Requirements Engineering for Model Transformation Development

Yassipour Tehrani, Sobhan

Awarding institution:
King's College London

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:

- State of Practice: Requirements Engineering Process in Model Transformation Development. In *International Journal on Software and Systems Modeling 2017* (Submitted)
  Yassipour Tehrani, S., Lano, K., Zschaler, S.
  This publication is based on Chapter 3 and Chapter 4 of the thesis.

- Requirements Engineering in Model-Transformation Development: An Interview-Based Study. In *International Conference on Theory and Practice of Model Transformations. Springer International Publishing, 2016.*
  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.

- Requirements Engineering in Model Transformation Development: A Technique Suitability Framework for Model Transformation Applications. In *The International Journal on Advances in Software published by IARIA.*
  Yassipour Tehrani, S. & Lano, K.
  This publication is based on Chapter 5 and Chapter 6 of the thesis.

- Temporal Logic Specification and Analysis for Model Transformations. In *Verification Of ModeL Transformation (VOLT 2015).*
  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.

- Agile model-driven engineering of financial applications. In *International Conference on New Trends in Information and Communication Technologies.*
  Lano, K., Haughton, H., ***Yassipour Tehrani, S.*** & Alfraihi, H. A. A
  This publication is based on Chapter 6 of the thesis.

Partially related publications:

- The use of Model Transformation Design Patterns in Practice . In *Journal of Systems and Software Publishing, 2017.*
  Lano, K., ***Yassipour Tehrani, S.*** & Kolahdouz-Rahimi, S.

- A Survey of Model Transformation Design Pattern Usage. In *International Conference on Theory and Practice of Model Transformations. Springer International Publishing, 2017.*
  Lano, K., Kolahdouz-Rahimi S. & ***Yassipour Tehrani, S.***

- Verified Bidirectional Transformations by Construction . In *Verification Of ModeL Transformation at MODELS 2016*
  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.

- Model Transformation Semantic Analysis by Transformation. In *Verification Of ModeL Transformation (VOLT 2015).*
  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

List of Figures

List of Figures

List of Tables

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 [46], 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 *trans*ML [1] 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

[1] *trans*ML 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.

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.

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

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

the functional and non-functional requirements before building the software and to make sure that the system or software will achieve the set objectives. Selič [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.

TABLE 2.1. Project failure analysis (Standish Group (1995))

	Incomplete requirements	13.1%
	Lack of user involvement	12.4%
	Lack of resources	10.6%
	Unrealistic expectations	9.9%
	Lack of executive support	9.3%
	Changing requirements/specifications	8.7%
	Lack of planning	8.1%
	Did not need it any longer	7.5%

TABLE 2.2. Project success analysis (Standish Group (1995))

	User involvement	15.9%
	Management support	13.9%
	Clear statement of requirements	13%
	Proper planning	9.6%
	Realistic expectations	8.2%
	Smaller milestones	7.7%
	Competent staff	7.2%
	Ownership	5.3%

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.

TABLE 2.3. Project success analysis(Standish Group (2015))

	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

TABLE 2.4. Definition of non-functional requirements

Source	Definition
Anton [5]	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.
Kotonya *et al.* [80]	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.
Paech *et al.* [25]	"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."
Landes [82]	"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."

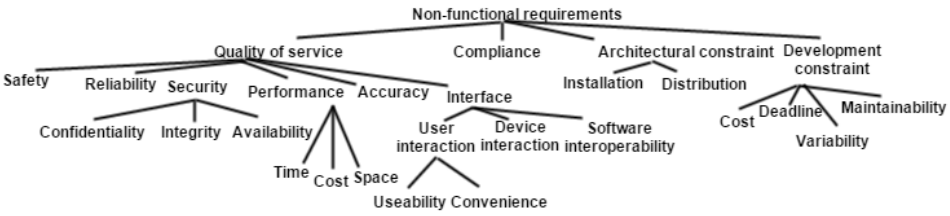


Figure 2.1. A taxonomy of non-functional requirements [143]

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,

portability and etc.[143]).

By having requirements taxonomies, *confined* and *cross-cutting* concerns [143] 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 [113].

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.

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].

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

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

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.*

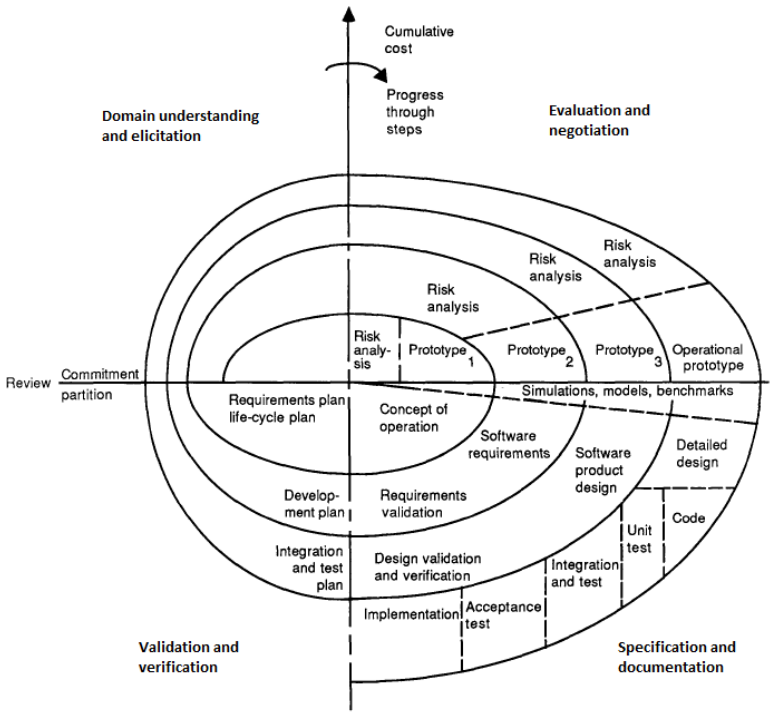


Figure 2.2. Spiral model of the software process [18]

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

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

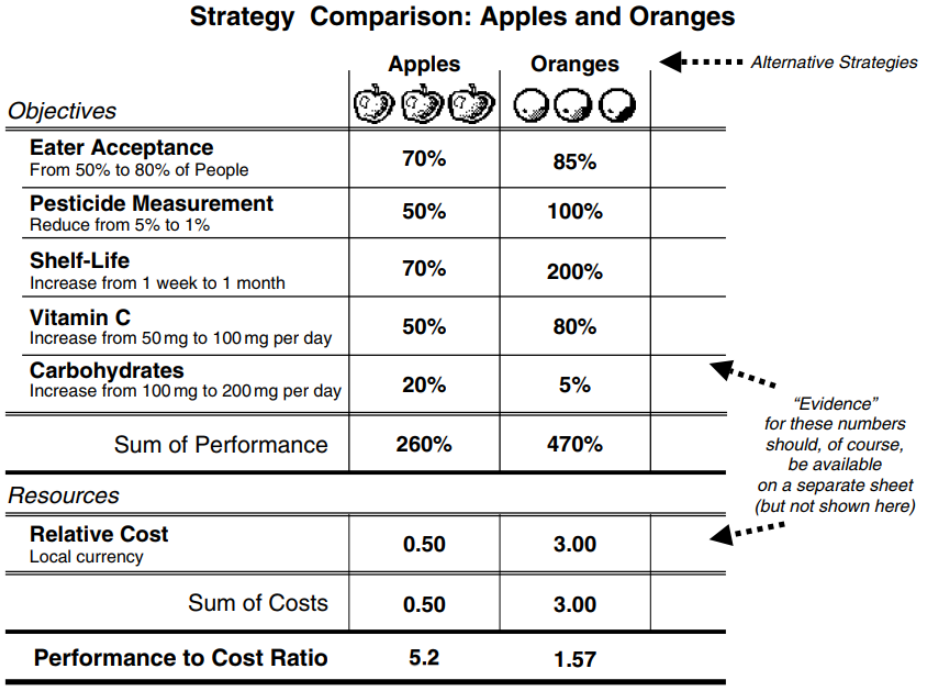


Figure 2.3. Comparison table using impact estimation [44]

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

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.

TABLE 2.5. ISO/IEC 9126-1 quality characteristics [40]

Characteristics	**Sub-characteristics**
Functionality	Suitability
	Accuracy
	Interoperability
	Security
	Functionality compliance
Reliability	Maturity
	Fault tolerance
	Recoverability
	Reliability compliance
Usability	Understandability
	Learnability
	Operability
	Attractiveness
	Usability compliance
Efficiency	Resource utilization
	Time behaviour
Maintainability	Analysability
	Changeability
	Stability
	Testability
	Maintainability compliance
Portability	Adaptability
	Installability
	Co-existence
	Replaceability
	Portability compliance

- **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

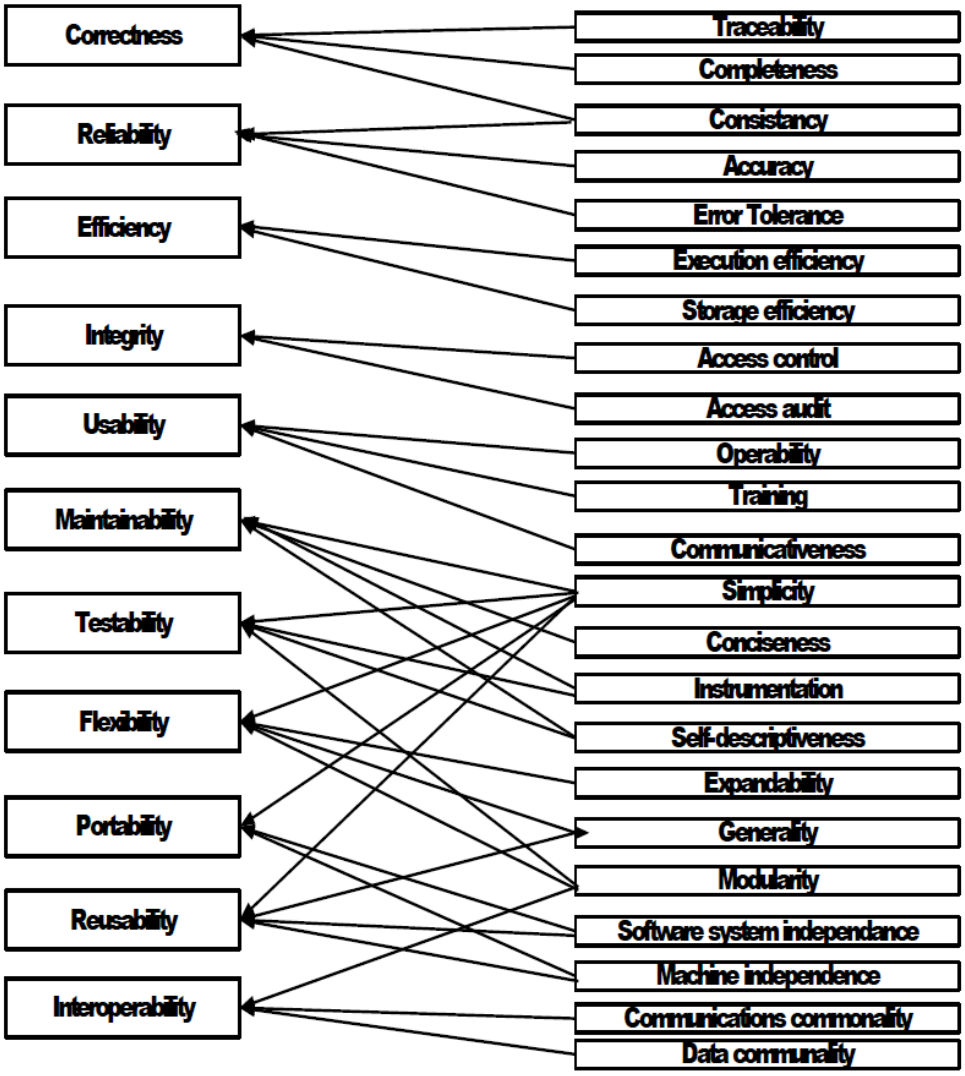


Figure 2.4. McCall Quality Model [106]

Model.

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

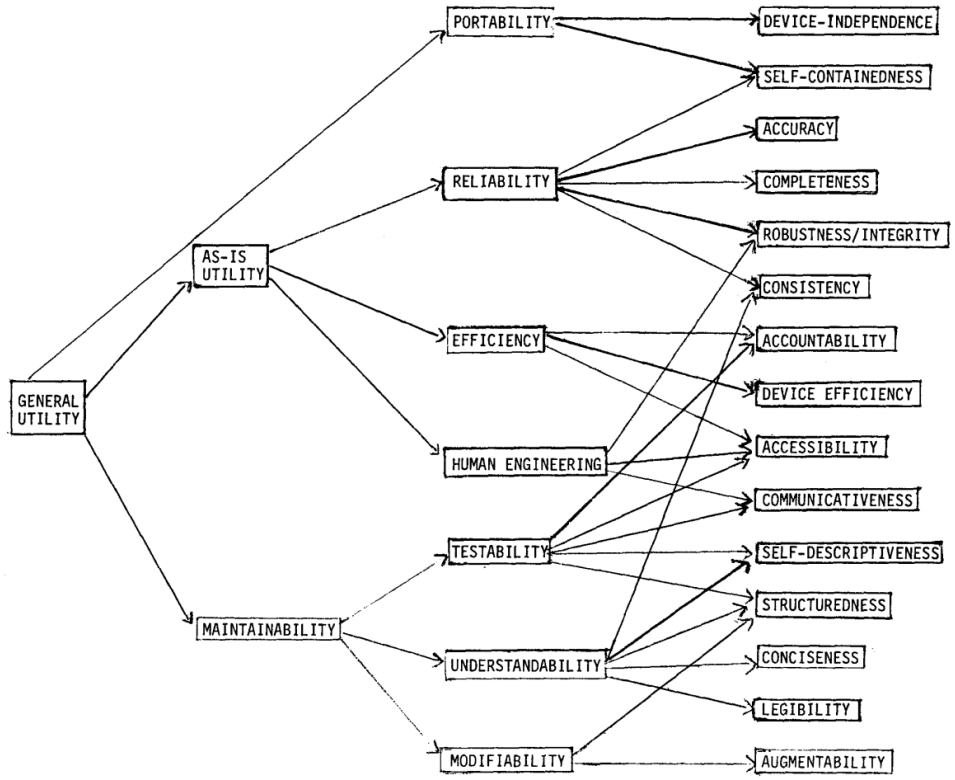


Figure 2.5. Boehm Quality Model [19]

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.

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].

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.

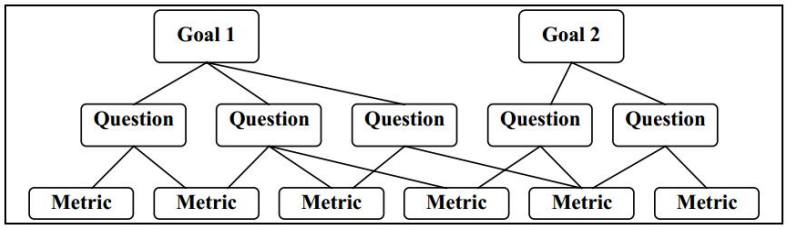


Figure 2.6. GQM model [10]

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

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 [143], 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

[129]. 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).

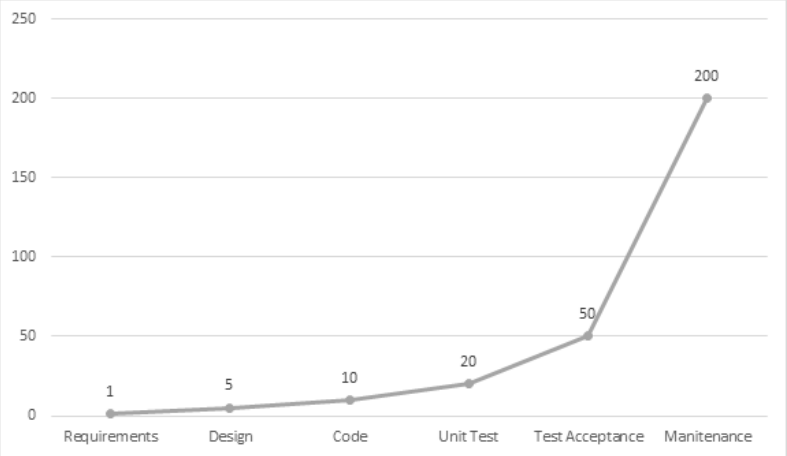


Figure 2.7. Cost of late correction (Boehm)

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

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

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* [143] 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 [143] 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

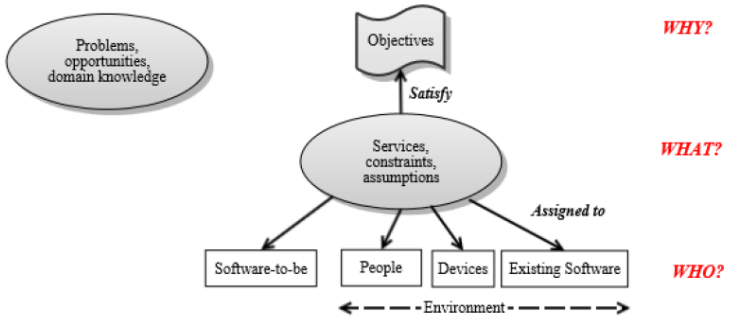


Figure 2.8. Three dimensions of requirements engineering [143]

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 [143]

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 [143]. 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 [143]. 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 [143]. In Figure 2.9, the classification of RE statements is illustrated in a diagram.

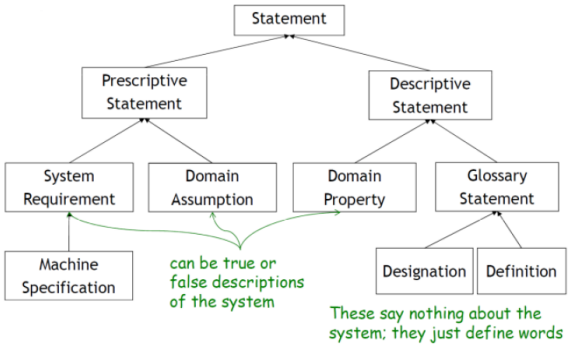


Figure 2.9. Classification of statements in RE [97]

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:

- 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.

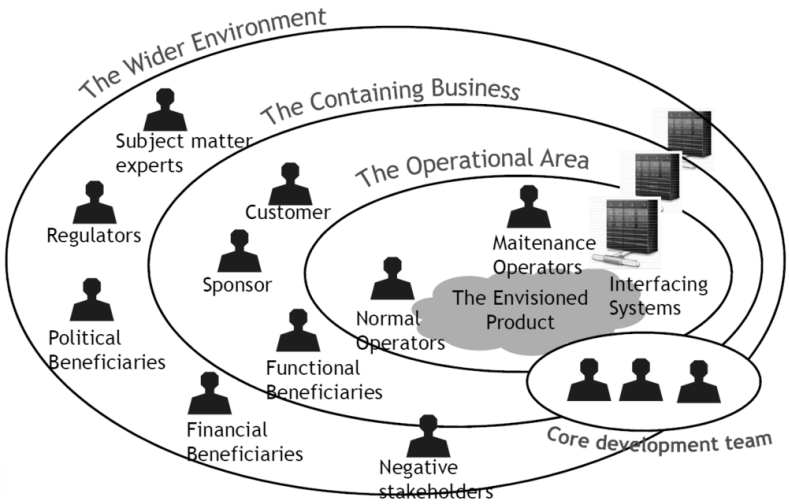


Figure 2.10. Stakeholder onion model [98]

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 [98].

