates user-software interaction aspects.

Regardless of functional or user-interface, prototyping techniques are usually derived from the following iterative procedures [143]:

36
2.4. Requirements Engineering

repeat
   build a prototype version from the selected requirements
   show execution of prototypes
   get feedback from stakeholders
   updates from feedback
until prototypes get full agreements from stakeholders

Scenario based approach

According to [145], scenario based approaches are widely used in the software development life-cycle. From an RE perspective, scenarios are useful for two main reasons. Firstly, they explain the current software system through concrete examples of a real set of interactions. Secondly, they explore how the required software system (system-to-be) will run via concrete examples of hypothetical sequences of interactions. There are different types of scenario based approaches from which only two are presented below.

- **Positive scenario vs Negative scenario**
  A positive scenario identifies what action should occur for a behaviour that the software system can cover, whereas a negative scenario demonstrates behaviour that the system should exclude.

- **Normal scenario vs Abnormal scenario**
  A normal scenario includes a sequence of interactions which proceed normally as expected, whereas an abnormal scenario captures a sequence of interactions based on unusual conditions.

The scenario based approach technique has some advantages and disadvantages like any other technique. The ease of usage of scenarios by different stakeholders with different backgrounds in order to share an understanding, could be regarded as a positive side of scenario based approaches. On the negative side, because they are represented as a lim-
2.4. Requirements Engineering

ited number of examples, they do not cover all possible behaviour under different circumstances [143].

Goals-Operators-Methods-Selection rules

Goals-Operators-Methods-Selection rules (GOMS) [71] model represents the procedural knowledge that developers require to be able to start the project. In other words, it is an analytic model for identifying tasks. GOMS analysis consists of defining and describing user’s Goals, Operators, Methods, and Selection rules in formal notations. The model consists of:

- **Goals**: The intention of the user that must be achieved.
- **Operators**: The required actions that must be performed to accomplish the goal.
- **Methods**: Sequences of operators to achieve a goal. There may be more than one method available to accomplish a single goal, if this is the case then:
  - **Selection rules**: When a user would select a certain method among others [71].

Similar to any given technique in the requirements engineering field, GOMS has some advantages and disadvantages. One of its advantages is that an estimation of a given interaction can be done with little effort, at little cost and in a short amount of time, which has made GOMS very practical. On the other hand, in order to define the goals, the analysts must define the task which needs to be accomplished in detail which often goes beyond the system specifications [71]. This may be difficult as the analysts must consider all concepts of the system in order to identify the main goals. GOMS analysis has to define what to do and what not to. Explaining all these specifications is usually very difficult especially in complex systems.
2.4. Requirements Engineering

2.4.2 Evaluation and Negotiation

At the stage of Evaluation & Negotiation, it is assumed that the previous stage, that of Domain Analysis & Requirements Elicitation, has been performed effectively. This section will introduce techniques and methods for evaluating the elicited requirements, along with possible negotiations that might occur between developers and stakeholders.

The evaluation stage is a necessary process that must be carried out during the software development process. It is possible for the existence of an inconsistency amongst the requirements, the chances of which will increase if the requirements have been gathered from multiple and different stakeholders. Sometimes, this inconstancy could even result in having conflicts between the requirements. Some requirements might increase the probability of different types of risk such as: safety, security and development risks. Therefore, an appropriate evaluation is essential as part of the development process.

Furthermore, requirements’ evaluation allows for the discovery of alternative solutions. This would be useful especially in cases where there are inconsistencies or conflicts among the requirements. Once the alternative solutions have been identified, then negotiations could take place between both sides regarding the alternative solution(s) based on the budget and the delivery time. In general, the aim of requirements evaluation is to ensure the system will have a low level of risk, that there are no conflicts and that there is agreement between the stakeholders about the requirements. The following methods should be applied during the Evaluation & Negotiation stage according to the type of project:

- Inconsistency management
- Risk analysis
- Evaluating alternative options
- Requirements prioritization
2.4. Requirements Engineering

In the following section, some RE techniques have been selected based on their importance and relevance to this thesis which can be applied during the Evaluation & Negotiation stage.

**Representation and Maintenance of Process knowledge**

Representation and Maintenance of Process knowledge (REMAP) is a requirements engineering technique which could be applied during the Evaluation & Negotiation phase. The REMAP model is based on the ‘Issue-Based Information Systems’ (IBIS) design rationale model [81] and uses goals to provide the context in which design deliberations occur in RE. IBIS is a method based on deliberation by the articulation of questions. Every question is regarded as an issue which is associated by a position (answer) as a solution to the issue. Positions are followed by arguments as means of support [81]. In REMAP, a goal expresses a capability that must be met in order to meet user needs, to solve a problem or to achieve an objective. Goals drive the argumentation process, the outcome of which is the definition of a design solution that satisfies the initial goals. Satisfying the goals generally requires the introduction of further goals leading to a network of goals [81].

**Unified Modelling Language**

The Unified Modelling Language (UML) [20] is based on graphical notations. UML is a language for expressing requirements, specification models and designs in a platform-independent manner. It includes multiple types of diagram each of which allows for a specific design aspect of the software system to be represented based on the type of diagram. According to [143] the following diagrams of UML (from [84]) are relevant to the requirements engineering process:

- **Class diagram**
  Class diagrams illustrate classes of the system in terms of objects as well as any relationship between them. Throughout the devel-
2.4. Requirements Engineering

In the development process, class diagrams can be used at different stages such as:

- Conceptual modelling of problem domain
- Specification modelling; recording in precise but implementation independent manner based on agreed requirements of the system
- Design modelling; detailing design structures of the system as well as all dependencies between classes

- **Use Case diagram**
Use case diagrams can be used to describe:

- the system-to-be; the system to be constructed
- Actors; representing a role played by a person or an entity that interacts with the system
- Use cases; families of usage scenarios of an application, grouped into coherent cases of functionality

A use case is a general group of possible scenarios of using the system; it could be said that a scenario is an instance of a use case. It can be of either two types: inclusion or extension.

- *Includes*: use case uc1 includes use case uc2 if doing uc1 always involves doing uc2. It is particularly useful if uc2 is a common subtask of two or more use cases.
- *Extends*: use case uc1 extends use case uc2 if uc1 provides additional functionality used to carry out uc2 in certain cases.

- **Sequence diagram**
Sequence diagrams consist of:

- Object lifelines, represented by vertical dashed lines
2.4. Requirements Engineering

- Vertical rectangles, indicating activities of the object and its duration
- Arrows, from one object lifeline to another represent messages, usually method invocations

- **State diagram**

  State machines graphically represent the dynamic behaviour of objects. It also shows the life history of objects over time and patterns of inter-communication.

  It consists of the following elements:

  - States
  - Transitions
  - Default initial state
  - Termination of state machine

2.4.3 Specification and Documentation

The Specification & Documentation phase of the RE process is mainly based on the result of the two previous phases: Domain Analysis & Requirements Elicitation and Evaluation & Negotiation. It begins with the specification process which contains a set of agreed statements by all relevant sides of the project such as: requirements, assumptions, and system properties. Based on the results of the specification, the requirements documentation can be drafted. In this section, some RE techniques will be introduced that could be applied during the Specification & Documentation stage.

**Formal specification**

A formal specification, documents RE items formally. It formalizes the RE statements with precise notations according to mathematical concepts which would be necessary to validate the requirements and deals

42
2.4. Requirements Engineering

with them if requirements change. KAOS methodology [143] is a goal-oriented requirements engineering approach which defines goals by using a formal mathematical method of analysis. It is a useful methodology as it supports the entire requirements elaboration process. It defines requirements as high-level goals that need to be achieved and assigns objects and operations to responsible agents [144]. In KAOS methodology, goals are formalised by using temporal logic according to the pattern of behaviour they require. The goals of KAOS can be expressed using the following formulae in temporal logic [66]:

- Achieve: $G \Rightarrow \Diamond Q$
  
  Q holds in some future state

- Cease: $G \Rightarrow \Diamond \neg Q$
  
  There will be some point in the future that Q will not hold

- Maintain: $G \Rightarrow \Box Q$
  
  Q holds in all future states

- Avoid: $G \Rightarrow \Box \neg Q$
  
  Q will never hold in the future

Model transformation systems necessarily involve a notion of time. Propositional logic is not expressive enough to describe these features in terms of requirements engineering. Yet, describing those using natural languages are even less precise once we involve time. Here we have formulated the general temporal properties of MT as follows:
### 2.4. Requirements Engineering

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Liveness</strong></td>
<td>Every request is followed by a response</td>
</tr>
<tr>
<td></td>
<td>$\square (\text{request} \rightarrow \text{response})$</td>
</tr>
<tr>
<td><strong>Safety</strong></td>
<td>$p$ never happens</td>
</tr>
<tr>
<td></td>
<td>$\square \neg p$</td>
</tr>
<tr>
<td><strong>Fairness</strong></td>
<td>if $p$ happens infinitely often, then $\varphi$ will be true</td>
</tr>
<tr>
<td></td>
<td>$\Diamond \diamondsuit p \rightarrow \varphi$</td>
</tr>
<tr>
<td><strong>Invariance</strong></td>
<td>At some point, $p$ will hold forever</td>
</tr>
<tr>
<td></td>
<td>$\Diamond \square p$</td>
</tr>
<tr>
<td><strong>Partial correctness</strong></td>
<td>if $p$ is true, then $q$ will be true when the task is completed</td>
</tr>
<tr>
<td></td>
<td>$p \rightarrow \square (\text{done} \rightarrow q)$</td>
</tr>
<tr>
<td><strong>Mutual exclusion</strong></td>
<td>two processes cannot enter their critical sections simultaneously</td>
</tr>
<tr>
<td></td>
<td>$\square \neg (\text{in}<em>{CS}1 \land \text{in}</em>{CS}2)$</td>
</tr>
<tr>
<td>$p$ oscillates every step</td>
<td>$\square ((p \land X\neg p) \lor (\neg p \land Xp))$</td>
</tr>
</tbody>
</table>

The formalised rules need to be checked for internal correctness properties such as definedness and determinacy, which should hold for meaningful rules. A prototype implementation can be generated and its behaviour on a range of input models, covering all of the scenarios considered during requirements elicitation, can be checked. When a precise expression of the functional and non-functional requirements has been defined, these can be validated with the stakeholders to confirm that they do indeed accurately express the stakeholders’ intentions and needs for the system.

The formalised requirements of a transformation $\tau: S \rightarrow T$ can also be verified to check that they are consistent:

- The functional requirements must be mutually consistent
2.4. Requirements Engineering

- The assumptions and invariant of $\tau$, and the language constraints of $S$ must be jointly consistent.

- The invariant and post conditions of $\tau$, and the language constraints of $T$ must be jointly consistent.

- Each mapping rule $LHS$ must be consistent with the invariant, as must each mapping rule $RHS$.

**Non-functional requirements (NFR) framework**

The NFR framework approach is focused on the non-functional requirements (software quality attributes such as security, performance, etc.) throughout the entire developing process. NFR framework aims to help developers to consider the non-functional requirements as important and give them as much attention as they can. It helps to identify NFR for the domain by acquiring knowledge about the system and its domain. It also applies trade-offs and prioritization techniques among the NFR. Figure 2.11 presents a catalogue of some NFR types [26].
2.4. Requirements Engineering

Natural language

Free documentation in natural language is a requirements documentation technique. Using the natural language technique to document requirements would be a suitable option, mainly because there would be no limitation in terms of expressiveness on what is needed in natural languages, it can be understood by all stakeholders, and there is no communication barrier. Nevertheless, this lack of limitation may result in some negative outcomes as well, such as ambiguity (requirements with no unique interpretation), opacity (requirements with no visible rationality and independencies) and noises (requirements with no information on any problem) [143]. In order to avoid these flaws, we can introduce some discipline and structure in the documentation process using natural languages, for instance, SBVRSE [138].

2.4.4 Validation and Verification

In this stage, specifications must be analysed. They should be validated by stakeholders in order to be evaluated according to their actual need. Specifications should be verified in order to check consistency and avoid conflicts and omissions. Any potential error and flaw must be fixed during this phase and before the actual development in order to save cost, effort and time. The aim of validation in the requirements engineering concept is to check the achievement of the requirements. In other words, it is the process of checking whether or not a completed project specification (system-to-be) has met the stakeholders’ expectations. To this end, we can use a scenario based technique (section 2.3.1). A sample of validation scenarios can be explored by stakeholders in order to check the validity of the system. Another approach could be to animate parts of the system by generating an executable model of the final system according to the specifications [143]. In this approach operational behaviour is graphically illustrated as a model where the model moves through time [127]. Visualization of a simulation would help the stakeholders to validate the system’s behaviour. Last but not least, the verification process
2.4. Requirements Engineering

can be done through formal checks. For this purpose, specifications must be formal. The verification process checks whether or not the system is correct according to semantics and requirements that were identified earlier in the software life-cycle.

Validation in model transformation denotes checking whether or not the inputs, outputs and the transformation itself fulfil the specifications (quality criteria). This process can be done in a variety of manners, such as inspection and review, check-list and testing.

Requirements inspection and review [143] are applicable techniques for model transformation development. According to our investigations (Chapters 3 and 4), this technique is widely used within the MT community which consists of selecting an individual or a group of people to analyse the transformations for possible defects. Then a meeting takes place to discuss the findings and once there is agreement regarding the defects then appropriate solutions are suggested. This technique is known to be an effective source code validation technique [38]. This is due to the fact that it can be applied to any kind of software development and project with any sort of specification format.

During the inspection process [143], general questions such as what, who, when and where should be asked in order to find any potential defects. The result of the inspection process must be approved by all the involved members during the process (inspectors). As the main objective of this technique is to find defects, inspectors must not only be independent from each other but also from the author of the requirements documentation in order to avoid any potential conflict and/or interest.

- What and Who: the inspection process should be precise and accurate about a particular subject and must be based on facts rather than opinions or predictions.

- When and Where: the inspection process should not take place at a too early stage of any given project, since potential errors may be discovered by the author of the requirements documentation himself or another person involved in the project at the early stages.
2.4. Requirements Engineering

The inspection process should take place after the author of the requirements documentation and others involved in the project have had a chance to check and verify the process. “Empirical evidence from software testing suggests that the more defects are found at a particular place, the more scrutiny is required at that place and the places impacting on it or impacted by it” [143]. For instance, safety and security related projects.

Depending on the type of transformation project, requirements can be more or less structured. Figure 2.12 presents the general structure of requirements inspection and review [143] that can be used in MT.

Model checking [9] is another increasingly used RE technique by which the properties of formally specified models can be verified. It is a technique to verify finite state concurrent systems such as model transformations. One of the main advantages of this technique is that it allows for automatic and systematic performance. The general idea behind this technique is to explore the models systematically in order to find any possible error by creating a counterexample that does not satisfy the required properties.

Spin [57] is a model checker by which the properties of a given system can be checked if properties are formalized in Linear Temporal Logic (LTL). It is a well-aligned approach with requirements engineering of model transformation in KAOS, which supports requirements elaboration using temporal logic. In model transformations, formalised requirements in temporal logic could then be checked for particular implementations using model-checking techniques, as in [121].
Validation and verification of a model transformation system involves checking whether or not the transformation in question behaves as it was initially designed to behave. Requirements of the transformation should satisfy the system requirements and there must not exist any incompleteness, conflict and inconsistency amongst the requirements. This can be achieved by using formal methods, such as model checking.

By using model checking, a formal checking process would go through the behavioural property (specifications) of the system (model) in order to verify it either by an exhaustive enumeration (explicit or symbolic) of all of the reachable states of the system or any internal behaviour that might result in the system’s transition between them.

A counterexample will be produced whenever the specification does not hold in all of the system (model) execution which should consist of a trace of the model from a start state to an error state in which the specification is violated, providing a very helpful tool for debugging the system design [126].

2.5 Model Driven Engineering

Model driven engineering [12] is an approach to software development in which the primary focus is on models rather than programs. Models can include various information such as functionality, time constraints, security, maintainability etc. The intention of MDE is to use models in a productive way that can be manipulated by programs. The productivity of a model is determined by how complete and formally it is defined. In the concept of MDE, metamodels are used to build the formal definition of the models. A metamodel describes the structure of a model that the model needs to follow in order to be valid. In general, it could be said that a metamodel is the prerequisite of the model transformation’s context [16]. MDE provides a framework which integrates software development activities along with metamodels and model transformations. It presumes models as primary entities in the software development cy-
2.5. Model Driven Engineering

cle. In order to make models to be entities of software development, they need to be formally defined and automatically manipulated by programs.

The increasing complexity and size of today’s software systems has resulted in the proliferation of many different kinds of development environment, to create these systems. As current technologies are mainly geared towards code-centric software development, it is not a trivial task to develop software using different environments. MDE [68] is a development methodology which allows developers to investigate software from the low-level implementation to the more abstract level. Models are central to the MDE software development process. One of the main aspects of MDE is applying operations on models automatically. Models encompass information of different phases of the development process. They are also capable of representing the system at different levels of view and abstraction [16]. It could be said that models are an abstract representation of the system-to-be. Models are defined by using different modelling languages according to specific syntax and semantic rules which allow the developers to analyse different properties of the system. To understand the advantages of using MDE in the development process, a clear description of its structure and the relationships between its components and different sections is required.

The first challenge one might face regarding the model-driven universe is that it contains a variety of different acronyms whose exclusivities might lead to confusion in different paradigms. To this end the fundamental components of MDE and their relations will be reviewed and a clear explanation regarding the main and the basic acronyms in the model-driven universe and their relationships will be presented.

MDE could be regarded as the superclass of other model-driven concepts as it consists of the main engineering process. An instance of MDE is Model Driven Development (MDD) [116] which is mainly focused on development activities. It is a paradigm which applies models as the primary artifact in the cycle of development. In general, MDD generates implementations from models in a (semi) automatical manner [22]. Model Driven Architecture (MDA) [16] is another element of the model-driven
2.5. Model Driven Engineering

universe which itself is a subset of MDD. MDA is a particular approach which was introduced by the Object Management Group (OMG) [132] where Platform Independent Models (PIM) are transformed to Platform Specific Models (PSM). In PIM, the model does not contain information about the platform used, whereas in PSM, the model does contain information about the platform used. Figure 2.13 shows the relationship between MDE (a field), MDD (a field overlapping MDE) and MDA (a framework) in the model-driven universe.

![Model-driven frameworks](image)

Figure 2.13. Model-driven frameworks, adapted from [22]

2.5.1 Model Driven Development in MDE

Software systems are simply much larger and do more complex things than ever before. One of the main reasons behind this complexity is the semantic gap between the problem domain and the solution domain. MDD aims to fulfil this gap by specifying the problem with a high abstraction level model and then transforming it into implementation of the actual software [128]. This would drive developers to shift their focus from low-level programming code to high-level models. This would enable the developers to focus more on solving the actual problem rather than focusing on the details of the implementation.

MDD is a software development approach in which models are considered to be the main elements throughout the development process.
This would allow the implementation to be generated in an automatic or semi-automatic way. In this approach, the main goal is to have automation as much as possible during the software development life cycle.

### 2.5.2 Model Driven Architecture in MDE

As mentioned earlier, MDA is the framework defined by the Object Management Group (OMG) as the realization of MDE. It provides developers with an architectural view of how the OMG perceives MDE should be done in different stages such as the analysis phase, the design and implementation phase.[67] Another advantage of the MDA framework is that it can be defined at different levels of abstraction. The levels of abstraction are as follows:[73]:

- **Computational Independent Model (CIM): analysis**
- **Platform Independent Model (PIM): high-level design**
- **Platform Specific Model (PSM): detailed design**

Computational Independent Model[73] provides a view of a system from a computation independent viewpoint. It does not contain any details of the system’s structure. It can be regarded as an informational concept of a model which describes the requirements and domains of the system from a high point of view by avoiding details and any computational implementation. In short, CIM can be regarded as the analysis level of MDA.

Platform Independent Model[22] describes the behaviour and structure of the application. It is a role where the model does not contain any information regarding the platform used. It provides the developer with a sufficient degree of independence to map to one or more concrete implementation platforms. In short, PIM can be regarded as the high-level design of MDA.
2.6. Model Transformation

Platform Specific Model [22] concerns specific platforms. Even if a model is not being executed itself, it must contain all the necessary information about the behaviour and structure of an application to a specific platform. In short, PSM can be regarded as the detailed design level of MDA.

MDA treats model transformation as its main artifact which provides an automated transformation among different types of representation. For instance, a CIM captures information regarding the requirements from the domain and by applying model transformation, it can be transformed to a PIM (Figure 2.14). This would make the model independent of any implementation while the model contains the complete specifications. Similarly, a model transformation can be executed to transform the PIM into PSM while the system is maintained with sufficient functionalities. Finally, an executable code can be produced by a model transformation on PSM.

Figure 2.14. Transformation between different representations of a model
2.6. Model Transformation

captured information in models is common to all these tasks, therefore it can be reused and avoids the process of creating the artifacts from scratch. Model transformation allows developers to use the information that was once captured as a model and build on it [29].

Model transformation consists of the words model and transformation. According to Stachowiak [15] a model must possess the following three features:

- **Mapping**: it should always be based on an origin
- **Reduction**: it should only represent a relevant subset of the original’s properties
- **Pragmatic**: it should be usable in place of the original for a particular given pre-defined purpose

Fowler [22] has classified models into three groups: models as sketches, models as blueprints and models as programs. When a model is referred to as a sketch, only a partial section of the actual system is specified by it. On the other hand, models as blueprints are used to provide a complete and detailed specification of the system. Finally models as programs are used instead of programs where models are directly used to develop the system.

Transformations are often used for:

- Restructuring and refactoring models
- Migrating models according to the metamodel
- Refining models from PIM to PSM

In general, it could be said that transformations are generally useful to translate the semantic content of a model from one language to that of another [29].

Model transformations perform a mapping between different models. They take the source models and transform them to the target models. Nevertheless, first it has to be declared what needs to be transformed.
2.6. Model Transformation

If the artifacts that need to be transformed are programs such as source code, byte code or machine code, then the term program transformation must be used. On the other hand, if the software artifacts are models, the term model transformation should be used [108]. Figure 2.15 illustrates the general architecture of the transformation paradigm.

Model transformation is one of the core elements in MDE in the development of a software. "Transformations are used to refine models from platform-independent forms to platform-specific, to migrate models in response to metamodel evolution, and generally to translate the semantic content of a model from one language to that of another" [75]. Moreover, transformations could be used as a means to restructure a model in order to increase its quality.

MDE aims to develop, maintain and evolve software by performing model transformations and relying on models as first-class entities. A large number of transformation tools and approaches have been defined across the MDE community. Transformations can be differentiated regarding their input (source) model, output (target) model, specification notation and style. From an engineering point of view, a model can be useful if it is an aid in deciding the appropriate series of actions that need to be taken to reach and maintain the system’s goal.

2.6.1 Transformation Types and Properties

Before developing any type of model transformation application, the developer needs to identify certain properties regarding the actual transformation. In this section, the different types of model transformation will
2.6. Model Transformation

be categorised according to their characteristic properties. This is useful as it allows developers to decide which model transformation language and engine is suitable for a specific type of transformation. We will give a brief explanation of some general properties of model transformation developments as presented in Table 2.6.

<table>
<thead>
<tr>
<th>Transformation properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>model-to-text, model-to-model, text-to-model</td>
</tr>
<tr>
<td>Number of models</td>
</tr>
<tr>
<td>one model, one-to-one, one-to-many, many-to-one, many-to-many</td>
</tr>
<tr>
<td>Change of abstraction</td>
</tr>
<tr>
<td>vertical, horizontal</td>
</tr>
<tr>
<td>Change of metamodel</td>
</tr>
<tr>
<td>endogenous, exogenous</td>
</tr>
<tr>
<td>Properties preservation</td>
</tr>
<tr>
<td>semantic, behaviour, syntax</td>
</tr>
<tr>
<td>Rule application control</td>
</tr>
<tr>
<td>implicit and explicit control, external control, rule application scoping</td>
</tr>
<tr>
<td>Rule scheduling</td>
</tr>
<tr>
<td>rule selection, rule iteration, phasing</td>
</tr>
<tr>
<td>Traceability</td>
</tr>
<tr>
<td>implicit, explicit</td>
</tr>
<tr>
<td>Directionality</td>
</tr>
<tr>
<td>unidirectionality, multidirectionality</td>
</tr>
</tbody>
</table>

• Type
Model transformation types can be categorised as follows:

– Model to text
Model to text transformations can be divided into two categories: model to actual text transformations and model to source code transformations, also called model to code transformation/code generation. In the case of a model to text transformation, it is necessary for the engineer to identify the
2.6. Model Transformation

type of text. If it is a source code, the programming language it should produce as well as appropriate notations need to be established. If the output is from a documentation generation type, then the structure and format of the documentation must be identified.

– Model to model

In model to model transformations, the general idea is to create elements of target models. Elements in the source model have to be mapped to elements in the target model. It is important to identify properties of the target model [16]. Depending on the requirements, extra restrictions may be added on the output model. This is due to the fact that all possible instances of the input metamodel are not always transformed automatically to the output and there is a need to add certain restrictions to them.

– Text to model

The main idea in a text to model transformation, also called reverse engineering or design recovery, is to create models from text. In order to do that, a parser must be used for the text. The type of parser can be identified through different methods such as interviewing the stakeholders or in some cases the stakeholders might prefer to use their own parser.

The supported target type is a property of model transformation which can be used to distinguish the model transformation according to its target model. The target model could be either in the form of a model or text. Model to model transformations create elements of the target model, then elements of the source model are mapped into the target model. In model to text transformations, instead of creating elements of the target model, the transformation creates arbitrary text, and the elements of the source model are mapped into fragments of text. This type of transformation is also called model to code transformation if the target’s text consists
2.6. Model Transformation

of program source code [108].

(For the purposes of this thesis, if not clarified otherwise, when talking about ‘model transformation’ we are referring to model to model transformations).

• Number of Models

The number of models in a transformation is another factor that needs to be established (one-to-one, one-to-many, many-to-one and many-to-many). In a transformation, there must be at least one model involved, and in such cases where only one model is involved, the model is the source model and the target model at the same time. The target model is created by transforming the existing elements of the source model according to the specifications. Through this identification, it can be assumed that for this kind of transformation, the source and target models are identical, meaning they have the same language.

Transformations often contain two models: a source model, $S$, and a target model, $T$. The target model is usually assumed to be empty and in case there is information, it will be overwritten by the transformation and it will only contain the required generated information. It can also be the case that the transformation contains several source models. In this case, the engineer has to identify whether or not the transformation performs mapping (weaving $S_1$, $S_2$ into $T_1$) or whether it performs updating (merging $S_1$, $T_1$ into $T_1$).

• Change of Abstraction

Change of abstraction in model transformation means that the amount of information that a model can contain could be varied. Not only the amount of detail could be varied (details unchanged or reduced), but also it is possible to introduce new details. Another factor that needs to be identified in a transformation is its change of abstraction, whether it is vertical or horizontal. In a ver-
2.6. Model Transformation

tical transformation, the level of abstraction is modified whereas in a horizontal transformation, the abstraction level remains unchanged while the representation of the model is modified.

- **Change of Metamodel**
  Change of metamodels is based on whether the transformation is *endogenous* or *exogenous*. Endogenous refers to those types of transformations which only operate on a single metamodel to express the models (the metamodel of the source and target models are the same). The term exogenous refers to the transformations that are expressed using different metamodels (the metamodel of the source and target models are different).

Transformations can be either endogenous or exogenous. If it is an endogenous transformation, then there is no need to use two different metamodels and the engineer only needs to find out what the required transformation language is for both source and target models. On the other hand, if the transformation is exogenous, then metamodels of the source and target models need to be identified [16].

- **Properties Preservation**
  Preservation of properties in model transformation means that every transformation preserves certain aspects in the target model from the source model. Source and target models could have common properties depending on the transformation type. In general, there are three types of preservation in model transformation: *semantic preservation*, *behavioural preservation* and *syntactic preservation* [8].

Semantic preserving transformations refer to those kinds of transformations where the structure of the overall computation is changed without incurring any changes in the computed values. In these types of transformation, the source metamodel and the metamodel of the target are similar as is their mapping regarding semantic
2.6. Model Transformation

preservation; this means that the meaning of both models are similar while they are represented by different abstract syntax [L3].

A transformation is called behaviour preserving, if the target model fulfils the behaviour constraints in the source model. For instance, a model to text transformation can be regarded as behaviour preserving if the output code (text) produces values that slightly differ from the predicted values by the corresponding simulation model [L6], even though the source and target models are not semantic preserving.

Syntax preserving transformations are those in which the abstract syntax of the model remains unchanged. For instance, improving the graphical layout of a model is an example of syntax preserving transformation, where the abstract syntax of the transformation is preserved while the concrete syntax is changed by replacing old graphical elements with new ones [L6].

The target model obeys all the same constraints as the source model. The idea is that the constraints are what is being preserved, rather than the whole model.

• Rule Application Control and Scheduling

Every transformation language provides a different mechanism regarding when and where transformation rules should be applied, therefore it is necessary for the engineer to find out what the required characteristics of the transformation language are. Different aspects need to be considered such as: implicit control by which the order of rule application is not defined explicitly; explicit control by which the rule application and the transformations execution order is specified. Rule application scope means that “the transformation affects only parts of the model. The restriction can be either on the source model or on the target model” [L6]. Moreover, the order of rule application is determined by rule scheduling: Rule selection controls when a rule is applied. Rule iteration uses recursion, loop-
2.6. Model Transformation

Phasing determines that in a certain phase only certain rules can be executed. **Phasing**

- **Traceability**

  Traceability might be required to be performed by the transformation in order to perform different analyses such as how changing a model could affect other related models.

- **Directionality**

  The engineer needs to determine whether the transformation is unidirectional (where mapping is just from source to target model) or bidirectional (where mapping is from source to target model and from target model to source model).

2.6.2 Model Transformation Languages

“A model transformation language is a vocabulary and a grammar with well-defined semantics for performing model transformations” [16]. There are a large number of model transformation languages each with a different nature and structure intended for a specific kind of transformation task. Depending on the transformation task some MT languages are better equipped to handle the general characteristic of a software (i.e. complexity, accuracy, fault tolerance, execution time, modularity, etc.). There are different language paradigms that model transformation languages can follow. The main paradigms of model transformation languages are: **imperative, declarative and hybrid**. Imperative languages are mainly concerned with how the transformation should be executed, whereas declarative languages mainly focus on what needs to be transformed [117]. Hybrid transformation languages offer both imperative and declarative languages according to the user’s choice of language.

We have selected four MT languages to review, compare and analyse, namely: UML-Rigorous Systems Design Support (UML-RSDS), ATLAS (ATL), Epsilon Transformation Language (ETL), Query/View/Transformation (QVT).
2.6. Model Transformation

UML-Rigorous Systems Design Support (UML-RSDS)

UML-RSDS is a model transformation tool which is able to manufacture software systems in an automated manner. It is a tool which is designed for Model Driven Development and it supports by Java, C++, C# and JSP/servlets. It presents a high-level Unified Modelling Language (UML) specifications and uses standards in UML2 and OCL2. One of the main advantages of UML-RSDS is that in order to facilitate its use for the user, it uses simplified OCL notations. “The use of two-valued logic and having all collections as either sets or sequences would facilitate the verification task in cases of high-integrity systems requiring a high degree of assurance” [87]. It uses a declarative approach which is suitable for abstract transformations and could be used at the early stages of software development such as the requirement phase. UML-RSDS which uses use cases as the main behavioural description of systems, does not require implementation platform modelling, and the transformation process has a sequential execution model. The functionality of use cases is in turn defined using the data and operations of classes in the class diagram. State machines can be used in UML-RSDS to model the intended life histories of objects, and detailed behaviour of operations, but are optional. UML-RSDS has a high level of abstraction (Figure 2.16).

![Figure 2.16. General process of UML-RSDS](image-url)
2.6. Model Transformation

ATLAS (ATL)

ATL is another well-known model transformation tool which is used for model to model transformations. ATL transforms the source model into the target model according to the transformation definition. It is a hybrid transformation and has become a popular language tool due to its declarative and imperative aspects. Being declarative allows ATL to hide the details related to traceability, source elements, rule triggering and etc. It would result in having complex transformation algorithms underlying a simple syntax. However, it may not be possible to have a complete declarative solution for any given transformation. In that case ATL also allows developers to use imperative features [64].

ATL also uses Object Constraint Language (OCL) and is similar to the QVT standard provided by Object Management Group. It is a domain-specific language (DSL) [141]. ATL uses a specific approach and therefore instead of focusing on a general solution which may be suboptimal, it focuses on a specific solution for a specific set of problems [64]. Figure 2.17 presents the overall overview of ATL transformation approach.

In ATL a source model $S_a$ is transformed into a target model $T_a$ according to the transformation definition $mna2mmb.atl$ which is defined in the ATL language. “The transformation definition is a model conforming to the ATL metamodel. All metamodels conform to the Meta Object Facility (MOF)” [61].

![Figure 2.17. Overview of the ATL transformational approach](image)
2.6. Model Transformation

Epsilon Transformation Language (ETL)

ETL is also a hybrid model to model transformation language that is built as a component atop of ‘Epsilon Eclipse’ \[77\]. This would enable ETL to be consistent semantically and syntactically with other languages which also are being built atop of Eclipse \[76\]. Within ETL, the Epsilon Object Language (EOL) provides a set of reusable model management tasks. EOL uses the OCL mechanism by supporting other languages at the same time. ETL provides features such as: multiple model access, statement sequencing and model modification capabilities. “ETL needs to be able to capture and execute specifications of transformation scenarios that involve an arbitrary number of input and output models of different modelling languages and technologies at a high level of abstraction” \[78\].

The overall structure of ETL consists of having one or more sets of modules that includes a number of rules and operations. “Rules are declared with their name, one source, and one or more target elements. The rules can be independent or be an extension of other transformation rules. It is possible to assign applicability of the rules to the particular elements in the source model by defining a guard. The guard can be specified optionally by using EOL expression or a block of EOL statements” \[74\].

Query View Transformation (QVT)

QVT \[115\] can be used either as an imperative language or declarative language defined by OMG. The declarative part of QVT consists of a two-level architecture which embodies the same semantics at two different levels of abstraction: \textit{User-friendly Relations} and \textit{Core language}. User-friendly relation level is responsible for pattern matching of complex objects and creating a template of objects. Both unidirectional and bidirectional types of transformation can be written in QVT. The imperative nature of QVT consists of an \textit{Operational} part which is designed to write unidirectional transformations. \textit{Black box} is another component of QVT.
2.6. Model Transformation

language which acts as a mechanism to invoke facilities of a transforma-
tion to be expressed in other transformation languages. It can be said
that QVT is an architectural basis, and that individual vendors have to
implement it. Figure 2.18 presents an overall architecture of QVT.

![QVT architecture](image)

**Figure 2.18.** QVT architecture [115]

<table>
<thead>
<tr>
<th>TABLE 2.7. Comparison of transformation languages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>UML-RSDS</td>
</tr>
<tr>
<td>ATL</td>
</tr>
<tr>
<td>QVT</td>
</tr>
<tr>
<td>ETL</td>
</tr>
</tbody>
</table>

**Comparison of Transformation Languages**

In this section, we are going to compare the functionality of model trans-
formation languages. ATL has a restricted form of update-in-place pro-
cessing, called *refining mode*. This makes a copy of the source model
and then updates this copy based on the (read-only) source model. Thus
the effects of updates cannot affect subsequent rule applications. QVT
also adopts this approach, but repeatedly applies the copy and update
process until no further changes occur. In contrast, UML-RSDS and
ETL directly apply updates to the source model. Bidirectionality in
UML-RSDS is partly supported by the synthesis of inverse transforma-
tions from mapping transformations. QVT provides the capability to
2.6. Model Transformation

apply a transformation in different directions between the domains (parameters) of the transformation rules. However, as with UML-RSDS, this capability is essentially limited to bijective mapping transformations [135]. QVT additionally supports change-propagation, by deleting, creating and modifying target model objects when an incremental change to the source model takes place. This also applies only to mapping transformations and not to update-in-place transformations. Table 2.7 illustrates a comparison of the four selected MT languages.