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 [143]. 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 [143]:

- 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]:

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-

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.

2.4.2 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

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-

opment 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

  - 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

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:

Liveness: Every request is followed by a response
$\Box(request \rightarrow response)$

Safety: p never happens
$\Box\neg p$

Fairness: if p happens infinitely often, then φ will be true
$\Box \Diamond \mathrm{p} \rightarrow \varphi$

Invariance: At some point, p will hold forever
$\Diamond \Box \mathrm{p}$

Partial correctness: if p is true, then q will be true when the task is completed
$\mathrm{p} \rightarrow \Box\ (\mathrm{done} \rightarrow \mathrm{q})$

Mutual exclusion: two processes cannot enter their critical sections simultaneously
$\Box\neg\ (in_C S1 \wedge in_C S2)$

p *oscillates* every step
$\Box((p \wedge X\neg p) \vee (\neg p \wedge Xp)$

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 τ: $\mathtt{S} \rightarrow \mathtt{T}$ can also be verified to check that they are consistent:

- The functional requirements must be mutually consistent

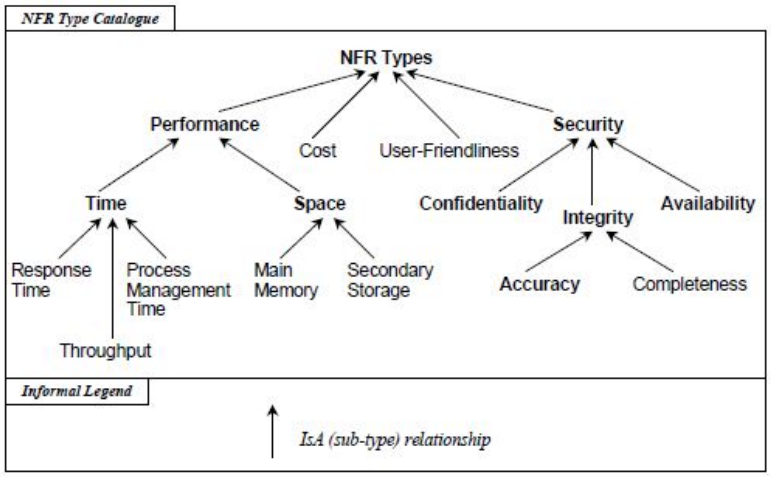


Figure 2.11. NFR types [26]

- The assumptions and invariant of τ, and the language constraints of `S` must be jointly consistent
- The invariant and post conditions of τ, 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].

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

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.

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.

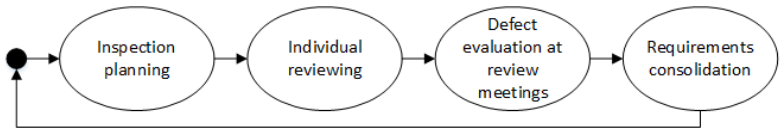


Figure 2.12. Requirements inspection and review process

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-

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

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.

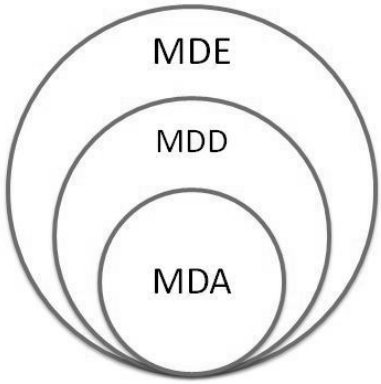


Figure 2.13. Model-driven frameworks, adapted from [22]

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 [22].

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.

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.

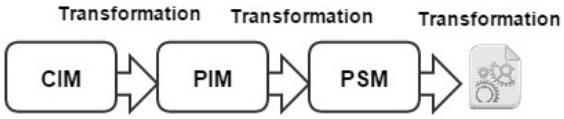


Figure 2.14. Transformation between different representations of a model

2.6 Model Transformation

According to the Oxford Dictionary [34], transformation is defined as "a marked change in form, nature, or appearance". In general, the aim of model transformation is to create a new model or to improve an existing model according to established specifications expected to solve a specific problem. Models are the primary artefacts. The model used as the input of a transformation is referred to as the source model and the model used as the output of the transformation is referred to as the target model [16]. Model transformation can be used for different tasks such as: modifying, creating, adapting, merging, weaving or filtering models. The

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 [75].

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.

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.

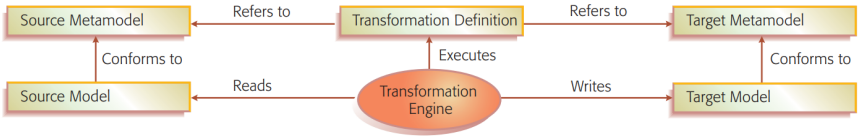


Figure 2.15. The general architecture of model transformation [29]

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

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 2.6. General properties of transformations

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

- **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

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

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 `S1`, `S2` into `T1`) or whether it performs updating (merging `S1`, `T1` 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-

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

preservation; this means that the meaning of both models are similar while they are represented by different abstract syntax [148].

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 [16], 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 [16].

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" [16]. Moreover, the order of rule application is determined by rule scheduling: Rule selection controls when a rule is applied. Rule iteration uses recursion, loop-

ing or fixed point iteration. *Phasing* determines that in a certain phase only certain rules can be executed [16].

- **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).

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).

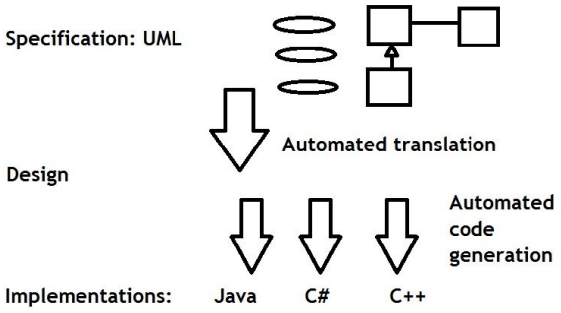


Figure 2.16. General process of UML-RSDS

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 `Sa` is transformed into a target model `Ta` according to the transformation definition `mma2mmb.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)" [64].

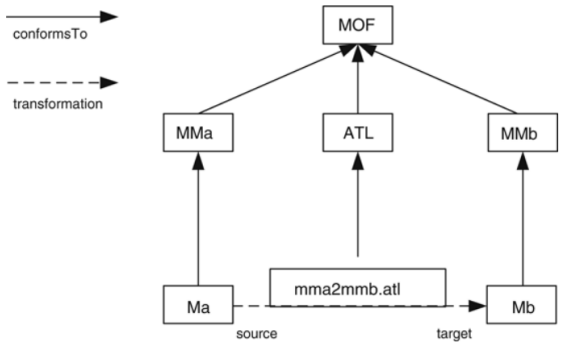


Figure 2.17. Overview of the ATL transformational approach [64]

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: *User-friendly Relations* and *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 *Operational* part which is designed to write unidirectional transformations. *Black box* is another component of QVT

language which acts as a mechanism to invoke facilities of a transformation 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.

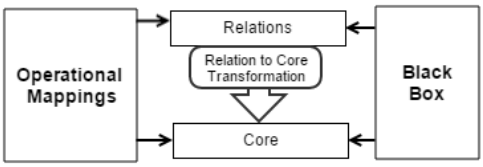


Figure 2.18. QVT architecture [115]

TABLE 2.7. Comparison of transformation languages

Language	Mapping	Update-in-place	Bidirectionality	Change-propagation
UML-RSDS	✓	✓	✓	×
ATL	✓	✓ *(partial)*	×	×
QVT	✓	✓ *(partial)*	✓	✓
ETL	✓	✓	×	×

Comparison of Transformation Languages

In this section, we are going to compare the functionality of model transformation languages. ATL has a restricted form of update-in-place processing, 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 transformations from mapping transformations. QVT provides the capability to

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.

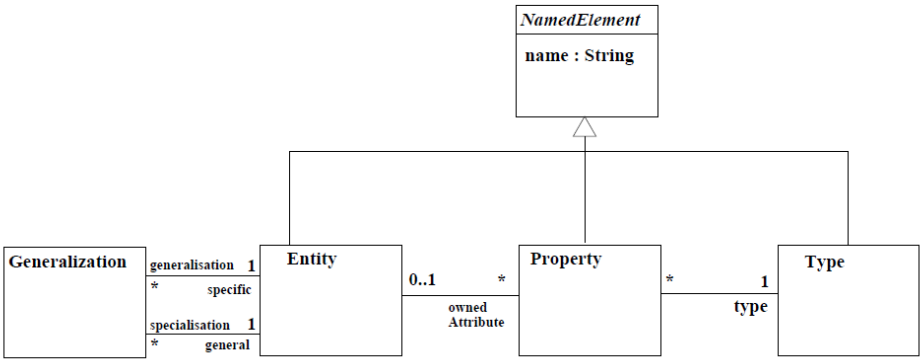


Figure 2.19. Class diagram metamodel [75]

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.specific* 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).

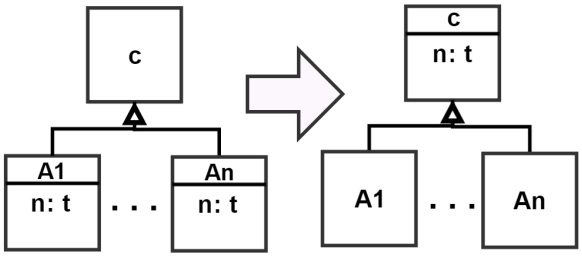


Figure 2.20. Rule 1 [75]

- A new subclass created for duplicated attributes: If there is a class called `c` that has two or more direct subclasses `g`, `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 that 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. 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 `c1` and remove the copies from each of its direct subclasses" [75] (Figure 2.21).

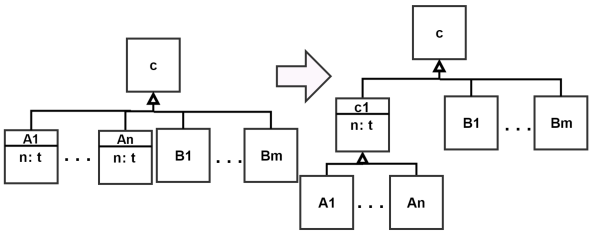


Figure 2.21. Rule 2 [75]

- 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

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).

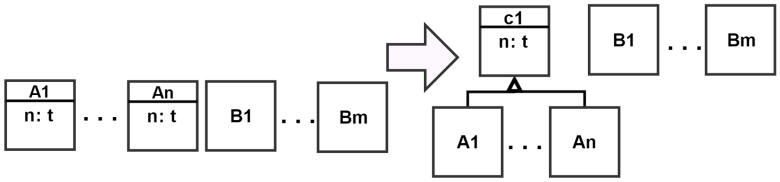


Figure 2.22. Rule 3 [75]

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

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 [123], 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)

- 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]

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

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].

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 ⇒ 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].

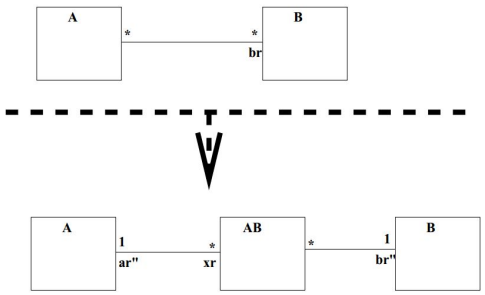


Figure 2.23. Removing a many-to-many association

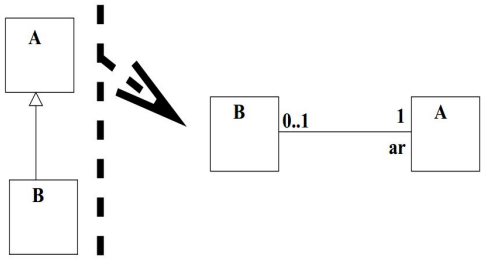


Figure 2.24. Replacing an inheritance by an association

- 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].

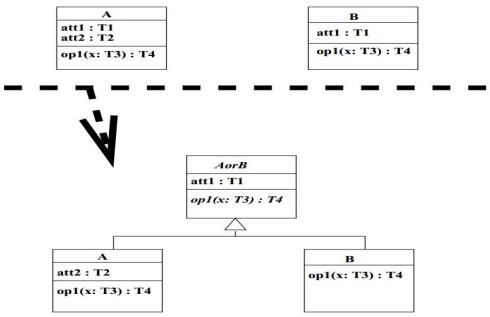


Figure 2.25. Introducing a superclass

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 a 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

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

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 Section 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, followed 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 participants' 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 involved (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 participants. 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-

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 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 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

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 *trans*ML 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

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.

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,

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 3.1. Types of MT project

Case	Brown-field	Green-field	Customer-driven	Market-driven	In-house	Out-sourced	Single-product	Product-line
Project 1		✓	✓		✓		✓	
Project 2		✓	✓		✓		✓	
Project 3	✓		✓		✓		✓	
Project 4		✓	✓		✓		✓	
Project 5		✓	✓		✓		✓	
Project 6		✓	✓		✓		✓	
Project 7	✓		✓		✓		✓	
Project 8		✓	✓		✓		✓	
Project 9	✓			✓	✓		✓	
Project 10		✓	✓		✓		✓	

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).

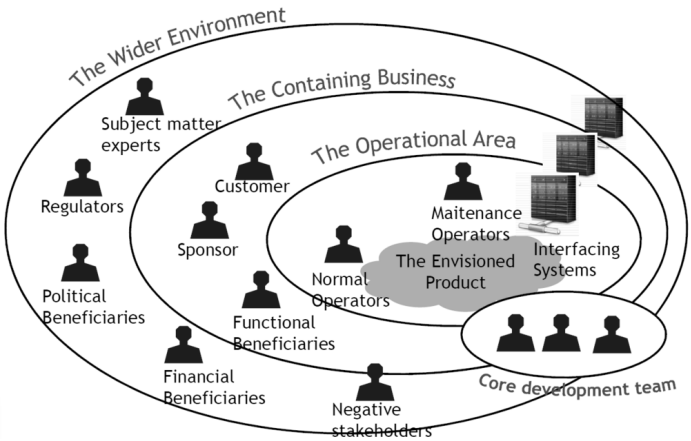


Figure 3.1. Onion model of stakeholder general relationships [3]

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):

- 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 3.2. Stakeholders of model transformation projects

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

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

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.

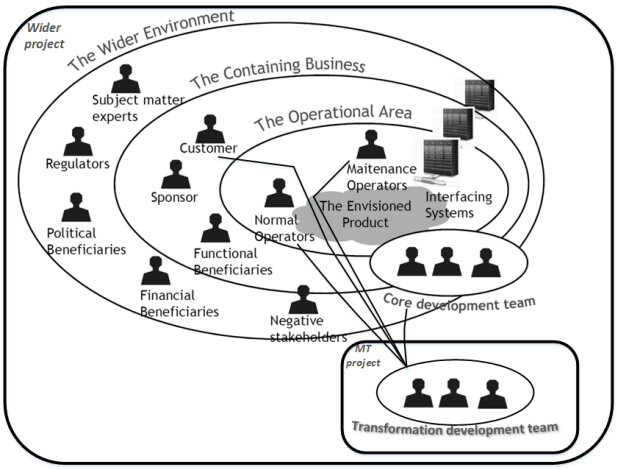


Figure 3.2. Adapted onion model of MT stakeholder relationships

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.

- **Reusebility 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].

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

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 techniques and approaches that were used is listed.

TABLE 3.3. Requirements engineering techniques in MT projects

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

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

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 3.4. RE revision activity in MT projects

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

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.

> "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,

> "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.

TABLE 3.5. RE techniques in MT projects

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

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-

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 (under-determined) 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

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

TABLE 3.6. Outcomes of MT projects

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

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-

tion design (JAD) workshops are recommended by [100] 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.

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

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 [99]:

- 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.

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.

4.3.5 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.

- 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.

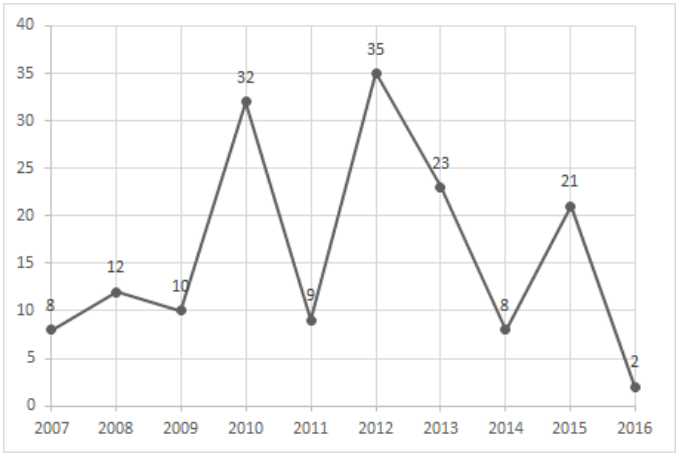


Figure 4.1. Surveyed cases per year

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

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 4.1. Number of reviews

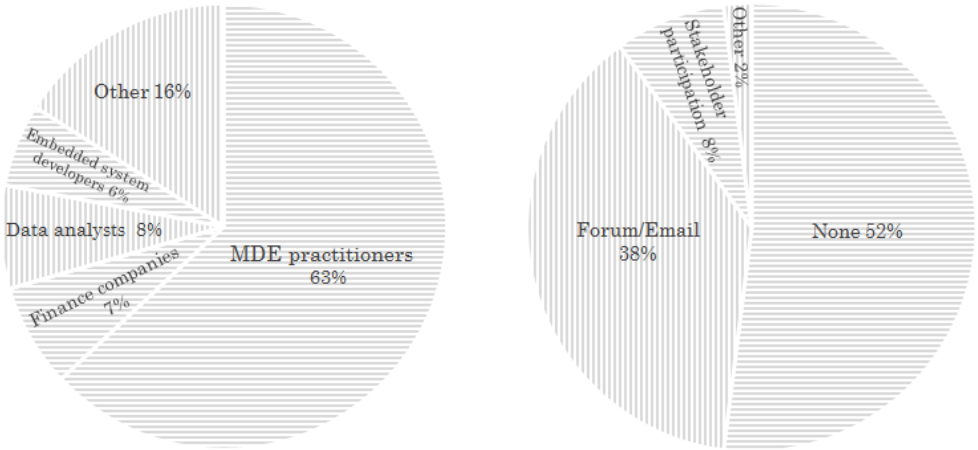
	Automatic search				**Manual search**		
Source	IE	SL	ACM	RG	TTC/ MODELS/STAF/SoSyM	[11]	Total
Initial result	5	8	5	16	73	82	189
Final result	2	6	3	13	56	80	160

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,



(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

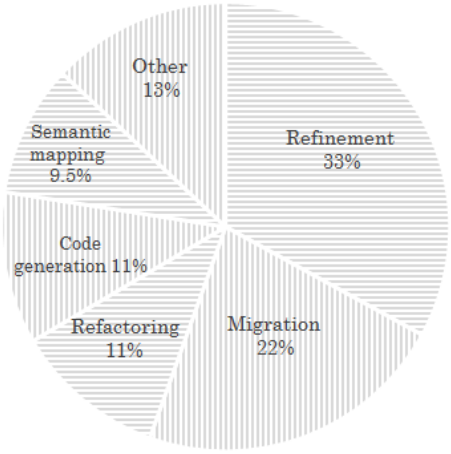


Figure 4.3. Transformation types

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.