2.6.3 Model Transformation Examples

In this section, we have chosen to review three types of model transformation: refactoring, migration and refinement.

Refactoring

The general idea behind refactoring is to improve the structure of the model to make it easier to understand, and to make it more maintainable and amenable to change. According to Fowler, refactoring could be defined as “changing a software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure” [37].

We will go through a case study [91], an example categorized under refactoring/restructuring transformations. It is an example of an update-in-place model transformation, which carries out a refactoring of a class diagram to improve its quality. The aim of the transformation is to remove situations of apparently duplicated attributes in different classes from the diagram. For example, if all subclasses (more than one) of a given class have an attribute with identical name and type, then these copies can be replaced by a single attribute in the superclass. Figure 2.19 is a representation of the metamodel for the source and target transformation language where the metamodel is represented by UML 2.0 class diagram language.
2.6. Model Transformation

The initial assumption for models in this case is:

- Class name uniqueness
- Type name uniqueness
- Property name uniqueness in classes
- Single inheritance

According to the assumptions, two classes with the same name, two types with the same name, two attributes of a class with a distinct name with another attribute of its own class and with the attributes of any superclass must not exist. Moreover, there must not be any multiple inheritance, i.e., the multiplicity of generalisation is restricted to 0:1. Not only must these assumptions be preserved, but also the following properties of the transformation itself must hold.

- Moving up common attributes of all direct subclasses to their superclass: If the set $g = c.specialisation.specifc$ of all the direct subclasses of a particular class $c$ has two or more elements, and all classes in $g$ have an owned attribute with the same name $n$ and type $t$, then add an attribute of the same name and type to $c$, and remove the copies from each element of $g$ (Figure 2.20).
2.6. Model Transformation

- A new subclass created for duplicated attributes: If there is a class called \( c \) that has two or more direct subclasses \( g, g = c.\text{specialisation } .\text{specific} \), and there is a “subset \( g_1 \) of \( g \), of size at least 2, all the elements of \( g_1 \) have an owned attribute with the same name \( n \) and type \( t \), but there are elements of \( g - g_1 \) without such an attribute that introduce a new class \( c_1 \) as a subclass of \( c \). \( c_1 \) should also be set as a direct superclass of all those classes in \( g \) which own a copy of the cloned attribute. In order to minimise the number of new classes introduced, the largest set of subclasses of \( c \) which all contain a copy of the same attribute should be chosen. Add an attribute of name \( n \) and type \( t \) to \( c_1 \) and remove the copies from each of its direct subclasses” [75] (Figure 2.21).

- A root class created for duplicated attributes: If there are two root classes or more, “then all of which have an owned attribute with the same name \( n \) and type \( t \), create a new root class \( c \). Make \( c \) the direct superclass of all root classes with such an attribute, and
2.6. Model Transformation

add an attribute of name $n$ and type $t$ to $c$ and remove the copies from each of the direct subclasses” [75] (Figure 2.22).

Figure 2.22. Rule 3 [75]

The refactoring example operates on UML class diagrams to remove cases where all subclasses of a given class have an attribute with a common name and type. The requirements of the case study are described in more detail in [75], but here we suffice to briefly list the functional and non-functional requirements:

- **Functional requirements:** the assumptions are precisely defined; three intended transformation steps (refactoring requirements) are described in text and concrete syntax diagrams.

- **Non-functional requirements:** invariance of the assumptions is specified. Effectiveness is specified in terms of minimising the number of new classes created.

Missing are requirements for model-level semantic preservation for model quality improvement, reliability, extensibility, efficiency and comprehensibility.

The above MT example (refactoring) was a case study proposed by Transformation Tool Contest (TTC), where participants submitted their solutions by using their preferred transformation tool and language. The outcome of the submitted solutions for this case study were that:

- None of the five published solutions in [75] provide proof of model-level semantic preservation. Some solutions do not achieve such
2.6. Model Transformation

preservation, because they use a different set of rules to those indicated in the functional requirements. This was due to the fact that requirements were not stated clearly and explicitly which resulted in requirements’ ambiguity and misinterpretation amongst the participants.

- None of the solutions in [75] achieve more than a neutral measure of usability or extensibility.

- Only two solutions in [75] have a practical efficiency in processing large models.

Migration

Model migration transformation is part of model transformation which itself is a key element of model-driven software development. In model migration, instances of models are updated in order to conform to the evolved metamodel. For this type of transformation, a case study, TTC 2010 [128], has been chosen which involves the transformation of models of the UML 1.4 activity diagram language [137] into models of the UML 2.2 activity diagram language [139].

The Unified Modeling Language (UML) has changed in a number of ways from version 1.4 to version 2.2. For instance, in UML 1.4 model elements, which defines what or who is responsible for a particular role, are represented as partitions, while in UML 2.2 it is represented as activity partitions. In general the most important changes between these two UML versions are activity diagrams. The structure of activity diagrams has changed between UML versions of 1.4 and 2.2. Activity diagrams are used in UML to model low-level behaviours by emphasising sequencing and co-ordination conditions. An activity diagram consists of three elements:

- Activities (as rounded rectangles)
- Transitions (as directed arrows)
2.6. Model Transformation

- Decisions (as diamonds)

On an abstract level, Table 2.8 provides information regarding the differences between UML 1.4 and UML 2.2 [6].

**TABLE 2.8. Model elements in UML [6]**

<table>
<thead>
<tr>
<th>UML 1.4 element names</th>
<th>UML 2.2 element names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association end</td>
<td>Member end and property</td>
</tr>
<tr>
<td>Object (when used in activity diagrams)</td>
<td>Object node</td>
</tr>
<tr>
<td>Object (when used in sequence diagrams)</td>
<td>Lifeline</td>
</tr>
<tr>
<td>Collaboration diagrams</td>
<td>Communication diagrams</td>
</tr>
<tr>
<td>Swimlane (or partition)</td>
<td>Activity partition</td>
</tr>
<tr>
<td>Activity</td>
<td>Structured activity node</td>
</tr>
<tr>
<td>Decision</td>
<td>Decision node or merge node</td>
</tr>
<tr>
<td>Final state or end state</td>
<td>Final activity node</td>
</tr>
<tr>
<td>Initial state or start state</td>
<td>Initial node</td>
</tr>
<tr>
<td>Object instance (in activity diagrams)</td>
<td>Central buffer node</td>
</tr>
<tr>
<td>State</td>
<td>Structured activity node</td>
</tr>
<tr>
<td>State machine</td>
<td>Structured activity node</td>
</tr>
<tr>
<td>Synchronization bar (synch bar)</td>
<td>Fork node or join node</td>
</tr>
<tr>
<td>Transition (on an activity diagram)</td>
<td>Control flow</td>
</tr>
<tr>
<td>Transition condition (guard condition)</td>
<td>Control flow guard</td>
</tr>
<tr>
<td>Formal argument</td>
<td>Template parameter substitution</td>
</tr>
<tr>
<td>Formal arguments (collection of formal arguments)</td>
<td>Template binding</td>
</tr>
<tr>
<td>Three-tiered diagrams</td>
<td>Class diagrams</td>
</tr>
<tr>
<td>Class instance</td>
<td>Lifeline</td>
</tr>
<tr>
<td>Self-link</td>
<td>Message pathway</td>
</tr>
<tr>
<td>Connection relationship</td>
<td>Communication path</td>
</tr>
<tr>
<td>Process (in a deployment diagram)</td>
<td>Artifact</td>
</tr>
<tr>
<td>Processor</td>
<td>Execution environment</td>
</tr>
<tr>
<td>Destruction marker</td>
<td>Stop node</td>
</tr>
<tr>
<td>Focus of control</td>
<td>Execution occurrence</td>
</tr>
<tr>
<td>Action</td>
<td>UML activity</td>
</tr>
<tr>
<td>State diagram</td>
<td>Statechart diagram</td>
</tr>
</tbody>
</table>
2.6. Model Transformation

The structure and rules of a model are defined by its metamodel, therefore if a metamodel evolves, instances of the models might no longer conform to the metamodel. In that case, instances cannot be manipulated by the editor and managed with model management operations and may not even be loaded with the modelling tools. The aim of this case study is to explore the ways in which model transformation languages can be used to update models in response to metamodel changes.

The model migration example concerns the migration of UML 1.4 activity diagram models to UML 2.2 activity diagrams [123]. The requirements consist of:

- Functional requirements: the transformation should successfully migrate one example activity diagram which includes all of the core elements of UML 1.4 activity diagrams.
- Non-functional requirements: size and comprehensibility of the specification should be optimised.

It can be seen that many of the categories of requirements are missing for this case study, in particular: what assumptions can be made upon the input model, model-level semantic preservation, confluence, reliability, time performance. The required functionality is only indicated by one examplar case of a source model and its intended target.

The poor quality of the published solutions [123] may result in part from these incomplete requirements:

- The highest score for correctness for the 9 solutions was 5.5 out of 10.
- No solution provided a proof of model-level semantic preservation. The proposed migration strategy would lose semantic information (the action performed in action states).
- The issue that some valid UML 1.4 activity diagrams are not valid as UML 2.2 activities (when directly translated according to the example given) was not addressed by any solution [89].
2.6. Model Transformation

All of these aspects would hinder the practical use of transformations for this migration problem.

Refinement

In general, refinement \[23\] can be interpreted as replacing some artefact \(S\) safely by a refinement \(R\) in a way that properties of \(S\) are preserved. Refinement transformations on UML specifications can be used to improve the structure of a model, or to progress the model towards implementation \[83\]. For instance, transforming a UML PIM to a relational database PSM is a type of refinement transformation where some constructs should be removed. The following is some of the main structures that need to be removed in order to transform a UML class diagram to a relational database:

- Many-to-many associations removal: In relational databases, explicit many-to-many associations cannot be constructed by using foreign keys. All many-to-many associations must be replaced by a new class which has two many-to-one associations. The new class must associate only those objects that were connected by the original association, and must not duplicate such links: \(c1 : C \& c2 : C \& |c1.ar \& ca.br = c2.br \Rightarrow c1 = c2 \) \[83\]. For instance, \(a : A\) and \(b : B\) are associated by \(A_B\), and there is an \(AB\) object, \(x\), such that \(x.ar'' = a\) and \(x.br'' = b\), and vice-versa. Thus the original \(a.br\) set is \(a.xr.br''\) in the new model (Figure 2.23) \[84\].

- Inheritance-association replacement: During the refinement of a PIM to a PSM, inheritance should be removed as the PSM platform does not support inheritance. Therefore, all inheritance relationships between two classes must be replaced by associations. “For instance, inheritance of \(A\) by \(B\) must be replaced by a direct reference from \(B\) to \(A\). Features \(f\) of \(A\) used in \(B\) must be referred to as \(ar.f\) in the new model” (Figure 2.24) \[84\].
2.6. Model Transformation

- Introducing superclass: This construct can be applied if two classes have common features. “A new superclass is usually abstract, because only A and B instances existed in the original model. Common features are moved to superclass, common operations become abstract in superclass” (Figure 2.25) [84].
2.7. Summary

2.7 Summary

In this chapter we investigated and analysed the background and related literature of the requirements engineering concept as well as model transformation. It provides a detailed overview of the principles of requirements engineering and requirements engineering process. It describes what MDE is and provides an explanation regarding MDE’s branches such as MDD, MDA and MT. Moreover, transformation types, properties, and languages are presented.

We have given a detailed explanation about the most important and elementary concepts of each term through examples and case studies. To conclude, this chapter provides a substantial amount of information regarding RE and MT and their current applications.
Chapter 3

Requirements Engineering in MT Development

In order to have a better understanding of model transformation development in the industrial world and the role of RE, we decided to carry out an interview-based study. This chapter is the result of an exploratory interview study based on industrial model transformation projects involving seven industry experts in model transformation. We discuss the types of projects often seen in model transformation development, their embedding in the context of other projects and organisations, the roles of stakeholders, and the requirements engineering techniques employed in practice, and we consider future research directions. The aim of this study was to explore transformation projects from a requirements engineering perspective. Specifically, we are interested in finding out what requirements engineering techniques, if any, are applied in model transformation development.

3.1 Introduction

The size and complexity of model transformations grow as they face more wide-spread use in industry. As a result, systematic approaches to the development of high-quality and highly reliable model transformations
3.1. Introduction

become increasingly important. However, because little is known about the context in which model transformations are developed, it is very difficult to know what would be required from such systematic approaches. This section provides some initial results and analysis of an interview-based study of requirements engineering in MT developments. We have interviewed industry experts in MT development, with the goal of understanding the contexts and ways in which transformations are developed and how their requirements are established. The types of stakeholders of transformations were identified, as well as their role in the transformation development. We also discovered a possible differentiation amongst the development of model transformation projects and general software development projects.

Model transformations are central to model driven engineering [131]. They can be used for a range of purposes, including to improve the quality of models, to refactor models, to migrate or translate models from one representation to another, and to generate code or other artifacts from models [108]. Model transformations are used to transform one model into another, generate text (such as code) from a model or to do reverse engineering (i.e. extracting models from text/code). In any case, they aim to automate repetitive development tasks, ensuring different situations are treated in a generalised manner.

As MDE is being used more intensively [59], systematic development of the transformations becomes more important [46]. However, as Selic argues [129]: “we are far from making the writing of model transformations an established and repeatable technical task”. The software engineering of model transformations has only recently been considered in a systematic way, and most of this work has focussed on design and verification rather than on requirements engineering.

We are interested in understanding what requirements engineering for model transformation development should look like. To this end, we need to understand the context in which model transformations are typically developed and what, if any, requirements-engineering techniques are already applied. This will help us understand how existing RE techniques
3.2. Methodology

might be applied (or may have to be adapted) for the context of MT
development.

The remainder of this chapter is structured as follows: After a brief
discussion of our methodology in Section 3.2 and related work in Sec-
tion 3.2.1, we present some of our findings from the interviews. We
begin with a discussion of the types of projects in Section 3.3, followed
by a discussion of stakeholders in Section 3.4. Section 3.5 discusses the
requirements engineering techniques identified by our participants, fol-
lowed by a brief analysis of project outcomes in Section 3.6.

3.2 Methodology

We identified seven participants that are experts in the MT development
field and have industrial experience. The selection was based on partic-
ipants’ experience and the work that they have done. Our participants
have between eight to twenty years of experience in MT development.
We asked participants to focus their responses on self-selected recent
projects. All participants had a leading role in these projects. Participants
were interviewed regarding the project(s) in which they were in-
volved (ten projects in total), and their views regarding the requirements
engineering process in relation to these projects.

We conducted semi-structured interviews of approximately one hour
duration. The same questions in the same order were given to all par-
ticipants. The questions concerned the project context and scale, the
stakeholders, the requirements engineering techniques and process used,
and the project outcomes.

Our approach is thus qualitative, investigating in depth the ‘why’
and ‘how’ of decision making for particular requirements engineering
techniques and activities in model-transformation development. More
information about the interview prompts can be found in the Appendix
B.

Threats to the validity of conclusions drawn from the interviews in-
3.2. Methodology

clude: (i) that the interviewees and examined cases are not representative of transformation developers and projects, (ii) that interviewees selected unrepresentative projects, (iii) that interview questions were aimed at eliciting a particular response.

We tried to avoid potential problems (i) by requesting interviews with a wide range of MT experts. The candidates for interviews were selected from our previous literature surveys of RE in MT. 15 candidates were approached, of whom seven agreed to be interviewed. These represent a diverse range of organisations, and the projects cover a range of domains: embedded systems, finance, re-engineering, defence and business. Regarding (ii), projects with poor outcomes, such as 3 and 6, were included in addition to successful projects. Regarding (iii), the questionnaire and methodology was examined by an expert committee for ethical approval.

3.2.1 Related Work

There has been very limited empirical research into model-transformation development. The only relevant studies have been based on MDE in general, such as that of [59, 147], which used interviews as well as a questionnaire-based survey. The main aim of this study was to capture the success and failure factors for MDE based on industry evidence. They conducted 22 interviews with MDE practitioners. The survey found that some use of MDE is made in a wide range of companies and industry sectors, however this use tended to be based on Domain-Specific Languages (DSLs) and modelling of narrow specialised domains. Transformations were used to generate artefacts from the DSL models, however code generation was not itself a primary benefit of MDE, instead the benefits came from the ability to abstract system architectures and concepts into models. The evidence from this survey suggests that transformations are often developed based on the expert knowledge of software developers, to encode and automate previously manual procedures.

A high degree of domain knowledge appears essential for the successful construction of the transformations. The survey of [110] considered in
3.3. Transformation Development Projects

depth four companies adopting MDE, but did not specifically consider requirements engineering. One concern of the companies in [110] was the cost of developing transformations, a factor which could be improved by more systematic RE for MT.

The transML methodology defines an outline RE approach and methods for the RE of MT [46]. They adopt SysML and scenarios to analyse and document requirements.

In our work, we focus specifically on model transformation developments, whether as part of an MDE process or as an independent development. For MT developments, we examine how RE techniques and the RE process is carried out.

3.3 Transformation Development Projects

In this section, we will describe the MT projects which our participants focused on in their descriptions. All of our interviewees are either the sole developers or the lead developers for these projects. Each project has been categorised according to the MT field that it belongs to. The scale, developer’s time and effort for some of these projects will also be described.

Ten MT development projects were considered in this study:

1. **Automated generation of documentation for international standards.** This transformation concerns the generation of standard documentation text from metamodels, to ensure consistency of the documentation. The source metamodels are of the order of 600 meta-classes. The development effort was not available.

2. **Reverse-engineering and re-engineering of banking systems and web-services.** The idea of this project was to build transformations to construct models of existing applications, and to forward-engineer these models to new platforms. The scale of the finance system re-engineering is approximately three million
3.3. Transformation Development Projects

LOC extracted from 100 million LOC legacy code, the scale of the web services re-engineering is approx 15 million LOC. The re-engineering process must be done in a way that not only reveals the actual functionality of the system, but also enables further analysis according to system requirements. The development effort was not available.

3. **Code-generation of embedded software from DSLs.** In this project transformations are defined to map between embedded system DSLs forming C extensions, and from these DSLs to C code. These extensions are used by embedded software developers. More than 25 different DSLs are involved, and approx 30 person-years of effort.

4. **Petri-net to statechart mapping.** This model transformation maps Petri-net models to statecharts, in order to analyse the Petri-nets. It involves both refactoring and migration aspects. The transformation is intended to map large-scale models with thousands of elements. Effort was three person-months.

5. **Big Data analysis of IMDb.** The Internet Movie Database (www.imdb.com) can be regarded as a Big Data case. It has information about the title of movies, names of actors, rating of movies and actors playing roles in which movies. In this case, a model transformation was developed to implement IMDb searches by users. Effort was 3 person-months.

6. **UML to C++ code generator.** This case involved the construction of a transformation for the generation of multi-threaded/multi-processor code from UML. The transformation generates C++ code as well as providing a run-time layer to support the generator. Effort was four person-years.

7. **Reverse-engineering of a code generator.** This MT project was an example of re-engineering of an existing transformation.
3.3. Transformation Development Projects

In this case study an existing code-generation transformation was analysed and re-engineered to improve its functionality. Effort was four person-months.

8. Automation of railway network engineering. This case involved using models and transformations to support railway network design. This is a safety-critical case, with an approximate value of £150,000 per year. The metamodels operated on have about 200 classes.

9. MDE Platform. A generative software platform based on transformations and DSLs. This ongoing project consumes up to 300 person days per year.

10. SOA for insurance. Generation of middleware from DSLs, for service integration. The effort was approximately 20 person years.

3.3.1 Types of Project

Here is a recap of the types of software development projects [143] as mentioned in Chapter 2:

• Greenfield vs Brownfield
  In a Greenfield type of project, the system is completely new, therefore the developers have to start from scratch and build the system from the beginning. On the other hand, in Brownfield projects, a system already exists but it has to be further developed and improved.

• Customer vs Market Driven
  Software could be either a solution for a particular type of client in the market (customer driven) or a solution which would cover the need of a large percentage of the market (market driven). In customer-driven types of projects, the software is designed according to the needs of a specific type of client, whereas in market-driven
projects, a larger scope of solution is considered covering more than just one particular type of client.

- **In-House vs Outsourced**
  A project could be regarded either as an in-house project where it is assigned to a particular organization in order to carry out all the project’s life-cycle processes or it could be outsourced where it is assigned to different companies according to different project phases. In an in-house type of project, one team/company will carry out all the phases in the project, whereas in an outsourced project, usually once the requirements have been identified different teams from different companies will carry out the different phases such as design, implementation, testing, etc.

- **Single Product vs Product Line**
  The outcome of a project could have only one version which would satisfy the customer’s need or it could have different versions each of which would cover particular needs in a large organisation. “In a single-product project, a single product version is developed for the target customer(s). In a product-line project, a product family is developed to cover multiple variants” [143].

According to our interviews, three out of the ten projects, (3), (7) and (9), can be regarded as Brownfield projects mainly due to the fact that these projects were either extending, adapting or re-engineering existing transformations. The customers already had an existing transformation and required to improve or extend it. Seven projects were Greenfield as the transformation had to be done from scratch because either the project was completely new, or because developers wanted to use their own tools and technology.

All projects except for (9) were customer-driven as they were specified for particular client(s). All the projects were in-house, single-product projects. Each project was assigned to a different, single company to do all the transformations, therefore there was no need of outsourcing,
3.4. Stakeholders in MT

and only a single version of the project was developed. Eight of the ten projects were industrial, and two were academic. Table 3.1 summarises this classification of the MT projects in our study.

<table>
<thead>
<tr>
<th>Case</th>
<th>Brownfield</th>
<th>Greenfield</th>
<th>Customer-driven</th>
<th>Market-driven</th>
<th>In-house</th>
<th>Outsourced</th>
<th>Single-product</th>
<th>Product-line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Project 10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

MT development often occurs within a wider software development project, although there are also cases where MT development is the main part of software development (e.g. Project 1).

As a result, we have found it useful to differentiate explicitly between properties of the transformation-development project and the project this was embedded in. For example, while most of containing projects were Brownfield projects, most of the transformation-development projects were Greenfield as no previous transformation existed for the specific required purpose.

3.4 Stakeholders in MT

A stakeholder can be defined as an individual or an organisation or group of people who is either affected by or has an effect on the outcome of a given project [122]. Stakeholders are categorized into three different types: Operational, Containing Business and Wider Environment (Figure 3.1).
3.4. Stakeholders in MT

Figure 3.1. Onion model of stakeholder general relationships [3]

The Operational stakeholders have a direct interaction with the system. They consist of normal operators, people who will eventually operate and use the developed product, and maintenance operators, people from which the maintainability requirements can be discovered.

Stakeholders in the Containing Business area are those who somehow benefit from the system and consist of the Sponsor, the Customer and the Functional Beneficiaries. More specifically, sponsors are stakeholders that have the responsibility to pay for the developed product. The customer buys the product and sometimes it can be the case where the customer is also the end user of the developed product.

The Wider Environment area contains stakeholders which have an effect on or interest in the system, but only an in-direct influence. For example, Subject Matter Experts could consist of “internal and external consultants, may include domain analysts, business consultants, business analysts, or anyone else who has some specialized knowledge of the business subject” [122].

The Core Development Team consists of developers that are in charge of developing the product.

We have adapted the onion model to classify the stakeholders in MT development based on our participants’ descriptions and have identified the following for all of the MT projects (Figure 3.2):
3.4. Stakeholders in MT

- The Core Development team consisted solely of transformation developers.

- The Customers consisted of the committee that was interacting with the transformation developers in order to explain the problem space and what was needed.

- The Sponsors were the companies which were represented by the customers, and did not interact with MT developers directly.

- The Normal and Maintenance Operators consisted of the people who were going to use the result of the transformations as end users.

Table 3.2 presents the Sponsors, Customers and the Operators of the MT projects.

<table>
<thead>
<tr>
<th>Case</th>
<th>Sponsor and Customer</th>
<th>Normal and Maintenance Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Technology standards consortium</td>
<td>Users of the standards</td>
</tr>
<tr>
<td>2</td>
<td>Financial/Telecom organisations</td>
<td>Users of re-engineered systems</td>
</tr>
<tr>
<td>3</td>
<td>Commercial companies</td>
<td>Embedded software developers</td>
</tr>
<tr>
<td>4</td>
<td>External customer</td>
<td>Users of the produced models</td>
</tr>
<tr>
<td>5</td>
<td>External customer</td>
<td>Users searching the data</td>
</tr>
<tr>
<td>6</td>
<td>Government &amp; defence industries</td>
<td>Users of C++ applications</td>
</tr>
<tr>
<td>7</td>
<td>Commercial client</td>
<td>Users of the code generator</td>
</tr>
<tr>
<td>8</td>
<td>External customer; parent company</td>
<td>Railway engineers and operators</td>
</tr>
<tr>
<td>9</td>
<td>Own company; MDE users</td>
<td>Consultants, toolkit customers</td>
</tr>
<tr>
<td>10</td>
<td>Insurance company</td>
<td>IT team of company</td>
</tr>
</tbody>
</table>

As mentioned earlier, the MT projects that we analysed are typically embedded within wider projects. As a result, the role of stakeholders of the wider project changed according to the embedded MT project. For
3.4. Stakeholders in MT

example, in one case (Project 2) the members of the core development team of the wider project turned into the customers interacting with transformation developers for technical issues. Therefore, the transformation developers were facing two types of customers for this project: one to explain the general requirements of the overall system and one to deal with more detailed requirements and technical difficulties of the transformation. The impact of other stakeholders of the containing project (i.e. from the Containing Business or Wider Environment) on the transformation development has become more indirect. Understanding fully the role of these stakeholders in the context of transformation development seems important for successfully developing requirements engineering techniques for MT development. For example, the indirect nature of contact with the stakeholders of the enclosing development project is likely to have an impact on the use of RE techniques that require stakeholder interaction. Figure 3.2 is a first attempt at showing some of the relationships amongst the MT developers and general stakeholders in a generalised onion model.

![Figure 3.2. Adapted onion model of MT stakeholder relationships](image-url)

88
3.5. Requirements Engineering Process

3.5 Requirements Engineering Process

In this section, we will present our findings regarding the requirements engineering process applied in the ten examined projects. We start by showing the overall RE process used, before focusing on requirements elicitation and cataloguing typical RE techniques employed.

3.5.1 Overall RE Process

Requirements engineering is specialized and unique for any type of software development, similarly model transformation is no exception and it is specialized and unique in its own right. There are some key aspects which cause this uniqueness, as listed below:

- **Type of system**
  Critical systems need a complete and consistent set of requirements that can be analysed in advance. For business systems, work can start with an outline of the requirements that are then refined during development.

- **Type of development process**
  Plan-based processes require all requirements to be available at the start of the project, whereas in an agile approach, requirements are developed incrementally.

- **Type of environment**
  In some cases, users and other stakeholders are available to provide information about the requirements; in others they are not. These require different approaches for RE in their starting point for implementation.

- **Reusability extent**
  The extent to which other systems are reused in a system that is being developed, needs to be measured. Generally, requirements for reused systems are not available. Thus, the RE process needs to reverse engineer these requirements from the existing system [134].
3.5. Requirements Engineering Process

Sommerville et al. [133] have proposed a process model for the RE process which is widely accepted by researchers and professional experts. In this study, we used this model as our template to investigate the MT projects. The most important phases of RE which have to be applied are: Domain Analysis & Requirements Elicitation, Evaluation & Negotiation, Specification & Documentation and Validation & Verification.

The initial step in the RE process, that of Domain Analysis & Requirements Elicitation, is the act of obtaining detailed knowledge regarding the domain of the current problem, the organization or company confronting the problem and the existing system that is facing the problem. Once the required knowledge has been acquired, a draft document is provided which would help system developers to understand the context of the actual problem as well as enabling them to identify the stakeholders’ actual needs and requirements.

At the next stage of Evaluation & Negotiation, it is assumed that the previous stage (Domain Analysis & Requirements Elicitation) has been performed effectively. The Evaluation & Negotiation stage identifies inconsistencies and conflicts between requirements. The likelihood of such conflicts will increase if the requirements have been gathered from multiple and different stakeholders. Negotiation with stakeholders takes place to resolve conflicts and potentially infeasible requirements.

The third phase, that of Specification & Documentation of the RE process makes a precise set of agreed statements by all relevant sides of the project such as: requirements, assumptions, and system properties. Based on the specification, the requirements documentation can be drafted.

During the last phase, namely the Validation & Verification stage, the specifications are analysed and then validated by the stakeholders to ensure they satisfy their actual needs. Furthermore, specifications should be verified in order to check their consistency and to avoid conflicts and omissions. Any potential error and flaw must be fixed during this phase and before the actual development in order to save cost, effort and time.

Table 3.3 shows the requirements engineering process that was used
### 3.5. Requirements Engineering Process

in the examined MT development projects. Each MT project has been
categorised into the four RE stages: Domain Analysis & Requirements
Elicitation, Evaluation & Negotiation, Specification & Documentation,
Validation & Verification and for each project the individualized tech-
niques and approaches that were used is listed.

<table>
<thead>
<tr>
<th>Case</th>
<th>Elicitation</th>
<th>Evaluation</th>
<th>Specification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>document mining, prototyping</td>
<td>informal conflict resolution</td>
<td>UML/OCL</td>
<td>inspection</td>
</tr>
<tr>
<td>2</td>
<td>brainstorming, prototyping, reverse engineering</td>
<td>impact analysis</td>
<td>UML, graphs</td>
<td>testing</td>
</tr>
<tr>
<td>3</td>
<td>informal techniques, prototyping</td>
<td>none</td>
<td>none</td>
<td>testing</td>
</tr>
<tr>
<td>4</td>
<td>prototyping</td>
<td>scenario analysis</td>
<td>UML/OCL</td>
<td>testing, inspection, proof</td>
</tr>
<tr>
<td>5</td>
<td>prototyping</td>
<td>scenario analysis</td>
<td>UML/OCL</td>
<td>testing, inspection</td>
</tr>
<tr>
<td>6</td>
<td>prototyping</td>
<td>goal decomposition</td>
<td>UML, metamodelling</td>
<td>testing</td>
</tr>
<tr>
<td>7</td>
<td>reverse-engineering</td>
<td>goal decomposition</td>
<td>formal/logic</td>
<td>proof</td>
</tr>
<tr>
<td>8</td>
<td>prototyping</td>
<td>customer feedback, prioritisation</td>
<td>UML class diagram</td>
<td>testing, static analysis</td>
</tr>
<tr>
<td>9</td>
<td>domain analysis, prototyping</td>
<td>customer feedback, prioritisation</td>
<td>UML, BPMN</td>
<td>testing</td>
</tr>
<tr>
<td>10</td>
<td>prototyping, interviews, workshops</td>
<td>joint reviews with customers, conflict resolution</td>
<td>UML class diagram</td>
<td>testing</td>
</tr>
</tbody>
</table>

### 3.5.2 Changes and Conflicts in Requirements

It is always possible that requirements may change in the middle of the life cycle development. This can be due to stakeholder’s change of mind or circumstances or the need for more requirements in addition to the existing ones. Based on our study, we realized that the transformation developers also experienced a need to change the requirements in the
3.5. Requirements Engineering Process

middle of the life cycle when dealing with requirements modifications, unrealistic requirements and conflict amongst the requirements.

“Don’t do what you are told, but always do what is needed” (Study participant).

In Table 3.4 we have identified MT developer’s responses when confronted with common problems that may occur during the MT development.

<table>
<thead>
<tr>
<th>Project</th>
<th>Problem</th>
<th>Reaction, paraphrased from participant comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3, 4, 6, 7, 8</td>
<td>Unrealistic requirements</td>
<td>- Implementing “what is needed” rather than what is wanted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implementing “the underlying system”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Feedback to customers the estimated cost</td>
</tr>
<tr>
<td>1, 2, 3, 6, 7, 8</td>
<td>Change of requirements</td>
<td>- Agile provides sufficient time via weekly deployments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Confirming the requirements at the beginning of every iteration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Charging extra for additional requirements</td>
</tr>
<tr>
<td>1, 2, 3, 4, 5, 10</td>
<td>Requirements conflict</td>
<td>- Resolving conflicts through common sense</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Trade-off amongst the conflict requirements</td>
</tr>
<tr>
<td>2, 3, 4, 5, 6, 7, 8</td>
<td>Requirements uncertainty</td>
<td>- Contacting the stakeholders for clarifications</td>
</tr>
</tbody>
</table>

3.5.3 Requirements

According to our investigation, the requirements elicitation process in model transformations often starts with having an initial meeting with customers. Their input is central to the process at this stage.

“It is the process and an engagement that starts with the customer” (Study participant).

Customers often only have a very high-level view of what they need the transformation to achieve. For instance, a customer may only be aware of the language that his/her company wants the code to be generated into or the kind of platform it wants it to operate in.
3.5. Requirements Engineering Process

“Stakeholders are not very technical but they know what they need to see from the system at the end” (Study participant).

Therefore, transformation developers suggest joint sessions with the stakeholders in order to gain explicit knowledge about the system. During these sessions, brainstorming methods are applied to confirm the functional and non-functional requirements and specifications in more detail.

Customers often leave it up to the MT developers to flesh out the nature of those high-level requirements based on their expertise. The task of requirements elicitation and requirements engineering in general is done by MT developers. Not only are they in charge of implementation, but also eliciting the requirements is carried out by them as well.

“Stakeholders give high-level goals and it is for you to decide how to get there and what to use” (Study participant).

Therefore, initially the customer provides the developers with some high-level goals. Next, developers decompose the goals into sub requirements and once they have analysed them then they meet the customers again for a confirmation. Once there is an initial confirmed draft of the requirements of the overall system then the implementation phase is started. During the implementation, at the end of every stage developers provide prototypes for stakeholders.

“It starts with the customer, proof of concept, then taking some code from the customer and presenting what can be done by prototyping, with a tool, which provides analysis on code” (Study participant).

Once the prototype is delivered to the stakeholders, they will have the opportunity to raise any issues should there be something wrong or missing, otherwise the next stage of implementation will start. Prototypes were very popular amongst the model transformation projects that we analysed as these help both developers and stakeholders to understand the problem space. According to a participant’s paraphrased quote,
3.5. Requirements Engineering Process

“A good prototype is one which is a subset of the complete system”.

3.5.4 RE Techniques

There are several methods and techniques proposed by the requirements engineering community, however, selecting an appropriate set of requirements engineering techniques for a project is a challenging issue. Most of these methods and techniques were designed for a specific purpose and none cover the entire RE process. Researchers have classified RE techniques and categorised them according to their characteristics. For instance, Hickey et al. [54] proposed a selection model of elicitation techniques, Maiden et al. [101] came up with a framework that provides requirements acquisition methods and techniques. According to our study, in MT projects, RE techniques are selected and applied mainly based on personal preference or companies’ policy rather than characteristics and specifications of a project.

There are several different requirements engineering techniques from a variety of sources that can be employed during MT development. In Table 3.5 we present some of those that were more widely used in the MT projects. In the first column a Category is defined, where RE techniques have been categorized into: groups of human communication, process techniques, knowledge development and requirements documentation. In the second column the applied RE techniques are listed. Column three shows the MT project in which the RE techniques were applied. In the fourth column that of Rationale, the selection criteria of the techniques as described by interviewees are listed.
3.6. Outcomes

**TABLE 3.5.** RE techniques in MT projects

<table>
<thead>
<tr>
<th>Category</th>
<th>RE Technique</th>
<th>Project</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human communication</td>
<td>Online conference</td>
<td>1, 2, 3, 6</td>
<td>- Distribution of stakeholders</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Lack of accessibility</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Convenience</td>
</tr>
<tr>
<td></td>
<td>Brainstorming</td>
<td>1, 2, 6</td>
<td>- Enabling both stakeholders and developers to understand each other</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>as well as the requirements</td>
</tr>
<tr>
<td>Process techniques</td>
<td>Joint requirements</td>
<td>2, 10</td>
<td>- Resolving any possible issue which is not clear</td>
</tr>
<tr>
<td></td>
<td>development session</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Categorisation</td>
<td>1, 2, 3, 4, 5, 6, 7, 10</td>
<td>- Identifying functional and non-functional requirements</td>
</tr>
<tr>
<td>Knowledge development</td>
<td>Prototyping</td>
<td>1, 2, 3, 4, 5, 6, 8, 9, 10</td>
<td>- Receiving feedback based on the prototype</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Informing the stakeholders of the progress</td>
</tr>
<tr>
<td></td>
<td>Scenario</td>
<td>4, 5</td>
<td>- To decompose the requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Identify implications</td>
</tr>
<tr>
<td></td>
<td>Negotiation</td>
<td>2, 3, 6, 8, 10</td>
<td>- To prioritize the requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Identify trade-offs</td>
</tr>
<tr>
<td>Requirement documentation</td>
<td>Diagrams</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9, 10</td>
<td>- Providing a general view of the system</td>
</tr>
<tr>
<td></td>
<td>Textual</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9, 10</td>
<td>- Presenting the system formally</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Providing a contract for stakeholders</td>
</tr>
</tbody>
</table>

3.6 Outcomes

In evaluating the outcomes of the MT projects, the development effort and the encountered problems are considered, together with the degree to which the delivered transformation achieved the customer’s expectations. We use qualitative five point scales (Very Large, Large, Medium, Small, Very Small) for project scale, business value and customer satisfaction. Table 3.6 summarises the outcomes of the different MT projects.

**Project 1:** There were development problems stemming from the complexity and size of the metamodels to be processed. The intent and rationale for certain UML/OCL constructs needed to be clarified, as these were not clear from the metamodels or the existing docu-
3.6. Outcomes

mentation. The results of the developed transformation have been adopted by the customer. We regarded the size of this transformation as large, since there are of the order of approximately 600 rules.

Project 2: In this project also, development problems arose mainly from the nature of the transformation input data namely the large-scale legacy system code, and the effort needed to understand this before regenerating a modernised version. There was generally good communication between the developers/analysts and the customers, and thus negotiation over requirements was effective. There was good acceptance of the re-engineering work by customers. We regarded the size of this transformation as very large, as the size of the data was extremely high and over 1500 transformation rules were used.

Project 3: The transformation language used (Java-based syntax tree processor) was too procedural in style, which made analysis difficult, and in particular obstructed analysis of the semantic interaction between different transformations (code generators). The development process used was an agile ‘implement first’ process, with minimal documentation. This led to high costs in reworking the translators when errors were discovered. The customer was unwilling to participate in any structured requirements engineering process, and some of their requirements were infeasible. For these reasons we categorise the development costs as high. Not all of the desired customer requirements could be achieved, so we give a value of moderate here. We considered this transformation to be large in terms of size, since it involved about 500 transformation rules.

Project 4: Although the size of this transformation was quite small (about 20 rules), it had a complex behaviour due to the (underdetermined) interaction of the rules, which simultaneously refactor
and translate Petri-Net models to state machines. This made it difficult to verify correctness properties such as confluence. Many development iterations (over 10) were needed. The resulting transformation did satisfy all the customer’s functional requirements, but capacity requirements to handle large models were not fully achieved.

**Project 5:** This was a relatively small transformation (approximately 30 rules), but with large-scale data (100MB and larger). Many development iterations (over 10) were needed. The resulting transformation did not satisfy all of the efficiency requirements, and was not able to process the complete movie database data. Thus, we have given a value of moderate for customer acceptance.

**Project 6:** This case study involved a complex and difficult code generation problem for C++ multi-threaded and multi-processor code on multiple platforms. We consider this transformation to be of large size as a few hundred transformation rules were used. The semantic complexity of the target language and platforms caused the development effort and costs to be significantly higher than for other code generators developed by the company. In addition, the complexity of the resulting generator has hindered its adoption, which has been limited. Thus we give a rating of moderate for customer acceptance in this case.

**Project 7:** This case involved re-engineering of an existing code generator: extracting its requirements from its code and then writing a new improved generator to satisfy these requirements. The scale of the work was relatively low, and the main difficulties concerned extracting the requirements and identifying incorrect functionality in the existing translator. The new translator was accepted by the customer.

**Project 8:** This case involved the development of transformations to support railway network design. The scale of the models was
3.6. Outcomes

relatively large, with over 200 entities and transformation rules. An agile methodology was followed throughout the development, enabling rapid customer feedback, requirements prioritisation and fast responses to changing requirements. The project was a success and was accepted by the customer.

**Project 9:** This was a substantial MDE platform, which was the basis of the company’s business. The scale was large. An evolutionary and incremental methodology was followed for its development. The project was successful, with application of the tools in several commercial projects.

**Project 10:** This case involved the generation of middleware code from DSLs. The transformations were written partly in Java and partly in a custom MT language. It was of moderate scale, although substantial resources were used. The project was a success from the standpoint of customer satisfaction.

Development problems were encountered in Projects 1 and 6 due to the scale of the metamodels and in Projects 2 and 5 due to the models and programs involved in the transformations. The complexity of metamodels in Project 6 and the need to manage multiple versions of metamodels in Project 3 also caused problems. One conclusion that can be provisionally drawn is that large-scale transformation developments with relatively low levels of application of requirements engineering tend to have poor outcomes such as Projects 3 and 6. In contrast, a more detailed RE process can help to counteract the impact of the scale of a model such as Projects 2 and 8.

3.6.1 MT Project Failures

From the interview study, we identified three project cases that partially failed. In the terms of [100], these were: a process failure in the case
3.6. Outcomes

<table>
<thead>
<tr>
<th>Project</th>
<th>Transformation scale</th>
<th>Development cost</th>
<th>Customer satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Large</td>
<td>Moderate</td>
<td>High</td>
</tr>
<tr>
<td>2</td>
<td>Very Large</td>
<td>Moderate</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>Large</td>
<td>High: specifications too procedural, hard to analyse or modularise</td>
<td>Moderate</td>
</tr>
<tr>
<td>4</td>
<td>Small</td>
<td>Moderate</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>Medium</td>
<td>Moderate</td>
<td>Moderate</td>
</tr>
<tr>
<td>6</td>
<td>Large</td>
<td>High: complex and detailed semantics</td>
<td>Moderate</td>
</tr>
<tr>
<td>7</td>
<td>Medium</td>
<td>Moderate</td>
<td>High</td>
</tr>
<tr>
<td>8</td>
<td>Large</td>
<td>Moderate</td>
<td>High</td>
</tr>
<tr>
<td>9</td>
<td>Large</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>10</td>
<td>Medium</td>
<td>Moderate</td>
<td>High</td>
</tr>
</tbody>
</table>

of Project 3, where the process cost was excessive; an interaction failure in the case of Project 6, with relatively low uptake of the system; and an expectation failure in the case of Project 5, with the capacity of the developed transformation being inadequate to meet expectations. The causes of these failures correlate with those described in [100] as follows:

- Project 3: the cost of the process was due in part to poor communication with stakeholders, and to a lack of a systematic process.

- Project 6: the poor uptake seemed in part due to poor communication with the end users of the transformation and lack of knowledge of what they needed or would use.

- Project 5: the failure was in part due to lack of understanding of the IMDb and how it supports query optimisation, and also due to technical inadequacy of MT technology to process big data.

Poor communication with stakeholders is a potential problem in many MT developments due to the fact that these are often embedded within larger MDE projects. The MT developers receive requirements from the MDE team, but these may contain incorrect or incomplete understanding of the complete system requirements. Techniques such as joint applica-
3.7. Summary

JAD workshops are recommended to enable active participation of end users and other stakeholders in the RE process. Technological problems are also a factor in MT project failure due to the relative immaturity and non-standardised nature of MT languages and tools. It may be infeasible to solve a problem such as that in Project 5, with MT technologies and infeasible to manage and maintain the specifications and implementations in MT languages such as that in Project 3. The Validation & Verification stage of the RE process is particularly affected by the lack of tools for MT analysis and verification. Experimental techniques may be used in production projects (Project 6), resulting in high costs.

3.7 Summary

In this chapter, we have reported on the results of an exploratory study of requirements engineering for model transformation development. We have presented our initial findings from seven semi-structured interviews with industrial experts in the field. Clearly, more research is needed, but some interesting points have already emerged from this study and are worth closer attention:

First, we have been able to identify that model transformation projects are typically individual projects that are embedded in wider software development projects. We have briefly commented on how this impacts the identification of and communication with stakeholders in the transformation development. The projects we have discussed are mainly Greenfield projects, which is different from the wider software development reality. This may be because model transformations are still a relatively young technology in industrial practice.

Threats to the validity of conclusions drawn from the interviews include: (i) that the interviewees and examined cases are not representative of transformation developers and projects, (ii) that interviewees selected unrepresentative projects, (iii) that interview questions were aimed at
elicitating a particular response. We tried to avoid potential problems (i) by requesting interviews with a wide range of MT experts. The candidates for interviews were selected from our previous literature surveys of RE in MT. 15 candidates were approached, of whom seven agreed to be interviewed. These represent a diverse range of organisations, and the projects cover a fairly wide range of domains such as: embedded systems, finance, re-engineering, transport, defence and business. Regarding (ii), projects with poor outcomes, such as Project 3 and Project 6, were included in addition to successful projects. Regarding (iii), the questionnaire and methodology were examined by an expert committee for ethical approval.

The interaction between the needs of the wider project and the highly technical nature of model transformation development seems to have an impact on the requirements elicitation process in particular. We have seen that while prototyping and example-based generalisation seem to play an important role in understanding the requirements in model transformations, no systematic process seems to be followed. Although developers apply some requirements engineering techniques in transformation projects this is often based on their experience and common sense as there is no specific requirements engineering process designed for model transformation development. At present, the focus of transformation development is mainly on the specification and implementation stages and the development team is responsible for all development process activities including the requirements engineering process.
Chapter 4

Systematic Literature Survey

After the interview-based study, we decided to expand our understanding of requirements engineering in model transformation through further investigation of model development projects by means of a systematic literature review (SLR). In this chapter, we present the results of a systematic literature survey of the current process of requirements engineering in MT developments. 160 papers have been reviewed and analysed from the past 10 years rendering it one of the largest existing surveys in MT. All the papers contained either industrial or academic MT developments.

4.1 Introduction

Transformations are used widely in model-driven engineering (MDE) and model-based development (MBD). Their uses include migration of models from one language to another, refactoring of models to improve quality, refinement of models from a specification to a design, or from design to implementation, code generation to generate program code from models, bidirectional transformations to synchronise two different models and to maintain their consistency. Transformations can also be used for data analysis to analyse and extract information from models. Semantic mapping transformations map a model to a semantic domain to support precise analysis.
4.2. Related Work

Similar to any other field, MT also requires an appropriate RE process in order to develop correct transformation applications. In this chapter we consider specific aspects of RE for model transformation. In order to achieve any given goal using software (such as in a transformation), having a scheme in which its requirements have been identified is essential. Requirements engineering has been a relatively neglected aspect of model transformation development; the emphasis in transformation development has been upon specification and implementation. The failure to explicitly identify requirements may result in developed transformations which do not satisfy the needs of the transformation users. Problems may arise because implicitly-assumed requirements have not been explicitly stated. It might be possible to skip the requirements engineering process in small projects and jump directly into the implementation phase, however, it is unlikely to be possible and effective in large and industrial projects.

This chapter is organized as follows: Section 4.2 describes the related work. Section 4.3 describes the methodology we used to carry out the systematic literature review. Section 4.4 reveals the results obtained from the survey followed by Section 4.5 which discusses the threats to the validity of the results. Finally, Section 4.6 presents our concluding remarks from this survey study.

4.2 Related Work

The survey of [110] considered in depth four cases of MDE application, but did not specifically consider the requirements engineering of these cases. One concern of the companies in [110] was the cost of developing transformations, a factor which could be improved by more systematic RE for MT. The survey of [99] considers the use of RE in MDE, and concludes that the use of rigorous techniques for RE in MDE is limited: the majority of surveyed cases did not have tool support for RE, and in most cases the RE process was not integrated into the MDE process.
These results are consistent with our own findings for RE use in the more specialised field of MT development. The survey [11] of concrete MT developments analyses 82 MT cases with regard to their type and outcomes, but does not specifically consider RE aspects.

In our RE in model driven engineering SLR, we focus specifically on model transformation developments, whether as part of an MDE process or as independent developments. For MT developments, we examine how RE techniques and the RE process is carried out. We consider a wide spectrum of RE techniques and approaches.

4.3 Research Methodology

For the Systematic Literature Review (SLR), we follow the methodology proposed by Kitchenham [72] defining an SLR as a means to identify, evaluate and interpret a series of available relevant research topics for a particular research question, area or phenomenon of interest. According to this methodology, an SLR consists of different stages including planning, conducting and reporting the review. In the planning stage of the review, the actual need of the review as well as specific questions regarding the review’s protocol are defined. In the conducting stage of the review, the scope of the research as well as primary studies regarding the research are identified. The study quality assessment is defined, alongside with data extraction, monitoring and synthesis. Last but not least, in the reporting stage of the review the overall disseminated structure is specified and the review results are presented.

4.3.1 Research Question

The main question we consider is:

What requirements engineering process and techniques, if any, have been applied in model transformation development?

The purpose of this question is to investigate the recent and current role
4.3. Research Methodology

of RE in MT. We aim to collect the current available knowledge regarding MT developments and the role of RE in these developments. It will also be used to identify any potential gaps in research and practice, and guide the proposal of solutions and suggestions for further investigations and future work. We have defined the structure of the SLR according to the following procedure known as PICOC (population, intervention, comparison, outcome, context) criteria:

- Population: research papers presenting MT developments and case studies.
- Intervention: RE process and techniques.
- Comparison: analysis of the current state of RE in MT.
- Outcome: guidelines for a specific RE framework for MT.
- Context: MT engineering and development.

4.3.2 Source Selection

The selection process of relevant and appropriate papers was done in two different ways. Firstly, an automatic search method was used from the most credible scientific paper databases as follows: IEEE Xplore (IE), ACM Digital Library (ACM), Research Gate (RG), Google Scholar and SpringerLink (SL). Secondly, a manual search method was used from the following representative journals and conferences: Transformation Tool Contest (TTC), Model Driven Engineering Languages and Systems (MODELS), International Conference on Model Transformation (ICMT), International Journal on Software and Systems Modeling (SoSyM) and Journal of Systems and Software (JSS). One of the main advantages of applying the manual search was that it allowed us to carry out a more in-depth study of some particular works based on specific topics and areas. It also served as a verification method in order to verify the correctness of the automatic search method.
4.3. Research Methodology

In addition to the above, we included the 82 papers considered in the survey [11] of concrete MT developments in our initial selection.

4.3.3 Primary Studies Selection

In order to identify the primary studies, a search string was created based on the research question. There are two parts within the search string: the first part is about industrial and academic works that describe MT developments and the second part is related to the use of RE in MT development. There are several search strings that have been used throughout this SLR, the following is just a sample:

(Model transformations OR model transformation case studies OR requirements engineering in model transformations OR requirements engineering technique in model transformations) AND (MDA OR MDE OR model-driven OR model-based OR model*)

Since we are interested in discovering the prevalence of RE use in MT development, we do not include a term such as requirements as an obligatory part of the search string.

4.3.4 Selection Criteria

We imposed some limitations and restrictions on the reviewed papers and context. We only considered papers which presented MT development case studies, instead of purely theoretical papers. Furthermore, we only included published papers in conferences, workshop proceedings and scientific journals within the past decade (January 2006 to July 2016) for this SLR. Moreover we have excluded the papers with the following criteria: i) papers only describing MT languages and tools, ii) papers presenting RE in general and not related to MT development, iii) short papers (less than 5 pages), iv) PhD thesis, poster publications and tutorials, v) papers written in any language other than English. Appendix D lists all the reviewed papers.
4.3. Research Methodology

4.3.5 Information Extraction

We have extracted the data and information according to our research question. In this section, we will explain the research question according to the following criteria in more detail:

- **Type of transformation (criteria 1).** Model transformation is one of the most important parts of MDE approaches in today’s software development. Transformations are often used for: restructuring and refactoring models, migrating models and refining models from PIM to PSM. In general, it could be said that transformations are generally useful to translate the semantic content of a model from one language to that of another [34].

- **Transformation development scope (criteria 2).** Only information concerning the development of model transformation projects and case studies was collected.

- **Type of RE techniques (criteria 3).** In case there was usage of a particular type of requirements engineering technique throughout the MT development, the information regarding the technique and the requirements engineering is gathered. These techniques can be for either functional and non-functional requirements.

- **Transformation development projects (criteria 4).** We are interested to find out how MT development is currently applied and in what type of projects it is used. Moreover, the scale, developer’s time and effort for some of the MT development projects are other criteria of interest.

- **Type of project (criteria 5).** In general, software development projects can be classified into several types such as: Greenfield vs Brownfield, Customer vs Market driven, In-house vs Outsourced, Single product vs Product line.

- **Stakeholders (criteria 6).** We are interested to identify the type of stakeholders in MT development projects for this SLR.


4.3. Research Methodology

- Requirements elicitation (criteria 7). Identifying the requirements elicitation process in model transformations is an important factor that needs to be reviewed for this SLR.

- Requirements category (criteria 8). We are interested in the categories of requirements (functional, non-functional, local, global) which are considered in MT requirements engineering.

![Surveyed cases per year](image)

**Figure 4.1.** Surveyed cases per year

4.3.6 Conducting the Review

Following the format used in [99], we have conducted the review according to these activities: selecting primary studies, extracting the data and data synthesis. The primary sources, namely journals and conferences, were identified. The searching process resulted in over 600 items, from which we selected 189 papers that we considered to be most appropriate and useful for this research topic. Our primary searching method to find the relevant papers was an automatic search, however a manual searching method was also conducted to discover potential papers that were not identified via the automatic search.

Table 4.1 shows the initial results from both the automatic and the manual search as well as the final result. The initial result’s section shows
4.3. Research Methodology

the total number of papers obtained from different sources and the final result’s row indicates the actual number of papers that were selected to be reviewed. We have discarded duplicated papers along with the older and less updated version of a particular version. Remaining were 160 papers which satisfied our criteria for detailed analysis. Figure 4.1 shows the distribution of the surveyed papers over time.

<table>
<thead>
<tr>
<th>Source</th>
<th>IE</th>
<th>SL</th>
<th>ACM</th>
<th>RG</th>
<th>TTC/MODELS/STAF/SoSyM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial result</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>16</td>
<td>73</td>
<td>82</td>
</tr>
<tr>
<td>Final result</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>13</td>
<td>56</td>
<td>80</td>
</tr>
</tbody>
</table>

TABLE 4.1. Number of reviews

4.3.7 SLR Results

This section presents the outcome of the systematic literature survey together with the specified criteria as listed in Section 4.2.5. Table 4.1 shows a summary of the total number of papers analysed throughout this SLR. Tables A.1, A.2 and A.3 show information on the stakeholders of the considered cases. Table A.4 shows the categories of requirements occurring in the cases. Tables A.5 and A.6 shows information about the type of projects in each case, and Tables A.7, A.8, A.9 and A.10 show information about the RE methodology and techniques used.

Regarding the stakeholders (criterion 6), the results in Tables A.1 and A.2 show the main stakeholders of the analysed papers; of these 62.5% were MDE practitioners, 7.5% were financial companies, 7.5% were users requiring analysis of data, 6.25% were embedded system developers, and 16.25% were other types of stakeholders as represented in Figure 4.2a.

Of the projects, 86% were academic and 14% were industrial. The industry cases are numbers 25, 27, 29, 30, 58, 59, 60, 61, 62, 63, 64, 65,
4.3. Research Methodology

(a) Transformation stakeholders

(b) Reaching stakeholders

Figure 4.2. SLR MT stakeholders

66, 69, 114, 120, 141, 142, 151, 152, 155, 161 as listed in the Systematic Literature Survey Papers, Appendix D.

About 38% of the cases used online forums and email as a means of communication between the developers and the stakeholders. In many cases (52%) there seemed to be no attempt made to reach stakeholders, instead the transformation developers made their own assumptions about the needs of the stakeholders. Direct stakeholder participation in development or direct stakeholder consultation took place in only 8% of cases (Figure 4.2b).

The result for the transformation type (criterion 1) as presented in Table A.5 shows that about 33% of the transformation case studies were of the refinement type, 22.5% were of the migration type, 11% of the refactoring type, 11% were of the code generations type, and 9.5% were of the semantic mapping type (Figure 4.3).

The result for the scope of transformation developments (criterion 2) shows that from the reviewed papers, 100% of them were involved in developing a transformation project or case study.

The result for applying RE techniques (criterion 3) throughout the development as presented in Table A.7 and Table A.8 shows that 50% of
4.3. Research Methodology

Figure 4.3. Transformation types

cases applied some requirements engineering technique during the development life cycle as follows: scenario analysis: 43 cases, semi-formal or formal rule specifications (not in an MT language): 27 cases, prototyping: 18 cases, prioritization: 9 cases, participant interaction techniques (interviews, questionnaire, observation, survey): 10 cases. 50% of cases did not apply any RE technique, according to their presentation of the cases. Figure 4.4 shows these statistics in a graphical format.

Around 70% used UML class diagrams for documentation, 14% used no diagrams, and 12% used concrete syntax of the source or target languages.

Table A.4 identifies which kinds of requirements were considered in the surveyed papers. We divide requirements into functional and non-functional, and the former into local and global functional requirements.

Local requirements express how individual model elements or small groups of related model elements should be mapped to elements of another model, or should be refactored in-place. In the case of bidirectional transformations (bx), local correspondence requirements express how elements of one model should correspond to elements of the other. Global requirements concern properties of source or target models considered as a whole.
4.3. Research Methodology

The most common requirements are local mapping (63 cases), efficiency (54 cases), semantic correctness (44 cases), syntactic correctness (35 cases) and semantic preservation (27 cases).

Syntactic correctness is the property that the transformation produces models which conform to the constraints of the target language. Semantic correctness expresses that necessary semantic properties of the target model relative to the source model are satisfied. Semantic preservation states that the semantics of the source model are preserved in the target model. Efficiency concerns with the performance of the application.

The result for transformation development projects (criterion 4) in Table A.5 was not very explicit as not many developers explained the scale of the transformation and the time and effort that they consumed regarding the development in the analysed papers. The result for type of transformation in terms of Greenfield vs Brownfield, customer vs market driven, in-house vs outsourced, single product vs product line (criterion 5), shows that almost all cases were Green, customer driven, in-house and single transformation projects.
4.3. Research Methodology

The result of the RE process (criterion 7) regarding requirements engineering for model transformation in Tables A.7, A.8 and A.9 shows that around 95% of the MT projects did not follow any systematic requirements engineering process during the development, only 5% (8 cases) used such a process (cases 25, 27, 28, 37, 61, 125, 141, 161).

Tables A.11 and A.12 compare the relationship between RE quality and the project outcome for each case and Figure 4.5 shows a scatter plot of the values. Both the average quality of RE, measuring 3.4 out of 9, and the average outcome, measuring 2.9 out of 6, is low.

![Figure 4.5. SLR case RE rigour (x-axis) versus outcomes (y-axis)](image)

Figure 4.5. SLR case RE rigour (x-axis) versus outcomes (y-axis)
4.4. Comparison

The resulting correlation is 0.49. This is statistically significant at the 1% level using a 2-sided $t$-test with 146 degrees of freedom [146]. The strength of the correlation is low, representing that only about 25% of the difference in outcomes is due to differences in the RE rigour of the cases. Firstly, there are several cases where a development had a good outcome despite a low RE score, such as cases 15, 51, 52, 53. This can occur due to the high skill level of the developers, who in the reviewed literature cases generally have advanced degrees and qualification levels which probably would not be the situation for transformation developers in general. Secondly, there are cases with evidently good RE rigour but nonetheless had low evidence of successful outcomes, such as cases 124, 142, 145, 149. This is due to the presentation of the work at a too early stage, lack of evaluation by stakeholders or because of the large scale of the project. This low correlation is due to the fact that the source of the cases were mainly academic and a stronger correlation between RE and outcomes is expected in industrial cases.

The results of the SLR show a lack of a systematic RE process for MT, and a lack of guidelines for the use of RE techniques for transformations. In particular, the survey has shown the relevance of scenario analysis and prototyping as RE techniques for MT development, and the need to consider specific forms of requirements for transformations. We will incorporate the findings of this survey into our work on an RE framework for MT as presented in Chapter 5.

4.4 Comparison

The results of the SLR and the interview study can be compared as shown in Table 4.2. We aimed to focus on industrial case studies for our interview study, and this resulted in a much higher percentage of industrial cases compared to the SLR. There is a close correlation between industrial cases and large project scale: in the interview study 60% of the cases were either large or very large, in the SLR only 10% were in
4.4. Comparison

this category (Figure 4.6b).

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Interviews</th>
<th>SLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry/Academic</td>
<td>80%/20%</td>
<td>14%/86%</td>
</tr>
<tr>
<td>Scale (Large/Medium/Small)</td>
<td>60%/30%/10%</td>
<td>10%/23%/67%</td>
</tr>
<tr>
<td>Used RE techniques</td>
<td>90%</td>
<td>50%</td>
</tr>
<tr>
<td>Used RE process</td>
<td>30%</td>
<td>5%</td>
</tr>
</tbody>
</table>

Likewise, larger scale and industrial cases were more likely to use some RE techniques: 90% of the interview cases versus 50% in the SLR cases. In both, however, a systematic RE process was only used by a small minority of cases.

![Figure 4.6](image_url)

**Figure 4.6.** MT projects scale

Considering the 170 combined interview and SLR cases, of the 30 industrial cases, 27 used some RE techniques (90%), whereas only 63 of the 140 academic cases (45%) used RE techniques. 8 of the 30 industrial cases used a systematic RE process (27%), whilst only 3 of the 140 academic cases did so (2%).
4.4. Comparison

With regard to elicitation techniques, the interview cases more often used prototyping (90% of the cases) compared to the SLR cases (11%), and were less likely to use scenario analysis (20% versus 27%). There was a higher rate of usage of prioritisation (20% compared to 6%) and of UML class diagrams (80% compared to 70%).

Evaluating outcomes against RE rigour for the interview cases gives the scatter diagram of Figure 4.7. Here the average RE quality is much higher (6.5) than for the SLR cases, as is the outcome measure (5.4).
4.5 Threats to Validity

In this section, we discuss the potential threats which might affect the validity of our SLR. One possible limitation of the SLR study is that the authors may not have explicitly and fully explained the RE process that they have applied throughout the transformation development, nor other details of the case context and outcomes. In many cases only outline information about the transformation details and development process are given. We have made the assumption that if details of RE or development processes are not provided, then that means that the processes were not used. Moreover other factors such as publication bias, data extraction and misunderstandings are other factors that may have a negative impact towards the validity of this SLR [72]. One of the ways that we have validated our review is the selection of credible resources such as digital libraries in order to maximise the correctness and completeness of our review toward the specified objectives. As mentioned earlier, the search string was expressed via a combination of different terms and expressions from the type of papers concerning model transformation development case studies and requirements engineering. We have also attempted to reduce the misunderstandings and data extraction inaccuracy by having two independent reviewers for every selected paper.

4.6 Summary

In this chapter, we have reported on an exploratory study of requirements engineering for model-transformation development. We have reported on our initial findings from the analysis of 160 published papers of MT developments. Clearly, more research is needed, but some interesting points have already emerged from this study and are worth closer attention.

After analysing the result of both our interviews and SLR, we can conclude that at present, the focus of transformation development is mainly on the specification and implementation stages. Developers may apply some requirements engineering techniques in transformation projects,
4.6. Summary

however this is often based on their experience and common sense as there is no systematic requirements engineering process designed for model transformation development. Almost none of the cases we considered in the interview study or SLR allocated a substantial phase for RE during the MT development. Most of the RE techniques were applied in an informal manner supported by mainly personal experience regarding that particular technique. Developers do not model the functional and non-functional requirements for a transformation, and they often only concentrate on implementing the main goal(s) of a transformation (i.e. refactoring or model to model migration). Developers usually start the validation process after the transformation has been developed and they check the transformation to see which of the quality requirements are satisfied.

To summarise, the obstacles for the use of RE in MT appear to be:

- Restricted access to stakeholders by the MT developers, limiting the use of RE techniques such as interviews and brainstorming.
- The lack of RE techniques and guidelines specific to MT.
- The absence of a systematic RE process defined for MT.
- The lack of published case studies of RE systematically applied to MT projects.

Chapter 5 will show our research work in this area which includes a more systematic process for requirements engineering in the context of MT development [151].
Chapter 5

Requirements Engineering Activity for MT

This chapter presents recommendations for the application of requirements engineering in model transformation followed by an adaptation of the Sommerville et al. model for MT. Furthermore, it presents the criteria for selecting appropriate requirements engineering techniques for MT, and proposes a framework for this selection process. This framework is aimed to facilitate the process of choosing the appropriate set of requirements engineering techniques according to the type of project (where the competing techniques in a given selection-process are the candidates for a given RE phase) in this case model transformation. Different sections of this chapter have been published in several conferences [149, 150, 151, 152], one journal [153] and a book [86].

5.1 Application of RE in MT

As concluded from Chapter 3 and 4, the development of model transformation is mainly focused on the specification and implementation phases, whereas there is a lack of support in other phases including: requirements, analysis, design and testing. Furthermore, there is a lack of cohesive support for transformations including: notations, methods and tools within
all phases during the development process, which makes the maintenance and understandability of the transformation code problematic [46].

In model transformation, requirements and specifications are very similar and sometimes are considered as the same element. Requirements determine what is needed and what needs to be achieved while taking into account the different stakeholders, whereas specifications define how to achieve the requirements.

Once the functional and non-functional requirements are identified, the scenarios of the transformation can be written as use cases in UML. Failure to explicitly identify requirements may result in developing transformations which do not completely satisfy the client’s needs or may lead to developing something that differs from what is actually needed. If the requirements are not being expressed explicitly and they are assumed implicitly, problems may arise during the transformation. For instance, in a migration or refactoring transformation that the semantics of its source model should be preserved in the target model, or that the transformation should only be required to operate on a restricted range of input models. Without thorough requirements engineering, important requirements may be omitted from consideration, resulting in a developed transformation which fails to achieve its intended purpose [149].

In model transformation, the initial requirements, which describe the intended functional behaviour of the transformation, are often displayed in terms of concrete syntax. This is especially helpful from a requirements engineering’s point of view which has a direct interaction with the stakeholders. Having the requirements and specifications of the intended functionality of the transformation expressed in concrete syntax rule (as opposed to abstract syntax rule) is more convenient for the stakeholders. This is because concrete syntax is usually more familiar to the stakeholders (the requirements engineer needs to be aware of these issues, not necessarily the stakeholder). However, concrete syntax may not always be completely precise since there may be significant details of models which have no representation in concrete syntax, or there may be ambiguities in the concrete syntax representation. Therefore, conversion of
5.1. Application of RE in MT

the concrete syntax rules into precise abstract syntax rules is a necessary step as part of the formalisation of the requirements [149].

Requirements engineering for model transformation involves specialised techniques and approaches, because transformations: (i) have highly complex behaviour, involving non-deterministic application of rules and construction of complex model data, (ii) are often high-integrity and business-critical systems, with strong requirements for reliability and correctness, (iii) are often embedded in large MDE projects and (iv) do not usually involve much user interaction since they are batch-processing systems, but may have security requirements if they process secure data.

The source and target languages of a transformation must be precisely specified by metamodels. However the requirements for its processing may initially be quite unclear.

For a migration transformation, analysis will be needed to identify how elements of the source language should be mapped to elements of the target; there may not be a clear relationship between parts of these languages, there may be ambiguities and choices in mapping, and there may be necessary assumptions on the input models for a given mapping strategy to be well-defined. There are specialist tools and languages for migrations, such as COPE [53] and Epsilon Flock [124], which may be selected. The requirements engineer should identify how each entity type and feature of the source language should be migrated.

For refactorings, the additional complications arising from update-in-place processing need to be considered. For instance, the application of one rule to a model may enable further rule applications which were not originally enabled. The choice of transformation technology will need to consider the level of support for update-in-place processing. Some languages such as ATL [64] and QVT [115] have limited update-in-place support. The requirements engineer should identify all the distinct situations which need to be processed by the transformation such as arrangements of model elements and their inter-relationships and significant feature values.

Code-generation transformations may be very large, with hundreds of
5.1. Application of RE in MT

rules. The effective organisation and modularisation of the transformation, and the selection of appropriate processing strategies, are important aspects to consider. Template-based generation of program language text is a useful facility for code generators, and is provided by transformation technologies such as Epsilon Generation Language (EGL) [125] and ATL templates. The requirements engineer needs to identify how each source language construct should be translated into code.

5.1.1 Requirements Taxonomy

In order to make the requirements engineering process more systematic, we have created a functional and non-functional requirements taxonomy. Taxonomizing the requirements according to their type not only would make it clearer to understand what the requirements refer to, but also by having this type of distinction among them will allow for a more semantic characterization of requirements.

We propose that requirements are distinguished into local requirements and global requirements:

- Local requirements are concerned with the mappings between one localised part of one or more models. Mapping local requirements define when and how a part of one model should be mapped onto a part of another. Refactoring local requirements dictate when and how a part of a model should be transformed in-place.

- Global requirements identify properties of an entire model. For example, that some global measure of complexity or redundancy is decreased by a refactoring transformation. Assumptions, model quality improvement, postconditions and invariants often have an effect on the entire model level.

Figure 5.1 shows a taxonomy of functional requirements for model transformations based on our findings of transformation requirements.
5.1. Application of RE in MT

Figure 5.1. A taxonomy of functional requirements

Figure 5.2 shows a taxonomy of non-functional requirements that need to be considered during the RE process. It shows a general decomposition of non-functional requirements for model transformations. The quality of service categories correspond closely to the software quality characteristics identified by the IEC 25010 software quality standard [21].

Non-functional requirements for model transformations could be further detailed. For instance, regarding the performance requirements, boundaries (upper/lower) could be set on: execution time, memory usage for models of a given size, and the maximum capability of the transformation (the largest model it can process within a given time). Restrictions can also be placed upon the rate of growth of execution time with input model size, for example, that this should be linear.

Similarly, reliability requirements for a transformation which are categorized into maturity and fault tolerance, could also be more detailed. For maturity requirements, it can be measured depending on its history and the extent to which the transformation has been used. For fault tolerance requirements, it can be quantified in terms of the proportion of execution errors which are successfully caught by an exception handling
5.1. Application of RE in MT

mechanism, and in terms of the ability of the transformation to detect and reject invalid input models.

The accuracy characteristic includes two sub-characteristics: correctness and completeness. Likewise, correctness requirements could be further divided into the following [90]:

- **Syntactic correctness**: a transformation \( \tau \) is syntactically correct if wherever \( \tau \) terminates, when applied to a valid model \( m \) of source language \( S \), it produces a valid target model \( n \) in terms of conformance to the \( T \)'s language constraints.

- **Termination**: a transformation \( \tau \) will always terminate if applied to a valid \( S \) model.