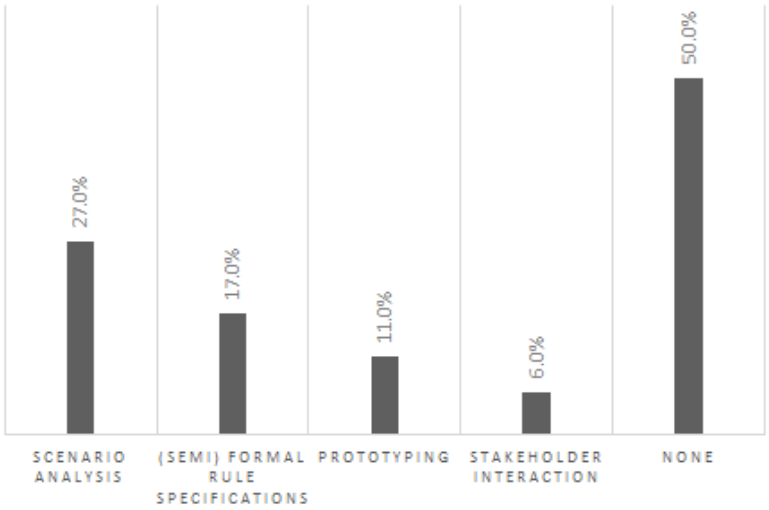


Figure 4.4. RE technique used in MT cases

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.

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)

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

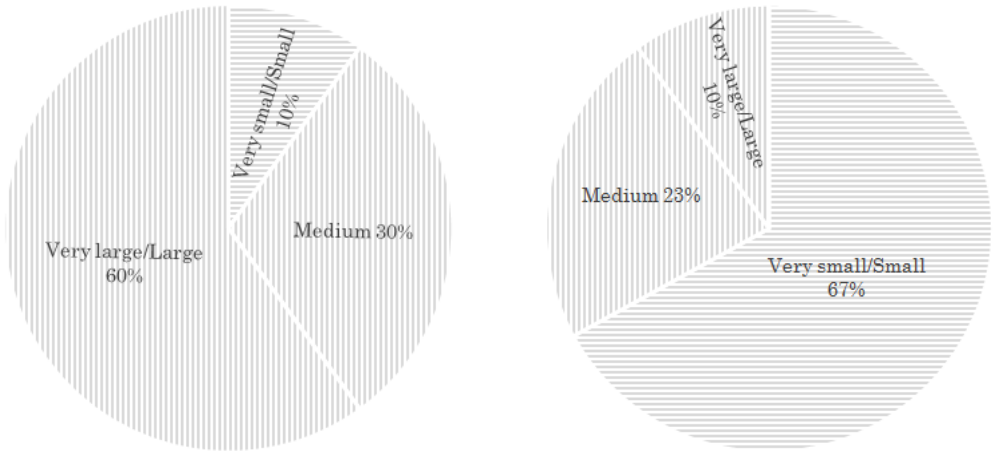
The results of the SLR and the interview study can be compared as shown in Table 4.2. We aimed to focus on industrial case studies for our interview study, and this resulted in a much higher percentage of industrial cases compared to the SLR. There is a close correlation between industrial cases and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

this category (Figure 4.6b).

TABLE 4.2. Comparison of results

Aspect	Interviews	SLR
Industry/Academic	80%/20%	14%/86%
Scale (Large/Medium/Small)	60%/30%/10%	10%/23%/67%
Used RE techniques	90%	50%
Used RE process	30%	5%

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.



(a) interview cases **(b)** SLR cases

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%).

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).

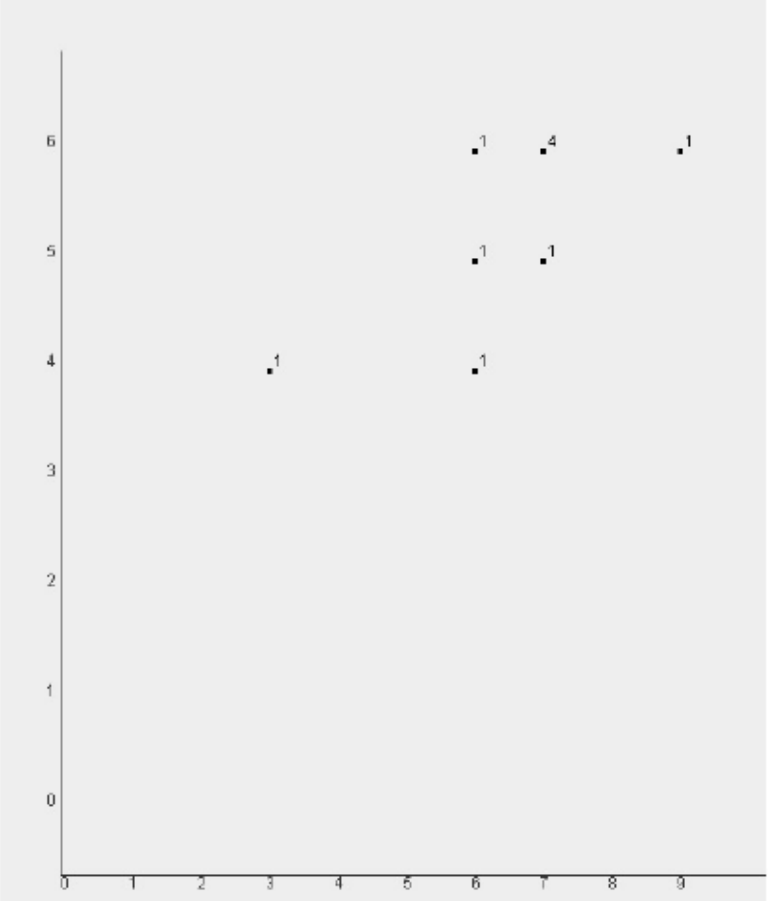


Figure 4.7. Interview case RE rigour (x-axis) versus outcomes (y-axis)

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,

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

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

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.

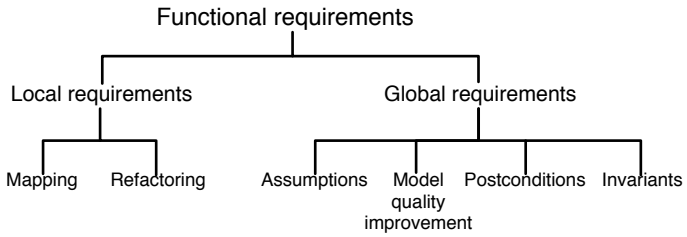


Figure 5.1. A taxonomy of functional requirements

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].

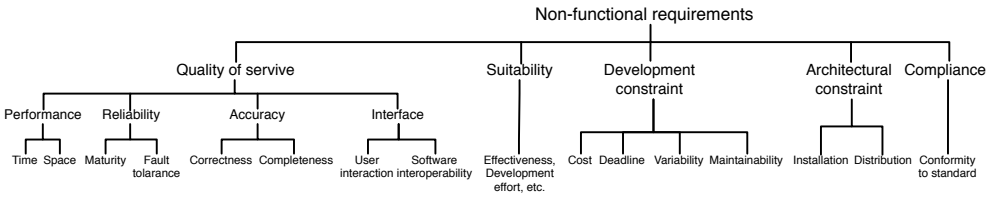


Figure 5.2. A taxonomy of non-functional requirements for MT

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

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 τ is syntactically correct if wherever τ terminates, when applied to a valid model `m` of source language `S`, it produces a valid target model `n` in terms of conformation to the `T`'s language constraints.
- *Termination*: a transformation τ will always terminate if applied to a valid `S` model.
- *Confluence*: all result models produced by transformation τ from a single source model are isomorphic.
- *Model-level semantic preservation*: a transformation τ 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 τ [90].

An additional accuracy property that could be considered is the existence of invertibility in a transformation $\sigma : \mathtt{T} \rightarrow \mathtt{S}$, which inverts the effect of τ. Given a model `n` derived from `m` by τ, σ applied to `n` produces a model $\mathtt{m}'$ 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

TABLE 5.1. Transformation requirements catalogue

	Refactoring	**Refinement**	**Migration**
Local Functional	Rewrites/ Refactorings	Mappings	Mappings
Local Non-functional	Completeness (all cases considered)	Completeness (all source entities, features considered)	Completeness (all source entities, features considered)
Global Functional	Improvement in quality measure(s), Invariance of language constraints, Assumptions, Postconditions	Invariance, Assumptions, Postconditions	Invariance, Assumptions, Postconditions
Global Non-functional	Termination, Efficiency, Modularity, Model-level semantic preservation, Confluence, Fault tolerance, Security	Termination, Efficiency, Modularity, Traceability, Confluence, Fault tolerance, Security	Termination, Efficiency, Modularity, Traceability, Confluence, Fault tolerance

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 5.2. Standard quality framework (ISO 9126)

Characteristics	**Subcharacteristics**
Functionality	Suitability, Accuracy, Interoperability, Security, Functionality
Reliability	Maturity, Fault tolerance, Recoverability, Reliability compliance
Usability	Understandability, Learnability, Operability, Attractiveness, Usability compliance
Efficiency	Time behaviour, Resource utilisation, Efficiency compliance
Maintainability	Analysability, Changeability, Stability, Testability, Maintainability compliance
Portability	Adaptability, Installability, Co-existence, Replaceability, Portability compliance

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

Aa 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-

tions, 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

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.

TABLE 5.3. Requirements priority for different types of transformation

Category	**Primary requirement**	**Secondary requirement**
Refactoring	Model quality improvement Model-level semantic preservation Syntactic correctness Termination	 Invariance Confluence
Migration	Syntactic correctness Model-level semantic preservation Termination	Invertibility Confluence Traceability
Refinement	Syntactic correctness Model-level semantic preservation Confluence Termination	Traceability

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:

- *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 φ which should be removed will eventually be removed, and will not be reintroduced:

$$\textit{model elements x1, ..., xn satisfy property } \varphi \Rightarrow$$
$$\Diamond \Box \textit{ (x1, ..., xn do not satisfy } \varphi)$$

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

$$Asm \Rightarrow \Box(Inv)$$

General postconditions can be expressed using the '*Achieve*' pattern:

$$Asm \Rightarrow \Diamond \Box\ (Post_1 \wedge ... \wedge Post_m)$$

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 τ: $\mathtt{S} \rightarrow \mathtt{T}$ can also be verified to check that they are consistent as follows:

- The functional requirements must be mutually consistent
- The assumptions and invariant of τ, and the language constraints of S must be jointly consistent
- The invariant and postconditions of τ, and the language constraints of T must be jointly consistent

- 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:Integer* could be identified which is always non-negative and which is strictly decreased by each application of a mapping rule [90]. Formalised requirements in temporal logic could then be checked for particular implementations using model-checking techniques, as in [152].

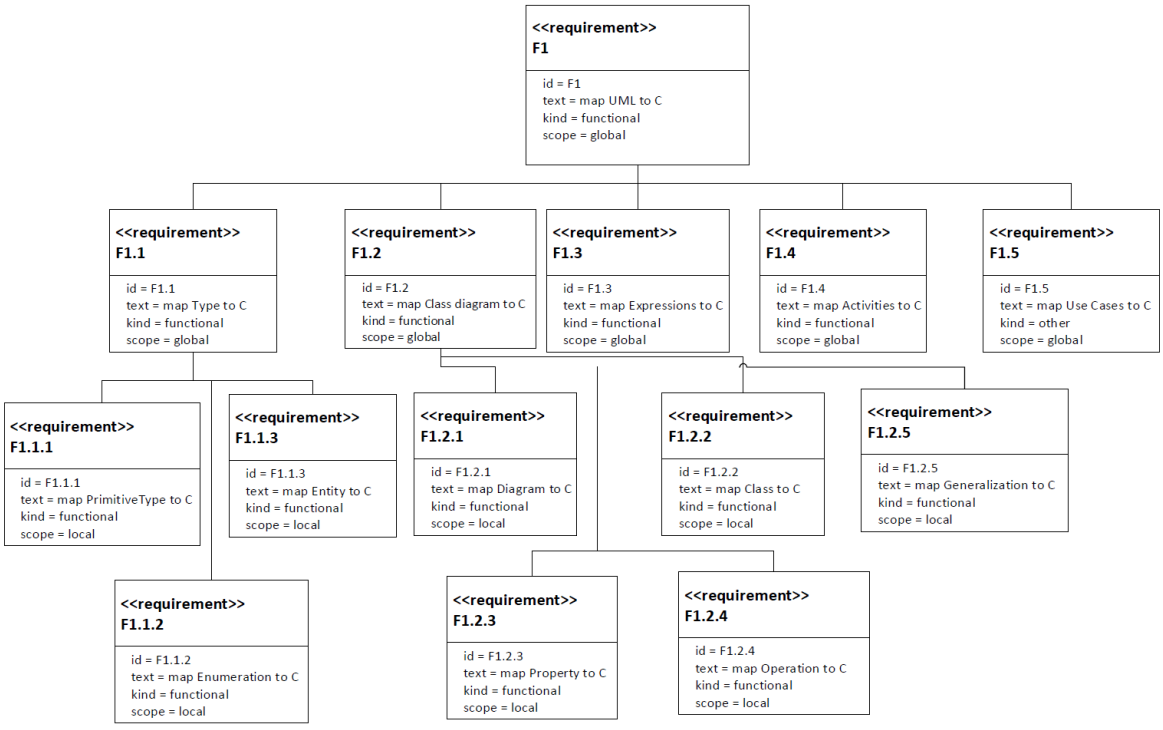


Figure 5.3. Functional requirements decomposition

5.2.5 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), semi-formal (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 `T1` instance `t1` [and to a `Tj` instance `tj`]* such that `P`(`s`, `t1`, ..., `tn`)
>
> Each `S1` is considered to be a SCond if it has `SP`

are formalised as:

```
S1::
SP => T1 -> exists( t1 |...
    Tn->exists( tn | P(self,t1,...,tn) ) ... )
```

SCond is a linguistic term to represent `S1` $\rightarrow$ *select* (`SP`). The constraint expresses that each such `S1` instance maps to `T1`, `T2`, ..., `Tn` 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 post-condition 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

checks on use case definedness, determinacy and data-dependency relations [87]. Formal proof of invariance, syntactic correctness and model-level semantic preservation is also supported via the B formal method [88].

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

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

relevance of a technique is the product of the three measures for *Q1, Q2* and *Q3*.

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

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.

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

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

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 $\mathtt{V}(\mathtt{X},\mathtt{Y})$ of $\mathtt{X}$ (technique) for $\mathtt{Y}$ (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 $\mathtt{V}(a_x, t)$

Attribute	**Interview**	**Question-naire**	**Document Mining**	**Brainstor-ming**	**Proto-typing**	**Scenario**	**Ethno Methodology**
Eliciting MT requirements	1	0.8	1	0.8	1	1	0.8
Facilitating communication	1	1	0	0.8	0.8	1	0.6
Understanding social issues	0.8	1	0.8	0.4	0.2	0.6	0.8
Getting domain knowledge	0.6	0.6	1	1	0.4	0.4	1
Getting implicit knowledge	0.2	0.2	0.2	0.2	0.2	0.2	1
Identifying MT stakeholders	1	0.8	0.2	1	0	0.4	0.6
Identifying non-functional requirements	1	0.6	0.8	1	0.2	0.2	0.4
Identifying viewpoints	0.8	0.6	0.4	0.8	0	0.8	0.4

TABLE 5.6. Requirements Evaluation & Negotiation technique attributes evaluation $\mathtt{V}(a_x, t)$

Attribute	**Proto-typing**	**Scenario**	**UML**	**Goal-oriented Analysis**	**Functional Decom-position**
Modelling MT requirements	0.8	1	1	0.8	0.6
Analysing requirements with relevant MT notations	0.6	1	0.8	0.8	0.8
Analysing non-functional requirements	0.2	0.2	0	0.6	0.2
Facilitating negotiation with stakeholders	0.8	0.6	0.8	0.8	0.4
Prioritizing requirements based on stakeholders	0.2	0.4	0	0.4	0.2
Identifying accessibility of the transformation	0.8	0.8	0.6	0.6	0.2
Modeling interface requirements	0.6	1	1	0.4	0.2
Re-usability of MT requirements	0	0	1	0.2	0

TABLE 5.7. Requirements Specification & Documentation technique attributes evaluation $\mathtt{V}(a_x, t)$

Attribute	**SysML**	**KAOS**	**Structured language template**	**SADT**	**Evolutionary Prototyping**	**UML**
Representing MT requirements	0.8	0.8	0.8	0.6	0.8	1
Requirements verification	1	1	0	0.4	0.8	0.8
Semantic completeness	0.8	1	0.4	0.6	0.2	0.8
Representing requirements using MT notations	0.6	0.4	0.4	0.4	0.2	1
Writing complete requirements	0.8	0.8	0.6	0.6	0.4	0.8
Requirements management	0.8	0.4	0.6	1	0	0.8
Designing highly modular systems	0.8	0.6	0.2	0	0	0.8
Implementability of the notation(s)	1	1	0	0	0.8	0.8

TABLE 5.8. Requirements Validation & Verification technique attributes evaluation $\mathtt{V}(a_x, t)$

Attribute	**Inspection**	**Desk-Checks**	**Rapid Prototyping**	**Checklist**
Identifying ambiguous requirements	0.4	0	0.4	0
Identifying inconsistency and conflict	0.4	1	0.8	1
Identifying incomplete requirements	0.8	0.8	0.8	0.8

The set $\mathtt{T}$ 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 $\mathtt{t} \in \mathtt{T}$, a value $\mathtt{RA}(\mathtt{t})$ (Requirement Attribute of a technique) is calculated, which represents the suitability of applying $\mathtt{t}$ in the project, which is based on the technique attributes. The function $\mathtt{RA} : \mathtt{T} \longmapsto [0, 1]$ is defined as:

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

This expresses that 'there are ($\sum$) attributes $a_x \in \mathtt{A}$, important ($\mathtt{I}$) to the project and for which $\mathtt{t}$ is relevant ($\mathtt{V}$)'. 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 $\mathtt{A}$. For instance, $\mathtt{A}$ = {eliciting MT requirements, facilitating communication, ..., identifying incomplete requirements}.
- $\mathtt{I}(a_x)$ which is a value in the range $[0, 1]$, represents the importance of an attribute $a_x \in \mathtt{A}$ for the project. A low $\mathtt{I}(a_x)$ value for an

attribute $a_x \in \mathtt{A}$ means a_x is not important for the MT project and a high $\mathtt{I}(a_x)$ value for an attribute $a_x \in \mathtt{A}$ represents high importance for the project. The assignment of $\mathtt{I}(a_x)$ to each $a_x \in \mathtt{A}$ 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 "a_x = identifying MT stakeholders" is assigned a lower $\mathtt{I}(a_x)$ value (than $\mathtt{I}(a_x)$ of documentation), because it is a secondary task compared to the documentation requirement.