- **Confluence**: all result models produced by transformation \( \tau \) from a single source model are isomorphic.

- **Model-level semantic preservation**: a transformation \( \tau \) is preserved model-level semantically, if \( m \) and \( n \) have equivalent semantics under semantic-assigning maps \( Sem_S \) on models of \( S \) and \( Sem_T \) on models of \( T \).

- **Invariance**: some properties \( Inv \) should be preserved as true during the entire execution of transformation \( \tau \) [90].

An additional accuracy property that could be considered is the existence of invertibility in a transformation \( \sigma : T \rightarrow S \), which inverts the effect of \( \tau \). Given a model \( n \) derived from \( m \) by \( \tau \), \( \sigma \) applied to \( n \) produces a model \( m' \) of \( S \) isomorphic to \( m \). A related property is change propagation, which means that small changes to a source model can be propagated to the target model without re-executing the transformation on the entire source model. A further property, verifiability, is important for transformations, which is part of a business-critical or safety-critical process. This property identifies how effectively a transformation can be verified. Size, complexity, abstraction level and modularity are contributory factors to this property.
The traceability property is the requirement that an explicit trace between mapped target model elements and their corresponding source model elements should be maintained by the transformation, and be available at its termination.

The interface property is subdivided into user interaction, which in turn is subdivided into usability and convenience, and software interoperability.

The suitability property is defined according to [28] as the capability of a transformation approach to provide an appropriate means to express the functionality of a transformation problem at an appropriate level of abstraction, and to solve the transformation problem effectively and with acceptable use of resources such as developer time, computational resources, etc. In [75] the following subcharacteristics for the suitability quality characteristic of model transformation specifications were identified as: abstraction level, size, complexity, effectiveness and development effort.

Taxonomy Example

We have applied the requirements taxonomies according to model transformation characteristics as shown in Table 5.1. All types of functional requirements for model transformations including: mapping, refactoring, assumptions, model quality improvement, post-conditions and invariants requirements have been categorised.

Concrete syntax is often used at the early stages (RE stage) of the development cycle in order to validate the requirements by the stakeholders since the concrete syntax level is more convenient for them, whereas abstract syntax rule is often used in the implementation phase for developers. However, there should be a direct correspondence between the concrete syntax elements in the informal or semi-formal expression of the requirements, and the abstract syntax elements in the formalised versions.

Non-functional requirements mainly consider the necessary quality of
5.1. Application of RE in MT

<table>
<thead>
<tr>
<th></th>
<th>Refactoring</th>
<th>Refinement</th>
<th>Migration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local Functional</strong></td>
<td>Rewrites/Refactorings</td>
<td>Mappings</td>
<td>Mappings</td>
</tr>
<tr>
<td><strong>Local Non-functional</strong></td>
<td>Completeness (all cases considered)</td>
<td>Completeness (all source entities, features considered)</td>
<td>Completeness (all source entities, features considered)</td>
</tr>
<tr>
<td><strong>Global Functional</strong></td>
<td>Improvement in quality measure(s), Invariance of language constraints, Assumptions, Postconditions</td>
<td>Invariance, Assumptions, Postconditions</td>
<td>Invariance, Assumptions, Postconditions</td>
</tr>
</tbody>
</table>

**TABLE 5.1. Transformation requirements catalogue**
5.1. Application of RE in MT

a transformation. There may be a wide range of different non-functional requirements for a system [143], in categories such as quality of service, compliance, development constraint, etc. Some requirements categories, such as safety and security, are not generally properties of concern for model transformations. This is due to the fact that transformations are usually used internally within the organization. Amstel et al. [140] propose a set of quality characters regarding model transformation including: understandability, modifiability, reusability, modularity, completeness, consistency, and conciseness. Obviously the number of these taxonomies could be extended and varied depending on the type of model transformation.

For quality of service requirements, specific quantifiable measures for the properties of interest should be identified, and precise bounds on the permitted values of these measures (or ranges of acceptable values) specified. We have chosen the ISO 9126-1 [21] quality framework as our standard from which the requirements are to be measured. The International Organisation for Standardization provides a set of metric and quality standards for measuring the quality of developed software. The ISO 9126 contains different metrics for all kinds of software. Table 5.2 shows some of the quality characteristics of model transformations as well as their sub-characteristics according to the ISO framework.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Subcharacteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability, Accuracy, Interoperability, Security, Functionality</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity, Fault tolerance, Recoverability, Reliability compliance</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability, Learnability, Operability, Attractiveness, Usability compliance</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour, Resource utilisation, Efficiency compliance</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Analysability, Changeability, Stability, Testability, Maintainability compliance</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability, Installability, Co-existence, Replaceability, Portability compliance</td>
</tr>
</tbody>
</table>

TABLE 5.2. Standard quality framework (ISO 9126)
5.2 RE Process Adaptation for MT

In this section, it will be shown how the Sommerville et al. [133] RE process model can be adapted for model transformation based on our studies.

5.2.1 Domain Analysis and Requirements Elicitation

As a result of our study, a large number of requirements elicitation techniques have been surveyed. We summarise these and consider their relevance for the requirements analysis of transformations as follows:

- Observation
  This involves the requirements engineer observing the current manual or semi-automated process used for the transformation. It is relevant if a currently manual software development or transformation process is to be automated as a transformation. For example, if a procedure for constructing web applications or Enterprise Information System (EIS) of a particular architectural form is to be automated. Observation can capture the elements of the manual process currently used by developers. The technique is relevant for refinement, code generation, migration and refactoring transformations.

- Unstructured interviews
  In this technique the requirements engineer asks stakeholders open-ended questions about the domain and current status of the transformation. The technique is relevant in identifying the important issues which a transformation should have as goals. For refactorings, these could be what are the important goals for quality improvement of a model or a system. For refinements, what are the important properties of the generated code (e.g. efficiency, conformance to a coding standard, readability, etc.). For migra-
5.2. RE Process Adaptation for MT

... what is the scope of mapping (which forms of input models are intended to be processed), what semantic properties should be preserved, and what required restrictions are there on the output model structure.

- Structured interviews
In this technique the requirements engineer asks stakeholders prepared questions about the domain and the system. The requirements engineer needs to define appropriate questions which help to identify the scope of the transformation and the required properties of the product (output model requirements). This technique is relevant to all forms of transformation problems. We have defined a catalogue of MT requirements for refactorings, refinements and migrations, as an aid for structured interviews, and as a checklist to ensure that all forms of requirements appropriate for the transformation are considered as shown in Table 5.1.

- Brainstorming
In this technique the requirements engineer asks a group of stakeholders to generate ideas about the system and problem. This may be useful for very open-ended and new transformation problems where there is no clear understanding of how to carry out the transformation. For example, for complex forms of migration where it is not yet understood how data in the source and target languages should correspond, likewise for complex refinements, perhaps involving synthesis of information from multiple input models to produce a target model. Complex refactorings such as the introduction of design patterns could also use this approach.

- Rapid prototyping
In this technique a stakeholder is asked to comment on a prototype solution. This technique is relevant for all forms of transformation, where the transformation can be effectively prototyped. Rules could be expressed in a concrete grammar form and reviewed by
5.2. RE Process Adaptation for MT

stakeholders, along with visualisations of input and output models. This approach fits well with an Agile development process for transformations. Some transformation tools and environments are well-suited to rapid prototyping, such as GROOVE (GRaph-based Object-Oriented VErification) \[120\], a software model checking of object-oriented systems. For others, such as ETL \[77\] or QVT \[115\], the complexity of rule semantics may produce misleading results.

- **Scenario analysis**
  In this approach the requirements engineer formulates detailed scenarios or use cases of the system for discussion with the stakeholders. This is highly relevant for MT requirements elicitation. Scenarios can be defined for different required cases of transformation processing. The scenarios can be used as the basis of requirements formalisation. This technique is proposed for transformations in \[46\]. A risk with scenario analysis is that this may fail to be complete and may not cover all cases of expected transformation processing. It is more suited to the identification of local rather than global requirements.

- **Ethnographic methods**
  This approach involves systematic observation of actual practice in a workplace. Like Observation, this may be useful to identify current work practices (such as coding strategies) which can be automated as transformations. In general, techniques capturing process and behaviour information are more relevant than those capturing data, because the data (metamodels) of transformations are often explicitly provided and are fixed.

5.2.2 Evaluation and Negotiation

Prototyping techniques are useful for evaluating requirements, and for identifying deficiencies and areas where the intended behaviour is not yet understood. A goal-oriented analysis technique such as KAOS \[143\]
or SysML [42] can be used to decompose requirements into sub-goals. A formal modelling notation such as OCL, state machines or state charts can be used to expose the implications of requirements. For transformations, state machines may be useful to identify implicit orderings or conflicts of rules which arise because the effect of one rule may enable or disable the occurrence of another.

Requirements have to be prioritized according to their importance and the type of transformation. In general, all requirements must be satisfied according to their importance. Primary requirements have a higher priority compared to the secondary requirements. All primary requirements have to be satisfied first and then secondary requirements have to be satisfied according to their importance. A transformation may still be valid if a secondary requirement is not met, but with a less degree of validation. For instance, in a refinement transformation, the semantics of the source and target models have to be equivalent as the primary requirement and to have the traceability feature as a secondary requirement. In Table 5.3 we have categorised the requirements according to the type of transformation.

Furthermore, there should be no conflict among the requirements. For instance, there is often a conflict between the time, quality and budget of a project. The quality of the target model should be satisfactory with respect to the performance (time, cost and space) of the transformation.
### 5.2. RE Process Adaptation for MT

**TABLE 5.3.** Requirements priority for different types of transformation

<table>
<thead>
<tr>
<th>Category</th>
<th>Primary requirement</th>
<th>Secondary requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refactoring</td>
<td>Model quality improvement</td>
<td>Invariance</td>
</tr>
<tr>
<td></td>
<td>Model-level semantic preservation</td>
<td>Confluence</td>
</tr>
<tr>
<td></td>
<td>Syntactic correctness</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Termination</td>
<td></td>
</tr>
<tr>
<td>Migration</td>
<td>Syntactic correctness</td>
<td>Invertibility</td>
</tr>
<tr>
<td></td>
<td>Model-level semantic preservation</td>
<td>Confluence</td>
</tr>
<tr>
<td></td>
<td>Termination</td>
<td>Traceability</td>
</tr>
<tr>
<td>Refinement</td>
<td>Syntactic correctness</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Model-level semantic preservation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Confluence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Termination</td>
<td>Traceability</td>
</tr>
</tbody>
</table>

### 5.2.3 Specification and Documentation

Regarding the specification and documentation process, the following techniques could be applied for model transformations:

- Structured language template
  Templates can be used to impose a standard structure on the documentation of transformations. There are several templates, the following is an example of an IEEE Standard-830 [114], a well-known template:
5.2. RE Process Adaptation for MT

- Introduction
  - Purpose of the requirements document
  - Scope of the transformation
  - Definition, acronyms and abbreviations
  - References
  - Overview of the remainder of the documents

- General description
  - Transformation perspective
  - Transformation functions
  - Transformation/Transformer characteristics
  - General constraints
  - Assumptions and dependencies
  - Apportioning of requirements

- Specific requirements
  - Functional requirements
  - External interface requirements
  - Performance requirements
  - Design constraints
  - Software quality attributes
  - Other requirements

- Appendices

- Index

- Diagrammatic notations
  Another way of documenting is by using semi-formal specification languages. There are several types of diagrams through which a problem can be presented. Interaction scenario is a useful technique to specify and document transformations. In this technique, a set of possible events are simulated and documented. The aim of this technique is to think about the current problem, different possibilities, assumptions related to these possibilities, action opportunities and risks [60]. Through the SLR and interview-based
study it can be deduced that UML and OCL are popular techniques in model transformations. UML is used as a diagrammatic notation and OCL is often used to express transformation rules.

- Formal specifications

In general, a software application can be seen as a formal description that can be analysed by using logic. Logic allows developers to express reasoning steps explicitly. One of the main roles of requirements engineering is to fill the gap between the informal needs of stakeholders and the formal needs of the software. There are different types of logic which express different aspects of the required transformation.

An approach which seems particularly well-aligned with requirements engineering of model transformations is KAOS [143], which supports requirements elaboration using temporal logic. Temporal logic [113] describes information regarding timing, it identifies permissions and it imposes obligations.

Moreover, another advantage of using logic based approaches is that they are amenable to perform reasoning and analysis tasks automatically which aligns with the nature of transformations. The ‘Cease’ goal pattern of KAOS fits the usual case of refactoring transformations which must remove structures of particular kinds in the model. Each local refactoring requirement can be expressed by such a goal pattern, asserting that each occurrence of a condition \( \varphi \) which should be removed will eventually be removed, and will not be reintroduced:

\[
\begin{align*}
\text{model elements } x_1, \ldots, x_n \text{ satisfy property } \varphi & \Rightarrow \\
\Diamond \Box (x_1, \ldots, x_n \text{ do not satisfy } \varphi)
\end{align*}
\]

Transformation invariants (Inv) can be expressed using the ‘Maintain’ goal pattern according to the assumptions (Asm):

\[Asm \Rightarrow \Box(Inv)\]
5.2. RE Process Adaptation for MT

General postconditions can be expressed using the ‘Achieve’ pattern:

\[ \text{Asm} \Rightarrow \square (Post_1 \land \ldots \land Post_m) \]

Formalised requirements in temporal logic could then be checked for particular implementations using model-checking techniques, as in [121].

Techniques for this stage include: UML and OCL modelling, structured natural language, formal modelling languages. We use SysML with SBVRSE [138] structured English descriptions of individual functional requirements. Structural assertions concerning the source and target languages can be mapped from SBVRSE to UML following the procedure in [56]. We have also defined mappings from behavioural assertions to OCL.

Abstract grammar transformation cases are used to formalise MT requirements in [45]. In UML-RSDS the specification of a transformation consists of one or more UML use cases, each consisting of one or more transformation rules defined by use case postcondition constraints in OCL. Similar structures are available in other MT languages, such as modules with OCL-based rules in ATL and QVT-R.

5.2.4 Validation and Verification

Techniques for this stage include: prototyping with testing, formal requirements inspection, requirements checklists, static analysis, formal modelling and model checking.

The formalised rules produced by the previous stage should be statically checked for internal correctness properties such as definedness and determinacy, which should hold for meaningful rules.

Correct data-dependency relations should hold within a use case definition. For instance, data should be defined before use, and should not be written after it has been used. These checks are performed by the Generate Design option of the UML-RSDS tools, and the results are displayed
on the console window. A prototype implementation can be generated, and its behaviour on a range of test case input models, covering all of the scenarios considered during requirements elicitation, can be checked against stakeholder expectations. Global correctness requirements should be refined to local rule correctness requirements as follows:

- Model-level semantic preservation for an update-in-place transformation is refined into an invariance requirement (that the semantics of the model is equal to its original semantics), and then decomposed into local requirements that this invariant is preserved by each rule application.

- Syntactic correctness of update-in-place transformations is refined to an invariance property that the model satisfies (is conformant with) the metamodel, and then further decomposed into subgoals that each rule application maintains this invariant.

Invariance properties for UML-RSDS, QVT-R and ATL specifications can be checked by proof, using the B formalism \[88\]. Proof can also be used to show that model quality measures are increased by rule applications for refactoring transformations.

When a precise expression of the functional and non-functional requirements has been defined, these can be validated with the stakeholders to confirm that they do indeed accurately express the stakeholders’ intentions and needs for the system. The formalised requirements of a transformation \( \tau: S \rightarrow T \) can also be verified to check that they are consistent as follows:

- The functional requirements must be mutually consistent

- The assumptions and invariant of \( \tau \), and the language constraints of \( S \) must be jointly consistent

- The invariant and postconditions of \( \tau \), and the language constraints of \( T \) must be jointly consistent
5.2. RE Process Adaptation for MT

- Each mapping rule left-hand side (LHS) must be consistent with the invariant, as must each mapping rule right-hand side (RHS).

These consistency properties can be checked using tools such as Z3 or Alloy, given suitable encodings [4, 31]. Model-level semantic preservation requirements can in some cases be characterised by additional invariant properties which the transformation should maintain. For each functional and non-functional requirement, justification should be given as to why the formalised specification satisfies these requirements. For example, to justify termination, some variant quantity $Q: \text{Integer}$ could be identified which is always non-negative and which is strictly decreased by each application of a mapping rule [10]. Formalised requirements in temporal logic could then be checked for particular implementations using model-checking techniques, as in [152].

**Figure 5.3.** Functional requirements decomposition
5.2. RE Process Adaptation for MT

5.2.5 Tool Support for RE in MT

The UML-RSDS tools provide a requirements editor, which uses SysML diagrams to document requirements and their decompositions (Figure 5.3).

The numeric ordering of subgoals indicates sequential composition (for separate model transformations) or relative priorities (for update-in-place transformations). Scenarios are attached to local requirements, and may have three forms of descriptions: informal (text, sketches), semiformal (structured text, concrete grammar rules), formal (OCL, abstract grammar rules). The informal and semiformal descriptions are more suitable for evaluation by stakeholders. Structured English descriptions of the form:

It is necessary that each SCond instance $s$ maps to a $T_1$ instance $t_1$ [and to a $T_j$ instance $t_j]^*$ such that $P(s, t_1, ..., t_n)$

Each $S_1$ is considered to be a SCond if it has $SP$

are formalised as:

$S_1::$

$SP \Rightarrow T_1 \rightarrow \exists( t_1 | ... T_n \rightarrow \exists( t_n | P(self, t_1, ..., t_n) ) ... )$

SCond is a linguistic term to represent $S_1 \rightarrow select (SP)$. The constraint expresses that each such $S_1$ instance maps to $T_1, T_2, ..., T_n$ instances such that the condition $P$ holds.

Requirements and scenarios are linked to the use cases which are defined to satisfy these requirements and scenarios. A use case postcondition can be derived as the ordered conjunction of its formalised scenarios.

Requirements specification is supported by the use of OCL constraints to specify the preconditions and postconditions of use cases and operations. Validation and verification is supported by several static analysis
5.3. A Framework for Selecting Suitable RE Techniques

There are several methods and techniques proposed by the requirements engineering community, however selecting an appropriate set of requirements engineering techniques for a project is a challenging issue. Most of these methods and techniques were designed for a specific purpose and none cover the entire RE process. Researchers have classified RE techniques and categorised them according to their characteristics. For instance, Hickey et al. [54] proposed a selection model of elicitation techniques, Maiden et al. [101] came up with a framework for requirements acquisition methods and techniques. However, lack of support for selecting the most appropriate set of techniques for a software project has made requirements engineering one of the most complex parts of the software engineering process.

While there are some approaches regarding the selection of RE technique for general projects, there is no systematic guideline available for MT projects. Traditionally, the selection of RE techniques is mainly based on personal preference rather than on characteristics and specifications of the project, and MT projects are no exception. From our SLR and interview-based study, we discovered that there is a lack of guidance on RE in MT in general, which includes lack of guidance regarding the selection of RE techniques for certain activities in MT context.

This section focuses on a description of our proposed framework for selecting suitable RE techniques designed for MT. It aims to help MT developers to find the most suitable RE technique in an MT project for any particular requirement. This framework is based on in-depth
5.3. A Framework for Selecting Suitable RE Techniques

research (SLR and interview-based study) into MT applications as well as RE techniques via analysis, synthesis, and classification mechanisms. Our proposed framework allows MT developers to:

- Identify MT project attributes and RE technique attributes and a possible link between them
- Provide a systematic RE process designed for MT developments

This framework for selecting a suitable RE technique has been applied in two real case studies as presented in Chapter 6, which shows the framework provided an effective decision support for RE selection with a positive outcome.

We use Basili’s GQM [10] framework to define the framework and to identify the metrics of the RE technique in order to make precise the selection criteria. The top-level goal is “identifying the most relevant requirements engineering techniques for a particular MT project”. This goal could be decomposed into three questions:

- **Q1.** How do the characteristics of the MT project requirements affect the relevance of the RE techniques for the project?
- **Q2.** How does the size, cost and other attributes of the project affect the relevance of RE techniques for the project?
- **Q3.** How does the experience level of the MT developers in the RE techniques affect their relevance for the project?

For **Q1**, we adapt and extend the measures of RE techniques’ relevance designed by Jiang [62]. For **Q2**, we identify nine project attributes and evaluate the relevance of the RE techniques for each of these. For **Q3**, we assign a measure in [0, 1] to represent the experience level. We consider separately the relevance of techniques in each stage: Domain Analysis & Requirements Elicitation, Evaluation & Negotiation, Specification & Documentation and Validation & Verification. Overall, the
5.3. A Framework for Selecting Suitable RE Techniques

relevance of a technique is the product of the three measures for $Q_1$, $Q_2$ and $Q_3$.

Classification of RE techniques have a direct relation with the type of the proposed project, the organization and the internal attributes of a specific technique. In general, a project is assigned to an organization in order to be developed. Usually the software developing organization is selected according to the type of project. In the following section, we present attributes of these three factors, namely that of: (i) the technique (technique attribute), (ii) the project (project attribute) and (iii) the organization (organizational attribute) in order to identify the most well-suited set of techniques to use in MT for a particular type of project.

5.3.1 Technique Attribute

As mentioned earlier, multiple RE techniques can be used during the RE stage. Each technique has some attributes that would render it more suitable for a particular RE activity. Identifying the technique attributes could be very useful as they allow us to compare the different techniques.

We have identified 33 attributes from which 23 were defined by [62]. These attributes are categorized according to the RE phases that they belong to based on Sommerville et al. [133]. These attributes are selected based on characteristics of RE techniques as well as other researchers’ criteria and frameworks [100, 101, 133]. The following are the attributes that we have identified according to MT characteristics: ability to elicit MT requirements, ability to identify MT stakeholders, ability to analyse and to model requirements with relevant MT notations, ability to identify accessibility of the transformation, ability to prioritize requirements according to the transformation, ability to use re-usability of MT requirements, ability to specify completeness of semantics and notations, ability to write precise requirements using MT notation, ability to support MT language, maturity of supporting tool.

For instance, some RE techniques are well-suited for identifying non-functional requirements. Therefore, if non-functional requirements in
5.3. A Framework for Selecting Suitable RE Techniques

a particular project have high priority, then the attribute of *ability to help identify non-functional requirements* is important and applying the appropriate RE technique such as NFR framework [26] to find non-functional requirements would be necessary. In Table 5.4 we have adapted the attributes of [62] and have made additions to these attributes to make them specific for MT.
### 5.3. A Framework for Selecting Suitable RE Techniques

#### TABLE 5.4. RE technique attributes and classifications adapted and extended from [62]

<table>
<thead>
<tr>
<th>ID</th>
<th>Categories</th>
<th>Attributes of techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Domain Analysis &amp; Requirements</td>
<td>Ability to elicit MT requirements</td>
</tr>
<tr>
<td>2</td>
<td>Elicitation</td>
<td>Ability to facilitate communication</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Ability to understand social issues</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Ability to get domain knowledge</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Ability to get implicit knowledge</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Ability to identify MT stakeholders</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Ability to identify non-functional requirements</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Ability to identify viewpoints</td>
</tr>
<tr>
<td>9</td>
<td>Evaluation &amp; Negotiation</td>
<td>Ability to model and understand requirements</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Ability to analyse and to model requirements with relevant MT notations</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Ability to analyse non-functional requirements</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>Ability to facilitate negotiation with customers</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>Ability to prioritize requirements according to stakeholders need</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>Ability to prioritize requirements according to the transformation</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>Ability to identify accessibility of the transformation</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Ability to model interface requirements</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>Ability to use re-usability of MT requirements</td>
</tr>
<tr>
<td>18</td>
<td>Specification &amp; Documentation</td>
<td>Ability to represent MT requirements</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>Ability to verify requirements</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>Ability to specify completeness of semantics and notations</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>Ability to write precise requirements using MT notation</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>Ability to write complete requirements</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>Ability to consider requirements management</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>Ability to design highly modular systems</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>Ability to implement used notation</td>
</tr>
<tr>
<td>26</td>
<td>Validation &amp; Verification</td>
<td>Ability to identify ambiguous requirements</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>Ability to identify inconsistency and conflict</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>Ability to identify incomplete requirements</td>
</tr>
<tr>
<td>29</td>
<td>Other aspects</td>
<td>Ability to support MT language</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>Maturity of supporting tool</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>Learning curve</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>Application cost</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>Complexity of technique</td>
</tr>
</tbody>
</table>
5.3. A Framework for Selecting Suitable RE Techniques

Tables 5.5–5.8 present technique attributes and sample assessment data of techniques provided by Jiang [61]. These scored attributes are static and fixed, independent of particular projects and represent fitness \( V(X, Y) \) of \( X \) (technique) for \( Y \) (specific requirement attribute). The technique scoring tables are based on Jiang’s survey of many RE projects. Jiang’s work has been published in several well-known conferences and journals and it is highly credible.

It is worth mentioning that these RE technique attributes are only a sample of available RE techniques and attributes. Depending on the nature of any given project, they can be extended and modified. For this research, we are only taking a sample of 33 RE technique attributes, however there is no limit for having further attributes.

**TABLE 5.5.** Domain Analysis & Requirements Elicitation technique attributes evaluation \( V(a_x, t) \)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Interview</th>
<th>Questionnaire</th>
<th>Document Mining</th>
<th>Brainstorming</th>
<th>Proto-typing</th>
<th>Scenario</th>
<th>Ethno Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliciting MT requirements</td>
<td>1</td>
<td>0.8</td>
<td>1</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>Facilitating communication</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.8</td>
<td>0.8</td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>Understanding social issues</td>
<td>0.8</td>
<td>1</td>
<td>0.8</td>
<td>0.4</td>
<td>0.2</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>Getting domain knowledge</td>
<td>0.6</td>
<td>0.6</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>Getting implicit knowledge</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>Identifying MT stakeholders</td>
<td>1</td>
<td>0.8</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>Identifying non-functional requirents</td>
<td>1</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Identifying viewpoints</td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
<td>0.8</td>
<td>0</td>
<td>0.8</td>
<td>0.4</td>
</tr>
</tbody>
</table>
### 5.3. A Framework for Selecting Suitable RE Techniques

**TABLE 5.6. Requirements Evaluation & Negotiation technique attributes evaluation $V(a, t)$**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Proto-typing</th>
<th>Scenario</th>
<th>UML</th>
<th>Goal-oriented Analysis</th>
<th>Functional Decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling MT requirements</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Analysing requirements with relevant MT notations</td>
<td>0.6</td>
<td>1</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Analysing non-functional requirements</td>
<td>0.2</td>
<td>0.2</td>
<td>0</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>Facilitating negotiation with stakeholders</td>
<td>0.8</td>
<td>0.6</td>
<td>0.8</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>Prioritizing requirements based on stakeholders</td>
<td>0.2</td>
<td>0.4</td>
<td>0</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Identifying accessibility of the transformation</td>
<td>0.8</td>
<td>0.8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>Modeling interface requirements</td>
<td>0.6</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Re-usability of MT requirements</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
</tr>
</tbody>
</table>

**TABLE 5.7. Requirements Specification & Documentation technique attributes evaluation $V(a, t)$**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>SysML</th>
<th>KAOS</th>
<th>Structured language template</th>
<th>SADT</th>
<th>Evolutionary Prototyping</th>
<th>UML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representing MT requirements</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>Requirements verification</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.4</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Semantic completeness</td>
<td>0.8</td>
<td>1</td>
<td>0.4</td>
<td>0.6</td>
<td>0.2</td>
<td>0.8</td>
</tr>
<tr>
<td>Representing requirements using MT notations</td>
<td>0.6</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>Writing complete requirements</td>
<td>0.8</td>
<td>0.8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.4</td>
<td>0.8</td>
</tr>
<tr>
<td>Requirements management</td>
<td>0.8</td>
<td>0.4</td>
<td>0.6</td>
<td>1</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>Designing highly modular systems</td>
<td>0.8</td>
<td>0.6</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>Implementability of the notation(s)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>
5.3. A Framework for Selecting Suitable RE Techniques

**TABLE 5.8.** Requirements Validation & Verification technique attributes evaluation $V(a_x, t)$

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Inspection</th>
<th>Desk-Checks</th>
<th>Rapid Prototyping</th>
<th>Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifying ambiguous requirements</td>
<td>0.4</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>Identifying inconsistency and conflict</td>
<td>0.4</td>
<td>1</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>Identifying incomplete requirements</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

The set $T$ of all RE techniques to be considered (e.g. interview, prototype) in each category (Domain Analysis & Requirements Elicitation, Evaluation & Negotiation, Specification & Documentation and Validation & Verification) should be identified. For each requirement technique that has been identified for the project, the RE technique $t \in T$, a value $RA(t)$ (Requirement Attribute of a technique) is calculated, which represents the suitability of applying $t$ in the project, which is based on the technique attributes. The function $RA : T \mapsto [0, 1]$ is defined as:

$$RA(t) = \frac{\sum_{a_x \in A} I(a_x) \times V(a_x, t)}{\sum_{a_x \in A} I(a_x)}$$

This expresses that ‘there are (\sum) attributes $a_x \in A$, important (I) to the project and for which $t$ is relevant (V)’. Normalization can be defined by dividing the result by $\sum(I)$.

In the definition:

- The set of all technique attributes in the MT project (i.e. facilitating communication, identifying MT stakeholders) is $A$. For instance, $A = \{eliciting MT requirements, facilitating communication, \ldots, identifying incomplete requirements\}$.

- $I(a_x)$ which is a value in the range $[0, 1]$, represents the importance of an attribute $a_x \in A$ for the project. A low $I(a_x)$ value for an
attribute $a_x \in A$ means $a_x$ is not important for the MT project and a high $I(a_x)$ value for an attribute $a_x \in A$ represents high importance for the project. The assignment of $I(a_x)$ to each $a_x \in A$ is done by MT developers according to the initial project description and the stakeholders. For instance, in a project where the stakeholders are not accessible and “documentation” is identified as an important requirement, then the technique attribute “identifying MT stakeholders” is assigned a lower $I(a_x)$ value (than $I(a_x)$ of documentation), because it is a secondary task compared to the documentation requirement.

- $V(a_x, t)$ is a function $V : A \times T \mapsto [0, 1]$ which given a technique attribute and an RE technique, assigns a $[0, 1]$ value. These values are static and fixed, independent of the project and are based on the technique attribute measures of [61] as well as other attributes that we have added to make them specific for MT for this research. Tables 5.5 5.6 5.7 5.8 give examples of these adapted attribute measures.
5.3. A Framework for Selecting Suitable RE Techniques

5.3.2 Project Attribute

A transformation project’s attribute is also an important factor in selecting the most suitable RE techniques. Each project has a set of attributes and the priority of each attribute may vary based on the characteristics of a project. For instance, the category of a project that it belongs to is an attribute, therefore RE techniques for a category of safety-critical system may vary from a non-critical system. In this research, we have identified nine attributes, which shall be analysed in more detail according to the type of transformation project. These attributes are only a sample of all possible existing attributes. According to [36, 63, 107, 155], these attributes are considered important factors as their values determine the essential characteristics of the software project. We have defined some transformation project attributes in more detail as follows:

Size

- **Very Big**: when the estimated number of transformation rules are more than 300
- **Big**: when the estimated number of transformation rules are between 150 and 300
- **Medium**: when the estimated number of transformation rules are between 100 and 150
- **Small**: when the estimated number of transformation rules are between 50 and 100
- **Very Small**: when the estimated number of transformation rules are less than 50
5.3. A Framework for Selecting Suitable RE Techniques

**Volatility**

*Very High:* transformation requirements keep changing throughout the entire development (more than 50% change of requirements)

*High:* transformation requirements keep changing throughout the entire development (25%-50% change of requirements)

*Medium:* Some of the requirements change during the development (10%-25% change of requirements)

*Low:* A few requirements may change during the development (5%-10% change of requirements)

*Very Low:* Change of requirements is unlikely to happen

**Complexity**

*Very High:* transformation correctness, completeness and effectiveness are very complicated (at least three complicating factors apply)

*High:* transformation correctness, completeness and effectiveness are complicated (at least two complicating factors apply)

*Medium:* transformation correctness, completeness and effectiveness are medium level (at least one complicating factor applies)

*Small:* transformation correctness, completeness and effectiveness are clear (no complicating factors, some non-trivial functionality)

*Very small:* transformation correctness, completeness and effectiveness are easy to achieve (only simple processing is present, e.g. copying of data)

---

1 Due to factors such as (i) complex rule logic, (ii) repeated refactoring process, (iii) complex computations, (iv) non-standard processing (e.g. genetic algorithm), (v) use of multiple MT languages, (vi) bidirectional or change propagation
5.3. A Framework for Selecting Suitable RE Techniques

**Relationship**  
*Very High:* there is a very good and constant interaction amongst the developer and the customer (the customer is directly available when required)  
*High:* there is a good and constant interaction amongst the developer and the customer (the customer is available with delay of one day)  
*Medium:* there is some contact between the developer and the customer when necessary (there is limited access to the customer (delay of > day))  
*Low:* there are few meetings between the two parties, only when essential (there is very limited access to the customer (delay > week))  
*Very Low:* there is no contact between the customer and developer throughout the development (no access to the customer)  

**Safety**  
*Very High:* there is a very high likelihood that the transformation will have safety consequences (it will be used to produce, modify, or analyse safety-critical systems)  
*High:* there is a high likelihood that the transformation will have safety consequences (it may be used to produce, modify, or analyse safety-critical systems)  
*Medium:* there is moderate likelihood that the transformation will have safety consequences (it will be used to produce, modify, or analyse safety-related systems but not safety-critical system)  
*Low:* there is low possibility that the transformation could cause any danger (it may be used to produce, modify, or analyse safety-related systems but not safety-critical system)  
*Very Low:* the transformation has no possibility of causing any danger (it will not be used for any safety-related systems)
5.3. A Framework for Selecting Suitable RE Techniques

<table>
<thead>
<tr>
<th>Quality</th>
<th>Very High: the transformation has a very high level of functionality, reliability and usability requirements ($\geq 100$ requirements)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>High: the transformation has a high level of functionality, reliability and usability requirements ($\geq 50$ requirements)</td>
</tr>
<tr>
<td></td>
<td>Medium: the transformation has a medium level of functionality, and usability requirements ($\geq 20$ requirements)</td>
</tr>
<tr>
<td></td>
<td>Low: there are low reliability, etc. requirements ($\geq 10$ requirements)</td>
</tr>
<tr>
<td></td>
<td>Very Low: there are very low levels of reliability, etc. requirements ($&lt; 10$ requirements)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Very High: the transformation has very restrictive development time constraints (less than 5% extension is possible)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>High: the transformation has a high level of development time constraints (less than 10% extension is possible)</td>
</tr>
<tr>
<td></td>
<td>Medium: the transformation has a medium level of development time constraints (less than 20% extension is possible)</td>
</tr>
<tr>
<td></td>
<td>Low: the transformation has low development time constraints (up to 50% extension is possible)</td>
</tr>
<tr>
<td></td>
<td>Very Low: the transformation has very low development time constraints (more than 50% extension is possible)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Very High: the budget is very tight (less than 5% extension is possible)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>High: the budget is tight (less than 10% extension)</td>
</tr>
<tr>
<td></td>
<td>Medium: the transformation has a limited budget (less than 20% extension)</td>
</tr>
<tr>
<td></td>
<td>Low: the transformation has the budget to cover different aspects and unforeseen circumstances (up to 50% extension permitted)</td>
</tr>
<tr>
<td></td>
<td>Very Low: the budgets are flexible (more than 50% extension is possible)</td>
</tr>
</tbody>
</table>
5.3. A Framework for Selecting Suitable RE Techniques

<table>
<thead>
<tr>
<th>Domain knowledge</th>
<th>Very High: developers have good background knowledge and previous experience regarding the domain (at least 5 years experience)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>High: there is a good amount of knowledge and experience regarding the domain (at least 2 years experience)</td>
</tr>
<tr>
<td></td>
<td>Medium: there is some background knowledge and experience regarding the domain (at least 1 year experience)</td>
</tr>
<tr>
<td></td>
<td>Low: the amount of experience and knowledge regarding the domain is low (some experience, less than 1 year)</td>
</tr>
<tr>
<td></td>
<td>Very Low: there is no experience or knowledge about the domain (no experience)</td>
</tr>
</tbody>
</table>

Table 5.9 shows these transformation project attributes’ weightings in a tabular format.

**TABLE 5.9. Project attributes weighting**

<table>
<thead>
<tr>
<th>Project attribute C</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very high/large</td>
<td>0.8 - 1</td>
</tr>
<tr>
<td>High/large</td>
<td>0.6 - 0.8</td>
</tr>
<tr>
<td>Medium</td>
<td>0.4 - 0.6</td>
</tr>
<tr>
<td>Low/small</td>
<td>0.2 - 0.4</td>
</tr>
<tr>
<td>Very low/small</td>
<td>0 - 0.2</td>
</tr>
</tbody>
</table>

These MT attributes are only a selection of available project attributes. Depending of the nature of a given project, they can be extended and modified. The value given for each transformation project attribute is assigned by the developer according to the initial project description. For this research, we are only taking a sample of nine MT project attributes, however there is no limit for having further attributes.

In the following section, we will show how to identify the (MT) project attributes. For each requirement \( r \) identified within the project, each RE technique \( t \in T \) is assigned a value \( PD(t) \) (for Project Description) representing the suitability of applying \( t \) to fulfil this requirement, based
5.3. A Framework for Selecting Suitable RE Techniques

on the project’s descriptions. The function $PD : T \mapsto [0, 1]$ is defined as:

$$
PD(t) = \prod_{d_x \in D} \begin{cases} 
1 - \mathcal{W}(d_x) & \text{if } d_x \in ID_t \\
\mathcal{W}(d_x) & \text{otherwise}
\end{cases}
$$

This expresses that ‘the technique $t$ is relevant based upon all ($\prod$) the project attributes $d_x \in D$’.

Where:

- The set of all project descriptors (e.g. size, complexity) is $D$. In this thesis, we assume $D = \{\text{size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}\}$. It should be noted that these are only an arbitrary selection of all possible existing attributes of a project.

- $\mathcal{W}(d_x)$ is a function $\mathcal{W} : D \mapsto [0, 1]$ which represents the magnitude of a specific descriptor in the project. For example, for $d = \text{cost}$, a high value represents that the budget of the project is tight, while a low value indicates that the budget is flexible. Then for $d = \text{estimated size}$, a high value means that the project involves a large number of transformation rules while a low value indicates a small number of rules is involved.

- $ID_t \subseteq D$ is a set containing all descriptors with inverse impact for a specific RE technique $t$. More specifically, for each $d \in ID_t$, the higher the value of $\mathcal{W}(d_x)$ the more negative the impact of applying $t$ in that project. An example of such a descriptor for technique “interview” is time, where the higher the value of $\mathcal{W}(\text{time})$ in a specific project, the more negative the relevance of interviewing as a technique for this project. Table 5.10 is an example of RE technique weight descriptor values, where $-1$ indicates that $1 - \mathcal{W}(d_x)$ is used and 1 idicates that $\mathcal{W}(d_x)$ is used. This is static and fixed, independent of particular projects.
### 5.3. A Framework for Selecting Suitable RE Techniques

#### TABLE 5.10. Technique weight descriptor values

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Size</th>
<th>Complexity</th>
<th>Volatility</th>
<th>Relation</th>
<th>Safety</th>
<th>Quality</th>
<th>Time</th>
<th>Cost</th>
<th>Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interview</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Questionnaire</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Scenario</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Prototype</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Rapid prototyping</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For a detailed description of a particular RE technique and its correlating project attributes refer to Appendix C.

#### 5.3.3 Organisational Attribute

Every software developing organization applies the RE process in a different manner. This difference is caused by the behaviour of developers and stakeholders involved in the project. This behaviour is influenced by different factors of the organization such as: size, culture, policy and complexity. These factors have a direct effect on the way the RE process is performed. For instance, in a small organization, new technologies and expensive RE techniques may not be the first choice due to the high cost of it, whereas in a large and complex organization more flexible and disciplined techniques are required to do RE tasks. Although there is no limit to the attributes of an organization, we have identified the level of experience and familiarity with a particular RE technique as the main organization attribute for this research study.

In this section, we are going to identify the level of experience regarding particular RE techniques for a particular MT requirement. Evaluating the degree $E$ (for Experience) of experience/expertise regarding the RE technique $t$ available in the development team. $E : T \rightarrow [0, 1]$ is a function where $E(t)$ represents the level of experience and practical and theoretical knowledge of the developer regarding $t$. For instance, depending on the number of projects in which the developer has applied a particular RE technique, the weight of $E$ may vary. If the developer
5.4. Application Framework Example

has used a particular RE technique in more than 20 different projects, then the weight of \( E \) should be closer to 1.

It is worth mentioning that the given values of the technique attribute, the project attribute and the organization attribute may be modified according to the progress of the MT project and the MT developer’s learning capability regarding the project’s domain and RE techniques.

Once all attributes have been identified, we can calculate the technique suitability score, \( S(t) \), of a particular technique \( t \). By using \( S(t) \), the overall suitability score of a particular RE technique \( t \in T \) can be determined, and hence it would be possible to define a ranking of techniques \( t \) based on their suitability scores \( S(t) \) for use in the project. Techniques can thus be ranked according to their suitability score. The higher the value of \( S(t) \) the more suitable is that particular technique. \( S(t) \) is defined in terms of the requirement attribute score \( RA(t) \), the project description score \( PD(t) \), and the experience score \( E(t) \) of RE technique \( t \) as follows:

\[
S(t) = RA(t) \times PD(t) \times E(t)
\]

This expresses that ‘the suitability of a technique, \( S(t) \), is based upon the requirement attributes of that technique, \( RA(t) \), the project description attributes of that technique, \( PD(t) \), and the experience attributes of that technique, \( E(t) \)’.

5.4 Application Framework Example

Our overall procedure for selecting RE techniques for a MT project is presented in a running example consisting of six sections.

Example 1 (Running Example). We will choose a refactoring type of MT project as an example and will apply our proposed technique framework step by step. The general idea behind refactoring is to improve the structure of the model to make it easier to understand, and to make
5.4. Application Framework Example

it more maintainable and amenable to change. We describe an example [92] of an in-place endogenous transformation which refactors class diagrams to improve their quality by removing redundant feature declarations where: (i) there is a complex rule logic and (ii) there are repeated refactoring steps. In this section, we are going to describe a systematic procedure by which the requirements attribute of, RA, a technique, t, in the MT project is identified as follows, using the formula:

\[
RA(t) = \frac{\sum_{a_x \in A} I(a_x) \times V(a_x, t)}{\sum_{a_x \in A} I(a_x)}
\]

**Example 2** (Continuation of Example 1). In this section, we apply the framework to find the RA value for the refactoring example:

- **category: Domain Analysis & Requirements Elicitation**
  - The techniques in T_elicitation are our chosen sample because the developers had some experience with these techniques.
  - T_elicitation = \{interview, prototyping, questionnaire, document mining, brainstorming, scenario, ethno methodology\}
  - t_1 = interview, t_2 = questionnaire are chosen arbitrarily and any number of these techniques can be chosen.
  - I(a) has a dynamic weighting which can be assigned from a range \([0,1]\) according to the importance of the technique attributes, A_1 and A_2, which is determined by the developers according to the initial project description and stakeholders.
  - I(a) value is 1 for A_1, 0.8 for A_2 and 0 for the remaining attributes:

\[2\] The set of attributes \((A_1, A_2)\) has been grouped according to stakeholder information and developer understanding.
5.4. Application Framework Example

* $A_1 = \{\text{eliciting MT requirements, getting domain knowledge, identifying non-functional requirements}\}$
* $A_2 = \{\text{identifying MT stakeholders, facilitating communications}\}$
* $A_3 = A - (A_1 + A_2)$

$\nu(a_x, t_1)$ for $A_1$ is $\{1, 0.6, 1\}$
$\nu(a_x, t_1)$ for $A_2$ is $\{1, 1\}$
$\nu(a_x, t_2)$ for $A_1$ is $\{0.8, 0.6, 0.6\}$
$\nu(a_x, t_2)$ for $A_2$ is $\{0.8, 1\}$

\[
\text{RA}(t_1) = \frac{(1 \times 1) + (1 \times 0.6) + (1 \times 1) + (0.8 \times 1) + (0.8 \times 1)}{4.6} = 0.91
\]

\[
\text{RA}(t_2) = \frac{(1 \times 0.8) + (1 \times 0.6) + (1 \times 0.6) + (0.8 \times 0.8) + (0.8 \times 1)}{3.8} = 0.9
\]

• category: Evaluation & Negotiation

- $T_{Evaluation} = \{\text{prototyping, UML, scenario, goal-oriented analysis}\}$
- $t_3 = \text{scenario}$, $t_4 = \text{prototyping}$
- $I(a)$ value is 1 for $A_1$, 0.8 for $A_2$ and 0 for the remaining attributes:
  * $A_1 = \{\text{modelling MT requirements, identifying accessibility of the transformation}\}$
  * $A_2 = \{\text{prioritizing requirements based on stakeholders, analysing non-functional requirements}\}$
5.4. Application Framework Example

* $A_3 = A - (A_1 + A_2)$

- $v(a_x, t_3)$ for $A_1$ is $\{1, 0.8\}$
- $v(a_x, t_3)$ for $A_2$ is $\{0.4, 0.2\}$
- $v(a_x, t_4)$ for $A_1$ is $\{0.8, 0.8\}$
- $v(a_x, t_4)$ for $A_2$ is $\{0.2, 0.2\}$

\[
RA(t_3) = \frac{[(1 \times 1) + (1 \times 0.8) + (0.8 \times 0.4) + (0.8 \times 0.2)]}{2.4} = \frac{2.28}{2.4} = 0.95
\]

\[
RA(t_4) = \frac{[(1 \times 0.8) + (1 \times 0.8) + (0.8 \times 0.2) + (0.8 \times 0.2)]}{2} = \frac{1.92}{2} = 0.96
\]

- **category:** Specification & Documentation

  - $T = \{\text{interview, UML, evolutionary prototyping, questionnaire, formal methods, structured language template, checklist}\}$
  - $t_5 = \text{evolutionary prototyping}, t_6 = \text{UML}$
  - $I(a)$ value is 1 for $A_1$, 0.8 for $A_2$ and 0 for the remaining attributes:
    * $A_1 = \{\text{requirements verification, semantic completeness}\}$
    * $A_2 = \{\text{representing requirements using MT notations}\}$
    * $A_3 = A - (A_1 + A_2)$
  - $v(a_x, t_5)$ for $A_1$ is $\{0.8, 0.2\}$
  - $v(a_x, t_5)$ for $A_2$ is $\{0.2\}$
  - $v(a_x, t_6)$ for $A_1$ is $\{0.8, 0.8\}$
5.4. Application Framework Example

- $v(a_x, t_5)$ for $A_2$ is $\{1\}$

\[
RA(t_5) = \frac{[1 \times 0.8 + (1 \times 0.2) + (0.8 \times 0.2)]}{1.2} = 0.96
\]

- $RA(t_6) = \frac{[1 \times 0.8 + (1 \times 0.8) + (0.8 \times 1)]}{2.6} = 0.92$

- **category:** Validation & Verification

- $T = \{\text{interview, UML, rapid prototyping, questionnaire, formal methods, structured language template, checklist}\}$

- $t_7 = \text{rapid prototyping}$

- $I(a)$ value is 1 for $A_1$, 0.8 for $A_2$ and 0 for the remaining attributes:
  * $A_1 = \{\text{identifying incomplete requirements, identifying inconsistency and conflict}\}$
  * $A_2 = \{\text{identifying ambiguous requirements}\}$
  * $A_3 = A - (A_1 + A_2)$

- $v(a_x, t_7)$ for $A_1$ is $\{0.8, 0.8\}$

- $v(a_x, t_7)$ for $A_2$ is $\{0.4\}$

- $RA(t_7) = \frac{[1 \times 0.8 + (1 \times 0.8) + (0.8 \times 0.4)]}{2} = 0.96$
5.4. Application Framework Example

Example 3 (Continuation of Example 2). In this section, we are going to apply the framework to calculate the value for PD(t) on the refactoring example based on the following formula:

\[
PD(t) = \prod_{d_x \in D} \begin{cases} 
1 - W(d_x) & \text{if } d_x \in ID_t \\
W(d_x) & \text{otherwise}
\end{cases}
\]

- According to the transformation project attributes, the size of this transformation is small, there are two complicating factors, there is 5%-10% change of requirements, there is limited access to the customer, the transformation may be used to refactor safety-related systems, there exists approximately 20 requirements, up to 50% extension is possible regarding the time, the budget is tight (less than 10% extension) and developers are quite familiar with the domain (at least one year of experience). In other words, we have the following values:

- size: small (0.2), complexity: high (0.8), volatility: low (0.2), customer-developer relationship: low (0.4), safety: low (0.2), quality: medium (0.5), time: low (0.2), cost: high (0.8), domain knowledge: medium (0.6)

- \(D = \{\text{size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}\}\)

- \(ID_{\text{interview}} = \{\text{time, cost}\}\)

- \(ID_{\text{questionnaire}} = \{\text{volatility, relationship}\}\)

- \(ID_{\text{scenario}} = \{\text{size, volatility, time, cost}\}\)

- \(ID_{\text{prototype}} = \{\text{size, complexity, volatility, cost}\}\)

- \(ID_{\text{evolutionaryprototyping}} = \{\text{size, complexity, volatility, cost}\}\)

\(^3\) The values have been determined from the transformation project attributes weighting (Table 5.9)
5.4. Application Framework Example

- \( ID_{UML} = \{\text{complexity, volatility}\} \)
- \( ID_{rapidprototyping} = \{\text{size, complexity, cost}\} \)
- \( PD(t_1) = (0.2) \times (0.8) \times (0.2) \times (0.4) \times (0.2) \times (0.5) \times (0.8) \times (0.2) \times (0.6) = 0.0001 \)
- \( PD(t_2) = (0.2) \times (0.8) \times (0.8) \times (0.2) \times (0.5) \times (0.8) \times (0.6) = 0.0029 \)
- \( PD(t_3) = (0.8) \times (0.2) \times (0.8) \times (0.2) \times (0.5) \times (0.8) \times (0.2) \times (0.6) = 0.0019 \)
- \( PD(t_4) = (0.8) \times (0.2) \times (0.8) \times (0.4) \times (0.2) \times (0.5) \times (0.2) \times (0.6) = 0.0001 \)
- \( PD(t_5) = (0.8) \times (0.2) \times (0.2) \times (0.8) \times (0.4) \times (0.2) \times (0.5) \times (0.2) \times (0.6) = 0.0001 \)
- \( PD(t_6) = (0.2) \times (0.2) \times (0.8) \times (0.4) \times (0.2) \times (0.5) \times (0.2) \times (0.8) \times (0.6) = 0.0001 \)
- \( PD(t_7) = (0.8) \times (0.2) \times (0.2) \times (0.2) \times (0.4) \times (0.2) \times (0.5) \times (0.2) \times (0.2) \times (0.6) = 0.00003 \)

**Example 4** (Continuing Example 3). Evaluating the degree \( E \) (for Experience) of experience/expertise regarding the RE technique \( t \) available in the development team. \( E : T \rightarrow [0, 1] \) is a function where \( E(t) \) represents the level of experience and practical and theoretical knowledge of the developer regarding \( t \). The value of \( E \) is established based on developer experience and is determined by the developer.

**Example 5** (Continuing Example 4). For this refactoring example, here we list the suitability score \( S(t) \) of the different techniques \( t \in T \) that have been identified throughout this example. \( S(t) \) values reflect the relevance of a particular technique, the higher the \( S(t) \) value the higher the priority of that technique.
5.4. Application Framework Example

- \( S(t_1) = 0.91 \times 0.0001 \times 1 = 0.00009 \)
- \( S(t_2) = 0.9 \times 0.0029 \times 0.8 = 0.002 \)
- \( S(t_3) = 0.95 \times 0.0019 \times 0.6 = 0.001 \)
- \( S(t_4) = 0.96 \times 0.0001 \times 0.8 = 0.00007 \)
- \( S(t_5) = 0.96 \times 0.0001 \times 0.8 = 0.00007 \)
- \( S(t_6) = 0.92 \times 0.0001 \times 0.8 = 0.00007 \)
- \( S(t_7) = 0.96 \times 0.00003 \times 0.8 = 0.00002 \)

Table 5.11 shows the overall calculation of the metric framework of this example.

<table>
<thead>
<tr>
<th>RE technique</th>
<th>RA(t)</th>
<th>PD(t)</th>
<th>E(t)</th>
<th>S(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interview ((t_1))</td>
<td>0.91</td>
<td>0.0001</td>
<td>1</td>
<td>0.00009</td>
</tr>
<tr>
<td>Questionnaire ((t_2))</td>
<td>0.9</td>
<td>0.0029</td>
<td>0.8</td>
<td>0.002</td>
</tr>
<tr>
<td>Scenario ((t_3))</td>
<td>0.95</td>
<td>0.0019</td>
<td>0.6</td>
<td>0.001</td>
</tr>
<tr>
<td>Prototyping ((t_4))</td>
<td>0.96</td>
<td>0.0001</td>
<td>0.8</td>
<td>0.00007</td>
</tr>
<tr>
<td>Evolutionary prototyping ((t_5))</td>
<td>0.96</td>
<td>0.0001</td>
<td>0.8</td>
<td>0.00007</td>
</tr>
<tr>
<td>UML ((t_6))</td>
<td>0.92</td>
<td>0.0001</td>
<td>0.8</td>
<td>0.00007</td>
</tr>
<tr>
<td>Rapid prototyping ((t_7))</td>
<td>0.96</td>
<td>0.00003</td>
<td>0.8</td>
<td>0.00002</td>
</tr>
</tbody>
</table>

**Example 6** (Example Result). In this section, we present the result of applying our proposed suitability technique framework on the refactoring example according to the standard RE stages. The properties for this type of transformation are: endogenous, model-to-model, many-to-many (source to target model), horizontal, semantic preservation, explicit control/rule application scoping, rule iteration, traceable and that it is a unidirectional transformation.
5.4. Application Framework Example

- Domain Analysis & Requirements Elicitation for Refactoring

The initial requirements statement is to refactor a UML class diagram to remove all cases of duplicated attribute declarations in sibling classes (classes which have a common parent). This statement is concerned purely with functional behaviour. Through structured interviews with the customer (and with the end users of the refactored diagrams and the development team) we can further uncover non-functional requirements as follows: *efficiency*, the refactoring should be able to process diagrams with 1000 classes and 10,000 attributes in a practical time (less than 5 minutes); *correctness*, the start and end models should have equivalent semantics; *minimality*, the number of new classes introduced should be minimized to avoid introducing superfluous classes into the model; *confluence*, would be desirable but is not mandatory.

The functional requirements can also be clarified and more precisely scoped through the interview process. A global functional requirement is the invariance of the class diagram language constraints meaning that there is no multiple inheritance, and no concrete class with a subclass. It is not proposed to refactor associations because of the additional complications this would cause for the developers. Only attributes are to be considered. Through scenario analysis using concrete grammar sketches, the main functional requirement is decomposed into three cases: (i) where all (two or more) direct subclasses of one class have identical attribute declarations, (ii) where two or more direct subclasses have identical attribute declarations, (iii) where two or more root classes have identical attribute declarations.

- Evaluation & Negotiation for Refactoring

At this point we should ask whether these scenarios are complete and if they cover all intended cases of the required refactorings. Through the analysis of the possible structures of class diagrams,
and by taking into account the invariant of single inheritance, it can be deduced that they are complete. Through exploratory prototyping and execution on particular examples of class diagrams, we can identify that the requirement for minimality means that rule 1 Pull up attributes should be prioritised over rule 2 Create subclass or 3 Create root class. In addition, the largest set of duplicated attributes in sibling classes should be removed.

- Specification & Documentation for Refactoring

To formalise the functional requirements, we express the three scenarios in abstract grammar of the language.

Rule 1: If the set \( g = c.specialisation\.specific \) of all direct subclasses of a class \( c \) has two or more elements, and all classes in \( g \) have an owned attribute with the same name \( n \) and type \( t \), add an attribute of this name and type to \( c \), and remove the copies from each element of \( g \).

Rule 2: If a class \( c \) has two or more direct subclasses \( g = c\.specialisation\.specific \), and there is a subset \( g1 \) of \( g \) of size at least 2, all the elements of \( g1 \) have an owned attribute with the same name \( n \) and type \( t \), but there are elements of \( g - g1 \) without such an attribute, introduce a new class \( c1 \) as a subclass of \( c \). \( c1 \) should also be set as a direct superclass of all those classes in \( g \) which own a copy of the cloned attribute. Add an attribute of name \( n \) and type \( t \) to \( c1 \) and remove the copies from each of its direct subclasses.

Rule 3: If there are two or more root classes all of which have an owned attribute with the same name \( n \) and type \( t \), create a new root class \( c \). Make \( c \) the direct superclass of all root classes with such an attribute, and add an attribute of name \( n \) and type \( t \) to \( c \), and remove the copies from each of the direct subclasses.

- Validation & Verification for Refactoring

The functional requirements can be checked by executing the prototype transformation on test cases. In addition, informal reasoning
5.5. Framework Implementation

can be used to check that each rule application preserves the invariants. For example, no rule introduces new types, or modifies existing types, so the invariant that type names are unique is clearly preserved by rule applications. Likewise, the model-level semantics is also preserved. Termination follows by establishing that each rule application decreases the number of attributes in the diagram, i.e., $\text{Property.size}$ (since it is bounded below by 0, there can only be finitely many rule applications). The efficiency requirements can be verified by executing the prototype transformation on realistic test cases of increasing size.

5.5 Framework Implementation

Our proposed framework for selecting the most suitable RE technique would help MT developers to choose the most suitable RE technique for a specific requirement or set of requirements. Applying the framework manually could result in high human error-rate calculation and be quite time consuming for developers. For this reason, we decided to implement our framework in UML-RSDS in order to not only facilitate the calculation in an automated way, but also to reduce the calculation error. Moreover, we will be using UML-RSDS tool to do the two case studies in the next chapter. Figure 5.4 is an illustration of the metamodel of our framework. The ElicitationAttributes and ProjectDescriptors class data are defined for each project, whereas ElicitationTechnique and ElicitationTechniqueSuitability class data are fixed tables independent of each project.
5.5. Framework Implementation

In this section, we will go through the framework’s implementation in UML-RSDS for the Domain Analysis & Requirements Elicitation stage. Technique attributes and their values have been defined in a .csv file for each RE stage (Domain Analysis & Requirements Elicitation, Evaluation & Negotiation, Specification & Documentation and Validation & Verification). The formula regarding $\text{RA}(t)$ calculation has been generated as an operation according to UML-RSDS syntax as follows:

```plaintext
*** Operations of entity ElicitationTechnique:
RA(req: ElicitationAttributes): double
pre: true
post: result = ( req.elicitMTreqs * elicitMTreqs +
  req.facilitateComm * facilitateComm + req.understandSocial * understandSocial + req.getDomainKnowl * getDomainKnowl +
  req.getImplicitKnowl * getImplicitKnowl +
  req.identifyMTstakehs * identifyMTstakehs +
  req.identifyNFRreqs * identifyNFRreqs + req.identifyViewpoints *
  identifyViewpoints ) / req.sumFactors()
```

Project attributes and their values have been defined in a .csv file for a given MT project. The formula regarding $\text{PD}(t)$ calculation has been generated as an operation according to UML-RSDS syntax as follows:
5.6. Summary

*** Operations of entity ProjectDescriptors:
query PD(prj: ProjectDescriptors): double
pre: true
post: result = projectFactor(prj.projectSize,projectSize) * projectFactor(prj.complexity,complexity) * projectFactor(prj.volatility,volatility) * projectFactor(prj.custRelationship,custRelationship) * projectFactor(prj.safety,safety) * projectFactor(prj.quality,quality) * projectFactor(prj.time,time) * projectFactor(prj.cost,cost) * projectFactor(prj.domainKnowl,domainKnowl)

Moreover, we have given the value 1 for direct proportion and −1 for inverse proportion for every RE technique as follows:

*** Operations of entity projectFactor:
query projectFactor(att: double,proj: double): double
pre: true
post: ( proj < 0 => result = 1 - att ) & ( proj > 0 => result = att )

Using the $S$ operation, the overall suitability score of a particular RE technique, $S(t)$, can be calculated as follows:

"$S$ for technique X is: "->display() &
( ElicitationTechnique[s].RA(atts) * ElicitationTechniqueSuitability[s].PD(des) * TechniqueExperience[s].experience )->display()

5.6 Summary

We have identified ways in which requirements engineering can be applied systematically to model transformations. Comprehensive catalogues of functional and non-functional requirements categories for model transformations have been defined. We have examined a case study which
5.6. Summary

is typical of the current state of the art in transformation development, and identified how formal treatment of functional and non-functional requirements can benefit such developments. Moreover, we have defined our proposed metric for the suitability of RE techniques for MT as well as implementing the framework in UML-RSDS.
Chapter 6

Evaluation

In this section, we illustrate the application of our proposed RE process and RE selection procedure, we show how these have been used on two substantial and industrial MT case studies using UML-RSDS. We will evaluate the proposed framework, which has been implemented in UML-RSDS, in order to facilitate the calculation of the RE technique suitability value, by applying it to these two real industrial cases: UML to C Transformation and Collateralized Debt Obligations (CDO).

6.1 Case study 1: UML to C Transformation

This case study concerns the development of a code generator for the UML-Rigorous Systems Design Support (UML-RSDS) [85] dialect of UML. UML-RSDS is a model transformation tool, which is able to manufacture software systems in an automated manner. Given a valid UML-RSDS model, the UML2C transformation should produce a C application with the same semantics. The target code should be structured in the standard C style with header and code files and standard C libraries may be used. The produced code is then compared to the hand-written code to verify its efficiency. The code generation process should not take longer than 1 minute for class diagrams with fewer than 100 classes.
6.1. Case study 1: UML to C Transformation

Before applying any RE process, we need to identify the stakeholders, which are listed below:

(i) the UML-RSDS development team

(ii) users of UML-RSDS who require C code for embedded or limited resource systems

(iii) end-users of such systems

Through the use of Tables of 5.5–5.8, we are going to apply our proposed framework for the RE process, which calculates the suitability score of a particular RE technique, $S(t)$, through the use of this formula:

$$S(t) = RA(t) \times PD(t) \times E(t)$$

The following are calculations for the Domain Analysis & Requirements Elicitation stage of the translation of UML to C case, by using Table 6.1 adapted from Jiang [61]. These scored attributes are static and fixed, independent of particular projects and represent fitness of $X$ (technique) for $Y$ (specific requirement attribute). Step by step application of this stage is presented below:

- **category: Domain Analysis & Requirements Elicitation**
  - The techniques in $T_{elicitation}$ are our chosen sample because the developers had some experience with these techniques.
  - $T_{elicitation} = \{interview, prototyping, questionnaire, document mining, brainstorming, scenario, ethno methodology\}$
  - $t_1 = interview, t_2 = questionnaire, t_3 = document mining$ are chosen arbitrarily for this example (Note that any number of techniques can be chosen).
  - $I(a)$ has a dynamic weighting which can be assigned from a range $[0,1]$ according to the importance of the technique attributes, $A_1$ and $A_2$, which is determined by the developers according to the initial project description and stakeholders.
TABLE 6.1. Domain Analysis & Requirements Elicitation technique attributes evaluation

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Interview</th>
<th>Questionnaire</th>
<th>Document Mining</th>
<th>Brainstorming</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliciting MT requirements</td>
<td>1</td>
<td>0.8</td>
<td>1</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>Facilitating communication</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Understanding social issues</td>
<td>0.8</td>
<td>1</td>
<td>0.8</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Getting domain knowledge</td>
<td>0.6</td>
<td>0.6</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>Getting implicit knowledge</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Identifying MT stakeholders</td>
<td>1</td>
<td>0.8</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Identifying non-functional</td>
<td>1</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identifying viewpoints</td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
<td>0.8</td>
<td>0</td>
</tr>
</tbody>
</table>

− I(a) value is 1 for A₁, 0.8 for A₂ and 0 for the remaining attributes:

* A₁ = {eliciting MT requirements, getting domain knowledge, getting implicit knowledge}

* A₂ = {identifying MT stakeholders, facilitating communications, identifying non-functional requirements}

− v(aₓ, t₁) for A₁ is {1, 0.6, 0.2}
− v(aₓ, t₁) for A₂ is {0.8, 0.8, 0.8}
− v(aₓ, t₂) for A₁ is {0.8, 0.6, 0.2}
− v(aₓ, t₂) for A₂ is {0.8, 1, 0.6}
− v(aₓ, t₃) for A₁ is {1, 1, 0.2}
− v(aₓ, t₃) for A₂ is {0.2, 0, 0.8}
### 6.1. Case study 1: UML to C Transformation

1. \( RA(t_1) = \frac{(1 \times 1) + (1 \times 0.6) + (1 \times 0.2) + (0.8 \times 0.8) + (0.8 \times 0.8) + (0.8 \times 0.8)}{4.8} \)
   \[ = \frac{4.2}{4.8} = 0.77 \]

2. \( RA(t_2) = \frac{(1 \times 0.8) + (1 \times 0.6) + (1 \times 0.2) + (0.8 \times 0.8) + (0.8 \times 1) + (0.8 \times 0.6)}{4} \)
   \[ = \frac{3.52}{4} = 0.88 \]

3. \( RA(t_3) = \frac{(1 \times 1) + (1 \times 1) + (1 \times 0.2) + (0.8 \times 0.2) + (0.8 \times 0) + (0.8 \times 0.8)}{3.2} \)
   \[ = \frac{3}{3.2} = 0.93 \]

According to the transformation project attributes, the estimated size of this transformation is large as it has over 250 rules, therefore a value of 0.8 is given for size. There are two complicating factors: complex rule logic and bidirectionality, therefore a value of 0.8 is given for complexity. There is 5%-10% change of requirements, therefore a value of 0.2 is given for the volatility attribute. There is limited access to the customer, therefore a value of 0.2 is given for customer relationship. The transformation may be used for safety-related systems but not safely-critical systems, therefore a value of 0.5 is given for the safety project attributes. There exists approximately 50 requirements, therefore a value of 0.8 is given for the transformation quality attribute. Up to 20% extension is possible regarding the time attribute, therefore the value of 0.5 is given for the time attribute. The budget restriction is low (up to 50% extension), therefore a value of 0.2 is given for the budget.
attribute. Developers are not very familiar with the domain (some experience, less than one year of experience), therefore a value of 0.4 is given for the domain knowledge attribute.

In other words, we have the following:

- Project attributes: size: large (0.8), complexity: high (0.8), volatility (0.2): low, customer-developer relationship: low (0.2), safety: medium (0.5), quality: high (0.8), time: medium (0.5), cost: low (0.2), domain knowledge: medium (0.4)

- $D = \{\text{size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}\}$
- $ID_{\text{interview}} = \{\text{time, cost}\}$
- $ID_{\text{questionnaire}} = \{\text{relationship}\}$
- $ID_{\text{documentmining}} = \{\text{relationship, domain knowledge}\}$

- $PD(t_1) = (0.8) \times (0.8) \times (0.2) \times (0.2) \times (0.5) \times (0.8) \times (0.5) \times (0.4) = 0.0016$
- $PD(t_2) = (0.8) \times (0.8) \times (0.2) \times (0.2) \times (0.5) \times (0.8) \times (0.5) \times (0.2) \times (0.4) = 0.0004$
- $PD(t_3) = (0.8) \times (0.8) \times (0.2) \times (0.8) \times (0.5) \times (0.8) \times (0.5) \times (0.2) \times (0.6) = 0.0024$

- $E(t)$ values regarding each RE technique for this stage are listed in Table 6.2 for calculating $S(t)$.

- $S(t_1) = 0.77 \times 0.0016 \times 1 = 0.0012$
- $S(t_2) = 0.88 \times 0.0004 \times 0.6 = 0.0002$
- $S(t_3) = 0.93 \times 0.0024 \times 0.8 = 0.0017$

1 The values have been determined from the transformation project attributes weighting (Table 5.9)
6.1. Case study 1: UML to C Transformation

Table 6.2 presents the value of each attribute RA(t), PD(t), E(t) and S(t) for the selected RE techniques for the Domain Analysis & Requirements Elicitation stage.

**TABLE 6.2.** Domain Analysis & Requirements Elicitation technique attributes evaluation for UML to C case

<table>
<thead>
<tr>
<th>Measures</th>
<th>Brainstorming</th>
<th>Interview</th>
<th>Mining</th>
<th>Scenario</th>
<th>Questionnaire</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA(t)</td>
<td>0.88</td>
<td>0.77</td>
<td>0.93</td>
<td>0.9</td>
<td>0.88</td>
</tr>
<tr>
<td>PD(t)</td>
<td>0.0004</td>
<td>0.0016</td>
<td>0.0024</td>
<td>0.0016</td>
<td>0.0002</td>
</tr>
<tr>
<td>E(t)</td>
<td>0.4</td>
<td>1</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S(t)</td>
<td>0.0001</td>
<td>0.0012</td>
<td>0.0017</td>
<td>0.0014</td>
<td>0.0002</td>
</tr>
</tbody>
</table>

The S(t) results indicate that document mining (0.0017), scenario (0.0014) and interview (0.0012) techniques are best suited for this translation, therefore document mining, scenario and interview were used for the Domain Analysis & Requirements Elicitation stage.

As an initial phase of the requirement’s elicitation for this system, document mining, scenario and interview were conducted. Document mining consisted of research into the ANSI C language and existing UML to C translators. Scenario was used to consider different scenarios (model elements and structures of linked elements), and a semi-structured interview with the principal stakeholder was carried out.

- Document mining: this involves comprehensive background research into relevant documents and software, specifically C standards, textbooks, compilers and forums, and review of existing code generators for C and the UML-RSDS code generators.

- Scenario analysis: detailed consideration of specific scenarios (model elements and structures of linked elements) which the translator should process. Both normal and abnormal (error) scenarios can be considered. Scenario analysis is a widely-used technique, however it can suffer from incompleteness, since in general it is not possible to identify all scenarios. In the case of model transformations this problem can be addressed by systematically defining
6.1. Case study 1: UML to C Transformation

scenarios for each permitted construct of source models that satisfies the constraints of the source metamodel(s).

- Interview: elicitation of requirements from stakeholders via structured interviews.

This initial phase of Domain Analysis & Requirements Elicitation produced an initial set of functional (F) and non-functional requirements (NF) of the project as follows:

- Functional requirements:
  - F: Translate UML-RSDS designs (class diagrams, OCL, activities and use cases) into ANSI C code
  - F: Translation of types
  - F: Translation of class diagrams
  - F: Translation of OCL expressions
  - F: Translation of activities
  - F: Translation of use cases
  - F: Syntactic correctness: given correct input, a valid C program will be produced
  - F: Model-level semantic preservation: the semantics of the source and target models are equivalent
  - F: Traceability: a record should be maintained of the correspondence between source and target elements
  - F: Bidirectionality between source and target
  - F: Confluence

- Non-functional requirements
  - NF: Termination: given correct input
  - NF: Efficiency: input models with 100 classes and 100 attributes should be processed
6.1. Case study 1: UML to C Transformation

- NF: Modularity of the transformation
- NF: Flexibility: ability to choose different C interpretations for UML elements

After a further interview, the application of model-based testing and bidirectional transformations (bx) to achieve model-level semantic preservation was identified as an important area of work. Tests for the synthesised C code should, ideally, be automatically generated based on the source UML model. The bx property can be utilised for testing semantic equivalence by transforming UML to C, applying the reverse transformation, and comparing the two UML models to identify whether they are isomorphic.