- $\mathtt{V}(a_x, t)$ is a function $\mathtt{V} : \mathtt{A} \times \mathtt{T} \longmapsto [0, 1]$ 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.

5.3.2 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

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[1]

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) bidirectionality or change propagation

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)

Quality

Very High: the transformation has a very high level of functionality, reliability and usability requirements ($\geq$ 100 requirements)

High: the transformation has a high level of functionality, reliability and usability requirements ($\geq$ 50 requirements)

Medium: the transformation has a medium level of functionality, and usability requirements ($\geq$ 20 requirements)

Low: there are low reliability, etc. requirements ($\geq$ 10 requirements)

Very Low: there are very low levels of reliability, etc. requirements ($<$ 10 requirements)

Time

Very High: the transformation has very restrictive development time constraints (less than 5% extension is possible)

High: the transformation has a high level of development time constraints (less than 10% extension is possible)

Medium: the transformation has a medium level of development time constraints (less than 20% extension is possible)

Low: the transformation has low development time constraints (up to 50% extension is possible)

Very Low: the transformation has very low development time constraints (more than 50% extension is possible)

Cost

Very High: the budget is very tight (less than 5% extension is possible)

High: the budget is tight (less than 10% extension)

Medium: the transformation has a limited budget (less than 20% extension)

Low: the transformation has the budget to cover different aspects and unforeseen circumstances (up to 50% extension permitted)

Very Low: the budgets are flexible (more than 50% extension is possible)

Domain knowledge	*Very High:* developers have good background knowledge and previous experience regarding the domain (at least 5 years experience)
	High: there is a good amount of knowledge and experience regarding the domain (at least 2 years experience)
	Medium: there is some background knowledge and experience regarding the domain (at least 1 year experience)
	Low: the amount of experience and knowledge regarding the domain is low (some experience, less than 1 year)
	Very Low: there is no experience or knowledge about the domain (no experience)

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

TABLE 5.9. Project attributes weighting

Project attribute	**Value**
Very high/large	0.8 - 1
High/large	0.6 - 0.8
Medium	0.4 - 0.6
Low/small	0.2 - 0.4
Very low/small	0 - 0.2

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

on the project's descriptions. The function $\mathtt{PD} : \mathtt{T} \longmapsto [0,1]$ is defined as:

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

This expresses that 'the technique $\mathtt{t}$ is relevant based upon all ($\prod$) the project attributes $\mathtt{d}_x \in \mathtt{D}$'.

Where:

- The set of all project descriptors (e.g. size, complexity) is $\mathtt{D}$. In this thesis, we assume $\mathtt{D}$ = {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.

- $\mathtt{W}(d_x)$ is a function $\mathtt{W} : \mathtt{D} \longmapsto [0,1]$ which represents the magnitude of a specific descriptor in the project. For example, for $\mathtt{d}$ = 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 $\mathtt{d}$ = 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.

- $\mathtt{ID_t} \subseteq \mathtt{D}$ is a set containing all descriptors with inverse impact for a specific RE technique $\mathtt{t}$. More specifically, for each $\mathtt{d} \in \mathtt{ID_t}$, the higher the value of $\mathtt{W}(d_x)$ the more negative the impact of applying $\mathtt{t}$ in that project. An example of such a descriptor for technique "interview" is time, where the higher the value of $\mathtt{W}$(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 $-\mathtt{1}$ indicates that $\mathtt{1} - W(\mathtt{d}_x)$ is used and $\mathtt{1}$ idicates that $\mathtt{W(d}_x\mathtt{)}$ is used. This is static and fixed, independent of particular projects.

TABLE 5.10. Technique weight descriptor values

Attributes	**Size**	**Complexity**	**Volatility**	**Relation**	**Safety**	**Quality**	**Time**	**Cost**	**Knowledge**
Interview	1	1	1	1	1	1	-1	-1	1
Questionnaire	1	1	-1	-1	1	1	1	1	1
Scenario	-1	-1	-1	1	1	1	1	-1	1
Prototype	-1	-1	-1	1	1	1	1	-1	1
Rapid prototyping	-1	-1	1	1	1	1	1	-1	1

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 $\mathtt{E}$ (for Experience) of experience/expertise regarding the RE technique $\mathtt{t}$ available in the development team. $\mathtt{E} : \mathtt{T} \longmapsto [0, 1]$ is a function where $\mathtt{E(t)}$ represents the level of experience and practical and theoretical knowledge of the developer regarding $\mathtt{t}$. For instance, depending on the number of projects in which the developer has applied a particular RE technique, the weight of $\mathtt{E}$ may vary. If the developer

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

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, $\mathtt{t}$, in the MT project is identified as follows, using the formula:

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

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 $\mathtt{T}_{elicitation}$ are our chosen sample because the developers had some experience with these techniques.
  - $\mathtt{T}_{elicitation}$= {interview, prototyping, questionnaire, document mining, brainstorming, scenario, ethno methodology}
  - $\mathtt{t}_1$= interview, $\mathtt{t}_2$= questionnaire are chosen arbitrarily and any number of these techniques can be chosen.
  - $\mathtt{I}(a)$ has a dynamic weighting which can be assigned from a range [0,1] according to the importance of the technique attributes, $\mathtt{A}_1$ and $\mathtt{A}_2$, which is determined by the developers according to the initial project description and stakeholders.
  - $\mathtt{I}(a)$ value is 1 for $\mathtt{A}_1$, 0.8 for $\mathtt{A}_2$ and 0 for the remaining attributes[2] :

[2] The set of attributes ($\mathtt{A}_1$, $\mathtt{A}_2$) has been grouped according to stakeholder information and developer understanding

  - $\mathtt{A_1}$={eliciting MT requirements, getting domain knowledge, identifying non-functional requirements}
  - $\mathtt{A_2}$ = {identifying MT stakeholders, facilitating communications}
  - $\mathtt{A_3} = \mathtt{A} - (\mathtt{A_1} + \mathtt{A_2})$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_1})$ for $\mathtt{A_1}$ is $\{1, 0.6, 1\}$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_1})$ for $\mathtt{A_2}$ is $\{1, 1\}$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_2})$ for $\mathtt{A_1}$ is $\{0.8, 0.6, 0.6\}$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_2})$ for $\mathtt{A_2}$ is $\{0.8, 1\}$

- $$\mathtt{RA}(\mathtt{t_1}) = \frac{[(1 \times 1) + (1 \times 0.6) + (1 \times 1) + (0.8 \times 1) + (0.8 \times 1)]}{4.6}$$
$$= \frac{4.2}{4.6} = 0.91$$

- $$\mathtt{RA}(\mathtt{t_2}) = \frac{[(1 \times 0.8) + (1 \times 0.6) + (1 \times 0.6) + (0.8 \times 0.8) + (0.8 \times 1)]}{3.8}$$
$$= \frac{3.44}{3.8} = 0.9$$

- category: *Evaluation & Negotiation*
  - $\mathtt{T}_{Evaluation}$= {prototyping, UML, scenario, goal-oriented analysis}
  - $\mathtt{t_3}$ = scenario, $\mathtt{t_4}$ = prototyping
  - $\mathtt{I}(a)$ value is 1 for $\mathtt{A_1}$, 0.8 for $\mathtt{A_2}$ and 0 for the remaining attributes:
    - $\mathtt{A_1}$={modelling MT requirements, identifying accessibility of the transformation}
    - $\mathtt{A_2}$ = {prioritizing requirements based on stakeholders, analysing non-functional requirements}

    * $\mathtt{A_3} = \mathtt{A} - (\mathtt{A_1} + \mathtt{A_2})$
  - $\mathtt{v}(\mathtt{a_x}, \mathtt{t_3})$ for $\mathtt{A_1}$ is $\{1, 0.8\}$
  - $\mathtt{v}(\mathtt{a_x}, \mathtt{t_3})$ for $\mathtt{A_2}$ is $\{0.4, 0.2\}$
  - $\mathtt{v}(\mathtt{a_x}, \mathtt{t_4})$ for $\mathtt{A_1}$ is $\{0.8, 0.8\}$
  - $\mathtt{v}(\mathtt{a_x}, \mathtt{t_4})$ for $\mathtt{A_2}$ is $\{0.2, 0.2\}$
  -
$$\mathtt{RA}(\mathtt{t_3}) = \frac{[(1 \times 1) + (1 \times 0.8) + (0.8 \times 0.4) + (0.8 \times 0.2)]}{2.4}$$
$$= \frac{2.28}{2.4} = 0.95$$
  -
$$\mathtt{RA}(\mathtt{t_4}) = \frac{[(1 \times 0.8) + (1 \times 0.8) + (0.8 \times 0.2) + (0.8 \times 0.2)]}{2}$$
$$= \frac{1.92}{2} = 0.96$$

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

- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_6})$ for $\mathtt{A_2}$ is $\{1\}$
- $$\mathtt{RA}(\mathtt{t_5}) = \frac{[(1 \times 0.8) + (1 \times 0.2) + (0.8 \times 0.2)]}{1.2}$$
  $$\frac{1.16}{1.2} = 0.96$$
- $$\mathtt{RA}(\mathtt{t_6}) = \frac{[(1 \times 0.8) + (1 \times 0.8) + (0.8 \times 1)]}{2.6}$$
  $$\frac{2.4}{2.6} = 0.92$$

- category: *Validation & Verification*
  - $\mathtt{T}$ = {interview, UML, rapid prototyping, questionnaire, formal methods, structured language template, checklist}
  - $\mathtt{t_7}$ = rapid prototyping
  - $\mathtt{I}(a)$ value is 1 for $\mathtt{A_1}$, 0.8 for $\mathtt{A_2}$ and 0 for the remaining attributes:
    - $\mathtt{A_1}$={identifying incomplete requirements, identifying inconsistency and conflict}
    - $\mathtt{A_2}$ = {identifying ambiguous requirements}
    - $\mathtt{A_3} = \mathtt{A} - (\mathtt{A_1} + \mathtt{A_2})$
  - $\mathtt{v}(\mathtt{a_x}, \mathtt{t_7})$ for $\mathtt{A_1}$ is $\{0.8, 0.8\}$
  - $\mathtt{v}(\mathtt{a_x}, \mathtt{t_7})$ for $\mathtt{A_2}$ is $\{0.4\}$
  - $$\mathtt{RA}(\mathtt{t_7}) = \frac{[(1 \times 0.8) + (1 \times 0.8) + (0.8 \times 0.4)]}{2}$$
    $$\frac{1.92}{2} = 0.96$$

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

$$\mathtt{PD}(\mathtt{t}) = \prod_{\mathtt{d}_x \in \mathtt{D}} \begin{cases} 1 - \mathtt{W}(\mathtt{d}_x) & \text{if } \mathtt{d}_x \in \mathtt{ID}_\mathtt{t} \\ \mathtt{W}(\mathtt{d}_x) & \text{otherwise} \end{cases}$$

- 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[3]:
  - *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)*
- $\mathtt{D}$ = {size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}
- $\mathtt{ID}_{\mathtt{interview}}$ = {time, cost}
- $\mathtt{ID}_{\mathtt{questionnaire}}$ = {volatility, relationship}
- $\mathtt{ID}_{\mathtt{scenario}}$ = {size, volatility, time, cost }
- $\mathtt{ID}_{\mathtt{prototype}}$ = {size, complexity, volatility, cost }
- $\mathtt{ID}_{\mathtt{evolutionaryprototyping}}$ = {size, complexity, volatility, cost}

[3] The values have been determined from the transformation project attributes weighting (Table 5.9)

- $\texttt{ID}_{\texttt{UML}} = \{\text{complexity, volatility}\}$
- $\texttt{ID}_{\texttt{rapidprototyping}} = \{\text{size, complexity, cost}\}$
- $\texttt{PD}(\texttt{t}_1) = (0.2) \times (0.8) \times (0.2) \times (0.4) \times (0.2) \times (0.5) \times (0.8) \times (0.2) \times (0.6) = 0.0001$
- $\texttt{PD}(\texttt{t}_2) = (0.2) \times (0.8) \times (0.8) \times (0.6) \times (0.2) \times (0.5) \times (0.8) \times (0.8) \times (0.6) = 0.0029$
- $\texttt{PD}(\texttt{t}_3) = (0.8) \times (0.8) \times (0.8) \times (0.4) \times (0.2) \times (0.5) \times (0.8) \times (0.2) \times (0.6) = 0.0019$
- $\texttt{PD}(\texttt{t}_4) = (0.8) \times (0.2) \times (0.8) \times (0.4) \times (0.2) \times (0.5) \times (0.2) \times (0.2) \times (0.6) = 0.0001$
- $\texttt{PD}(\texttt{t}_5) = (0.8) \times (0.2) \times (0.8) \times (0.4) \times (0.2) \times (0.5) \times (0.2) \times (0.2) \times (0.6) = 0.0001$
- $\texttt{PD}(\texttt{t}_6) = (0.2) \times (0.2) \times (0.8) \times (0.4) \times (0.2) \times (0.5) \times (0.2) \times (0.8) \times (0.6) = 0.0001$
- $\texttt{PD}(\texttt{t}_7) = (0.8) \times (0.2) \times (0.2) \times (0.4) \times (0.2) \times (0.5) \times (0.2) \times (0.2) \times (0.6) = 0.00003$

Example 4 (Continuing Example 3)**.** Evaluating the degree $\texttt{E}$ (for Experience) of experience/expertise regarding the RE technique $\texttt{t}$ available in the development team. $\texttt{E} : \texttt{T} \longmapsto [0, 1]$ is a function where $\texttt{E}(\texttt{t})$ represents the level of experience and practical and theoretical knowledge of the developer regarding $\texttt{t}$. The value of $\texttt{E}$ 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 $\texttt{S}(\texttt{t})$ of the different techniques ($\texttt{t} \in \texttt{T}$) that have been identified throughout this example. $\texttt{S}(\texttt{t})$ values reflect the relevance of a particular technique, the higher the $\texttt{S}(\texttt{t})$ value the higher the priority of that technique.

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

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

TABLE 5.11. Attributes calculation of RE techniques

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

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.

- 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

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., *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 *ProjectDecriptors* class data are defined for each project, whereas *ElicitationTechnique* and *ElicitationTechniqueSuitability* class data are fixed tables independent of each project.

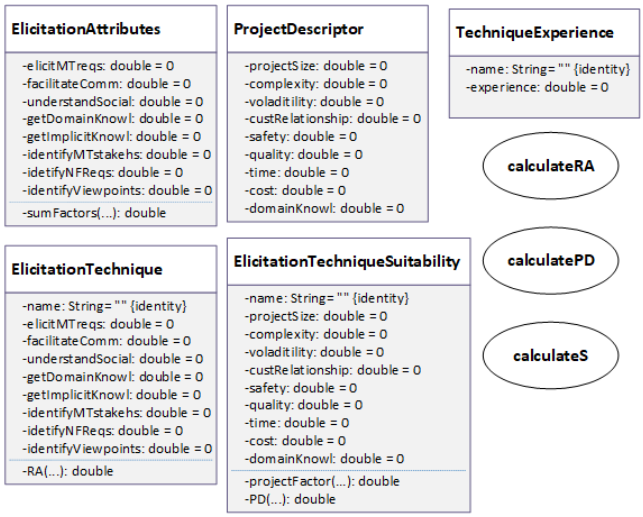


Figure 5.4. RE technique framework metamodel in UML-RSDS

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 `RA(t)` calculation has been generated as an operation according to UML-RSDS syntax as follows:

```
*** 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.identifyNFReqs * identifyNFReqs + 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 `PD(t)` calculation has been generated as an operation according to UML-RSDS syntax as follows:

```
*** 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

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.

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, $\mathtt{S}(\mathtt{t})$, through the use of this formula:

$$\mathtt{S}(\mathtt{t}) = \mathtt{RA}(\mathtt{t}) \times \mathtt{PD}(\mathtt{t}) \times \mathtt{E}(\mathtt{t})$$

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 $\mathtt{X}$ (technique) for $\mathtt{Y}$ (specific requirement attribute). Step by step application of this stage is presented below:

- category: *Domain Analysis & Requirements Elicitation*
  - The techniques in $\mathtt{T}_{elicitation}$ are our chosen sample because the developers had some experience with these techniques.
  - $\mathtt{T}_{elicitation}$ = {interview, prototyping, questionnaire, document mining, brainstorming, scenario, ethno methodology}
  - $\mathtt{t}_1$= interview, $\mathtt{t}_2$= questionnaire, $\mathtt{t}_3$= document mining are chosen arbitrarily for this example (Note that any number of techniques can be chosen).
  - $\mathtt{I}(a)$ has a dynamic weighting which can be assigned from a range [0,1] according to the importance of the technique attributes, $\mathtt{A}_1$ and $\mathtt{A}_2$, which is determined by the developers according to the initial project description and stakeholders.