The identified stakeholders included: (i) the UML-RSDS development team; (ii) users of UML-RSDS who require C code for embedded or limited resource systems; (iii) end-users of such systems. Direct access was only possible to stakeholders (i). Access to other stakeholders was substituted by research into the needs of such stakeholders, using document mining of sources such as C text books, the C standard, and specialised standards, particularly MISRA C [7].

An initial phase of requirements elicitation for this system used document mining (research into the ANSI C language and existing UML to C translators) and a semi-structured interview with the principal stakeholder. This produced an initial set of requirements, with priorities. It was determined that the complete set of language restrictions of MISRA C would not be followed, and instead the focus would be on supporting the implementation of UML in C for general users. Thus, we target the ANSI 89 standard version of C, as described in [69].

We distinguish between global and local requirements for MT: a global requirement concerns properties of the source/target model or transformation considered as a whole (such as the syntactic correctness of the target model with its metamodel), whilst local requirements concern properties specific to particular types of model elements or particular kinds of structures in the models (such as a mapping requirement for the
6.1. Case study 1: UML to C Transformation

mapping of UML inheritance to C).

In order to prioritise the requirements for this project, we used interview and brainstorming techniques based on the scoring result from the framework. We discussed the priority of the elicited requirements with the client through interview and brainstorming techniques. This produced an initial set of requirements, with the following priorities according to the stakeholder and the intended use of the system:

- high-level
- medium-level
- low-level

High-level functional requirement (F) of the translation is:

F1: Translate UML-RSDS designs (class diagrams, OCL, activities and use cases) into ANSI C code.

This high-level functional requirement was further decomposed into five high-level priority subgoals, each of which is responsible for a separate subtransformation as follows:

- F1.1: Translation of types
- F1.2: Translation of class diagrams
- F1.3: Translation of OCL expressions
- F1.4: Translation of activities
- F1.5: Translation of use cases

Each translation in this list is dependent upon all of the preceding translations. In addition, the translation of operations of classes depends upon the translation of expressions and activities. The development was therefore organised into five iterations, one for each translator part, and each iteration was given a maximum duration of one month.

Other high-level priority functional and non-functional (NF) requirements identified for the translator are as follows:
6.1. Case study 1: UML to C Transformation

- NF1: Termination: given correct input
- F2: Syntactic correctness: given correct input, a valid C program will be produced
- F3: Model-level semantic preservation: the semantics of the source and target models are equivalent
- F4: Traceability: a record should be maintained of the correspondence between source and target elements

Medium-level priority functional and non-functional requirements of the translation are:

- F5: Bidirectionality between source and target
- NF2: Efficiency: input models with 100 classes and 100 attributes should be processed within 30 seconds
- NF3: Modularity of the transformation
- NF6: Produce efficient code, of similar or higher efficiency as equivalent hand-produced code
- NF7: Produce compact code, of the same or smaller size as equivalent hand-produced code

Low-level priority functional and non-functional requirements of the translation are:

- F6: Confluence
- NF4: Flexibility: ability to choose different C interpretations for UML elements

There are potential conflicts between the requirements:
6.1. Case study 1: UML to C Transformation

- NF2 conflicts with F4, F5 and NF3 because the additional structure needed for tracing and bx properties impairs efficiency, and the decomposition of the transformation into subtransformations composed sequentially also slows execution.

- NF4 conflicts with NF10 as the additional work required for NF4 would need substantial additional time resources.

- NF6 conflicts with F3 because in some cases semantic correctness will require inefficient coding, e.g., because OCL collection operators produce modified copies of their arguments instead of updating them in-place.

Figure 6.1 shows part of the requirements subdivisions and goal decomposition using SysML. It shows the prioritization and dependency relationship of the requirements.

For the RE technique suitability score, $S(t)$, of the remaining three stages (Evaluation & Negotiation, Specification & Documentation and Validation & Verification), we have to apply a similar procedure for each
6.1. Case study 1: UML to C Transformation

particular technique at each stage. However, for brevity reasons, we are going to give the overall result of the suitability score of each selected technique without presenting any further calculation steps.

The overall ranking of techniques in the Evaluation & Negotiation stage according to importance based on \( S(t) \) values is: (i) scenario, (ii) UML, (iii) prototyping, as shown in Table 6.3.

The overall ranking of techniques in the Specification & Documentation stage according to importance based on \( S(t) \) values is: (i) natural language and UML, (ii) SysML, as shown in Table 6.4.

The overall ranking of techniques in the Validation & Verification stage according to importance based on \( S(t) \) values is: (i) prototyping, (ii) checklist, (iii) inspection, as shown in Table 6.5. These tables are based upon Tables 5.5–5.8.

- Prototyping: parts of the transformation are implemented in an initial form and tested using example models to identify if the intended mappings are correctly defined. This is an iterative process with successive refinement of the implementation based upon stakeholder feedback. In our case, the evolved prototype is also the final deliverable.

- Inspection: systematic review of the specifications is performed to check their syntactic and semantic correctness, and their validity wrt requirements. As noted below, we found that inspection of specifications was substantially more time-efficient than code inspection, corresponding to a 4-fold size reduction of the specification compared to executable code.
6.1. Case study 1: UML to C Transformation

**TABLE 6.3.** Technique attributes evaluation of the Evaluation & Negotiation stage $\Psi(a_x, t)$ for UML to C case

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Prototyping</th>
<th>Scenario</th>
<th>UML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling MT requirements</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Analysing non-functional requirements</td>
<td>0.2</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>Modelling interface requirements</td>
<td>0.6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Facilitate negotiation</td>
<td>0.8</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>RA($t$)</td>
<td>0.93</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>PD($t$)</td>
<td>0.0004</td>
<td>0.0016</td>
<td>0.0004</td>
</tr>
<tr>
<td>S($t$)</td>
<td>0.0002</td>
<td>0.0015</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

**TABLE 6.4.** Technique attributes evaluation of the Specification & Documentation stage $\Psi(a_x, t)$ for UML to C case

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Natural language</th>
<th>UML</th>
<th>SysML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representing MT requirements</td>
<td>0.8</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>Semantic completeness</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>Write complete requirements</td>
<td>0.6</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Modularity</td>
<td>0.2</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>RA($t$)</td>
<td>0.80</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>PD($t$)</td>
<td>0.0004</td>
<td>0.0004</td>
<td>0.0004</td>
</tr>
<tr>
<td>S($t$)</td>
<td>0.0003</td>
<td>0.0003</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Decomposing the code generator into two sub-transformations improves its modularity, and simplifies the constraints, which would other-
6.1. Case study 1: UML to C Transformation

<table>
<thead>
<tr>
<th>Attribute</th>
<th>(rapid)Prototyping</th>
<th>Inspection</th>
<th>Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifying ambiguous requirements</td>
<td>0.4</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>Identifying inconsistency and conflict</td>
<td>0.8</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>Identifying incomplete requirements</td>
<td>0.8</td>
<td>0.8</td>
<td>8</td>
</tr>
<tr>
<td>RA(t)</td>
<td>0.96</td>
<td>0.95</td>
<td>1</td>
</tr>
<tr>
<td>PD(t)</td>
<td>0.0004</td>
<td>0.0001</td>
<td>0.0004</td>
</tr>
<tr>
<td>S(t)</td>
<td>0.0003</td>
<td>0.00005</td>
<td>0.0002</td>
</tr>
</tbody>
</table>

wise need to combine language translation and text production. Therefore, a suitable overall architecture for the transformation was a sequential decomposition of a model-to-model transformation design2C, and a model-to-text transformation genCtext. Figure 6.2 shows the overall transformation architecture. This decomposition means that each of the high-level requirements need to be satisfied by both design2C and genCtext. The requirements for bidirectionality and traceability are however specific to design2C.

Figure 6.2. C code generator architecture

In the following subsections we present the application of the selected RE techniques on the case study.
6.1. Case study 1: UML to C Transformation

6.1.1 F1.1: Translation of Types

This iteration was divided into three phases: detailed requirements analysis, specification, testing. Detailed requirements elicitation used structured interviews to identify: (i) the source language, (ii) the mapping requirements, (iii) the target language, (iv) other functional and non-functional requirements, for this sub-transformation. Scenarios and test cases were prepared.

Using goal decomposition, the requirements were decomposed into specific mapping requirements, these are the local functional requirements F1.1.1 to F1.1.4 in Figure 6.1. Table 6.6 shows the informal scenarios for these local mapping requirements, based on the concrete metaclasses of Type and the different cases of instances of these metaclasses. The schematic concrete grammar is shown for the C elements representing the UML concepts. As a result of requirements evaluation and negotiation with the principal stakeholder, using exploratory prototyping, it was determined that all these local requirements are of high priority except for the mapping of F1.1.2 (Figure 6.1) of enumerations (medium priority). The justification for this is that enumerations are not an essential UML language element. Bidirectionalness was considered a high priority for this sub-transformation. It was identified that to meet this requirement, all source model Property elements must have a defined type, and specifically that elements representing many-valued association ends must have some CollectionType representing their actual type. A limitation of the proposed mapping is that mapping collections of primitive values (integers, doubles, booleans) to C is not possible, because there is no means to identify the end of the collection in C (NULL is used as the terminator for collections of objects and collections of strings).

6.1.2 F1.2: Translation of Class Diagrams

This iteration also used a three-phase approach, to define a subtransformation classdiagram2C. The class diagram elements Property, Operation, Entity, Generalization were identified as the input language. Ex-
6.1. Case study 1: UML to C Transformation

<table>
<thead>
<tr>
<th>Scenario</th>
<th>UML element e</th>
<th>C representation e’</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1.1.1.1</td>
<td>String type</td>
<td>char*</td>
</tr>
<tr>
<td>F1.1.1.2</td>
<td>int, long, double types, boolean type</td>
<td>same-named C types</td>
</tr>
<tr>
<td>F1.1.1.3</td>
<td>boolean type</td>
<td>unsigned char</td>
</tr>
<tr>
<td>F1.1.2</td>
<td>Enumeration type</td>
<td>C enum</td>
</tr>
<tr>
<td>F1.1.3</td>
<td>Entity type E</td>
<td>struct E* type</td>
</tr>
<tr>
<td>F1.1.4.1</td>
<td>Set(E) type</td>
<td>struct E** (array of E, without duplicates)</td>
</tr>
<tr>
<td>F1.1.4.2</td>
<td>Sequence(E) type</td>
<td>struct E** (array of E, possibly with duplicates)</td>
</tr>
</tbody>
</table>

ploratory prototyping was used for requirements elicitation and evaluation. During requirements evaluation and negotiation it was agreed that the metafeatures isStatic, isReadOnly, isDerived, isCached would not be represented in C, nor would addOnly, aggregation, constraint or linked-Class. This means that aggregations, association classes and static or constant features are not specifically represented in C. Interfaces are also not represented, only single inheritance is represented.

The scenarios of the local mapping requirements for class diagram elements are shown in Table 6.7.

The source language was identified as the Type class and its subclasses in the standard UMLRSDS class diagram metamodel as illustrated in Figure 6.3.

6.1.3 F1.3: Translation of OCL Expressions

In this iteration, the detailed requirements for mapping OCL expressions to C are identified, then this subtransformation, expressions2C, is specified and tested. There are many cases to consider in the mapping requirements, so we divided these into four subgroups, mapping of: (i) basic expressions; (ii) logical expressions; (iii) comparator, numeric and
### 6.1. Case study 1: UML to C Transformation

#### TABLE 6.7. Informal scenarios for the mapping of UML class diagrams to C

<table>
<thead>
<tr>
<th>Scenario</th>
<th>UML element e</th>
<th>C representation e’</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1.2.1</td>
<td>Class diagram D</td>
<td>C program with D’s name</td>
</tr>
<tr>
<td>F1.2.2</td>
<td>Class E</td>
<td>struct E {...};</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Global variable struct E** e_instances;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Global variable int e_size;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>struct E* createE() operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>struct E** newEList() operation</td>
</tr>
<tr>
<td>F1.2.3.1</td>
<td>Property p : T (not principal identity attribute)</td>
<td>Member T’p; of the struct for p’s owner, where T’ represents T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operations T’ getE_p(E’ self)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and setE_p(E’ self, T’ px)</td>
</tr>
<tr>
<td>F1.2.3.2</td>
<td>Principal identity attribute p : String of class E</td>
<td>Operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>struct E* getEByPK(char* v)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Key member char* p; of the struct for E</td>
</tr>
<tr>
<td>F1.2.4</td>
<td>Operation op(p : P) : T of E</td>
<td>C operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T’ op(E’ self, P’ p)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with scope = entity</td>
</tr>
<tr>
<td>F1.2.5</td>
<td>Inheritance of A by B</td>
<td>Member struct A* super;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of struct B</td>
</tr>
</tbody>
</table>

string expressions; (iv) collection expressions. These were considered the natural groupings of operations and operators, and these follow in part the metaclass organisation of UML expressions.

#### Mapping of Basic Expressions

The basic expressions of OCL generally map directly to corresponding C basic expressions. Table 6.8 shows the mapping for these. These mapping requirements are grouped together as requirement F1.3.1 (Figure 6.1).

#### Mapping of Logical Expressions

Table 6.9 shows the mapping of logical expressions and operators to C. These mappings are grouped together as requirement F1.3.2 (Figure 6.1).
6.1. Case study 1: UML to C Transformation

Figure 6.3. UML-RSDS class diagram metamodel

Mapping of Comparator, Numeric and String Expressions

Table 6.10 lists the comparator operators and their mappings to C. These mappings are grouped as requirement F1.3.3 (Figure 6.1). Numeric operators for integers and real numbers are shown in Table 6.11. The types int, double and long are not guaranteed to have particular sizes in C. All operators take double values as arguments except mod and Integer subrange, which have int parameters.

Other math operators directly available in C are: log10, tanh, cosh, sinh, asin, acos, atan. These are double-valued functions of double-valued arguments. cbrt is missing and needs to be implemented as pow(x, 1.0/3).

Mapping of Collection Expressions

Tables 6.13 and 6.14 show the values and operators that apply to sets and sequences, and their C translations. Some operators (unionAll, intersect-

---

2 OCL library functions isIn, equalsSet and etc. are defined in a file ocl.h to support the execution of OCL expressions.
6.1. Case study 1: UML to C Transformation

<table>
<thead>
<tr>
<th>OCL expression e</th>
<th>C representation e'</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>self as an operation parameter</td>
</tr>
<tr>
<td>Variable v or v[ind]</td>
<td>v or v[ind - 1]</td>
</tr>
<tr>
<td>Data feature f with no objectRef</td>
<td>self \rightarrow f</td>
</tr>
<tr>
<td>Data feature f of instance ex</td>
<td>ex' \rightarrow f</td>
</tr>
<tr>
<td>Operation call op(e1,...,en) or obj.op(e1,...,en)</td>
<td>op(self, e1', ..., en') or op(obj', e1', ..., en')</td>
</tr>
<tr>
<td>Attribute f of collection exs</td>
<td>getAllE_f(exs')</td>
</tr>
<tr>
<td>Single-valued role r : F of collection exs</td>
<td>getAllE_r(exs') defined by (struct F **) collectE(exs', getE_r)</td>
</tr>
<tr>
<td>col[ind] ordered collection col</td>
<td>(col')[ind-1]</td>
</tr>
<tr>
<td>E[v] v single-valued</td>
<td>getEByPK(v')</td>
</tr>
<tr>
<td>E[vs] vs collection-valued</td>
<td>getEByPKs(vs')</td>
</tr>
<tr>
<td>E.allInstances</td>
<td>e_instances</td>
</tr>
<tr>
<td>value of enumerated type, numeric or string value</td>
<td>value</td>
</tr>
<tr>
<td>boolean true, false</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>

*tAll, symmetricDifference, subcollections* were considered a low priority, because these are infrequently used, and were not translated. The requirements are grouped as F1.3.6 (Figure 6.1). In addition, prototyping revealed that compiler differences made the use of qsort impractical, and instead a custom sorting algorithm, treesort, was implemented. This has the signature *treesort(void* col[], int (*comp)(void*, void*))* and the translation of \( x \rightarrow \text{sort()} \) is then: \( \text{rt} \text{treesort}((\text{void}**) x', \text{comp}) \) for the appropriate result type \( \text{rt} \) and comparator function \( \text{comp} \). Table 6.12 shows the translation of Select and Collect expressions. These mappings are grouped as requirement F1.3.7 (Figure 6.1).

Unlike the Types and Class diagram mappings, a recursive descent style of specification is needed for mappings of expressions and activities. This is because the subordinate parts of an expression are themselves ex-
### 6.1. Case study 1: UML to C Transformation

#### TABLE 6.9. Mapping scenarios for Logical Expressions

<table>
<thead>
<tr>
<th>OCL expression e</th>
<th>C representation e’</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = B</td>
<td>!A’</td>
</tr>
<tr>
<td>A &amp; B</td>
<td>A’ &amp;&amp; B’</td>
</tr>
<tr>
<td>A or B</td>
<td>A’</td>
</tr>
<tr>
<td>not(A)</td>
<td>!A’</td>
</tr>
<tr>
<td>E-&gt;exists(P)</td>
<td>existsE(e_instances,fP) fP evaluates P’</td>
</tr>
<tr>
<td>e-&gt;exists(P)</td>
<td>existsE(e’,fP)</td>
</tr>
<tr>
<td>E-&gt;exists1(P)</td>
<td>exists1E(e_instances,fP) fP evaluates P’</td>
</tr>
<tr>
<td>e-&gt;exists1(P)</td>
<td>exists1E(e’,fP)</td>
</tr>
<tr>
<td>E-&gt;forAll(P)</td>
<td>forAllE(e_instances,fP) fP evaluates P’</td>
</tr>
<tr>
<td>e-&gt;forAll(P)</td>
<td>forAllE(e’,fP)</td>
</tr>
</tbody>
</table>

expressions. Thus, in general it is not possible to map all the subordinate parts of an expression by prior rules; even for basic expressions, the parameters may be general expressions. In contrast, the element types of collection types cannot themselves be collection types or involve subparts that are collection types, so it is possible to map all element types before considering collection types. A recursive descent style of mapping specification uses operations of each source entity type to map instances of that type, invoking mapping operations recursively to map subparts of the instances.

#### 6.1.4 Translation of Activities

In this iteration, UML-RSDS activities are mapped to C statements by a subtransformation statements2C. UML-RSDS statements correspond closely to those of C. Table 6.15 shows the main cases of the mapping of UML activities to C statements.
### 6.1. Case study 1: UML to C Transformation

**TABLE 6.10.** Mapping scenarios for Comparator Expressions

<table>
<thead>
<tr>
<th>OCL expression e</th>
<th>C representation e’</th>
</tr>
</thead>
<tbody>
<tr>
<td>x : E</td>
<td>isIn((void*) x’, (void **) e_instances)</td>
</tr>
<tr>
<td>E entity type</td>
<td></td>
</tr>
<tr>
<td>x : s</td>
<td>isIn((void*) x’, (void **) s’)</td>
</tr>
<tr>
<td>s collection</td>
<td></td>
</tr>
<tr>
<td>s-&gt;includes(x)</td>
<td>Same as x : s</td>
</tr>
<tr>
<td>s collection</td>
<td></td>
</tr>
<tr>
<td>x / : E</td>
<td>!isIn((void*) x’, (void **) e_instances)</td>
</tr>
<tr>
<td>E entity type</td>
<td></td>
</tr>
<tr>
<td>x / : s</td>
<td>!isIn((void*) x’, (void **) s’)</td>
</tr>
<tr>
<td>s collection</td>
<td></td>
</tr>
<tr>
<td>s-&gt;excludes(x)</td>
<td>Same as x / : s</td>
</tr>
<tr>
<td>s collection</td>
<td></td>
</tr>
<tr>
<td>x = y</td>
<td>x’== y’</td>
</tr>
<tr>
<td>Numerics, Booleans</td>
<td></td>
</tr>
<tr>
<td>Strings</td>
<td>strcmp(x’, y’) == 0</td>
</tr>
<tr>
<td>Objects</td>
<td>x’== y’</td>
</tr>
<tr>
<td>Sets</td>
<td>equalsSet((void **) x’, (void **) y’)</td>
</tr>
<tr>
<td>Sequences</td>
<td>equalsSequence((void **) x’, (void **) y’)</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>x’&lt;y’</td>
</tr>
<tr>
<td>Numerics</td>
<td></td>
</tr>
<tr>
<td>Strings</td>
<td>strcmp(x’, y’) &lt;0</td>
</tr>
<tr>
<td>Similarly for &gt;, &lt;=, &gt;=, =</td>
<td></td>
</tr>
<tr>
<td>/=</td>
<td></td>
</tr>
<tr>
<td>s &lt;: t</td>
<td>containsAll ((void **) t’, (void **) s’)</td>
</tr>
<tr>
<td>s, t collections</td>
<td></td>
</tr>
<tr>
<td>t-&gt;includesAll(s)</td>
<td>Same as s &lt;: t</td>
</tr>
<tr>
<td>t-&gt;excludesAll(s)</td>
<td>disjoint((void**) t’, (void**) s’)</td>
</tr>
</tbody>
</table>

191
6.1. Case study 1: UML to C Transformation

**TABLE 6.11.** Mapping scenarios for Numeric Expressions

<table>
<thead>
<tr>
<th>OCL expression e</th>
<th>Representation in C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-x</td>
<td>-x’</td>
</tr>
<tr>
<td>x + y</td>
<td>x’ + y’</td>
</tr>
<tr>
<td>numbers</td>
<td></td>
</tr>
<tr>
<td>x - y</td>
<td>x’ - y’</td>
</tr>
<tr>
<td>x* y</td>
<td>x’ * y’</td>
</tr>
<tr>
<td>x / y</td>
<td>x’ / y’</td>
</tr>
<tr>
<td>x mod y</td>
<td>x’ % y’</td>
</tr>
<tr>
<td>x.sqr</td>
<td>(x’ * x’)</td>
</tr>
<tr>
<td>x.sqrt</td>
<td>sqrt(x’)</td>
</tr>
<tr>
<td>x.floor</td>
<td>oclFloor(x’) defined as: ((int) floor(x’))</td>
</tr>
<tr>
<td>x.round</td>
<td>oclRound(x’)</td>
</tr>
<tr>
<td>x.ceil</td>
<td>oclCeil(x’) defined as: ((int) ceil(x’))</td>
</tr>
<tr>
<td>x.abs</td>
<td>fabs(x’)</td>
</tr>
<tr>
<td>x.exp</td>
<td>exp(x’)</td>
</tr>
<tr>
<td>x.log</td>
<td>log(x’)</td>
</tr>
<tr>
<td>x.pow(y)</td>
<td>pow(x’, y’)</td>
</tr>
<tr>
<td>x.sin, x.cos, x.tan</td>
<td>sin(x’), cos(x’), tan(x’)</td>
</tr>
<tr>
<td>Integer.subrange(st,en)</td>
<td>intSubrange(st’,en’)</td>
</tr>
</tbody>
</table>

**TABLE 6.12.** Scenarios for the mapping of Selection and Collection Expressions

<table>
<thead>
<tr>
<th>UML expression e</th>
<th>C translation e’</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-&gt;select(P)</td>
<td>selectE(s’, fP) fP evaluates P’</td>
</tr>
<tr>
<td>s-&gt;select ( x</td>
<td>P )</td>
</tr>
<tr>
<td>s-&gt;reject(P)</td>
<td>rejectE(s’, fP)</td>
</tr>
<tr>
<td>s-&gt;reject ( x</td>
<td>P )</td>
</tr>
<tr>
<td>s-&gt;collect(e)</td>
<td>(et*) collectE(s’, fe)</td>
</tr>
<tr>
<td>e of type et</td>
<td>fe evaluates e’</td>
</tr>
<tr>
<td>s-&gt;collect ( x</td>
<td>e )</td>
</tr>
</tbody>
</table>
6.1. Case study 1: UML to C Transformation

TABLE 6.13. Scenarios for the translation of Collection Operators (1)

<table>
<thead>
<tr>
<th>Expression e</th>
<th>C translation e'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set{}</td>
<td>newEList()</td>
</tr>
<tr>
<td>Sequence{}</td>
<td>newEList()</td>
</tr>
<tr>
<td>Set{x1, x2, ..., xn}</td>
<td>insertE(... insertE(newEList(), x1'), ..., xn')</td>
</tr>
<tr>
<td>Sequence{x1, x2, ..., xn}</td>
<td>appendE(... appendE(newEList(), x1'), ..., xn')</td>
</tr>
<tr>
<td>s-&gt;size()</td>
<td>length((void**) s')</td>
</tr>
<tr>
<td>s-&gt;including(x)</td>
<td>insertE(s',x') or appendE(s',x')</td>
</tr>
<tr>
<td>s-&gt;excluding(x)</td>
<td>removeE(s',x')</td>
</tr>
<tr>
<td>s - t</td>
<td>removeAllE(s',t')</td>
</tr>
<tr>
<td>s-&gt;append(x)</td>
<td>appendE(s',x')</td>
</tr>
<tr>
<td>s-&gt;count(x)</td>
<td>count((void*) x', (void**) s')</td>
</tr>
<tr>
<td>s-&gt;indexOf(x)</td>
<td>indexOf((void*) x', (void**) s')</td>
</tr>
<tr>
<td>x\y</td>
<td>intersectionE(x',y')</td>
</tr>
<tr>
<td>x~ y</td>
<td>concatenateE(x',y)</td>
</tr>
<tr>
<td>x-&gt;union(y)</td>
<td>unionE(x',y')</td>
</tr>
<tr>
<td>x-&gt;intersection(y)</td>
<td>intersectionE(x', y')</td>
</tr>
<tr>
<td>x-&gt;any()</td>
<td>x'[0]</td>
</tr>
<tr>
<td>x-&gt;reverse()</td>
<td>reverseE(x')</td>
</tr>
<tr>
<td>x-&gt;front()</td>
<td>subrangeE(x',1,length((void**) x')-1)</td>
</tr>
<tr>
<td>x-&gt;tail()</td>
<td>subrangeE(x',2,length((void**) x'))</td>
</tr>
<tr>
<td>x-&gt;first()</td>
<td>x'[0]</td>
</tr>
</tbody>
</table>

6.1.5 Translation of Use Cases

In this iteration, the mapping usecases2C of use cases is specified and implemented. A large part of this iteration was also taken up with integration testing of the complete transformation.

F1.5.1: A use case uc is mapped to a C operation with application scope, and with parameters corresponding to those of uc. Its code is given by the C translation of the activity classifierBehaviour of uc.

F1.5.2: Included use cases are also mapped to operations, and invoked from the including use case.

F1.5.3: Operation activities are mapped to C code for the corresponding COperation.

F1.5.1 is formalised as:

UseCase::
COperation->exists( cop | cop.name = name &
### 6.1. Case study 1: UML to C Transformation

**TABLE 6.14.** Scenarios for the translation of Collection Operators (2)

<table>
<thead>
<tr>
<th>Expression e</th>
<th>C translation e'</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-&gt;last()</td>
<td>x'[length((void**) x')-1]</td>
</tr>
<tr>
<td>x-&gt;sort()</td>
<td>qsort((void**) x', length((void**) x'), sizeof(struct E*), compareToE)</td>
</tr>
<tr>
<td>x-&gt;sortedBy(e)</td>
<td>qsort((void**) x', length((void**) x'), sizeof(struct E*), compare)</td>
</tr>
<tr>
<td>x-&gt;sum()</td>
<td>sumString(x'), sumint(x'), sumlong(x'), sumdouble(x')</td>
</tr>
<tr>
<td>x-&gt;prd()</td>
<td>prdint(x'), prdlong(x'), prddouble(x')</td>
</tr>
<tr>
<td>x-&gt;max()</td>
<td>maxInt(x'), maxLong(x'), maxDouble(x'), maxString(x')</td>
</tr>
<tr>
<td>x-&gt;min()</td>
<td>minInt(x'), minLong(x'), minDouble(x'), minString(x')</td>
</tr>
<tr>
<td>x-&gt;asSet()</td>
<td>asSetE(x')</td>
</tr>
<tr>
<td>x-&gt;asSequence()</td>
<td>x'</td>
</tr>
<tr>
<td>s-&gt;isUnique(e)</td>
<td>isUniqueE(s',fe)</td>
</tr>
</tbody>
</table>

**TABLE 6.15.** Scenarios for mapping of UML Activities to C Statements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>UML activity st</th>
<th>C statement st'</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1.4.1</td>
<td>Creation statement x : T defaultT' is default value of T'</td>
<td>T' x = defaultT';</td>
</tr>
<tr>
<td>F1.4.2</td>
<td>Assign statement v := e</td>
<td>v' = e';</td>
</tr>
<tr>
<td>F1.4.3</td>
<td>Sequence statement st1 ; ... ; stn</td>
<td>st1' ... stn'</td>
</tr>
<tr>
<td>F1.4.4</td>
<td>Conditional statement if e then st1 else st2</td>
<td>if e' {st1'} else {st2'}</td>
</tr>
<tr>
<td>F1.4.5</td>
<td>Return statement return e</td>
<td>return e';</td>
</tr>
<tr>
<td>F1.4.6</td>
<td>Break statement break</td>
<td>break;</td>
</tr>
<tr>
<td>F1.4.7</td>
<td>Bounded loop for (x : e) do st on object collection e of entity element type E</td>
<td>int i = 0; for ( ; i &lt;length((void**) e') ; i++)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ struct E* x = e'[i]; st' }</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New index variable i</td>
</tr>
<tr>
<td>F1.4.8</td>
<td>Unbounded loop while e do st</td>
<td>while (e') { st' }</td>
</tr>
<tr>
<td>F1.4.9</td>
<td>Operation call ex.op(pars)</td>
<td>op(ex',pars')</td>
</tr>
</tbody>
</table>

```c

cop.scope = "application" &
cop.isQuery = false &
```
6.1. Case study 1: UML to C Transformation

\[
\begin{align*}
cop.\text{code} & = \text{classifierBehaviour.mapStatement()} \& \\
cop.\text{parameters} & = \text{parameters.mapExpression()} \& \\
cop.\text{returnType} & = \text{CType[returnType.typeId]} 
\end{align*}
\]

Similarly for the activities of UML operations.

This case study is the largest transformation, which has been developed using UML-RSDS, in terms of the number of rules (over 250 rules/operations in 5 subtransformations). By using a systematic requirements engineering and agile development approach, we were able to effectively modularise the transformation and to organise its structure and manage its requirements. Despite the complexity of the transformation, it was possible to use patterns to enforce bx and other properties, and to effectively prove these properties. The bx properties are discussed in \[70\].

6.1.6 Evaluation

In this section we evaluate the outcomes of the development, the effectiveness of UML-RSDS for the development, and the RE technique framework that we have used.

Comparison with requirements

Table \[6.16\] compares the functional and non-functional requirements and the actual achieved results. In some cases it is possible to prove by the construction of the transformation that some properties hold (e.g., termination and confluence). For syntactic and semantic correctness we can give rigorous arguments based on considering each mapping rule and checking that it produces valid C with the same semantics as its input. For some aspects, such as numeric computations, semantic correctness is only relative to the same definitions of numeric types being used in the input UML and output C; the specifier needs to use in its specification the same data type sizes (e.g., 16 bit int type) as the target code platform. For dynamic memory allocation, we assume that \textit{malloc} and \textit{calloc} always succeed. Select and other iterator expressions are restricted to depend on only one variable. Only collections containing string or entity
6.1. Case study 1: UML to C Transformation

instances can be explicitly constructed.

**TABLE 6.16.** Achievement of requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Priority</th>
<th>Achievement</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF1: Termination</td>
<td>High</td>
<td>Proved</td>
</tr>
<tr>
<td>NF10: Development time</td>
<td>High</td>
<td>Achieved</td>
</tr>
<tr>
<td>F2: Syntactic correctness</td>
<td>High</td>
<td>Rigorous argument</td>
</tr>
<tr>
<td>F3: Semantic preservation</td>
<td>High</td>
<td>Rigorous argument</td>
</tr>
<tr>
<td>F4: Traceability</td>
<td>High</td>
<td>Achieved</td>
</tr>
<tr>
<td>F5: Bidirectionality</td>
<td>Medium</td>
<td>Partly achieved</td>
</tr>
<tr>
<td>NF2: Transformation efficiency</td>
<td>Medium</td>
<td>Achieved</td>
</tr>
<tr>
<td>NF3: Transformation modularity</td>
<td>Medium</td>
<td>Achieved</td>
</tr>
<tr>
<td>NF5: Usability</td>
<td>Medium</td>
<td>Achieved</td>
</tr>
<tr>
<td>NF6: Efficient code</td>
<td>Medium</td>
<td>Partly achieved</td>
</tr>
<tr>
<td>NF7: Compact code</td>
<td>Medium</td>
<td>Partly achieved</td>
</tr>
<tr>
<td>F6: Confluence</td>
<td>Low</td>
<td>Proven</td>
</tr>
<tr>
<td>NF4: Flexibility</td>
<td>Low</td>
<td>Not achieved</td>
</tr>
</tbody>
</table>

In order to test NF6 and NF7 we wrote a test UML specification involving a fixed-point computation of the maximum-value node in a graph of nodes. This has one entity $A$, with an attribute $x : int$ and a self-association $\text{neighbours} : A \rightarrow \text{Set}(A)$. There is a use case $\text{maxnode}$ with the postcondition:

$$A::
\text{n : neighbours & n.x > x@pre} \Rightarrow x = n.x$$

This updates a node to have the maximum $x$ value of its neighbours. Because this constraint reads and writes $A :: x$, a fixed-point design is generated, with a running time of cubic order in the number of nodes.

We obtain an overall estimate for the C code generator in Table 6.17.

Table 6.18 compares the code size and the efficiency of the C code with the Java code for all applications, including library code and the
6.1. Case study 1: UML to C Transformation

TABLE 6.17. Overall development effort for C code generator

<table>
<thead>
<tr>
<th>Stage</th>
<th>Effort (person days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req. Elicitation</td>
<td>17</td>
</tr>
<tr>
<td>Eval./Negotiation</td>
<td>5</td>
</tr>
<tr>
<td>Specification</td>
<td>56</td>
</tr>
<tr>
<td>Review/Validation</td>
<td>57</td>
</tr>
<tr>
<td>Implementation//Testing</td>
<td>49</td>
</tr>
<tr>
<td>Total</td>
<td>184</td>
</tr>
</tbody>
</table>

efficiency of the C code with the Java code. The lcc compiler was used for C. These show that code size is halved by using C, and that efficiency is improved.

TABLE 6.18. Generated C code versus Java code

<table>
<thead>
<tr>
<th>C version</th>
<th>Java version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code size</td>
<td>17Kb</td>
</tr>
<tr>
<td>Execution time</td>
<td></td>
</tr>
<tr>
<td>A.size = 20</td>
<td>0</td>
</tr>
<tr>
<td>A.size = 50</td>
<td>15ms</td>
</tr>
<tr>
<td>A.size = 100</td>
<td>240ms</td>
</tr>
<tr>
<td>A.size = 200</td>
<td>1750ms</td>
</tr>
</tbody>
</table>

Comparison with/without RE Technique Framework

Several code generators have previously been developed for UML-RSDS in Java 4, Java 6, Java 7, C# and C++. Each of these was developed using an agile development process but with manual coding in Java and without any RE activity. Table 6.19 shows the approximate effort in person-months expended for each of these to date. The generators for Java 6, 7 and C# used very similar strategies and extensively reused the code of the Java 4 version generator.

The best comparison with the C code generator (case study with RE technique framework) is perhaps the C++ generator (case study with-
### 6.1. Case study 1: UML to C Transformation

<table>
<thead>
<tr>
<th></th>
<th>Java 4</th>
<th>Java 6</th>
<th>Java 7</th>
<th>C#</th>
<th>C++</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req. Anal.</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4.5</td>
</tr>
<tr>
<td>Coding</td>
<td>12</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Testing</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>Maintenance</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>30</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>17</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 6.19. Development effort for code generators (person months)

out RE technique framework), which involved considerable background research into the semantics, language and libraries of C++, and significant revision of the existing Java-oriented code generator. Likewise, the C code generator involved substantial new research work on the code generation strategy, in addition to the technical challenge of implementing this strategy.

The development effort amounts to 4.5 person months for requirements analysis/specification activities, compared to 6 months for the manually-developed C++ generator. 49 days were spent on implementation and testing, compared to 8 months for the C++ generator (Table 6.17). A major factor in this difference is the simpler and more concise transformation specification of the C code generator (expressed in UML-RSDS) compared to the Java code of the C++ code generator. Not only is the UML-RSDS specification 4 times shorter than the Java code, but the latter is scattered over multiple source files (eg., Attribute.java, Association.java, Entity.java, etc.), making debugging and maintenance more complex compared to the C translator, which is defined in 2 specification files. The C++ generator does not construct a C++ language model, instead language mapping and text production are mixed together, resulting in complex and duplicated processing. In total, the core code of the UML-RSDS tools is 90,500 lines of Java code, of which approximately 20% (18,100 lines) is the C++ code generator. In contrast the UML2C specification is 2,200 (uml2Ca) and 2,700 (uml2Cb) lines, in total 4,900 lines. The OCL specification of UML2C is highly declarative and corresponds directly to the informal requirements, hence it is easier
6.1. Case study 1: UML to C Transformation

to understand and modify compared to a programming language implementation. In iterations 3 and 4 the specification style is less purely declarative than in iterations 1, 2 and 5, but instead is in a functional programming style. It was found that this was also more concise and easier to understand and change than the imperative Java coding of the C++ code generator transformation.

Whilst UML2C is explicitly divided into 5 main stages, each subdivided into model to model and model to text modules, the C++ generator has a monolithic structure. Only two design patterns (Iterator and Visitor) are used in the C++ generator, whilst 13 are used to organise UML2C.

Table 6.20 summarises the differences in software quality measures between the C++ generator and UML2C.

### TABLE 6.20. Software quality measures of C++ and C code generators

<table>
<thead>
<tr>
<th>Measure</th>
<th>C++ generator</th>
<th>UML2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (LOC)</td>
<td>18,100</td>
<td>4,900</td>
</tr>
<tr>
<td>Abstraction level</td>
<td>Low (code)</td>
<td>High (specification)</td>
</tr>
<tr>
<td>Software architecture</td>
<td>Partial</td>
<td>Detailed</td>
</tr>
<tr>
<td>Modularity</td>
<td>Low (one module)</td>
<td>High (10 Modules)</td>
</tr>
<tr>
<td>Cohesion</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Coupling</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Design patterns</td>
<td>2</td>
<td>13</td>
</tr>
</tbody>
</table>

We can also compare the level of design flaws or technical debt in the C++ translator and in UML2C. For the C++ translator the data has been calculated using the PMD code size library (https://pmd.github.io). For UML2C we have used the following measures of technical debt:

- **ETS**: Excessive transformation size (total complexity >1000, where complexity is the sum of the number of operator and identifier occurrences)
6.1. Case study 1: UML to C Transformation

- **ENR**: Excessive number of rules (nrules >10)
- **ENO**: Excessive number of helpers/operations (nops >10)
- **ERS**: Excessive rule size (>100 identifiers + operators in a rule)
- **EHS**: Excessive helper size (>100 identifiers + operators in a helper)
- **EPL**: Excessive parameter list (for transformation, rules, and helpers: >10 parameters including auxiliary rule variables)
- **CC**: Cyclomatic complexity (of rule logic or of procedural code: >10)

Measures EFO, DC and CBR are not measured by PMD, so are omitted. There are substantial numbers of code clones and inter-operation dependencies in the Java code, however, ETS is taken to be the same as Excessive Class Size in PMD. The threshold values used in PMD are: ETS 1000 LOC; EHS 100 LOC; EPL 10 parameters; CC 10; ENO 10 per class. For comparison, the Technical Debt (TD) figures for UML2C are also given, and these are generally lower than that of the C++ translator (Table 6.21).

**TABLE 6.21. Software quality comparison**

<table>
<thead>
<tr>
<th>Transformation</th>
<th>ETS</th>
<th>EHS + ERS</th>
<th>CC</th>
<th>ENO + ENR</th>
<th>EPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ translator</td>
<td>5</td>
<td>16.5</td>
<td>110.6</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>UML2C</td>
<td>2</td>
<td>13</td>
<td>62</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Case study 2 is the adaptation of a procedure for calculating and evaluating the risk of multiple-share financial investments. The procedure modelled by Hammerlind [51] had to be adapted for our client’s own company. The risk analysis model was implemented in UML-RSDS according to the client’s specific needs in order that it could be used in his company.

This case study has been worked on by our research group. Some of this work has been previously published in [86] and [93]. It concerns the risk evaluation of multiple-share financial investments known as Collateralized Debt Obligations (CDO), where a portfolio of investments is partitioned into a collection of sectors, and there is the possibility of contagion of defaults between different companies in the same sector [30, 51]. Risk analysis of a CDO contract involves computing the overall probability \( P(S = s) \) of a financial loss \( s \) based upon the probability of an individual company defaulting and the probability of default infection within sectors. For this case study, it was required to have an approximate version of the loss estimation function \( P(S = s) \). The case study was carried out in conjunction with a financial risk analyst, who was also the customer of the development. Implementations in Java, C# and C++ were required.

We have used the following formulas: Theorem 1.1, Theorem 3.1 and equations 1 and 2, from Hammarlid [51] in order to calculate the probability of the financial loss. The attribute \( L \) represents the credit loss per default, in each sector. The attribute \( n \) stands for the number of bonds that are subject to risk. The attribute \( k \) stands for a sector out of all possible sectors \( K \). We will convert and adapt the following formula according to UML-RSDS specification in the following sections for this case study to compute the probability of risk.
Theorem 1.1:

\[ P(N_k = m) = \binom{n}{m} (p_k^n (1 - p_k)^{n-m} (1 - q_k)^{n-m}) + \]

\[ \sum_{i=1}^{m-1} \binom{m}{i} p_k^i (1 - p_k)^{n-x-i} (1 - (1 - q_k)^i)^{n-i} \times (1 - q_k)^i (n-m) \]

“When an outbreak occurs in sector \( k \), each single default causes an integer valued credit loss of \( L_k \) and the total loss of \( S_k = N_k L_k \), where \( N_k > 0 \). Conditioned on an outbreak in a sector, the distribution of the number of defaults is”:

**Equation (1):**

\[ P(N_k = m|N_k > 0) = P(N_k = m)/(1 - (1 - p_k)^{n_k}), m \geq 1, \]

and the probability of total credit loss given an outbreak is:

**Equation (2):**

\[ P(S_k = mL_k) = P(N_k = m|N_k > 0) \]

Theorem 3.1:

\[ P(S = 0) = \exp(-\sum_{k=1}^{K} \mu_k) \]

and

\[ P(S = s) = \frac{1}{s} \sum_{k=1}^{K} \sum_{n_k=1}^{\lfloor s/L_k \rfloor} \mu_k m_k L_k P(N_k = m_k L_k|N_k > 0) \times P(S = s - m_k L_k) \]

The requirements for this case study were quite straightforward:

- F1: Compute \( P(S = s) \) for CDO portfolios which is derived from Theorem 3.1 based on the computation of \( P(N_k = m) \) of Hammarlid [51].
6.2. Case Study 2: CDO Risk Estimation

- F1.1: Calculate probability of no contagion.
- F1.2: Calculate probability of contagion.

- F2: Calculate risk function \( P(S \geq s) \).

- NF1: The system must be able to compute results in a practical time (less than 30 seconds for each \( s \) for a portfolio of 20 sectors and 100 companies).

- NF2: The system should be accurate, within 5% of the theoretical exact result.

- F3: The system should be extensible to handle the case of cross-sector companies and cross-sector infection.

Table 6.22 is based upon Tables 5.5–5.8.

**TABLE 6.22.** Technique attributes of the Domain Analysis & Requirements Elicitation stage \( \psi(a_x, t) \) for CDO case

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Brainstorming</th>
<th>Interview</th>
<th>Mining</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting domain knowledge</td>
<td>1</td>
<td>0.6</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>Eliciting implicit knowledge</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Eliciting MT requirements</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Identifying MT stakeholders</td>
<td>1</td>
<td>1</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Facilitating communications</td>
<td>0.8</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Identifying non-functional requirements</td>
<td>1</td>
<td>1</td>
<td>0.8</td>
<td>0.2</td>
</tr>
<tr>
<td>RA((t))</td>
<td>0.88</td>
<td>0.87</td>
<td>0.93</td>
<td>0.9</td>
</tr>
</tbody>
</table>

We applied our RE framework as follows:
6.2. Case Study 2: CDO Risk Estimation

1. category: Domain Analysis & Requirements Elicitation

- $T_{elicit} = \{\text{interview, prototyping, questionnaire, document mining, brainstorming, scenario, ethno methodology}\}$
- $t_1 = \text{interview}, t_2 = \text{brainstorming}, t_3 = \text{document mining}$
- $I(a)$ value is 1 for $A_1$, 0.8 for $A_2$ and 0 for the remaining attributes:
  - $A_1 = \{\text{eliciting MT requirements, getting domain knowledge, getting implicit knowledge}\}$
  - $A_2 = \{\text{identifying non-functional requirements, facilitating communication}\}$

- According to the transformation project attributes, the size of this transformation is small as it has approximately less than 100 rules, therefore a value of 0.2 is given for size. There are two complicating factors: complex rule logic and complex computations, therefore a value of 0.8 is given for complexity. There is 5%-10% change of requirements, therefore a value of 0.2 is given for the volatility attribute. There is good access to the customer, therefore a value of 0.8 is given for customer relationship. It may be used to produce, modify, or analyse safety-related systems but not safety-critical system, therefore a value of 0.2 is given for the safety project attribute. There exists approximately 50 requirements, therefore a value of 0.8 is given for the transformation quality attribute. Up to 20% extension is possible regarding the time attribute, therefore the value of 0.5 is given for the time attribute. The budget is low (up to 50% extension), therefore a value of 0.2 is given for the budget attribute. Developers are not very familiar with the domain (some experience, less than one year of experience), therefore a value of 0.4 is given for the domain knowledge attribute.
6.2. Case Study 2: CDO Risk Estimation

In other words, we have the following:

- Project attributes: size: small (0.2), complexity: high (0.8), volatility: low (0.2), customer-developer relationship: high (0.8), safety: low (0.2), quality: high (0.8), time: medium (0.5), cost: low (0.2), domain knowledge: medium (0.4)

- \(D = \{\text{size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}\}\)

\[S(t_1) = 0.0002\]
\[S(t_2) = 0.003\]
\[S(t_3) = 0.0001\]

2. category: Evaluation & Negotiation

- \(T_{evaluation} = \{\text{prototyping, scenario, UML, functional decomposition, goal oriented analysis}\}\)

- \(t_1 = \text{prototyping}, t_2 = \text{scenario}\)

- \(I(a)\) value is 1 for \(A_1\), 0.8 for \(A_2\) and 0 for the remaining attributes:
  - \(A_1 = \{\text{analysing non-functional requirements, prioritizing requirements}\}\)
  - \(A_2 = \{\text{facilitating negotiation}\}\)

- Project attributes: size: small (0.2), complexity: high (0.8), volatility: low (0.2), customer-developer relationship: high (0.8), safety: low (0.2), quality: high (0.8), time: medium (0.5), cost: low (0.2), domain knowledge: medium (0.4)

- \(D = \{\text{size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}\}\)

\[S(t_4) = 0.0013\]

\[\text{The values have been determined from the transformation project attributes weighting (Table 5.9)}\]
6.2. Case Study 2: CDO Risk Estimation

- \( S(t_2) = 0.0003 \)

3. category: Specification & Documentation

- \( T_{\text{specification}} = \{ \text{SysML, KAOS, structured language template, SADT, UML, evolutionary prototyping} \} \)
- \( t_1 = \text{Structured language template}, t_2 = \text{UML}, t_3 = \text{evolutionary prototyping} \)
- \( I(a) \) value is 1 for \( A_1 \), 0.8 for \( A_2 \) and 0 for the remaining attributes:
  - \( A_1 = \{ \text{semantics completeness, requirements verification} \} \)
  - \( A_2 = \{ \text{writing complete requirements} \} \)
  - \( D = \{ \text{size, complexity, volatility, relationship, safety, quality, time, cost, domain understanding} \} \)
- \( S(t_1) = 0.0002 \)
- \( S(t_2) = 0.0003 \)
- \( S(t_3) = 0.0059 \)

4. category: Validation & Verification

- \( T_{\text{validation}} = \{ \text{prototyping, inspection, desk-checks, GQM, checklist} \} \)
- \( t_1 = \text{prototyping}, t_2 = \text{GQM} \)
- \( I(a) \) value is 1 for \( A_1 \) and 0 for the remaining attributes:
  - \( A_1 = \{ \text{identifying incomplete requirements, identifying inconsistency and conflict} \} \).
  - Project attributes: size: small (0.2), complexity: high (0.8), volatility: low (0.2), customer-developer relationship: high (0.8), safety: low (0.2), quality: high (0.8), time: medium (0.5), cost: low (0.2), domain knowledge: medium (0.4)
6.2. Case Study 2: CDO Risk Estimation

- \( D = \{ \text{size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge} \} \)
  
- \( S(t_1) = 0.0013 \)
  
- \( S(t_2) = 0.0001 \)

First, a phase of research was needed to understand the problem and to clarify the actual computations required. Brainstorming, interview and document mining with the stakeholder were carried out to elicit detailed requirements. The work items were prioritised, with tasks F1.1 and F1.2 being scheduled for a first development iteration, as these were considered more critical than other functionalities. Exploratory and evolutionary prototyping were used within this iteration, with the specification being progressively elaborated and tested until the functionalities were complete and correctly passed all tests. Then, functionality F2 was developed in iteration 2. A further external requirement F3 was introduced prior to this iteration in order to handle the case of crosssector contagion.

The required use cases and subtasks are given in Table 6.23. Use case 3 depends upon tasks F1.1 (2a) and F1.2 (2b) of use case 2.
6.2. Case Study 2: CDO Risk Estimation

<table>
<thead>
<tr>
<th>Use case</th>
<th>Subtasks</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Load data</td>
<td></td>
<td>Read data from a .csv spreadsheet</td>
</tr>
<tr>
<td>2. Calculate Poisson approximation of loss function</td>
<td>2a. Calculate probability of no contagion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2b. Calculate probability of contagion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2c. Combine 2a, 2b</td>
<td></td>
</tr>
<tr>
<td>3. Calculate precise loss function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Write data</td>
<td></td>
<td>Write data to a .csv spreadsheet</td>
</tr>
</tbody>
</table>

The following activities were employed during the RE development:

- **Refactoring**: the solutions of F1.1 and F1.2 were initially expressed as operations nocontagion, contagion of the CDO class (Figure 6.4). It was then realised that they would be simpler and more efficient if defined as Sector operations. The refactoring Move Operation was used. This refactoring did not affect the external interface of the system.

- **Customer collaboration in development**: the risk analyst gave detailed feedback on the generated code as it was produced, and carried out their own tests using data such as the realistic dataset of [51].

- **Iterations**: short iterations were completed within three weeks.

Figure 6.4 shows the class diagram of the solution produced at the end of the first development iteration.

This specification expresses the problem in terms of domain concepts from the CDO financial theory. The attribute \( L \) represents the credit...
6.2. Case Study 2: CDO Risk Estimation

![Diagram of CDO version 1 system specifications]

Figure 6.4. CDO version 1 system specifications

loss per default, in each sector. The attribute p is the probability of a company defaulting, independently from the companies in the sector. The attribute q is the probability of default infection of a company in a sector due to another company defaulting in that same sector, and n is the number of companies in the sector. The attribute mu is the Poisson approximation parameter. It represents the probability of a specific number of events that occur in a particular period of time [49].

![Diagram of CDO version 2 system specifications]

Figure 6.5. CDO version 2 system specifications

In UML-RSDS, use cases define the externally-usable functionalities provided by a system. Their effect is specified by a sequence of OCL postconditions. The declarative interpretation is that the conjunction of these postconditions is established by the use case. The procedural interpretation is that the postconditions are executed sequentially as statements which are UML-structured activities.
6.2. Case Study 2: CDO Risk Estimation

The specification of requirement F1 is expressed as a use case test with the following postconditions:

CDO::
  s : sectors =>
  s.mu = 1 - ( ( 1 - s.p )->pow(s.n) )

CDO::
  ps0 = -sectors.mu.sum->exp()

CDO::
  s : Integer.subrange(0,20) =>
  PS(s)->display()

The first constraint initialises the mu value for each sector s. The second initialises ps0 using these values. The third constraint calculates and displays PS(s) for integer values s from 0 to 20. Note that the arrow operator arg→op(p) is used generally for function application of op(p) to arg.

PS(s) computes the loss function P(S = s) which has been decomposed into combinations of failures in individual companies. P(k,m) is the probability of m defaults in sector k, PCond(k,m) the conditional probability of m defaults in sector k, given at least one default as shown in the following code:

CDO::
  query P(k : int, m : int) : double
  pre: true
  post:
      result = StatFunc.comb(sectors[k].n, m) *
      ( sectors[k].nocontagion(m) +
        Integer.Sum(1,m - 1,i, sectors[k].contagion(i,m)) )

CDO::

210
6.2. Case Study 2: CDO Risk Estimation

query PCond(k : int, m : int) : double
pre: true
post:
(m >= 1 => result = P(k,m) / (1 - ((1 - sectors[k].p)->pow(sectors[k].n))) & (m < 1 => result = 0)

The operation definitions are directly based upon the mathematical specifications. Integer.Sum(a, b, i, e) represents $\Sigma_{i=a}^{b} e$.

PS(s) sums up the sector’s loss function VS(k, s), which sums probability-weighted loss amounts resulting from each of the possible non-zero number of defaults in sector k:

CDO::
query cached PS(s : int) : double
pre: true
post:
(s < 0 => result = 0 ) &
(s = 0 => result = ps0 ) &
(s > 0 => result = Integer.Sum(1,sectors.size,k,VS(k,s))/s )

CDO::
query VS(k : int, s : int) : double
pre: true
post:
result = Integer.Sum(1, maxfails(k,s), mk, ( sectors[k].mu * mk * sectors[k].L * PCond(k,mk) * PS(s - mk * sectors[k].L )))

PS depends upon VS, which in turn depends upon PS. This mutual
6.2. Case Study 2: CDO Risk Estimation

Recursion suggests that optimisation using caching/memoization is necessary for PS.

Here, we will go through a simple example in order to have a better understanding regarding the problem. In this example, (Figure 6.6), we have Sector 1 and Sector 2, each of which contains three companies. Let’s assume that Sector 1 has the following values for its attributes: \( n = 3, \ p = 0.02, \ q = 0.01, \) \( L = 10 \) and Sector 2 has the following values for its attributes: \( n = 3, \ p = 0.05, \ q = 0.03, \) \( L = 8 \). We would like to calculate \( PS(s) \). Note that we have already specified that for this case study, the display \( PS(s) \) for integer value \( s \) is from 0 to 20.

![Figure 6.6. CDO example](image)

The following results have been obtained by UML-RSDS which calculates the loss function:

\[
\begin{align*}
    P(0) &= 0.8175583521347228 \\
    P(1) &= 0.0 \\
    P(2) &= 0.0 \\
    P(3) &= 0.0 \\
    P(4) &= 0.0 \\
    P(5) &= 0.0 \\
    P(6) &= 0.0 \\
    P(7) &= 0.0 \\
    P(8) &= 0.10413595347075202 \\
    P(9) &= 0.0
\end{align*}
\]
6.2. Case Study 2: CDO Risk Estimation

\[
\begin{align*}
  P(10) &= 0.04617347393199138 \\
  P(11) &= 0.0 \\
  P(12) &= 0.0 \\
  P(13) &= 0.0 \\
  P(14) &= 0.0 \\
  P(15) &= 0.0 \\
  P(16) &= 0.018554362881747166 \\
  P(17) &= 0.0 \\
  P(18) &= 0.005881315652160922 \\
  P(19) &= 0.0 \\
  P(20) &= 0.003178989411025038 \\
\end{align*}
\]

We have shown the probability of loss for different amounts. For instance, the probability of \( P = 4 \) is 0 because the quantity of loss must be composed of 8 or 10. This is used to calculate the exact loss of a particular company in a sector. By using the risk function, we can calculate the least amount of a particular loss: \( P(\text{loss} \geq x) = 1 - \sum_{i=0}^{i=x-1} P(i) \) where \( i = x - 1 \).

It was originally intended to use external hand-coded and optimised implementations of critical functions such as the combinatorial function \( \text{comb}(\text{int}n, \text{int}m) \). However, this would have resulted in the need for multiple versions of these functions to be coded, one for each target implementation language, and also it would have increased the time needed for system integration. It was found instead that platform-independent specifications could be given in UML-RSDS which were of acceptable efficiency.

The initial efficiency of the solution was too low, with calculation of \( P(S = s) \) for all values of \( s \leq 20 \) on the test data of [51] taking over 2 minutes on a standard Windows 7 laptop. To address this problem, the recursive operations and other operations with high usage were given the stereotype \( \langle\text{cached}\rangle \) to avoid unnecessary re-computation. This stereotype means that operations are implemented using the memoization technique of [109] to store previously-computed results as shown in Table 6.24. The resulting program is considerably more efficient than the
6.2. Case Study 2: CDO Risk Estimation

original manually-coded C++ version.

Figure 6.7 shows the refactored system specification at the end of the third development iteration (requirement F3).

![CDO version 3 system specifications](image)

**Figure 6.7. CDO version 3 system specifications**

<table>
<thead>
<tr>
<th>Version</th>
<th>Execution time for first 20 $P(S = s)$ calls</th>
<th>Execution time for first 50 $P(S = s)$ calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unoptimised Java</td>
<td>121s</td>
<td>- (more than 15 minutes)</td>
</tr>
<tr>
<td>Optimised Java</td>
<td>32ms</td>
<td>93ms</td>
</tr>
<tr>
<td>C#</td>
<td>10ms</td>
<td>20ms</td>
</tr>
<tr>
<td>C++</td>
<td>62ms</td>
<td>100ms</td>
</tr>
<tr>
<td>Original program</td>
<td>84s</td>
<td>- (more than 15 minutes)</td>
</tr>
</tbody>
</table>

### 6.2.1 Evaluation

Considering the time and effort spent on this case study, not only developers were satisfied with the result of the case study, but also according to the feedback, the client was very pleased with the overall performance. A risk evaluation application was developed according to the client’s request. Despite all the difficulties and problems that the developer team was confronted with throughout this case study, the project was completed before the assigned schedule. There were regular meetings from the beginning with the client to find out about the...
main goals and requirements of this case study. The client’s involvement was most efficient because there was thorough communication either in person through meetings or by sending messages via emails. The main elicitation techniques which were applied to evaluate the requirements of the project were brainstorming, interview and document mining. According to the result of the interview process, several prototypes were designed and presented to the client and the most appropriate ones were selected. Afterwards, the client was asked about the priority for each requirement.

User acceptance testing was another method used to evaluate the success level of the case study. This approach was applied at the completion of the case study. The main purpose of this type of testing was to evaluate the performance of the application with its initial requirements according to the client’s current needs. During one of the last meetings with the client, a live demo of the application was shown to the client which included all the functionalities. Then the customer had the opportunity to verify and validate the application according to a set of test scripts, which had been prepared before the meeting.

Table 6.25 shows the improvement in quality of the UML-RSDS version, where the RE technique framework was applied and the most appropriate RE process was then carried out accordingly, where the original C++ version, with no specific RE process.

<table>
<thead>
<tr>
<th></th>
<th>Size (LOC)</th>
<th>Call graph size</th>
<th>Large clones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original program</td>
<td>194</td>
<td>11 (one cyclic dependency)</td>
<td>2</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>33</td>
<td>11 (two cyclic dependencies)</td>
<td>0</td>
</tr>
</tbody>
</table>
6.3 Framework Evaluation

In this section, we present the result of evaluations for our proposed framework which has been implemented in UML-RSDS as specified in Section 5.5. In order to ensure the framework works efficiently with no problematic issues it was given to candidates for evaluation. Five candidates, both students and academics, were selected from the informatics field. Each candidate was given a five minute presentation regarding the overall idea of the framework and the scoring procedure, 1 being least satisfied and 5 most satisfied. Table 6.26 presents the overall scoring of the participants regarding the use of the framework.

\textbf{TABLE 6.26.} Framework evaluation form

\begin{tabular}{|l|c|}
\hline
\textbf{RE Technique Framework Evaluation Guidelines} & \\
\hline
\textbf{Specific Questions} & \textbf{Score (1-5)} \\
\hline
Is the program compatible with your computers and/or network? & 5 \\
\hline
Is technical assistance readily available via phone or email? & 3 \\
\hline
Is the level of language that the program offers clearly indicated? & 4 \\
\hline
Are the interface, navigation, and the directions clear and logical? & 2 \\
\hline
Does the program include scoring? & 5 \\
\hline
If a scoring system is used, does it make sense? & 5 \\
\hline
Can the learner easily quit something that is beyond his/her ability? & 5 \\
\hline
If the program includes pictures, are they (a) relevant, (b) an aid to understanding? & 1 (N/A) \\
\hline
If the program includes sound recordings, are they of an adequate quality? & 1 (N/A) \\
\hline
If the program includes video sequences, are they of an adequate quality? & 1 (N/A) \\
\hline
Overall satisfaction? & 4 \\
\hline
\end{tabular}

216
6.4 Summary

Systematic software engineering of model transformations is only practised in a minority of MT developments, according to surveys of MT development [11, 154]. The emphasis in MT developments has been on implementation, with less attention paid to requirements engineering. One example of a detailed development process is the migration case study of [130], which describes the techniques used in this industrial project. Details of the development process for an industrial transformation project are also provided in [111]. We have given a detailed description of the development process and engineering techniques used, together with evaluations of their effectiveness.

Code generation from UML to ANSI C is also an unusual topic, with only one recent publication describing such a translator [43]. This code generator is described in a high-level manner, and it is not clear how OCL expressions or UML activities are mapped to C using the transformation. In contrast, by means of detailed requirements analysis, we have produced explicit mappings for all elements of a substantial subset of UML, including OCL. The quality of the UML2C translator was a substantial improvement over manually-coded translator.

Regarding the CDO case study, scenario analysis led to the definition of use cases, which provide a good structuring mechanism for financial applications. Computation steps within a financial process can be expressed as successive postconditions within a use case, and separate stages in a process can be defined as separate use cases, which are then included in a use case which coordinates the sequencing of the stages. The quality of the CDO version was substantially improved from the previous C++ version.

The case studies have identified the need for a well-defined RE process, and to this effect a framework for selecting suitable RE techniques has been created for using UML-RSDS for MT development, and some techniques for improving the adoption and application of UML-RSDS, in addition to necessary technical improvements in the tools.
6.4. Summary

In general, it was found that a development approach using exploratory prototyping (of the system specification) at the initial stages, and evolutionary prototyping at later stages, was effective. By applying the RE framework on the case studies and achieving positive results and feedback from the stakeholders, we can conclude that the process can be used to develop specifications in a range of declarative and hybrid MT languages and projects. Overall, the requirements engineering framework provided a systematic basis for the construction of both case studies, leading to an improved outcome compared to the (implementation-focused) development of other UML-RSDS projects. In particular, premature commitment to poor code synthesis strategies in the UML to C case was avoided, and the modularity of the generator was considerably improved compared to the C++ translator. Similar improvements were achieved with refactoring transformations [151].

In this chapter we chose to apply our proposed framework to a specific language, namely UML-RSDS. However, since our proposed framework is designed in a language-independent manner, its usage is not limited to this specific language and can be applied to other languages such as ATL, ETL, etc.

We have identified ways in which requirements engineering can be applied systematically to model transformations. Comprehensive catalogues of functional and non-functional requirements categories for model transformations have been defined. We have examined a case study which is typical of the current state of the art in transformation development, and identified how formal treatment of functional and non-functional requirements can benefit such developments. We have proposed such a process, and identified RE techniques that can be used in this process. Moreover, we have identified a requirements engineering process for model transformations, and requirements engineering techniques that can be used in this process modelling. The use of a systematic requirements engineering process also helped to capture and make explicit all requirements, avoiding ambiguity over the development tasks. The process can be used to develop specifications in a range of declarative and
6.4. Summary

hybrid MT languages. We have evaluated the process and techniques on two large scale case studies, *UML to C* translation and the CDO.
Chapter 7

Summary and Concluding Remarks

7.1 Introduction

We have investigated the requirements engineering process in model transformations development on different case studies using a systematic RE framework.

In this thesis, we have suggested improvements to the requirements engineering process for transformations in the form of added rigor. Requirements in model transformation could be divided into two categories: functional and non-functional. Functional requirements mainly consider the functional effect on the transformed model, whereas non-functional requirements consider the quality of the transformation and the transformed model. In general, it could be said that at the present time RE process is not being performed efficiently in model transformation. RE techniques are used in the MT development, but these are not used as part of a structured RE process. Individual techniques are used in isolation and are not integrated across RE stages. The lack of any RE guidance or process specific for MTs means that RE techniques are used in an ad-hoc manner for MTs, without any justification that they are appropriate. Developers do not measure the degree of satisfaction of the
7.2 Objectives of Research

The thesis identified and defined a number of goals and objectives, specifically:

- Carrying out a survey on different industrial and academic transformation projects
- Carrying out an interview-based study on different industrial transformation projects
- Defining a requirements engineering process for MT
- Defining a taxonomy for functional and non-functional requirements for MT
- Defining a method for selecting suitable RE techniques for MT
- Validating the choice of RE methods and techniques through two case studies

7.3 Overview of Thesis

Chapter 2 presents a broad background about requirements engineering and model transformations and their related concepts. It provides some discussion on the requirements engineering process and methodology followed by an explanation of MT as a central concept of MDE. This chapter is about the current application of requirements engineering, its
advantages, process models, methodologies and techniques. It also explains model transformation, its current application, its languages, its relationship with model-driven engineering, and its context and various types with some use cases. Moreover, requirements have been categorised according to their type (functional and non-functional), and a RE process model has been investigated and analysed. Furthermore, it presents the definition and properties of MT and its relation to MDE community as well as comparing different MT languages and tools. Three different types of MT examples: refactoring, migration, and refinement, were also presented to provide a better understanding of MT.

Chapter 3 details an empirical analysis of RE in industrial MT projects and impact of RE in MT. In empirical research, we have carried out in-depth interviews with industrial practitioners, covering different MT applications. This chapter presents the requirements engineering techniques that were used in the MT development process. One conclusion that can be drawn from this chapter is that relatively few RE techniques are used in MT development, and these are not used as part of a structured RE process. Finally this chapter ends with evaluating the outcomes of the projects, the development effort and the encountered problems are analysed, together with the degree to which the delivered transformation achieved customer expectations.

Chapter 4 highlights the current status quo of RE in MT by providing a systematic literature survey in which several MT case studies have been analysed from a RE aspect. It provides the results of a systematic literature review (SLR) of the current process of requirements engineering in MT developments. 160 papers have been reviewed and analysed from the past 10 years. The overall result of this analysis shows that although developers apply some requirements engineering process and techniques in transformation developments, this is often based on their experience and common sense, and there are no systematic requirements engineering processes designed for model transformation development. It provides the results achieved from the SLR followed by discussing the shortcomings to the validity of the results. Finally, this chapter ends with
7.4 Limitations

a discussion of the advantages of RE in MT and provides suggestion areas for RE in MT investigation.

In Chapter 5, criteria for selecting appropriate requirements engineering techniques for MT have been identified, and we propose a framework for this selection process. This framework aims to facilitate the process of choosing an appropriate set of requirements engineering techniques according to the type of model transformation project. Furthermore, this chapter analyses the attributes of RE techniques and of the organization in which the project is delivered, and the actual type of MT project for selecting a suitable set of RE techniques for a specific MT project. Finally this chapter presents an MT example in which our proposed framework is applied.

Chapter 6, which could be regarded as the evaluation chapter of this thesis, illustrates the application of our RE process and RE selection procedure on two real substantial MT case studies using UML-RSDS:

- **UML to C**: This code-generation transformation is intended to map UML class diagrams, OCL and activity pseudocode into ANSI C.

- **Collateralized Debt Obligations**: CDO concerns the risk evaluation of multiple-share financial investments, where a portfolio of investments is partitioned into a collection of sectors, and there is the possibility of contagion of defaults between different companies in the same sector.

7.4 Limitations

The first limitation of this research was related to lack of resources regarding this research topic. As requirements engineering is quite a novel topic in the model transformation field, there is not much research and work available. Thus, it was not very convenient to evaluate the present status of RE in MT. Consequently, we decided to do a systematic literature review as well as an interview-based study to understand the
7.5. Future Work

There is substantial potential for the application of requirements engineering in model transformation development. This section will highlight a specific set of future work that could possibly be undertaken to extend this research.

7.5.1 Requirements Management in MT

Evolution is at the heart of model transformation technology which has a direct impact on the RE process as it may introduce new requirements and cycles. If we want to build an evolvable system, we are required to anticipate potential changes regarding the requirements. During MT development and after delivering the MT project, new problems or challenges may arise over time as the development of technology advances. Therefore, during the RE process beyond system-to-be (Chapter 2) we may need to consider system-to-be-next. “The process of anticipating, evaluating, agreeing on and propagating changes to requirements documentation items are so called requirements management” [143]. Requirements management is an activity which can be carried out during all four RE stages. The RE process model defined by [79] is illustrated in Figure 7.1.
7.5. Future Work

Missing from this research is the requirements management activity, its specific characteristics and the capability to identify these characteristics. Future work could consider the incorporation of this idea into the proposed RE framework for MT.

7.5.2 Applying the Framework to Several Cases

The proposed RE framework is applied on two substantial model transformation case studies in this thesis. We have identified a requirements engineering process for model transformations, and requirements engineering techniques that can be used in this process. The process can be used to develop specifications in a range of declarative and hybrid MT languages. We have evaluated the process and techniques on refactoring, migration and code generation cases with positive results. However, the framework could be used to evaluate the most suitable requirements engineering technique(s) for any kind of transformation project.
7.5. Future Work

7.5.3 Integration with transML

The modelling language transML \[15\] represents requirements in the form of SysML diagrams. transML aims to cover the whole life-cycle of transformation development such as requirements, analysis, design and testing. Traceability allows engineers to link requirements of a model transformation to its corresponding analysis and design models, code and other artifacts. It is important for engineers to be able to understand the connection between different artifacts in a model transformation process, which allows them to check whether or not all requirements have been applied correctly \[156\]. Although the significant role of traceability is well-established, in practice, it has not yet been widespread. This is due to the fact that different languages in model transformations do not support the traceability creation and maintenance sufficiently. This would cause problems especially in large developments which tend to create and maintain a quite large amount of traceability links throughout the transformation \[156\].