TABLE 6.1. Domain Analysis & Requirements Elicitation technique attributes evaluation

Attribute	**Interview**	**Question-naire**	**Document Mining**	**Brainstor-ming**	**Scenario**
Eliciting MT requirements	1	0.8	1	0.8	1
Facilitating communication	1	1	0	0.8	0.8
Understanding social issues	0.8	1	0.8	0.4	0.2
Getting domain knowledge	0.6	0.6	1	1	0.4
Getting implicit knowledge	0.2	0.2	0.2	0.2	0.2
Identifying MT stakeholders	1	0.8	0.2	1	0
Identifying non-functional requirements	1	0.6	0.8	1	0.2
Identifying viewpoints	0.8	0.6	0.4	0.8	0

- $\mathtt{I}(a)$ value is 1 for $\mathtt{A_1}$, 0.8 for $\mathtt{A_2}$ and 0 for the remaining attributes:
  - $\mathtt{A_1}$= {eliciting MT requirements, getting domain knowledge, getting implicit knowledge}
  - $\mathtt{A_2}$ = {identifying MT stakeholders, facilitating communications, identifying non-functional requirements}
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_1})$ for $\mathtt{A_1}$ is $\{1, 0.6, 0.2\}$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_1})$ for $\mathtt{A_2}$ is $\{0.8, 0.8, 0.8\}$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_2})$ for $\mathtt{A_1}$ is $\{0.8, 0.6, 0.2\}$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_2})$ for $\mathtt{A_2}$ is $\{0.8, 1, 0.6\}$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_3})$ for $\mathtt{A_1}$ is $\{1, 1, 0.2\}$
- $\mathtt{v}(\mathtt{a_x}, \mathtt{t_3})$ for $\mathtt{A_2}$ is $\{0.2, 0, 0.8\}$

- $$\texttt{RA}(\texttt{t}_1) = \frac{[(1 \times 1) + (1 \times 0.6) + (1 \times 0.2) + (0.8 \times 0.8) + (0.8 \times 0.8) + (0.8 \times 0.8)]}{4.8}$$
$$= \frac{4.2}{4.8} = 0.77$$

- $$\texttt{RA}(\texttt{t}_2) = \frac{[(1 \times 0.8) + (1 \times 0.6) + (1 \times 0.2) + (0.8 \times 0.8) + (0.8 \times 1) + (0.8 \times 0.6)]}{4}$$
$$= \frac{3.52}{4} = 0.88$$

- $$\texttt{RA}(\texttt{t}_3) = \frac{[(1 \times 1) + (1 \times 1) + (1 \times 0.2) + (0.8 \times 0.2) + (0.8 \times 0) + (0.8 \times 0.8)]}{3.2}$$
$$= \frac{3}{3.2} = 0.93$$

- 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[1]:

- 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 = {size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}
- $ID_{interview}$ = {time, cost}
- $ID_{questionnaire}$ = {relationship}
- $ID_{documentmining}$ = {relationship, domain knowledge}
- $PD(t_1) = (0.8) \times (0.8) \times (0.2) \times (0.2) \times (0.5) \times (0.8) \times (0.5) \times (0.8) \times (0.4) = 0.0016$
- $PD(t_2) = (0.8) \times (0.8) \times (0.2) \times (0.2) \times (0.5) \times (0.8) \times (0.5) \times (0.2) \times (0.4) = 0.0004$
- $PD(t_3) = (0.8) \times (0.8) \times (0.2) \times (0.8) \times (0.5) \times (0.8) \times (0.5) \times (0.2) \times (0.6) = 0.0024$
- $E(t)$ 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)

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

Measures	**Brainstorming**	**Interview**	**Mining**	**Scenario**	**Questionnaire**
RA(t)	0.88	0.77	0.93	0.9	0.88
PD(t)	0.0004	0.0016	0.0024	0.0016	0.0002
E(t)	0.4	1	0.8	1	1
S(t)	0.0001	0.0012	0.0017	0.0014	0.0002

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

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

- 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

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:

- 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:

- 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, eg., 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.

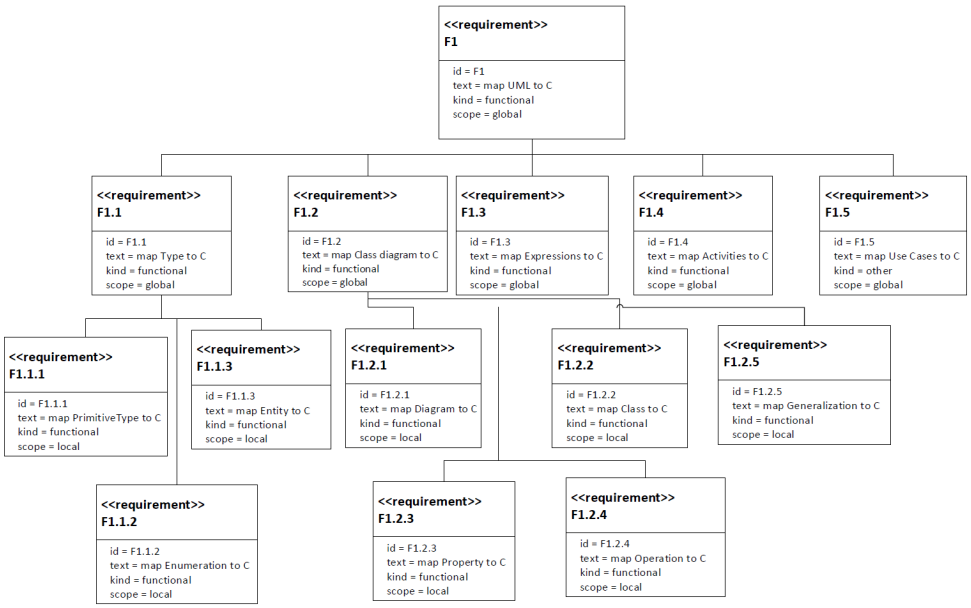


Figure 6.1. Functional requirements decomposition in SysML

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

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.

TABLE 6.3. Technique attributes evaluation of the Evaluation & Negotiation stage $\mathtt{V}(a_x, t)$ for UML to C case

Attribute	**Prototyping**	**Scenario**	**UML**
Modelling MT requirements	0.8	1	1
Analysing non-functional requirements	0.2	0.2	0
Modelling interface requirements	0.6	1	1
Facilitate negotiation	0.8	0.6	0.8
RA(t)	0.93	0.95	0.94
PD(t)	0.0004	0.0016	0.0004
S(t)	0.0002	0.0015	0.0003

TABLE 6.4. Technique attributes evaluation of the Specification & Documentation stage $\mathtt{V}(a_x, t)$ for UML to C case

Attribute	**Natural language**	**UML**	**SysML**
Representing MT requirements	0.8	1	0.8
Semantic completeness	0.6	0.8	1
Write complete requirements	0.6	0.8	0.8
Modularity	0.2	0.8	0.8
RA(t)	0.80	0.95	0.95
PD(t)	0.0004	0.0004	0.0004
S(t)	0.0003	0.0003	0.0001

Decomposing the code generator into two sub-transformations improves its modularity, and simplifies the constraints, which would other-

TABLE 6.5. Technique attributes evaluation of the Validation & Verification stage $\mathtt{V}(a_x, t)$ for UML to C case

Attribute	**(rapid)Prototyping**	**Inspection**	**Checklist**
Identifying ambiguous requirements	0.4	0.4	0
Identifying inconsistency and conflict	0.8	0.4	1
Identifying incomplete requirements	0.8	0.8	8
RA(t)	0.96	0.95	1
PD(t)	0.0004	0.0001	0.0004
S(t)	0.0003	0.00005	0.0002

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*.

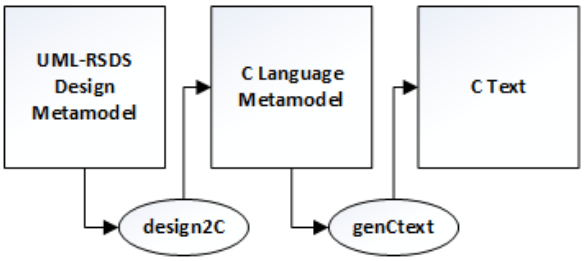


Figure 6.2. C code generator architecture

In the following subsections we present the application of the selected RE techniques on the case study.

6.1.1 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. Bidirectionality 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-

TABLE 6.6. Informal scenarios for types2C

Scenario	UML element e	C representation e'
F1.1.1.1	*String type*	*char**
F1.1.1.2	*int, long, double types,*	*same-named C types*
F1.1.1.3	*boolean type*	*unsigned char*
F1.1.2	*Enumeration type*	*C enum*
F1.1.3	*Entity type E*	*struct E* type*
F1.1.4.1	*Set(E) type*	*struct E** (array of E, without duplicates)*
F1.1.4.2	*Sequence(E) type*	*struct E** (array of E, possibly with duplicates)*

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 *linkedClass*. 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

TABLE 6.7. Informal scenarios for the mapping of UML class diagrams to C

Scenario	**UML element** e	**C representation e'**
F1.2.1	Class diagram D	C program with D's name
F1.2.2	Class E	struct E {...}; Global variable struct E** e_instances; Global variable int e_size; struct E* createE() operation struct E** newEList() operation
F1.2.3.1	Property p : T (not principal identity attribute)	Member T'p; of the struct for p's owner, where T' represents T Operations T' getE_p(E' self) and setE_p(E' self, T' px)
F1.2.3.2	Principal identity attribute p : String of class E	Operation struct E* getEByPK(char* v) Key member char* p; of the struct for E
F1.2.4	Operation op(p : P) : T of E	C operation T' op(E' self, P' p) with scope = entity
F1.2.5	Inheritance of A by B	Member struct A* super; of struct B

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).

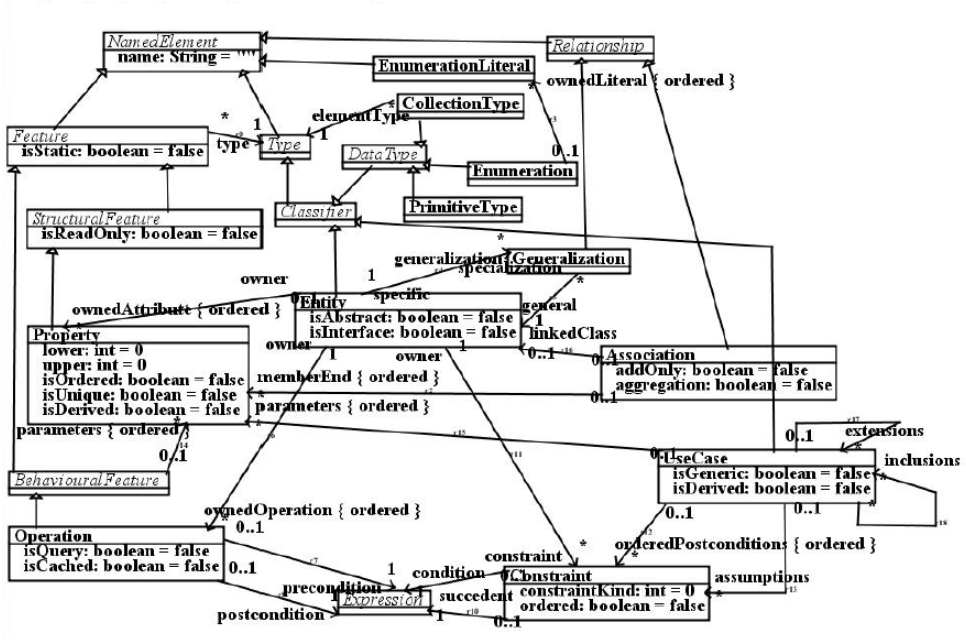


Figure 6.3. UML-RSDS class diagram metamodel

Mapping of Comparator, Numeric and String Expressions

Table 6.10[2] 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, intersec-*

[2] OCL library functions *isIn, equalsSet and etc.* are defined in a file ocl.h to support the execution of OCL expressions

TABLE 6.8. Mapping scenarios for Basic Expressions

OCL expression e	**C representation e'**
self	*self* as an operation parameter
Variable *v* or *v[ind]*	v *v[ind - 1]*
Data feature *f* with no objectRef	*self* → *f*
Data feature *f* of instance *ex*	*ex'* → *f*
Operation call *op(e1,...,en)* or *obj.op(e1,...,en)*	op(self, e1', ..., en') op(obj', e1', ..., en')
Attribute *f* of collection *exs*	get*AllE_f (exs')* (duplicate values preserved)
Single-valued role r : F of collection exs	*getAllE_r(exs')* defined by (*struct* F ∗∗) *collectE(exs', getE_r)*
col [ind] ordered collection col	(col')[ind-1]
E[v] *v* single-valued	getEByPK(v')
E[*vs*] *vs* collection-valued	getEByPKs(vs')
E.allInstances	*e_instances*
value of enumerated type, numeric or string value	*value*
boolean true, false	TRUE, FALSE

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→sort()* is then: (rt) treesort((void∗∗) x', comp) for the appropriate result type *rt* and comparator function *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-

TABLE 6.9. Mapping scenarios for Logical Expressions

OCL expression e	**C representation e'**
A =>B	!A' || B'
A & B	A' && B'
A or B	A' || B'
not(A)	!A'
E->exists(P)	existsE(e_instances,fP) fP evaluates P'
e->exists(P)	existsE(e',fP)
E->exists1(P)	exists1E(e_instances,fP) fP evaluates P'
e->exists1(P)	exists1E(e',fP)
E->forAll(P)	forAllE(e_instances,fP) fP evaluates P'
e->forAll(P)	forAllE(e',fP)

pressions. 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.

TABLE 6.10. Mapping scenarios for Comparator Expressions

OCL expression e	**C representation e'**
x : E E entity type	isIn((void∗) x', (void ∗∗) e_instances)
x : s s collection	isIn((*void*∗) x', (void ∗∗) s')
s->includes(x) s collection	Same as x : s
x / : E E entity type	!isIn((void∗) x', (void ∗∗) e_instances)
x / : s s collection	!isIn((void∗) x', (void ∗∗) s')
s->excludes(x) s collection	Same as x / : s
x = y Numerics, Booleans Strings Objects Sets Sequences	x'== y' strcmp(x', y') == 0 x'== y' equalsSet((void ∗∗) x', (void ∗∗) y') equalsSequence((void ∗∗) x', (void ∗∗) y')
x <y Numerics Strings	x'<y' strcmp(x', y') <0
Similarly for >, <=, >=, /=	>, <=, >=, ! =
s <: t s, t collections	containsAll ((void ∗∗) t', (void ∗∗) s')
t->includesAll(s)	Same as s <: t
t->excludesAll(s)	disjoint((void∗∗) t', (void∗∗) s')

TABLE 6.11. Mapping scenarios for Numeric Expressions

OCL expression e	Representation in C
-x	-x'
x + y	x' + y'
numbers	
x - y	x' - y'
x* y	x' * y'
x / y	x' / y'
x mod y	x' % y'
x.sqr	(x' * x')
x.sqrt	sqrt(x')
x.floor	oclFloor(x') defined as: ((int) floor(x'))
x.round	oclRound(x')
x.ceil	oclCeil(x') defined as: ((int) ceil(x'))
x.abs	fabs(x')
x.exp	exp(x')
x.log	log(x')
x.pow(y)	pow(x',y')
x.sin, x.cos, x.tan	sin(x'), cos(x'), tan(x')
Integer.subrange(st,en)	intSubrange(st',en')

TABLE 6.12. Scenarios for the mapping of Selection and Collection Expressions

UML expression e	C translation e'
s->select(P)	selectE(s', fP) fP evaluates P'
s->select(x | P)	as above
s->reject(P)	rejectE(s', fP)
s->reject(x | P)	as above
s->collect(e)	(et'$*$) collectE(s', fe)
e of type et	fe evaluates e'
s->collect(x | e)	as above

TABLE 6.13. Scenarios for the translation of Collection Operators (1)

Expression e	**C translation e'**
Set{}	newEList()
Sequence{}	newEList()
Set{x1, x2, ... , xn}	insertE(... insertE(newEList(), x1'), ..., xn')
Sequence{x1, x2, ..., xn}	appendE(... appendE(newEList(), x1'), ..., xn')
s->size()	length((void**) s')
s->including(x)	insertE(s',x') or appendE(s',x')
s->excluding(x)	removeE(s',x')
s - t	removeAllE(s',t')
s->append(x)	appendE(s',x')
s->count(x)	count((void*) x', (void**) s')
s->indexOf(x)	indexOf((void*) x', (void**) s')
x∨y	unionE(x',y')
x∧y	intersectionE(x',y')
x⌢ y	concatenateE(x',y)
x->union(y)	unionE(x',y')
x->intersection(y)	intersectionE(x', y')
x->any()	x'[0]
x->reverse()	reverseE(x')
x->front()	subrangeE(x',1,length((void**) x')-1)
x->tail()	subrangeE(x',2,length((void**) x'))
x->first()	x'[0]

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 &
```

TABLE 6.14. Scenarios for the translation of Collection Operators (2)

Expression e	**C translation e'**
x->last()	x'[length((void**) x')-1]
x->sort()	qsort((void**) x', length((void**) x'), sizeof(struct E*), compareToE)
x->sortedBy(e)	qsort((void**) x', length((void**) x'), sizeof(struct E*), compare) compare defines e-order
x->sum()	sumString(x'), sumint(x'), sumlong(x'), sumdouble(x')
x->prd()	prdint(x'), prdlong(x'), prddouble(x')
Integer.Sum(a,b,x,e)	sumInt(a',b',fe), sumDouble(a',b',fe) fe computes e'(x')
Integer.Prd(a,b,x,e)	prdInt(a',b',fe), prdDouble(a',b',fe)
x->max()	maxInt(x'), maxLong(x'), maxDouble(x'), maxString(x')
x->min()	minInt(x'), minLong(x'), minDouble(x'), minString(x')
x->asSet()	asSetE(x')
x->asSequence()	x'
s->isUnique(e)	isUniqueE(s',fe)
x->isDeleted()	killE(x')

TABLE 6.15. Scenarios for mapping of UML Activities to C Statements

Requirement	**UML activity st**	**C statement st'**
F1.4.1	Creation statement x : T defaultT' is default value of T'	T' x = defaultT';
F1.4.2	Assign statement v := e	v' = e';
F1.4.3	Sequence statement st1 ; ... ; stn	st1' ... stn'
F1.4.4	Conditional statement if e then st1 else st2	if e' {st1'} else {st2'}
F1.4.5	Return statement return e	return e';
F1.4.6	Break statement break	break;
F1.4.7	Bounded loop for (x : e) do st on object collection e of entity element type E	int i = 0; for (; i <length((void**) e'); i++) { struct E* x = e'[i]; st' } New index variable i
F1.4.8	Unbounded loop while e do st	while (e') { st' }
F1.4.9	Operation call ex.op(pars)	op(ex',pars')

```
cop.scope = "application" &
cop.isQuery = false &
```

```
cop.code = classifierBehaviour.mapStatement() &
cop.parameters = parameters.mapExpression() &
cop.returnType = CType[returnType.typeId] )
```

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 (eg., 16 bit int type) as the target code platform. For dynamic memory allocation, we assume that *malloc* and *calloc* always succeed. Select and other iterator expressions are restricted to depend on only one variable. Only collections containing string or entity

instances can be explicitly constructed.

TABLE 6.16. Achievement of requirements

Requirement	**Priority**	**Achievement**
NF1: Termination	High	Proved
NF10: Development time	High	Achieved
F2: Syntactic correctness	High	Rigorous argument
F3: Semantic preservation	High	Rigorous argument
F4: Traceability	High	Achieved
F5: Bidirectionality	Medium	Partly achieved
NF2: Transformation efficiency	Medium	Achieved
NF3: Transformation modularity	Medium	Achieved
NF5: Usability	Medium	Achieved
NF6: Efficient code	Medium	Partly achieved
NF7: Compact code	Medium	Partly achieved
F6: Confluence	Low	Proven
NF4: Flexibility	Low	Not achieved

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 $neighbours : A \to Set(A)$. There is a use case *maxnode* with the postcondition:

```
A::
n : neighbours & n.x > x@pre  =>  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

TABLE 6.17. Overall development effort for C code generator

Stage	**Effort (person days)**
Req. Elicitation	17
Eval./Negotiation	5
Specification	56
Review/Validation	57
Implementation//Testing	49
Total	184

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

	C version	**Java version**
Code size	17Kb	35Kb
Execution time		
A.size = 20	0	30ms
A.size = 50	15ms	70ms
A.size = 100	240ms	330ms
A.size = 200	1750ms	2500ms

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-

TABLE 6.19. Development effort for code generators (person months)

	Java 4	**Java 6**	**Java 7**	**C#**	**C++**	**C**
Req. Anal.	6	1	2	3	6	4.5
Coding	12	3	4	4	6	1
Testing	6	1	1	1	2	0.5
Maintenance	6	1	1	1	3	0
Total	30	6	8	9	17	6