Potential future work could integrate the RE framework in this thesis with transML by providing specific requirements (functional and non-functional) metrics for source, target and the transformation itself.

7.5.4 Further Extension of the Framework

This thesis has introduced a framework for selecting the most suitable RE technique based upon the concept of attributes, namely: technique attribute, transformation project attribute and organizational attribute. This framework can evidently be expanded to hold a broader range of attributes and attributes’ properties. We realize that the number of techniques, transformation projects and organization attributes can be expanded. Moreover, by introducing the requirements management phase, new properties can be introduced to these attributes which allows further research to be undertaken in the future.
7.6 Concluding Remarks

This thesis has proposed a systematic RE framework for MT development based upon the results of interviews with MT practitioners and a systematic literature review of the current process of requirements engineering in MT developments. Seven practitioners were interviewed, covering 10 projects, and 160 papers have been reviewed and analysed from the past 10 years. The framework has proposed different attributes of MT development such as: RE technique attribute, MT project attribute and the experience level of the developer in a particular RE technique. 33 technique attributes and nine MT project attributes have been introduced. The selection of techniques and project attributes was mainly based on the result from the SLR and the interview-study, however it has to be mentioned that there is no limit to the number of possible techniques and project attributes and in this thesis, only a sample of attributes has been selected.

The overall SLR and interview-based study analysis shows that although developers apply some requirements engineering process and techniques in transformation developments, this is often based on their experience and common sense, and there are no systematic requirements engineering processes designed for model transformation development. Having a systematic and validated framework to perform the requirements engineering process for different MT development projects would allow the MT developers to select the most appropriate (suitable) RE technique for a particular requirement/sub-requirement in a MT project. Applying this framework had a positive impact on time, cost and efficiency of the end product. The framework has been validated through real case studies. In the evaluation section, the capability and suitability of the proposed framework is validated, as the requirements of the case study were achieved successfully.
References


References


References


References


References


References


[56] HNATKOWSKA, B., AND MAZUREK, P. Verification of UML Class Diagrams against Business Rules Written in Natural Language. In
References


References


References


References


References


References


References


References


243


[154] Yassipour Tehrani, S., Zschaler, S., and Lano, K. Requirements Engineering in Model-Transformation Development:
References


Appendix A

SLR Tables
<table>
<thead>
<tr>
<th>Case #</th>
<th>How to find what was required</th>
<th>Stakeholders</th>
<th>Reaching stakeholders</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-11</td>
<td>Problem description, prototype and examples</td>
<td>Financial companies, case study proposers</td>
<td>Forum, email</td>
</tr>
<tr>
<td>12-20</td>
<td>Problem description, prototype and examples</td>
<td>Users requiring analysis of data</td>
<td>Forum, email</td>
</tr>
<tr>
<td>21</td>
<td>Research</td>
<td>ArchStudio users</td>
<td>None</td>
</tr>
<tr>
<td>22</td>
<td>Problem description, prototype and examples</td>
<td>MDE practitioners</td>
<td>None</td>
</tr>
<tr>
<td>23</td>
<td>Document mining</td>
<td>Software modellers/developers</td>
<td>None</td>
</tr>
<tr>
<td>24, 30, 124, 142, 157-159</td>
<td>Document mining</td>
<td>MDE practitioners</td>
<td>None</td>
</tr>
<tr>
<td>25</td>
<td>Interviews, document mining</td>
<td>Automotive system (VCS) developers, General Motors</td>
<td>Consultation with domain experts</td>
</tr>
<tr>
<td>26, 70, 71, 72, 76, 78, 80, 95-97, 99, 100, 103-106, 110, 111, 113, 116-119, 121, 125-130, 132, 133, 135, 138, 139, 146, 149, 150, 153-155, 160</td>
<td>Research</td>
<td>MDE practitioners</td>
<td>None</td>
</tr>
<tr>
<td>27</td>
<td>Brainstorming, customer as developer</td>
<td>End users of FrontArena software, FrontArena software teams, SunGard management</td>
<td>Constant feedback, participation in development</td>
</tr>
<tr>
<td>28</td>
<td>Problem statement</td>
<td>Software developers (users of migrated models)</td>
<td>None</td>
</tr>
<tr>
<td>29</td>
<td>Document mining</td>
<td>RTES developers, testers companies with RTES products</td>
<td>None</td>
</tr>
<tr>
<td>31-57</td>
<td>Problem description, prototype and examples</td>
<td>MDE practitioners</td>
<td>Forum, email</td>
</tr>
<tr>
<td>58</td>
<td>Document mining</td>
<td>Business process modeling users</td>
<td>None</td>
</tr>
<tr>
<td>59</td>
<td>Document mining</td>
<td>Risk analysts</td>
<td>Industry collaboration</td>
</tr>
<tr>
<td>Case #</td>
<td>How to find what was required</td>
<td>Stakeholders</td>
<td>Reaching stakeholders</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------</td>
<td>--------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>60</td>
<td>Document mining</td>
<td>Alloy Analyzer users</td>
<td>None</td>
</tr>
<tr>
<td>61</td>
<td>General problem description</td>
<td>Graph transformaion users</td>
<td>None</td>
</tr>
<tr>
<td>62</td>
<td>General problem description</td>
<td>Graphics and JavaScript application users</td>
<td>None</td>
</tr>
<tr>
<td>63</td>
<td>General problem description</td>
<td>MDE practitioners</td>
<td>None</td>
</tr>
<tr>
<td>64</td>
<td>General problem description</td>
<td>Use case model users</td>
<td>None</td>
</tr>
<tr>
<td>65</td>
<td>Document mining</td>
<td>C++ users (legacy code)</td>
<td>None</td>
</tr>
<tr>
<td>66</td>
<td>Research</td>
<td>Manufacturing system designers</td>
<td>None</td>
</tr>
<tr>
<td>67</td>
<td>Research</td>
<td>System migration engineers</td>
<td>None</td>
</tr>
<tr>
<td>68</td>
<td>Document mining</td>
<td>OCL users</td>
<td>None</td>
</tr>
<tr>
<td>69</td>
<td>Communication with domain experts</td>
<td>Volvo Cars Group</td>
<td>Consultations, demonstrations</td>
</tr>
<tr>
<td>73, 77</td>
<td>Research</td>
<td>Web application developers</td>
<td>None</td>
</tr>
<tr>
<td>74, 79</td>
<td>Research</td>
<td>Control engineers</td>
<td>None</td>
</tr>
<tr>
<td>75</td>
<td>Research</td>
<td>MT developers</td>
<td>None</td>
</tr>
<tr>
<td>81-94</td>
<td>Problem statement</td>
<td>MDE practitioners</td>
<td>Forum, email</td>
</tr>
<tr>
<td>98</td>
<td>Research</td>
<td>Process control practitioners</td>
<td>Discussions with experts</td>
</tr>
<tr>
<td>101</td>
<td>Research</td>
<td>Game application developers</td>
<td>None</td>
</tr>
<tr>
<td>102</td>
<td>Research</td>
<td>MDE practitioners, web application developers</td>
<td>None</td>
</tr>
<tr>
<td>107</td>
<td>Research</td>
<td>MDE practitioners, critical system developers</td>
<td>None</td>
</tr>
<tr>
<td>109</td>
<td>Business modelling</td>
<td>MDE practitioners</td>
<td>None</td>
</tr>
<tr>
<td>112</td>
<td>Research</td>
<td>Users exchanging web rules</td>
<td>None</td>
</tr>
<tr>
<td>114</td>
<td>Research, business analysis</td>
<td>Re-engineering projects</td>
<td>Via consortium with industrial users</td>
</tr>
<tr>
<td>120</td>
<td>Industry collaboration</td>
<td>Embedded sys. developers</td>
<td>Consultation</td>
</tr>
<tr>
<td>122, 123</td>
<td>Research, document mining</td>
<td>MDE practitioners</td>
<td>Experimenting with practitioners</td>
</tr>
<tr>
<td>Case #</td>
<td>How to find what was required</td>
<td>Stakeholders</td>
<td>Reaching stakeholders</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------</td>
<td>--------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>131</td>
<td>Research</td>
<td>WSN developers</td>
<td>None</td>
</tr>
<tr>
<td>134</td>
<td>Research</td>
<td>Software developers</td>
<td>None</td>
</tr>
<tr>
<td>136</td>
<td>Document mining</td>
<td>Activity diagram modellers</td>
<td>None</td>
</tr>
<tr>
<td>137</td>
<td>Document mining</td>
<td>Quantity surveyors</td>
<td>None</td>
</tr>
<tr>
<td>140</td>
<td>Research</td>
<td>Surveillance sys. developers</td>
<td>Participation in development</td>
</tr>
<tr>
<td>141</td>
<td>Document mining, consultation with domain experts</td>
<td>Embedded system developers</td>
<td>Collaboration with practitioners</td>
</tr>
<tr>
<td>143</td>
<td>Research</td>
<td>UI developers</td>
<td>None</td>
</tr>
<tr>
<td>145</td>
<td>Document mining</td>
<td>DES analysts</td>
<td>None</td>
</tr>
<tr>
<td>147</td>
<td>Research</td>
<td>Businesses requiring software modernisation</td>
<td>None</td>
</tr>
<tr>
<td>148</td>
<td>Research</td>
<td>Workflow analysts</td>
<td>None</td>
</tr>
<tr>
<td>151</td>
<td>Document mining</td>
<td>Satellite operators</td>
<td>Industrial collaboration</td>
</tr>
<tr>
<td>152</td>
<td>Document mining</td>
<td>Digital forensic experts</td>
<td>User collaboration</td>
</tr>
<tr>
<td>156</td>
<td>Research</td>
<td>MDE practitioners</td>
<td>Industrial collaboration</td>
</tr>
<tr>
<td>161</td>
<td>Problem statement</td>
<td>MDE practitioners</td>
<td>Consultation</td>
</tr>
<tr>
<td>162</td>
<td>Research</td>
<td>SOA developers</td>
<td>None</td>
</tr>
<tr>
<td>Category</td>
<td>Requirement</td>
<td>Cases</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Local</td>
<td>Local mapping requirements</td>
<td>1, 5, 6, 9, 12, 13, 15, 16, 19, 20, 24, 25, 35, 39, 54, 60, 65, 66, 68, 81-83, 85, 102, 104-111, 116-120, 122-132, 136, 139-141, 143, 147, 151-153, 156-161</td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td>Local refactoring requirements</td>
<td>23, 35, 36, 40, 41, 53, 57, 64, 70, 71, 135</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Local reactive requirements</td>
<td>26, 46, 47, 51, 56, 76, 145, 155</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Local correspondence requirements</td>
<td>32, 59, 154</td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td>Syntactic correctness</td>
<td>1, 2, 3, 5, 8, 10, 24, 25, 39, 58, 60, 65, 70, 71, 73, 75, 77-80, 116-119, 127, 128, 134, 143, 146-151</td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td>Completeness</td>
<td>3, 29, 38, 47, 55, 58, 61, 72, 119, 151</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Semantic correctness</td>
<td>14, 24, 25, 28, 29, 31, 44, 47, 50, 52, 53, 55, 57-69, 72-75, 79, 80, 84, 86-94, 105-107</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Semantic preservation</td>
<td>96, 101, 102, 105-107, 112-114, 119, 120, 124, 127, 128, 132, 133, 135, 136, 139, 142, 146, 148, 149, 156-159</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Confluence</td>
<td>36, 37, 72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bidirectionality</td>
<td>24, 32, 43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Model synchronisation/Change-propagation</td>
<td>21, 42, 45, 58, 22, 121, 138, 154</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Traceability</td>
<td>111, 121, 137</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Invariance</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accuracy</td>
<td>59, 75, 98, 137, 162</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Structural preservation</td>
<td>157, 158</td>
<td></td>
</tr>
<tr>
<td>Non-Functional</td>
<td>Efficiency</td>
<td>1, 3, 5, 8, 9, 11-19, 22, 31, 33, 34, 36-44, 46-54, 56, 57, 59, 61, 72, 84, 86-94, 101, 137, 138, 151</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Simplicity/Clarity</td>
<td>1, 28, 44, 47, 58, 81-86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>1, 5, 8, 9, 10, 11, 129</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modularity</td>
<td>1, 7, 9, 10, 11, 35, 63, 95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Size/Conciseness</td>
<td>6, 28, 34, 46, 81-86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scalability</td>
<td>12, 13, 15-19, 25, 27, 29, 147, 151</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Timing</td>
<td>26, 69, 76, 145, 155</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process Responsiveness</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reduced development effort</td>
<td>100, 110, 111</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Architectural quality</td>
<td>27, 95, 102</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flexibility</td>
<td>30, 78, 97, 103, 108, 129, 137, 140-142, 152</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Portability</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Usability</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Conformance to standards</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Alignment to business view</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Genericity</td>
<td>92-94</td>
<td></td>
</tr>
<tr>
<td>Case #</td>
<td>Type of transformation</td>
<td>Size/Scale</td>
<td>Green/Brownfield</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------</td>
<td>------------</td>
<td>------------------</td>
</tr>
<tr>
<td>1-11</td>
<td>Text-to-model, migration, code generation</td>
<td>Small</td>
<td>Green</td>
</tr>
<tr>
<td>12-20</td>
<td>Refinement, data analysis</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>21, 22</td>
<td>Bidirectional</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>23, 36-38, 71</td>
<td>Refactoring</td>
<td>Small</td>
<td>Green</td>
</tr>
<tr>
<td>24</td>
<td>Bidirectional</td>
<td>Large</td>
<td>Green</td>
</tr>
<tr>
<td>25, 60, 114</td>
<td>Migration</td>
<td>Large</td>
<td>Green</td>
</tr>
<tr>
<td>26, 46-48, 51, 56, 76</td>
<td>Reactive</td>
<td>Small</td>
<td>Green</td>
</tr>
<tr>
<td>27, 155</td>
<td>Refinement, migration</td>
<td>Large</td>
<td>Green</td>
</tr>
<tr>
<td>28, 78</td>
<td>Migration</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>29</td>
<td>Refinement, code generation</td>
<td>Large</td>
<td>Green</td>
</tr>
<tr>
<td>30</td>
<td>Refinement</td>
<td>Large</td>
<td>Green</td>
</tr>
<tr>
<td>31, 34, 35, 39, 147</td>
<td>Text/code-to-model</td>
<td>Small</td>
<td>Green</td>
</tr>
<tr>
<td>33, 70</td>
<td>Refactoring</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>43, 45, 50, 53, 55, 57</td>
<td>Java refactoring</td>
<td>Small</td>
<td>Green</td>
</tr>
<tr>
<td>44, 52, 54</td>
<td>Model execution</td>
<td>Small</td>
<td>Green</td>
</tr>
<tr>
<td>59, 141</td>
<td>Code generation</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>61</td>
<td>Semantic mapping</td>
<td>Large</td>
<td>Green</td>
</tr>
<tr>
<td>62</td>
<td>Migration/refinement</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>63, 69</td>
<td>Code generation</td>
<td>Large</td>
<td>Green</td>
</tr>
<tr>
<td>64</td>
<td>Refactoring</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>65, 151</td>
<td>Code-to-code</td>
<td>Large</td>
<td>Green</td>
</tr>
<tr>
<td>66</td>
<td>Migration</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>67, 81-88, 99, 106, 112, 118, 132, 146</td>
<td>Migration</td>
<td>Small</td>
<td>Green</td>
</tr>
<tr>
<td>68, 72, 107, 139</td>
<td>Semantic mapping</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>73, 77, 80</td>
<td>Refinement</td>
<td>Medium</td>
<td>Green</td>
</tr>
<tr>
<td>74, 79, 92-94, 103-105, 109-111, 122-125, 128, 130, 134, 137, 150, 157, 158, 160</td>
<td>Refinement</td>
<td>Small</td>
<td>Green</td>
</tr>
</tbody>
</table>
### TABLE A.6. MT project information (2)

<table>
<thead>
<tr>
<th>Case #</th>
<th>Type of transformation</th>
<th>Size/Scale</th>
<th>Green/Brownfield</th>
<th>Academic/Industry</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>Abstraction</td>
<td>Small</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>95, 162</td>
<td>Refinement, code generation</td>
<td>Small</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>98, 108, 140</td>
<td>Code generation</td>
<td>Medium</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>102</td>
<td>Migration, refactoring</td>
<td>Small</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>113</td>
<td>Reverse engineering</td>
<td>Medium</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>120, 161</td>
<td>Refinement</td>
<td>Medium</td>
<td>Green</td>
<td>Industry</td>
</tr>
<tr>
<td>121, 154</td>
<td>Bidirectional</td>
<td>Small</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>142</td>
<td>Migration</td>
<td>Small</td>
<td>Green</td>
<td>Industry</td>
</tr>
<tr>
<td>145</td>
<td>Reactive</td>
<td>Medium</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>149</td>
<td>Refinement</td>
<td>Small</td>
<td>Brown</td>
<td>Academic</td>
</tr>
<tr>
<td>152</td>
<td>Code generation</td>
<td>Small</td>
<td>Green</td>
<td>Industry</td>
</tr>
<tr>
<td>156</td>
<td>Abstraction</td>
<td>Medium</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>159</td>
<td>Refinement, refactoring</td>
<td>Small</td>
<td>Green</td>
<td>Academic</td>
</tr>
<tr>
<td>Case #</td>
<td>Methodology</td>
<td>Diagrams</td>
<td>RE technique/process</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1-11, 33, 34, 47, 68, 70, 71, 76-78, 81, 93, 102, 108, 110, 112, 113, 131, 134, 137, 138, 146</td>
<td>None</td>
<td>UML class diagram</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>12-20</td>
<td>None</td>
<td>UML class diagram</td>
<td>Exploratory prototyping and scenario analysis</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>MDE for MT</td>
<td>UML class diagram</td>
<td>None identified. Some concrete grammar diagrams used to express synchronisation rules</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Incremental development, evolutionary prototyping</td>
<td>UML class diagram</td>
<td>Experimental prototyping and scenario-based analysis, concrete grammar diagrams</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>None</td>
<td>UML class diagram</td>
<td>Not explicit; scenario-based experimental prototyping</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Incremental, iterative</td>
<td>UML class diagram</td>
<td>No explicit RE process, concrete grammar of the two languages used to express correspondence rules</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Incremental, iterative development using MD-Workbench, validation and testing</td>
<td>UML, class diagram</td>
<td>Structured interviews, scenarios, detailed RE process</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Incremental, experimentation</td>
<td>UML class diagram</td>
<td>Concrete grammar rules and scenario analysis are used for RE</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Agile MDE</td>
<td>None</td>
<td>Survey of developers, reverse engineering of legacy code</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>MDE</td>
<td>UML class diagram</td>
<td>Scenario analysis, exploratory prototyping, observation, interviews, concrete grammar used to express mapping examples</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Design patterns, verification through testing and inspection</td>
<td>UML class diagram</td>
<td>Observation (implicit), individual expert knowledge, literature analysis</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Incremental prototyping</td>
<td>UML class diagram, state machines</td>
<td>Scenario analysis, experimental prototyping, rules expressed using concrete grammar</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>None</td>
<td>UML class diagram</td>
<td>Experimental prototyping</td>
<td></td>
</tr>
<tr>
<td>32, 36, 38, 46</td>
<td>Incremental, iterative</td>
<td>UML class diagram</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>None</td>
<td>UML class diagram</td>
<td>Scenario analysis, informal mappings</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Incremental, iterative</td>
<td>UML class diagram</td>
<td>Survey and questionnaire, prioritisation, empirical studies with representative users and tasks</td>
<td></td>
</tr>
<tr>
<td>Case #</td>
<td>Methodology</td>
<td>Diagrams</td>
<td>RE technique/process</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------</td>
<td>------------------------------------</td>
<td>------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>39, 50, 57, 58</td>
<td>None</td>
<td>UML class diagram</td>
<td>Prioritization</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Design patterns, verification through testing and inspection</td>
<td>UML class diagram</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>41, 42, 51</td>
<td>None</td>
<td>None</td>
<td>Prioritization</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>None</td>
<td>None</td>
<td>Exploratory prototyping and scenario analysis</td>
<td></td>
</tr>
<tr>
<td>44, 45, 52, 53, 55, 56, 74, 82, 86, 87, 126, 132</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Incremental, iterative</td>
<td>UML class diagram</td>
<td>Validation scenarios</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>Incremental, iterative</td>
<td>Object diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>None</td>
<td>Concrete syntax</td>
<td>Prioritization</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>MDE for MT</td>
<td>UML class diagram,OCL</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>124, 127, 130, 162</td>
<td>None</td>
<td>UML class diagram</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>MDA for MT</td>
<td>OCL, UML class diagram</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>None</td>
<td>UML class diagram</td>
<td>Scenarios, OCL, UML, Constructive Query Containment (CQC) method</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>None</td>
<td>Feature diagrams</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>Refactoring</td>
<td>OCL, UML class diagram</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Use case-driven approach</td>
<td>OCL, UML class diagram</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>None</td>
<td>None</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>None</td>
<td>Concrete grammar</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>MDA for MT</td>
<td>Concrete grammar</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>Agile MDE</td>
<td>Simulink models</td>
<td>prototyping</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>None</td>
<td>UML activity diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>None</td>
<td>BPMN diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>None</td>
<td>Specialised</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>None</td>
<td>Control block diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Case #</td>
<td>Methodology</td>
<td>Diagrams</td>
<td>RE technique/process</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
<td>------------------------------------</td>
<td>---------------------------------------</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>MDE</td>
<td>UML class diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>83, 95, 129</td>
<td>None</td>
<td>UML class, activity diagrams</td>
<td>Scenario analysis</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>None</td>
<td>UML class, activity diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>85, 88</td>
<td>None</td>
<td>UML activity diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>89, 90</td>
<td>None</td>
<td>Graph diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>91, 101</td>
<td>None</td>
<td>UML class, state-machine diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>92, 155</td>
<td>None</td>
<td>Customised</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>94, 96, 109</td>
<td>None</td>
<td>UML class diagrams</td>
<td>Informal specifications</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>Iterative development</td>
<td>UI task models, state-charts</td>
<td>Prototyping</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>Iterative development</td>
<td>Process models</td>
<td>Prototyping</td>
<td></td>
</tr>
<tr>
<td>99, 160</td>
<td>MDE for MT</td>
<td>UML class diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>None</td>
<td>UML class diagrams, architecture diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Derive MTs from logical specifications</td>
<td>Architecture diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>MDA</td>
<td>UML class, state-machine diagrams</td>
<td>Informal mappings</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>Prototyping</td>
<td>UML class, state-machine diagrams</td>
<td>Informal specifications</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>Formal methods</td>
<td>Object diagrams/ graphs</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>Formal methods</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>None</td>
<td>UML activity, sequence diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>MT embedded in re-engineering process</td>
<td>UML class diagrams, activity diagrams</td>
<td>Domain analysis</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>None</td>
<td>UML class diagrams, concrete grammar</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>118, 119</td>
<td>None</td>
<td>UML class diagrams, concrete grammar</td>
<td>Formal specifications</td>
<td></td>
</tr>
<tr>
<td>120, 121, 128, 136</td>
<td>None</td>
<td>UML class diagrams, concrete grammar</td>
<td>Concrete grammar rule specifications</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>Specification by example</td>
<td>UML class diagrams</td>
<td>Concrete grammar rule specifications</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>None</td>
<td>UML class diagrams</td>
<td>Semi-formal rule specifications, experimental evaluation</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>MDE for MT</td>
<td>UML class diagrams</td>
<td>Scenarios, formal rule, specifications, partial RE process</td>
<td></td>
</tr>
<tr>
<td>Case #</td>
<td>Methodology</td>
<td>Diagrams</td>
<td>RE technique/process</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>None</td>
<td>None</td>
<td>Formal rule specifications</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>None</td>
<td>UML class diagrams, sequence diagrams</td>
<td>Formal rule specifications</td>
<td></td>
</tr>
<tr>
<td>139, 145, 150, 152</td>
<td>None</td>
<td>UML class diagrams</td>
<td>Formal rule specifications</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>Iterative, incremental</td>
<td>UML class diagrams, feature models</td>
<td>Stakeholder participation</td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>Iterative, incremental</td>
<td>UML class diagrams</td>
<td>Stakeholder identification, stakeholder collaboration</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>None</td>
<td>UML class diagrams, use case diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>Validation using industrial cases</td>
<td>UML class diagrams, concrete syntax diagrams,</td>
<td>Concrete syntax mapping rules</td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>Validation using industrial cases</td>
<td>Concrete syntax diagrams</td>
<td>Concrete syntax mapping rules</td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>MDA</td>
<td>UML class diagrams</td>
<td>Formal rule specifications</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>Phased</td>
<td>UML class diagrams</td>
<td>Graphical mapping specifications</td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>Formal methods</td>
<td>Concrete grammar diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>MDE</td>
<td>UML class diagrams</td>
<td>Semi-formal specification</td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>Formal methods</td>
<td>UML class diagrams, concrete syntax diagrams</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>Formal methods</td>
<td>Concrete syntax diagrams</td>
<td>Formal specification</td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>Formal methods</td>
<td>UML class diagrams</td>
<td>Formal specification</td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>Controlled experiments</td>
<td>UML use case, state-chart diagrams</td>
<td>Expert validation, goal decomposition, textual specifications</td>
<td></td>
</tr>
<tr>
<td>Cases</td>
<td>Requirements expression, techniques &amp; process</td>
<td>Requirements achievement</td>
<td>Stakeholder satisfaction</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------------------------</td>
<td>--------------------------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>high</td>
<td>high</td>
<td></td>
</tr>
<tr>
<td>2, 32, 93</td>
<td>1</td>
<td>low</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>3, 43, 50, 56, 57, 84, 85, 92</td>
<td>1</td>
<td>med</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>high</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>6, 7, 9, 33, 36, 47, 48, 62, 83, 86, 89, 91, 31, 39, 158, 74, 161, 108</td>
<td>2</td>
<td>med</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>8, 23, 54, 77, 90, 94, 95, 130, 131, 133, 134, 137, 138, 143, 150, 154, 156, 100-103, 110, 111, 113, 116, 121</td>
<td>3</td>
<td>med</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>10, 34, 38, 41, 63, 75</td>
<td>4</td>
<td>high</td>
<td>med</td>
<td></td>
</tr>
<tr>
<td>11, 19, 44, 49, 60, 80, 81, 88</td>
<td>3</td>
<td>med</td>
<td>med</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>low</td>
<td>med</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>low</td>
<td>med</td>
<td></td>
</tr>
<tr>
<td>14, 16</td>
<td>3</td>
<td>low</td>
<td>med</td>
<td></td>
</tr>
<tr>
<td>15, 51-53</td>
<td>2</td>
<td>high</td>
<td>med</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>high</td>
<td>med</td>
<td></td>
</tr>
<tr>
<td>18, 42</td>
<td>2</td>
<td>low</td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>20, 55, 82</td>
<td>2</td>
<td>med</td>
<td>med</td>
<td></td>
</tr>
<tr>
<td>21, 26, 76, 155, 120, 157, 159</td>
<td>4</td>
<td>high</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>25, 27, 152</td>
<td>7</td>
<td>high</td>
<td>high</td>
<td></td>
</tr>
<tr>
<td>28, 119</td>
<td>5</td>
<td>med</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>125, 145, 147, 160, 105, 107, 114</td>
<td>5</td>
<td>high</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>7</td>
<td>med</td>
<td>high</td>
<td></td>
</tr>
</tbody>
</table>
**TABLE A.12.** SLR case outcomes (2)

<table>
<thead>
<tr>
<th>Cases</th>
<th>Requirements expression, techniques &amp; process</th>
<th>Requirements achievement</th>
<th>Stakeholder satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>35, 61, 66</td>
<td>4</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>37, 98, 99, 151</td>
<td>5</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>med</td>
<td>low</td>
</tr>
<tr>
<td>45</td>
<td>1</td>
<td>med</td>
<td>low</td>
</tr>
<tr>
<td>46</td>
<td>1</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>58</td>
<td>2</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>64, 78, 96</td>
<td>5</td>
<td>high</td>
<td>med</td>
</tr>
<tr>
<td>65</td>
<td>5</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>69</td>
<td>7</td>
<td>high</td>
<td>med</td>
</tr>
<tr>
<td>87</td>
<td>0</td>
<td>low</td>
<td>unknown</td>
</tr>
<tr>
<td>123</td>
<td>8</td>
<td>high</td>
<td>medium</td>
</tr>
<tr>
<td>128</td>
<td>6</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td>130, 140, 162</td>
<td>3</td>
<td>low</td>
<td>unknown</td>
</tr>
<tr>
<td>148</td>
<td>6</td>
<td>high</td>
<td>medium</td>
</tr>
<tr>
<td>142, 149, 124, 104</td>
<td>6</td>
<td>high</td>
<td>unknown</td>
</tr>
</tbody>
</table>
Appendix B

Interview Guide

Note that this is a guide only. Questions listed in each grey box are potential prompts, interviews will be semi-structured and not all of the questions may be asked in each interview.

• Introduction
  – Recap of motivation, purpose and method of study, point out right to withdrawal, ask to sign consent form.

• Background and relevant expertise
  – What is your current position in the organisation? What responsibilities for model transformations does this entail?
  – What is your previous experience in model transformation development?

• Relevant requirements engineering techniques
  – What requirements engineering techniques are important for the kind of projects you have been involved with?

• Obtaining model transformation requirements
  – Can you talk me through an example project where there were conflicts between the requirements? What was the result of the trade-off? Did you use any particular technique? If yes, why?
• Role of requirements engineering in model transformation

  – How would you distinguish general “software development projects” and “transformation development projects”?

  – How do you categorise functional and non-functional requirements? In case you apply requirements engineering process in your projects, what techniques and methods would you use during the following stages and why?
    * Domain analysis and requirements elicitation
    * Evaluation and negotiation
    * Specification and documentation
    * Validation and verification

  – What kind of requirements engineering Process Model and Methodology do you use for your projects?

  – Do functional and non-functional requirements change frequently as a project evolves? If so, how do you account for this in your development process? Could you give an example of such a change occurring in your own experience and how you dealt with it?

• Triangulation

  – Is what we have discussed so far typical of software development projects in your view? Has this changed over time? How? Can you give concrete examples?

• Wrap up

  – Is there anything else you feel we should have talked about?

  – Do you have any other feedback on the interview?

• Thank participant for their time, explain what will happen next.
Thank you for your time today. I will now transcribe, anonymize, and analyse the interview recording. I will wait for at least two weeks before doing this. If, for any reason, you wish to withdraw from the study, you can do so within the next seven (7) days from now. Simply contact me to let me know.
Appendix C

Description of RE Techniques

In the following section, we have described some RE techniques regarding their attributes according to the SLR and interview-based study.

**Interview and Questionnaire:** They are very economical and easy to apply and are useful due to simplicity and a generic way to capture the requirements from multiple stakeholders [96]. If the size of the MT project is very large, it can be a time consuming technique to be implemented. It is an effective approach for an MT project with very high/high complexity as it allows the developers to elicit requirements regarding different aspects and give them a better understanding about the requirements as well as important quality criteria. If the volatility of the project is high then it is not recommended to apply this technique as it is not practical to have a high number of interview sessions with stakeholders especially if their availability is low. It is a suitable technique if the customer-developer relation is high.

**Document Mining:** This is an effective technique if the developer’s understanding regarding the project is low. Depending on the size of the project it can be time consuming. It is a suitable technique for low budget projects. It is effective if the level of relationship between developers and stakeholders is low. It is not an efficient
technique if the transformation requirements volatility is high as it is unlikely to have published documentations. If the transformation complexity is high, applying this technique can give a better understanding to developers by giving a better idea regarding the requirements.

**Brainstorming:** This is a useful method for complicated transformations as it allows discussions amongst the stakeholders and developers. This technique is quite expensive and difficult to conduct, especially if there are multiple stakeholders. It has an inverse proportion regarding the allocated time of project development, if time is limited the number of brainstorming sessions will also be limited. The suitability of this technique has a proportionally direct relation with the level of customer-developer relationship. It is an effective technique as it enables the developer to have a better understanding regarding the quality criteria and different types of requirements. It is an effective technique for small size MT projects as it allows the developers to discuss the details and quality criteria of the project. It is not well recommended if the volatility of the MT project is high as it is not practical to have several brainstorming sessions as it is often used in the preliminary stage of the project [158].

**Prototyping:** This is a suitable technique for Greenfield MT projects in which it is difficult to elicit the requirements and the stakeholder’s expectations regarding the project. It is recommended as one of the best techniques for representing the actual transformation in a functional and/or graphical way as it is able to capture all the details regarding the user interface. It can decrease the development time and effort if it is being used in an evolutionary manner (evolutionary prototyping) however it is more expensive than other RE techniques and more time consuming [48]. It is an effective method if there is a good relation between the customer and the developer as it allows the customer to play a more active role during the development process. It is not a well suited technique if the MT
project volatility is high as it requires much time and budget. If the size of MT is very large and the complexity is high, then building a prototype will be a challenging task as it is very expensive and time consuming to build a prototype that covers several requirements. It is a well suited technique for smaller MT projects.

**Scenario:** This is a very useful technique for small transformation projects as it makes it feasible to consider different circumstances and scenarios, on the other hand it is not very practical and affordable for very large and large projects. Defining different scenarios might also be time consuming. “Scenarios and user centred view provides flexibility to find the requirements while analysing different sessions and their user response after interaction with the scenarios” [65]. One of the advantages of this technique is that it represents the specifications in detail when the complexity is high [136]. If the project size is very large/large considering all the possible scenarios and outcome could be costly and time consuming. It is an effective technique if the level of volatility of the project is high. Moreover, it is a suitable technique if the customer is not available as it allows the developers to consider different circumstances.

**Ethno Methodology:** This is a useful technique for those kinds of MT projects that have low time constraints and have no budget constraints [17]. It is a very useful technique to implement if the developer’s understanding regarding the requirements is low. It is effective if the level of relationship between customer and developers is high. These involve systematic observation of actual practice in a workplace.

**UML:** This technique was one of the most popular techniques according to the data from the SLR and the interviews. This technique was applied to almost every project regardless of its attributes.

**Functional Decomposition:** This is an effective technique to be applied to various functional requirements relationships. It gives
a better understanding regarding the overall functionality of the project and the dependency amongst different components. Using this technique for large and complex MT projects allows the developer(s) to have a better understanding of the requirements as it breaks down the requirements into sub-requirements.
Appendix D

Surveyed Papers


[18] None. Duplicate paper.


[34] Tassilo Horn. Solving the TTC 2013 Flowgraphs Case with FunnyQT. In TTC, pages 57–68, 2013.


[61] Tassilo Horn. Solving the TTC Train Benchmark Case with FunQyQT. In *Proceedings of the 8th Transformation Tool Contest*, 2015.


[63] Tassilo Horn. Solving the TTC Java Refactoring Case with FunQyQT. In *Proceedings of the 8th Transformation Tool Contest*, 2015.


[66] Devinder Thapa, Suraj Dangol, and Gi-Nam Wang. Transformation from Petri nets model to programmable logic controller using one-to-one mapping technique. In International Conference
on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC’06), volume 2, pages 228–233. IEEE, 2005.


[84] Andreas Koch, Ruben Jubeh, and Albert Zündorf. UML 1. 4 to 2.1 activity diagram model migration with Fujaba TTC 2010 case study.


[105] Soon-Kyeong Kim, Toby Myers, Marc-Florian Wendland, and Peter A Lindsay. Execution of natural language requirements using


[127] Zekai Demirezen, Marjan Mernik, Jeff Gray, and Barrett Bryant. Verification of DSMLs using graph transformation: a case study with Alloy. In *Proceedings of the 6th International Workshop on


[144] None. Duplicate paper.


285