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

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

Measure	**C++ generator**	**UML2C**
Size (LOC)	18,100	4,900
Abstraction level	Low (code)	High (specification)
Software architecture	Partial	Detailed
Modularity	Low (one module)	High (10 Modules)
Cohesion	Low	High
Coupling	Low	Low
Design patterns	2	13

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)

- **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

Transformation	**ETS**	**EHS + ERS**	**CC**	**ENO + ENR**	**EPL**
C++ translator	5	16.5	110.6	28	2
UML2C	2	13	62	4	0

6.2 Case Study 2: CDO Risk Estimation

Case study 2 is the adaptation of a procedure for calculating and evaluating the risk of mutiple-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 $\texttt{P}(\texttt{S} = \texttt{s})$ 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 $\texttt{P}(\texttt{S} = \texttt{s})$. 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:

$$\mathtt{P}(\mathtt{N_k} = \mathtt{m}) = \binom{\mathtt{n_k}}{\mathtt{m}}(\mathtt{p_k^m}(1-\mathtt{p_k})^{\mathtt{n_k}-\mathtt{m}}(1-\mathtt{q_k})^{\mathtt{m}(\mathtt{n_k}-\mathtt{m})}+$$

$$\sum_{\mathtt{i}=1}^{\mathtt{m}-1}\binom{\mathtt{m}}{\mathtt{i}}\mathtt{p_k^i}(1-\mathtt{p_k})^{\mathtt{n_k}-\mathtt{i}}(1-(1-\mathtt{q_k})^{\mathtt{i}})^{\mathtt{m}-\mathtt{i}}\times(1-\mathtt{q_k})^{\mathtt{i}(\mathtt{n_k}-\mathtt{m})})$$

"When an outbreak occurs in sector $\mathtt{k}$, each single default causes an integer valued credit loss of $\mathtt{L_k}$ and the total loss of $\mathtt{S_k} = \mathtt{N_k}\mathtt{L_k}$, where $\mathtt{N_k} > 0$. Conditioned on an outbreak in a sector, the distribution of the number of defaults is":

Equation (1):

$$\mathtt{P}(\mathtt{N_k} = \mathtt{m}|\mathtt{N_k} > 0) = \mathtt{P}(\mathtt{N_k} = \mathtt{m})/(1-(1-\mathtt{p_k})^{\mathtt{nk}}),\mathtt{m} \geq 1,$$

and the probability of total credit loss given an outbreak is:

Equation (2):

$$\mathtt{P}(\mathtt{S_k} = \mathtt{mL_k}) = \mathtt{P}(\mathtt{N_k} = \mathtt{m}|\mathtt{N_k} > 0)$$

Theorem 3.1:

$$\mathtt{P}(\mathtt{S} = 0) = \mathtt{exp}(-\sum_{\mathtt{k}=1}^{\mathtt{K}}\mu_{\mathtt{k}})$$

and

$$\mathtt{P}(\mathtt{S} = \mathtt{s}) = \frac{1}{\mathtt{s}}\sum_{\mathtt{k}=1}^{\mathtt{K}}\sum_{\mathtt{m_k}=1}^{[\mathtt{s}/\mathtt{L_k}]}\mu_{\mathtt{k}}\mathtt{m_k}\mathtt{L_k}\mathtt{P}(\mathtt{N_k} = \mathtt{m_k}\mathtt{L_k}|\mathtt{N_k} > 0)\times\mathtt{P}(\mathtt{S} = \mathtt{s} - \mathtt{m_k}\mathtt{L_k})$$

The requirements for this case study were quite straight forward:

- F1: Compute $\mathtt{P}(\mathtt{S} = \mathtt{s})$ for CDO portfolios which is derived from Theorem 3.1 based on the computation of $\mathtt{P}(\mathtt{Nk} = \mathtt{m})$ of *Hammarlid* [51].

  - F1.1: Calculate probability of no contagion.
  - F1.2: Calculate probability of contagion.

- F2: Calculate risk function $\mathtt{P}(\mathtt{S} \geqslant \mathtt{s})$.
- NF1: The system must be able to compute results in a practical time (less than 30 seconds for each $\mathtt{s}$ 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 $\mathtt{V}(a_x, t)$ for CDO case

Attribute	**Brainstorming**	**Interview**	**Mining**	**Scenario**
Getting domain knowledge	1	0.6	1	0.4
Eliciting implicit knowledge	0.2	0.2	0.2	0.2
Eliciting MT requirements	0.8	1	1	1
Identifying MT stakeholders	1	1	0.2	0.4
Facilitating communications	0.8	1	0	1
Identifying non-functional requirements	1	1	0.8	0.2
$\mathtt{RA}(\mathtt{t})$	0.88	0.87	0.93	0.9

We applied our RE framework as follows:

1. category: *Domain Analysis & Requirements Elicitation*

   - $\mathtt{T}_{elicitation}$ = {interview, prototyping, questionnaire, document mining, brainstorming, scenario, ethno methodology}
   - $\mathtt{t}_1$ = interview, $\mathtt{t}_2$ = brainstorming, $\mathtt{t}_3$ = document mining
   - $\mathtt{I}(a)$ value is 1 for $\mathtt{A}_1$, 0.8 for $\mathtt{A}_2$ and 0 for the remaining attributes:
     - $\mathtt{A}_1$ = {eliciting MT requirements, getting domain knowledge, getting implicit knowledge}
     - $\mathtt{A}_2$ = {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.

In other words, we have the following[3]:

- 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)*
- $\mathtt{D}$ = {size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}
  - $\mathtt{S}(\mathtt{t}_1) = 0.0002$
  - $\mathtt{S}(\mathtt{t}_2) = 0.003$
  - $\mathtt{S}(\mathtt{t}_3) = 0.0001$

2. category: *Evaluation & Negotiation*

- $\mathtt{T}_{evaluation}$ = {prototyping, scenario, UML, functional decomposition, goal oriented analysis}
- $\mathtt{t}_1$ = prototyping, $\mathtt{t}_2$ = scenario
- $\mathtt{I}(a)$ value is 1 for $\mathtt{A}_1$, 0.8 for $\mathtt{A}_2$ and 0 for the remaining attributes:
  - $\mathtt{A}_1$ = {analysing non-functional requirements, prioritizing requirements}
  - $\mathtt{A}_2$ = {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)*
  - $\mathtt{D}$ = {size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge}
  - $\mathtt{S}(\mathtt{t}_1) = 0.0013$

[3] The values have been determined from the transformation project attributes weighting (Table 5.9)

- $\mathtt{S}(\mathtt{t}_2) = 0.0003$

3. category: *Specification & Documentation*

   - $\mathtt{T}_{specification}$ = {SysML, KAOS, structured language template, SADT, UML, evolutionary prototyping}
   - $\mathtt{t}_1$ = Structured language template, $\mathtt{t}_2$ = UML, $\mathtt{t}_3$ = evolutionary prototyping
   - $\mathtt{I}(a)$ value is 1 for $\mathtt{A}_1$, 0.8 for $\mathtt{A}_2$ and 0 for the remaining attributes:
     - $\mathtt{A}_1$ = {semantics completeness, requirements verification}
     - $\mathtt{A}_2$ = {writing complete requirements}
     - $\mathtt{D}$ = {size, complexity, volatility, relationship, safety, quality, time, cost, domain understanding}
     - $\mathtt{S}(\mathtt{t}_1) = 0.0002$
     - $\mathtt{S}(\mathtt{t}_2) = 0.0003$
     - $\mathtt{S}(\mathtt{t}_3) = 0.0059$

4. category: *Validation & Verification*

   - $\mathtt{T}_{validation}$= {prototyping, inspection, desk-checks, GQM, checklist}
   - $\mathtt{t}_1$ = prototyping, $\mathtt{t}_2$ = GQM
   - $\mathtt{I}(a)$ value is 1 for $\mathtt{A}_1$ and 0 for the remaining attributes:
     - $\mathtt{A}_1$={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)*

- $\mathtt{D} = \{$size, complexity, volatility, relationship, safety, quality, time, cost, domain knowledge$\}$
- $\mathtt{S}(\mathtt{t}_1) = 0.0013$
- $\mathtt{S}(\mathtt{t}_2) = 0.0001$

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.

TABLE 6.23. Use cases for CDO risk analysis application

Use case	Subtasks	Description
1. Load data		Read data from a .csv spreadsheet
2. Calculate Poisson approximation of loss function	2a. Calculate probability of no contagion	
	2b. Calculate probability of contagion	
	2c. Combine 2a, 2b	
3. Calculate precise loss function		
4. Write data		Write data to a .csv spreadsheet

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

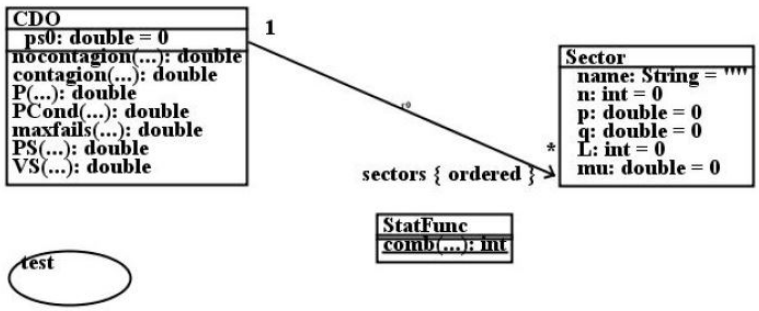


Figure 6.4. CDO version 1 system specifications

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].

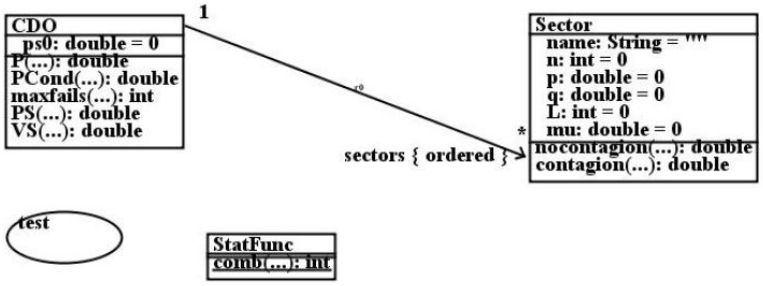


Figure 6.5. CDO version 2 system specifications

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.

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 $\mathtt{P}(\mathtt{S} = \mathtt{s})$ which has been decomposed into combinations of failures in individual companies. $\mathtt{P}(\mathtt{k}, \mathtt{m})$ is the probability of m defaults in sector k, $\mathtt{PCond}(\mathtt{k}, \mathtt{m})$ 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::
```

```
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 $\Sigma_{i=a^e}^{b}$.

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

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.

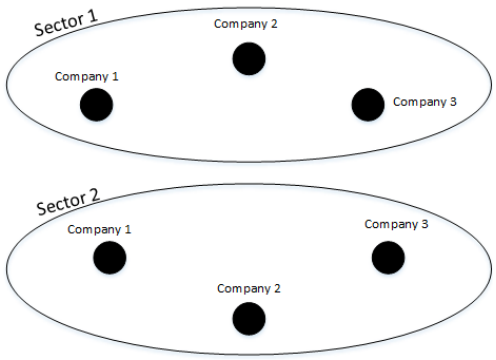


Figure 6.6. CDO example

The following results have been obtained by UML-RSDS which calculates the loss function:

```
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
```

```
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
```

We have shown the probability of loss for different amounts. For instance, the probability of ($\texttt{P} = \texttt{4}$) 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: $\texttt{P}(\texttt{loss} \geq \texttt{x}) = \texttt{1} - (\texttt{P}(\texttt{0}) + \texttt{P}(\texttt{1}) + \ldots + \texttt{P}(\texttt{i}))$ where $\texttt{i} = \texttt{x} - \texttt{1}$.

It was originally intended to use external hand-coded and optimised implementations of critical functions such as the combinatorial function $\texttt{comb}(\texttt{intn}, \texttt{intm})$. 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 $\texttt{P}(\texttt{S} = \texttt{s})$ for all values of $\texttt{s} \leq 20$ 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 ≪cached≫ 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

original manually-coded C++ version.

Figure 6.7 shows the refactored system specification at the end of the third development iteration (requirement F3).

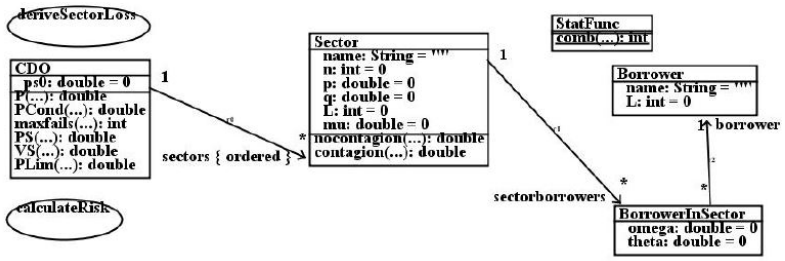


Figure 6.7. CDO version 3 system specifications

TABLE 6.24. Execution times for CDO versions

Version	**Execution time for first 20 P(S = s) calls**	**Execution time for first 50 P(S = s) calls**
Unoptimised Java	121s	- (more than 15 minutes)
Optimised Java	32ms	93ms
C#	10ms	20ms
C++	62ms	100ms
Original program	84s	- (more than 15 minutes)

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 6.25. CDO project comparison

	Size (LOC)	**Call graph size**	**Large clones**
Original program	194	11 (one cyclic dependency)	2
UML-RSDS	33	11 (two cyclic dependencies)	0

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.

TABLE 6.26. Framework evaluation form

RE Technique Framework Evaluation Guidelines	
Specific Questions	**Score (1-5)**
Is the program compatible with your computers and/or network?	5
Is technical assistance readily available via phone or email?	3
Is the level of language that the program offers clearly indicated?	4
Are the interface, navigation, and the directions clear and logical?	2
Does the program include scoring?	5
If a scoring system is used, does it make sense?	5
Can the learner easily quit something that is beyond his/her ability?	5
If the program includes pictures, are they (a) relevant, (b) an aid to understanding?	1 (N/A)
If the program includes sound recordings, are they of an adequate quality?	1 (N/A)
If the program includes video sequences, are they of an adequate quality?	1 (N/A)
Overall1 satisfaction?	4

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.

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-focussed) 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

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

functional and non-functional requirements for a transformation, rather they often only concentrate on implementing the main goal(s) of a transformation (e.g. refactoring or model to model migration). In fact, developers start the validation process after the transformation has been developed and they check the transformation to see which of the quality requirements are satisfied.

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

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

current status of RE in MT development and obtain more information regarding industrial and academic MT development. This would result in relying on the SLR survey and interview studies as the main source of information, rather than first-hand observations.

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.

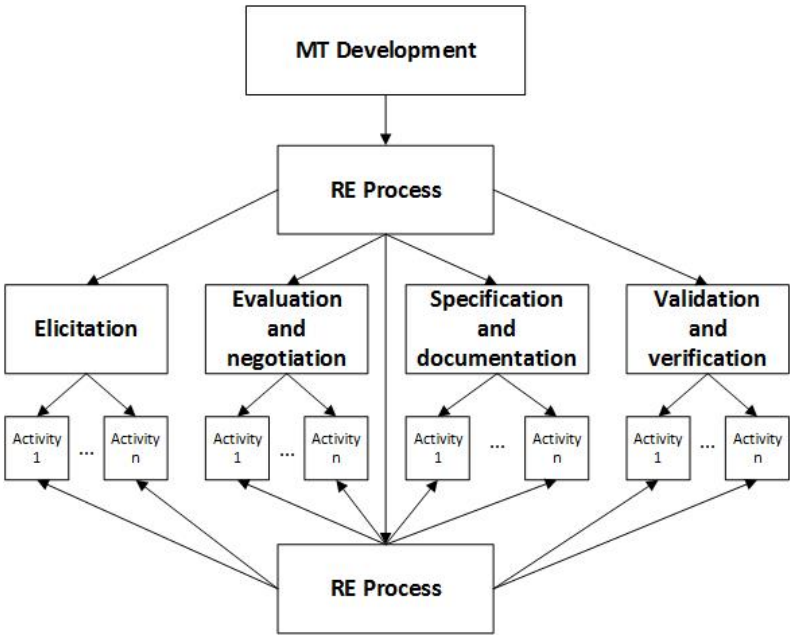


Figure 7.1. Requirements management process

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.

7.5.3 Integration with *trans*ML

The modelling language *trans*ML [45] represents requirements in the form of *SysML* diagrams. *trans*ML 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 *trans*ML 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

[1] Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J., et al. *Agile Software Development Methods: Review and Analysis*, vtt publications 478 ed. VTT Technical Research Centre of Finland, 2002.

[2] Albrecht, A. J. Function points help managers assess applications. *Computerworld, August 26* (1985).

[3] Alexander, I. F. A Taxonomy of Stakeholders: Human Roles in System Development. *International Journal of Technology and Human Interaction (IJTHI) 1*, 1 (2005), 23–59.

[4] Anastasakis, K., Bordbar, B., and Küster, J. M. Analysis of Model Transformations via Alloy. In *Proceedings of the 4th MoDeVVa workshop Model-Driven Engineering, Verification and Validation* (2007), pp. 47–56.

[5] Anton, A. I. *Goal Identification and Refinement in the Specification of Software-based Information Systems*. PhD thesis, Atlanta, GA, USA, 1997. UMI Order No. GAX97-35409.

[6] Arlow, J., and Neustadt, I. *UML 2 and the Unified Process: Practical Object-oriented Analysis and Design*. Pearson Education, 2005.

[7] Association, M. I. S. R., et al. *MISRA C 2012: Guidelines for the Use of the C Language in Critical Systems: March 2013*. Motor Industry Research Association, 2013.

[8] Back, R. J. Correctness Preserving Program Refinements: Proof Theory and Applications. *MC Tracts 131* (1980), 1–118.

[9] Baier, C., Katoen, J.-P., and Larsen, K. G. *Principles of Model Checking*. MIT press, 2008.

[10] Basili, V. R. Software modeling and measurement: the goal/question/metric paradigm. Tech. rep., 1992.

[11] Batot, E., Sahraoui, H., Syriani, E., Molins, P., and Sboui, W. Systematic mapping study of model transformations for concrete problems. In *Model-Driven Engineering and Software Development (MODELSWARD), 2016 4th International Conference on* (2016), IEEE, pp. 176–183.

[12] Baudry, B., Nebut, C., and Le Traon, Y. Model-driven Engineering for Requirements Analysis. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International* (2007), IEEE, pp. 459–459.

[13] Bell, T. E., and Thayer, T. Software Requirements: Are They Really a Problem? In *Proceedings of the 2nd international conference on Software engineering* (1976), IEEE Computer Society Press, pp. 61–68.

[14] Berry, D. M., Damian, D., Finkelstein, A., Gause, D., Hall, R., and Wassyng, A. To do or not to do: If the requirements engineering payoff is so good, why aren't more companies doing it? In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on* (2005), IEEE, pp. 447–447.

[15] Bezivin, J., and Gerbe, O. Towards a Precise Definition of the OMG/MDA Framework. In *Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on* (2001), pp. 273–280.

[16] Biehl, M. Literature Study on Model Transformations. *Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK* (2010).

[17] Boehm, B. W. Software engineering economics. *IEEE transactions on Software Engineering*, 1 (1984), 4–21.

[18] Boehm, B. W. A Spiral Model of Software Development and Enhancement. *Computer 21*, 5 (1988), 61–72.

[19] Boehm, B. W., Brown, J. R., and Lipow, M. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering* (1976), IEEE Computer Society Press, pp. 592–605.

[20] Booch, G. *The Unified Modeling Language User Guide*. Pearson Education India, 2005.

[21] Botella, P., Burgués, X., Carvallo, J., Franch, X., Grau, G., Marco, J., and Quer, C. ISO/IEC 9126 in Practice: What Do We Need to Know. In *Proceedings of the First Software Measurement European Forum (SMEF)* (2004).

[22] Brambilla, M., Cabot, J., and Wimmer, M. Model-driven Software Engineering in Practice. *Synthesis Lectures on Software Engineering 1*, 1 (2012), 1–182.

[23] Büttner, F., Egea, M., Guerra, E., and De Lara, J. Checking Model Transformation Refinement. In *Theory and Practice of Model Transformations*. Springer, 2013, pp. 158–173.

[24] Caldiera, V., and Rombach, H. D. The Goal Question Metric Approach. *Encyclopedia of software engineering 2*, 1994 (1994), 528–532.

[25] Chung, L., and do Prado Leite, J. C. S. On Non-functional Requirements in Software Engineering. In *Conceptual modeling: Foundations and applications*. Springer, 2009, pp. 363–379.

[26] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. The NFR Framework in Action. In *Non-Functional Requirements in Software Engineering*. Springer, 2000, pp. 15–45.

[27] Clancy, T. The Standish Group Report. *Chaos report retrieved Feb 20* (1995), 2008.

[28] Commission, I. O. F. S. E., et al. Software engineering–Product quality–Part 1: Quality model. *ISO/IEC 9126* (2001), 2001.

[29] Czarnecki, K., and Helsen, S. Feature-based survey of model transformation approaches. *IBM Systems Journal 45*, 3 (2006), 621–645.

[30] Davis, M., and Lo, V. Infectious Defaults. *Taylor & Francis Group* (2001).

[31] de Moura, L., and Bjørner, N. Z3–a tutorial, 2006.

[32] DeMarco, T. *Structured Analysis and System Specification*. Yourdon Press, 1979.

[33] DeMarco, T. *Controlling Software Projects: Management, Measurement, and Estimates*. Prentice Hall PTR, 1986.

[34] Dictionary, O. E. OED Online. *Oxford University Press. http://www. oed. com, Accessed Nov 30* (1989), 2006.

[35] Dromey, R. G. A model for software product quality. *IEEE Transactions on Software Engineering 21*, 2 (1995), 146–162.

[36] E., H., and P., D. Analyzing Projects to Decide How To Model the Requirements. T. C. U. JRCASE, Ed., Proceedings of The Fourth Australian Conference on RE, pp. 149–159.

[37] Ermel, C., Ehrig, H., and Ehrig, K. Refactoring of Model Transformations. *Electronic Communications of the EASST 18* (2009).

[38] Fagan, M. E. Advances in Software Inspections. In *Pioneers and Their Contributions to Software Engineering*. Springer, 2001, pp. 335–360.

[39] Fenton, N., and Bieman, J. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series. CRC Press, 2014.

[40] for Standardization, I. O., and Commission, I. E. *Software Engineering–Product Quality: Quality model*, vol. 1. ISO/IEC, 2001.

[41] Fowler, M., and Highsmith, J. The agile manifesto. *Software Development 9*, 8 (2001), 28–35.

[42] Friedenthal, S., Moore, A., and Steiner, R. *A Practical Guide to SysML: the Systems Modeling Language*. Morgan Kaufmann, 2014.

[43] Funk, M., Nyssen, E., and Lichter, H. From UML to ANSI-C-An Eclipse-Based Code Generation Framework. In *Proceedings of 3rd International Conference on Software and Data Technologies (ICSOFT)* (2008), Citeseer.

[44] Gilb, T. *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Butterworth-Heinemann, 2005.

[45] Guerra, E., De Lara, J., Kolovos, D. S., Paige, R. F., and dos Santos, O. M. transML: A family of Languages to Model Transformations. In *Model Driven Engineering Languages and Systems*. Springer, 2010, pp. 106–120.

[46] Guerra, E., Lara, J., Kolovos, D., Paige, R., and Santos, O. Engineering Model Transformations with transML. *Software & Systems Modeling 12*, 3 (2013), 555–577.

[47] Gunda, S. G. *Requirements Engineering : Elicitation Techniques.* PhD thesis, University West, Department of Economics and IT, 2008.

[48] Guoguen, J. Linden:Techniques for Requirement Elicitation. In *1st IEEE International Symposium on Requirements Engineering, San Diego, USA* (1993), pp. 4–6.

[49] Haight, F. A. Handbook of the Poisson Distribution.

[50] Halstead, M. H. *Elements of Software Science*, vol. 7. Elsevier New York, 1977.

[51] Hammarlid, O. Aggregating Sectors in the Infectious Defaults Model. *Quantitative Finance* (2003).

[52] Hastie, S., and Wojewoda, S. Standish Group 2015 Chaos Report-Q&A with Jennifer Lynch. *Retrieved 1*, 15 (2015), 2016.

[53] Herrmannsdoerfer, M., Benz, S., and Juergens, E. Cope-automating coupled evolution of metamodels and models. In *European Conference on Object-Oriented Programming* (2009), Springer, pp. 52–76.

[54] Hickey, A. M., and Davis, A. M. Requirements Elicitation and Elicitation Technique Selection: Model for Two Knowledge-intensive Software Development Processes. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on* (2003), IEEE, pp. 10–pp.

[55] Hickey, A. M., and Davis, A. M. A unified model of requirements elicitation. *Journal of Management Information Systems 20*, 4 (2004), 65–84.

[56] Hnatkowska, B., and Mazurek, P. Verification of UML Class Diagrams against Business Rules Written in Natural Language. In

Theory and Engineering of Complex Systems and Dependability. Springer, 2015, pp. 175–184.

[57] Holzmann, G. J. *The SPIN Model Checker: Primer and Reference Manual*, vol. 1003. Addison-Wesley Reading, 2004.

[58] Horowitz, E., and Boehm, B. W. *Practical Strategies for Developing Large Software Systems*. Addison-Wesley Reading, Ma, 1975.

[59] Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. Empirical Assessment of MDE in Industry. In *Proceedings of the 33rd International Conference on Software Engineering* (2011), ACM, pp. 471–480.

[60] Jarke, M. Scenarios for Modeling. *Communications of the ACM 42*, 1 (1999), 47–48.

[61] Jiang, L. *A Framework for the Requirements Engineering Process Development*. University of Calgary, 2005.

[62] Jiang, L., Eberlein, A., Far, B. H., and Mousavi, M. A Methodology for the Selection of Requirements Engineering Techniques. *Software & Systems Modeling 7*, 3 (2008), 303–328.

[63] Jones, C. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[64] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. ATL: A model Transformation Tool. *Science of Computer Programming 72*, 1 (2008), 31–39.

[65] Kausar, S., Tariq, S., Riaz, S., and Khanum, A. Guidelines for the Selection of Elicitation Techniques. In *Emerging Technologies (ICET), 2010 6th International Conference on* (2010), IEEE, pp. 265–269.

[66] Kavakli, E., and Loucopoulos, P. Goal Modeling in Requirements Engineering: Analysis and Critique. *Information Modeling Methods and Methodologies: Advanced Topics in Database Research: Advanced Topics in Database Research 102* (2004).

[67] Kempa, M., and Mann, Z. A. Model driven architecture. *Informatik-Spektrum 28*, 4 (2005), 298–302.

[68] Kent, S. Model Driven Engineering. In *Integrated Formal Methods*, M. Butler, L. Petre, and K. Sere, Eds., vol. 2335 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 286–298.

[69] Kernighan, B. W., and Ritchie, D. M. *The C Programming Language*. 2006.

[70] Kevin Lano, Sobhan Yassipour Tehrani, S. K.-R. The use of model transformation design patterns in practice. *Journal of Systems and Software* (2017).

[71] Kieras, D. GOMS Models for Task Analysis. *The handbook of task analysis for human-computer interaction* (2003), 83–116.

[72] Kitchenham, B. Procedures for Performing Systematic Reviews. *Keele, UK, Keele University 33*, 2004 (2004), 1–26.

[73] Kleppe, A. G., Warmer, J. B., and Bast, W. *MDA Explained: the Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional, 2003.

[74] Kolahdouz Rahimi, S. *A Comparative Study of Model Transformation Approaches through a Systematic Procedural Framework and Goal Question Metrics Paradigm*. PhD thesis, King's College London (University of London), 2013.

[75] Kolahdouz-Rahimi, S., Lano, K., Pillay, S., Troya, J., and Van Gorp, P. Evaluation of Model Transformation Ap-

proaches for Model Refactoring. *Science of Computer Programming 85* (2014), 5–40.

[76] Kolovos, D., Paige, R., and Polack, F. The Epsilon Transformation Language. In *Theory and Practice of Model Transformations*, A. Vallecillo, J. Gray, and A. Pierantonio, Eds., vol. 5063 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 46–60.

[77] Kolovos, D., Rose, L., Paige, R., and Garcia-Dominguez, A. The Epsilon Book. *Structure 178* (2010).

[78] Kolovos, D. S., Paige, R. F., and Polack, F. A. The Epsilon Object Language (EOL). In *Model Driven Architecture–Foundations and Applications* (2006), Springer, pp. 128–142.

[79] Kotonya, G., and Sommerville, I. RE, Processes and Techniques. *John Wiley and Sons Ltd., New York, ISBN 13* (1998), 9780471972082.

[80] Kotoyna, G., Sommerville, I., Wiley, J., et al. Requirements Engineering: Processes and Techniques, 1999.

[81] Kunz, W., and Rittel, H. W. *Issues as Elements of Information Systems*, vol. 131. Institute of Urban and Regional Development, University of California Berkeley, California, 1970.

[82] Landes, D., and Studer, R. *The Treatment of Non-functional Requirements in MIKE*. Springer, 1995.

[83] Lano, K. *Advanced Systems Design with Java, UML and MDA*. Elsevier, 2005.

[84] Lano, K. Object-oriented Specification and Design. University Lecture, 2010.

[85] Lano, K. The UML-RSDS Manual, 2014.

[86] Lano, K. Agile Model-Based Development Using UML-RSDS, 2016.

[87] Lano, K. *UML-Reactive Systems Design Support*. KCL, 2016.

[88] Lano, K., Clark, T., and Kolahdouz-Rahimi, S. A Framework for Model Transformation Verification. *Formal Aspects of Computing 27*, 1 (2015), 193–235.

[89] Lano, K., and Kolahdouz-Rahimi, S. Model Migration Transformation Specification in UML-RSDS. In *In TTC10: Transformation Tool Contest, 2010. and Model Transformation Tools for Model Migration 33* (2010), Citeseer.

[90] Lano, K., Kolahdouz-Rahimi, S., and Clark, T. Comparing Verification Techniques for Model Transformations. In *Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation* (2012), ACM, pp. 23–28.

[91] Lano, K., and Rahimi, S. K. Case study: Class Diagram Restructuring. *arXiv preprint arXiv:1309.0369* (2013).

[92] Lano, K., and Rahimi, S. K. Case study: Class Diagram Restructuring. In *Proceedings Sixth Transformation Tool Contest, TTC 2013, Budapest, Hungary, 19-20 June, 2013.* (2013), pp. 8–15.

[93] Lano, K., and Yassipour Tehrani, S. *Improving the Application of Agile Model-based Development: Experiences from Case Studies*. 11 2015.

[94] Laplante, P. A. *Requirements Engineering for Software and Systems*. CRC Press, 2013.

[95] Lapouchnian, A. Goal-oriented requirements engineering: An overview of the current research. *University of Toronto* (2005).

[96] Leite, J. D. P., and Gilvaz, A. P. P. Requirements Elicitation Driven by Interviews: The Use of Viewpoints. In *Software Specification and Design, 1996., Proceedings of the 8th International Workshop on* (1996), IEEE, pp. 85–94.

[97] Letier, E. Fundamentals of Requirements Engineering. University Lecture, 2013.

[98] Letier, E. Project initiation. University Lecture, 2013.

[99] Loniewski, G., Insfran, E., and Abrahão, S. A Systematic Review of the Use of Requirements Engineering Techniques in Model-driven Development. In *International Conference on Model Driven Engineering Languages and Systems* (2010), Springer, pp. 213–227.

[100] Macaulay, L. Requirements for Requirements Engineering Techniques. In *Requirements Engineering, 1996., Proceedings of the Second International Conference on* (1996), IEEE, pp. 157–164.

[101] Maiden, N., and Rugg, G. ACRE: Selecting Methods for Requirements Acquisition. *Software Engineering Journal 11*, 3 (1996), 183–192.

[102] Marinelli, V. *An Analysis of Current Trends in Requirements Engineering Practice*. Pennsylvania State University, Great Valley, 2008.

[103] Mashiko, Y., and Basili, V. R. Using the gqm paradigm to investigate influential factors for software process improvement. *Journal of Systems and Software 36*, 1 (1997), 17–32.

[104] McCabe, T. J. A complexity measure. *IEEE Transactions on software Engineering*, 4 (1976), 308–320.

[105] McCabe, T. J., and Butler, C. W. Design complexity measurement and testing. *Communications of the ACM 32*, 12 (1989), 1415–1425.

[106] McCall, J. A., Richards, P. K., and Walters, G. F. Factors in software quality. volume i. concepts and definitions of software quality. Tech. rep., General Electric CO SunnyVale CA, 1977.

[107] McPhee, C. *Requirements Engineering for Projects with Critical Time-to-market*. University of Calgary, 2001.

[108] Mens, T., and Van Gorp, P. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science 152* (2006), 125–142.

[109] Michie, D. Memo Functions and Machine Learning. *Nature 218*, 5136 (1968), 19–22.

[110] Mohagheghi, P., Gilani, W., Stefanescu, A., and Fernandez, M. A. An Empirical Study of the State of the Practice and Acceptance of Model-driven Engineering in Four Industrial Cases. *Empirical Software Engineering 18*, 1 (2013), 89–116.

[111] Nakićenović, M. B. An Agile Driven Architecture Modernization to a Model-Driven Development Solution. *International Journal on Advances in Software Volume 5, Number 3 & 4, 2012* (2012).

[112] Neill, C. J., and Laplante, P. A. Requirements engineering: the state of the practice. *IEEE software 20*, 6 (2003), 40–45.

[113] Nuseibeh, B., and Easterbrook, S. Requirements Engineering: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (New York, NY, USA, 2000), ICSE '00, ACM, pp. 35–46.

[114] of the IEEE Computer Society, S. E. S. C. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998* (Oct 1998), 1–40.

[115] Omg, Q. Meta Object Facility (mof) 2.0 Query/View/Transformation Specification. *Final Adopted Specification (November 2005)* (2008).

[116] Pastor, O., España, S., Panach, J. I., and Aquino, N. Model-driven Development. *Informatik-Spektrum 31*, 5 (2008), 394–407.

[117] Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., and Reijers, H. A. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In *Business Process Management Workshops* (2012), Springer, pp. 383–394.

[118] Pinto, J. K., and Mantel, S. J. The Causes of Project Failure. *IEEE transactions on engineering management 37*, 4 (1990), 269–276.

[119] Pohl, K. *Requirements Engineering: An Overview*. Aachener Informatik-Berichte. RWTH, Fachgruppe Informatik, 1996.

[120] Rensink, A. The groove simulator: A tool for state space generation. In *AGTIVE* (2003), vol. 3062, Springer, pp. 479–485.

[121] Rensink, A., Schmidt, Á., and Varró, D. Model Checking Graph Transformations: A Comparison of Two Approaches. In *International Conference on Graph Transformation* (2004), Springer, pp. 226–241.

[122] Robertson, S., and Robertson, J. *Mastering the Requirements Process (2Nd Edition)*. Addison-Wesley Professional, 2006.

[123] Rose, L. M., Kolovos, D. S., Paige, R. F., and Polack, F. Model Migration Case for TTC 2010. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain* (2010), 1.

[124] Rose, L. M., Kolovos, D. S., Paige, R. F., Polack, F. A., and Poulding, S. Epsilon flock: a model migration language. *Software & Systems Modeling 13*, 2 (2014), 735–755.

[125] Rose, L. M., Paige, R. F., Kolovos, D. S., and Polack, F. The epsilon generation language. *ECMDA-FA 8* (2008), 1–16.

[126] Rozier, K. Y. Linear Temporal Logic Symbolic Model Checking. *Computer Science Review 5*, 2 (2011), 163–203.

[127] Sargent, R. G. Verification and validation of simulation models. In *Proceedings of the 37th conference on Winter simulation* (2005), Winter Simulation Conference, pp. 130–143.

[128] Selic, B. The Pragmatics of Model-driven Development. *IEEE software 20*, 5 (2003), 19–25.

[129] Selic, B. What will it take? A view on adoption of model-based methods in practice. *Software & Systems Modeling 11*, 4 (2012), 513–526.

[130] Selim, G. M., Wang, S., Cordy, J. R., and Dingel, J. Model Transformations for migrating Legacy Deployment Models in the Automotive Industry. *Software & Systems Modeling 14*, 1 (2015), 365–381.

[131] Sendall, S., and Kozaczynski, W. Model Transformation the Heart and Soul of Model-driven Software Development. Tech. rep., Swiss Federal Institute of Technology in Lausanne (EPFL), 2003.

[132] Soley, R., et al. Model Driven Architecture. *OMG white paper 308*, 308 (2000), 5.

[133] Sommerville, I., and Kotonya, G. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., 1998.

[134] Sommerville, P. I. private communication by email, July 2015.

[135] Stevens, P. A Landscape of Bidirectional Model Transformations. In *Generative and transformational techniques in software engineering II*. Springer, 2008, pp. 408–424.

[136] Sutcliffe, A. Scenario-based Requirements Engineering. In *Proceedings 11th IEEE International Requirements Engineering Conference, 2003.* (Sept 2003), pp. 320–329.

[137] Taskforce, U. R. OMG UML Specification v. 1.4. *Object Management Group* (2001).

[138] Team, S., et al. Semantics of Business Vocabulary and Rules (SBVR). Tech. rep., Technical Report dtc/06–03–02, Object Management Group, Needham, Massachusetts, 2006.

[139] UML, O. Version 2.2. OMG Specification Superstructure and Infrastructure, 2009.

[140] Van Amstel, M. The Right Tool for the Right Job: Assessing Model Transformation Quality. In *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual* (2010), IEEE, pp. 69–74.

[141] Van Deursen, A., Klint, P., and Visser, J. Domain-specific Languages. *Centrum voor Wiskunde en Informatika 5* (2000).

[142] van Lamsweerde, A. Requirements Engineering: from Craft to Discipline. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (2008), ACM, pp. 238–249.

[143] van Lamsweerde, A. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley & Sons, 9 Jan 2009.

[144] Van Lamsweerde, A., and Letier, E. Handling Obstacles in Goal-oriented Requirements Engineering. *Software Engineering, IEEE Transactions on 26*, 10 (2000), 978–1005.

[145] Weidenhaupt, K., Pohl, K., Jarke, M., and Haumer, P. Scenarios in System Development: Current Practice. *Software, IEEE 15*, 2 (1998), 34–45.

[146] Wetherill, G. B. *Intermediate Statistical Methods*. Springer Science & Business Media, 2012.

[147] Whittle, J., Hutchinson, J., and Rouncefield, M. The State of Practice in Model-driven Engineering. *Software, IEEE 31*, 3 (2014), 79–85.

[148] Yang, W., Horwitz, S., and Reps, T. A Program Integration Algorithm that Accommodates Semantics-preserving Transformations. *ACM Transactions on Software Engineering and Methodology (TOSEM) 1*, 3 (1992), 310–354.

[149] Yassipour Tehrani, S., and Lano, K. Precise Requirements Engineering for Model. In *Software Technologies: Applications and Foundations (STAF)* (2014).

[150] Yassipour Tehrani, S., and Lano, K. The Significant role of Requirement Engineering in Model Transformation. In *In International Conference on New Trends in Information and Communication Technologies* (2014).

[151] Yassipour Tehrani, S., and Lano, K. Model Transformation Applications from Requirements Engineering Perspective. In *The 10th International Conference on Software Engineering Advances* (2015).

[152] Yassipour Tehrani, S., and Lano, K. Temporal Logic Specification and Analysis for Model Transformations. In *Verification of Model Transformations, VOLT 2015* (2015).

[153] Yassipour Tehrani, S., and Lano, K. Requirements Engineering in Model Transformation Development: A Technique Suitability Framework for Model Transformation Applications. *International Journal On Advances in Software* (2016).

[154] Yassipour Tehrani, S., Zschaler, S., and Lano, K. Requirements Engineering in Model-Transformation Development:

An Interview-Based Study. In *International Conference on Theory and Practice of Model Transformations* (2016), Springer, pp. 123–137.

[155] Yourdon, E. Death March: The Complete Software Developers Guide to Surviving Mission Impossible Projects. Paul, DB, 1997.

[156] Yue, T., Briand, L. C., and Labiche, Y. A Systematic Review of Transformation Approaches between User Requirements and Analysis Models. *Requirements Engineering 16*, 2 (2011), 75–99.

[157] Zave, P. Classification of Research Efforts in Requirements Engineering. *ACM Comput. Surv. 29*, 4 (Dec. 1997), 315–321.

[158] Zowghi, D., and Coulin, C. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*. Springer, 2005, pp. 19–46.

Appendix A

SLR Tables

TABLE A.1. Stakeholder information (1)

Case #	How to find what was required	Stakeholders	Reaching stakeholders
1-11	Problem description, prototype and examples	Financial companies, case study proposers	Forum, email
12-20	Problem description, prototype and examples	Users requiring analysis of data	Forum, email
21	Research	ArchStudio users	None
22	Problem description, prototype and examples	MDE practitioners	None
23	Document mining	Software modellers/developers	None
24, 30, 124, 142, 157-159	Document mining	MDE practitioners	None
25	Interviews, document mining	Automotive system (VCS) developers, General Motors	Consultation with domain experts
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	Research	MDE practitioners	None
27	Brainstorming, customer as developer	End users of FrontArena software, FrontArena software teams, SunGard management	Constant feedback, participation in development
28	Problem statement	Software developers (users of migrated models)	None
29	Document mining	RTES developers, testers companies with RTES products	None
31-57	Problem description, prototype and examples	MDE practitioners	Forum, email
58	Document mining	Business process modeling users	None
59	Document mining	Risk analysts	Industry collaboration

TABLE A.2. Stakeholder information (2)

Case #	How to find what was required	Stakeholders	Reaching stakeholders
60	Document mining	Alloy Analyzer users	None
61	General problem description	Graph transformation users	None
62	General problem description	Graphics and JavaScript application users	None
63	General problem description	MDE practitioners	None
64	General problem description	Use case model users	None
65	Document mining	C++ users (legacy code)	None
66	Research	Manufacturing system designers	None
67	Research	System migration engineers	None
68	Document mining	OCL users	None
69	Communication with domain experts	Volvo Cars Group	Consultations, demonstrations
73, 77	Research	Web application developers	None
74, 79	Research	Control engineers	None
75	Research	MT developers	None
81-94	Problem statement	MDE practitioners	Forum, email
98	Research	Process control practitioners	Discussions with experts
101	Research	Game application developers	None
102	Research	MDE practitioners, web application developers	None
107	Research	MDE practitioners, critical system developers	None
109	Business modelling	MDE practitioners	None
112	Research	Users exchanging web rules	None
114	Research, business analysis	Re-engineering projects	Via consortium with industrial users
120	Industry collaboration	Embedded sys. developers	Consultation
122. 123	Research, document mining	MDE practitioners	Experimenting with practitioners

TABLE A.3. Stakeholder information (3)

Case #	How to find what was required	Stakeholders	Reaching stakeholders
131	Research	WSN developers	None
134	Research	Software developers	None
136	Document mining	Activity diagram modellers	None
137	Document mining	Quantity surveyors	None
140	Research	Surveillence sys. developers	Participation in development
141	Document mining, consultation with domain experts	Embedded system developers	Collaboration with practitioners
143	Research	UI developers	None
145	Document mining	DES analysts	None
147	Research	Businesses requiring software modernisation	None
148	Research	Workflow analysts	None
151	Document mining	Satellite operators	Industrial collaboration
152	Document mining	Digital forensic experts	User collaboration
156	Research	MDE practitioners	Industrial collaboration
161	Problem statement	MDE practitioners	Consultation
162	Research	SOA developers	None

TABLE A.4. MT requirements

Category	Requirement	Cases
Local Functional	Local mapping requirements	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
	Local refactoring requirements	23, 35, 36, 40, 41, 53, 57, 64, 70, 71, 135
	Local reactive requirements	26, 46, 47, 51, 56, 76, 145, 155
	Local correspondence requirements	32, 59, 154
Global Functional	Syntactic correctness	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
	Completeness	3, 29, 38, 47, 55, 58, 61, 72, 119, 151
	Semantic correctness	14, 24, 25, 28, 29, 31, 44, 47, 50, 52, 53, 55, 57-69, 72-75, 79, 80, 84, 86-94, 105-107
	Semantic preservation	96, 101, 102, 105-107, 112-114, 119, 120, 124, 127, 128, 132, 133, 135, 136, 139, 142, 146, 148, 149, 156-159
	Confluence	36, 37, 72
	Bidirectionality	24, 32, 43
	Model synchronisation/ Change-propagation	21, 42, 45, 58, 22, 121, 138, 154
	Traceability	111, 121, 137
	Invariance	40
	Accuracy	59, 75, 98, 137, 162
	Structural preservation	157, 158
Non-Functional	Efficiency	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
	Simplicity/Clarity	1, 28, 44, 47, 58, 81-86
	Fault tolerance	1, 5, 8, 9, 10, 11, 129
	Modularity	1, 7, 9, 10, 11, 35, 63, 95
	Size/Conciseness	6, 28, 34, 46, 81-86
	Scalability	12, 13, 15-19, 25, 27, 29, 147, 151
	Timing	26, 69, 76, 145, 155
	Process Responsiveness	27
	Reduced development effort	100, 110, 111
	Architectural quality	27, 95, 102
	Flexibility	30, 78, 97, 103, 108, 129, 137, 140-142, 152
	Portability	99
	Usability	97
	Conformance to standards	30
	Alignment to business view	109
	Genericity	92-94

TABLE A.5. MT project information (1)

Case #	Type of transformation	Size/Scale	Green/ Brownfield	Academic/ Industry
1-11	Text-to-model, migration, code generation	Small	Green	Academic
12-20	Refinement, data analysis	Medium	Green	Academic
21, 22	Bidirectional	Medium	Green	Academic
23, 135, 36-38, 71	Refactoring	Small	Green	Academic
24	Bidirectional	Large	Green	Academic
25, 58, 60, 114	Migration	Large	Green	Industry
26, 46-48, 51, 56, 76	Reactive	Small	Green	Academic
27, 155	Refinement, migration	Large	Green	Industry
28, 78	Migration	Medium	Green	Academic
29	Refinement, code generation	Large	Green	Industry
30	Refinement	Large	Green	Industry
31, 32, 34, 35, 39, 147	Text/code-to-model	Small	Green	Academic
33, 40-42, 70	Refactoring	Medium	Green	Academic
43, 45, 50, 53, 55, 57	Java refactoring	Small	Green	Academic
44, 49, 52, 54	Model execution	Small	Green	Academic
59, 141	Code generation	Medium	Green	Industry
61	Semantic mapping	Large	Green	Industry
62	Migration/refinement	Medium	Green	Industry
63, 69	Code generation	Large	Green	Industry
64	Refactoring	Medium	Green	Industry
65, 151	Code-to-code	Large	Green	Industry
66	Migration	Medium	Green	Industry
67, 81-88, 99, 106, 112, 118, 132, 146	Migration	Small	Green	Academic
68, 72, 107, 139	Semantic mapping	Medium	Green	Academic
73, 77, 80	Refinement	Medium	Green	Academic
74, 79, 92-94, 103-105, 109-111, 122-125, 128, 130, 134, 137, 150, 157, 158, 160	Refinement	Small	Green	Academic

TABLE A.6. MT project information (2)

Case #	Type of transformation	Size/Scale	Green/ Brownfield	Academic/ Industry
75	Abstraction	Small	Green	Academic
89-91, 96, 119, 127, 129, 133, 136, 148	Semantic mapping	Small	Green	Academic
95, 162	Refinement, code generation	Small	Green	Academic
97, 100, 101, 116, 117, 126, 131, 138, 143, 153	Code generation	Small	Green	Academic
98, 108, 140	Code generation	Medium	Green	Academic
102	Migration, refactoring	Small	Green	Academic
113	Reverse engineering	Medium	Green	Academic
120, 161	Refinement	Medium	Green	Industry
121, 154	Bidirectional	Small	Green	Academic
142	Migration	Small	Green	Industry
145	Reactive	Medium	Green	Academic
149	Refinement	Small	Brown	Academic
152	Code generation	Small	Green	Industry
156	Abstraction	Medium	Green	Academic
159	Refinement, refactoring	Small	Green	Academic

TABLE A.7. Methodology information (1)

Case #	Methodology	Diagrams	RE technique/process
1-11, 33, 34, 47, 68, 70, 71, 76-78, 81, 93, 102, 108, 110, 112, 113, 131, 134, 137, 138, 146	None	UML class diagram	None
12-20	None	UML class diagram	Exploratory prototyping and scenario analysis
21	MDE for MT	UML class diagram	None identified. Some concrete grammar diagrams used to express synchronisation rules
22	Incremental development, evolutionary prototyping	UML class diagram	Experimental prototyping and scenario-based analysis, concrete grammar diagrams
23	None	UML class diagram	Not explicit; scenario-based experimental prototyping
24	Incremental, iterative	UML class diagram	No explicit RE process, concrete grammar of the two languages used to express correspondence rules
25	Incremental, iterative development using MD-Workbench, validation and testing	UML, class diagram	Structured interviews, scenarios, detailed RE process
26	Incremental, experimentation	UML class diagram	Concrete grammar rules and scenario analysis are used for RE
27	Agile MDE	None	Survey of developers, reverse engineering of legacy code
28	MDE	UML class diagram	Scenario analysis, exploratory prototyping, observation, interviews, concrete grammar used to express mapping examples
29	Design patterns, verification through testing and inspection	UML class diagram	Observation (implicit), individual expert knowledge, literature analysis
30	Incremental prototyping	UML class diagram, state machines	Scenario analysis, experimental prototyping, rules expressed using concrete grammar
31	None	UML class diagram	Experimental prototyping
32, 36, 38, 46	Incremental, iterative	UML class diagram	None
35	None	UML class diagram	Scenario analysis, informal mappings
37	Incremental, iterative	UML class diagram	Survey and questionnaire, prioritization, empirical studies with representative users and tasks

TABLE A.8. Methodology information (2)

Case #	Methodology	Diagrams	RE technique/process
39, 50, 57, 58	None	UML class diagram	Prioritization
40	Design patterns, verification through testing and inspection	UML class diagram	Scenario analysis
41, 42, 51	None	None	Prioritization
43	None	None	Exploratory prototyping and scenario analysis
44, 45, 52, 53, 55, 56, 74, 82, 86, 87, 126, 132	None	None	None
48	Incremental, iterative	UML class diagram	Validation scenarios
49	Incremental, iterative	Object diagrams	None
54	None	Concrete syntax	Prioritization
59	MDE for MT	UML class diagram,OCL	Scenario analysis
124, 127, 130, 162	None	UML class diagram	Scenario analysis
60	MDA for MT	OCL, UML class diagram	Scenario analysis
61	None	UML class diagram	Scenarios, OCL, UML, Constructive Query Containment (CQC) method
62	None	Feature diagrams	Scenario analysis
63	Refactoring	OCL, UML class diagram	Scenario analysis
64	Use case-driven approach	OCL, UML class diagram	Scenario analysis
65	None	None	Scenario analysis
66	None	Concrete grammar	Scenario analysis
67	MDA for MT	Concrete grammar	None
69	Agile MDE	Simulink models	prototyping
72	None	UML activity diagrams	None
73	None	BPMN diagrams	None
75	None	Specialised	None
79	None	Control block diagrams	None

TABLE A.9. Methodology information (3)

Case #	Methodology	Diagrams	RE technique/process
80	MDE	UML class diagrams	None
83, 95, 129	None	UML class, activity diagrams	Scenario analysis
84	None	UML class, activity diagrams	None
85, 88	None	UML activity diagrams	None
89, 90	None	Graph diagrams	None
91, 101	None	UML class, statemachine diagrams	None
92, 155	None	Customised	None
94, 96, 109	None	UML class diagrams	Informal specifications
97	Iterative development	UI task models, statecharts	Prototyping
98	Iterative development	Process models	Prototyping
99, 160	MDE for MT	UML class diagrams	None
100	None	UML class diagrams, architecture diagrams	None
103	Derive MTs from logical specifications	Architecture diagrams	None
104	MDA	UML class, statemachine diagrams	Informal mappings
105	Prototyping	UML class, statemachine diagrams	Informal specifications
106	Formal methods	Object diagrams/graphs	None
107	Formal methods	None	None
111	None	UML activity, sequence diagrams	None
114	MT embedded in re-engineering process	UML class diagrams, activity diagrams	Domain analysis
117	None	UML class diagrams, concrete grammar	None
118, 119	None	UML class diagrams, concrete grammar	Formal specifications
120, 121, 128, 136	None	UML class diagrams, concrete grammar	Concrete grammar rule specifications
122	Specification by example	UML class diagrams	Concrete grammar rule specifications
123	None	UML class diagrams	Semi-formal rule specifications, experimental evaluation
125	MDE for MT	UML class diagrams	Scenarios, formal rule, specifications, partial RE process

TABLE A.10. Methodology information (4)

Case #	Methodology	Diagrams	RE technique/process
133	None	None	Formal rule specifications
135	None	UML class diagrams, sequence diagrams	Formal rule specifications
139, 145, 150, 152	None	UML class diagrams	Formal rule specifications
140	Iterative, incremental	UML class diagrams, feature models	Stakeholder participation
141	Iterative, incremental	UML class diagrams	Stakeholder identification, stakeholder collaboration
143	None	UML class diagrams, use case diagrams	None
147	Validation using industrial cases	UML class diagrams, concrete syntax diagrams,	Concrete syntax mapping rules
148	Validation using industrial cases	Concrete syntax diagrams	Concrete syntax mapping rules
149	MDA	UML class diagrams	Formal rule specifications
151	Phased	UML class diagrams	Graphical mapping specifications
154	Formal methods	Concrete grammar diagrams	None
156	MDE	UML class diagrams	Semi-formal specification
157	Formal methods	UML class diagrams, concrete syntax diagrams	None
158	Formal methods	Concrete syntax diagrams	Formal specification
159	Formal methods	UML class diagrams	Formal specification
161	Controlled experiments	UML use case, statechart diagrams	Expert validation, goal decomposition, textual specifications

TABLE A.11. SLR case outcomes (1)

Cases	**Requirements expression, techniques & process**	**Requirements achievement**	**Stakeholder satisfaction**
1	4	high	high
2, 32, 93	1	low	unknown
3, 43, 50, 56, 57, 84, 85, 92	1	med	unknown
5	3	high	unknown
6, 7, 9, 33, 36, 47 48, 62, 83, 86 89, 91, 31, 39 158, 74, 161, 108	2	med	unknown
8, 23, 54, 77, 90 94, 95, 130, 131 133, 134, 137, 138 143, 150, 154, 156 100-103, 110, 111 113, 116, 121	3	med	unknown
10, 34, 38, 41 63, 75	4	high	med
11, 19, 44, 49 60, 80, 81, 88	3	med	med
12	4	low	med
13	2	low	med
14, 16	3	low	med
15, 51-53	2	high	med
17	3	high	med
18, 42	2	low	low
20, 55, 82	2	med	med
21, 26, 76, 155 120, 157, 159	4	high	unknown
24, 79, 97, 122 126, 127, 129, 132 135, 139, 141, 146 153, 106, 109, 112 117, 118	4	med	unknown
25, 27, 152	7	high	high
28, 119	5	med	unknown
125, 145, 147, 160 105, 107, 114	5	high	unknown
29	7	med	high

TABLE A.12. SLR case outcomes (2)

Cases	**Requirements expression, techniques & process**	**Requirements achievement**	**Stakeholder satisfaction**
35, 61, 66	4	med	med
37, 98, 99, 151	5	med	med
40	3	med	low
45	1	med	low
46	1	med	med
58	2	unknown	unknown
64, 78, 96	5	high	med
65	5	high	high
69	7	high	med
87	0	low	unknown
123	8	high	medium
128	6	medium	medium
136, 140, 162	3	low	unknown
148	6	high	medium
142, 149, 124, 104	6	high	unknown

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 [47]. 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

[1] Tassilo Horn. Solving the TTC FIXML Case with FunnyQT. In *TCC@ STAF*, pages 7–21. Citeseer, 2014.

[2] Christoph Eickhoff, Tobias George, Stefan Lindel, and Albert Zündorf. The SDMLib Solution to the MovieDB Case for TTC2014. In *TCC@ STAF*, pages 145–149, 2014.

[3] Dan Li, Danning Li, Xiaoshan Li, and Volker Stolz. FIXML to Java, C# and C++ Transformations with QVTR-XSLT. *TTC 2014*, page 27, 2014.

[4] K Lano, S Yassipour-Tehrani, and K Maroukian. Case study: FIXML to Java, C# and C+. *TTC 2014*, page 2, 2014.

[5] Frank Hermann, Nico Nachtigall, Benjamin Braatz, Thomas Engel, and Susann Gottmann. Solving the FIXML2Code-case Study with HenshinTGG. In *TCC@ STAF*, pages 32–46. Citeseer, 2014.

[6] Pablo Inostroza and Tijs van der Storm. The TTC 2014 FIXML Case: Rascal Solution. In *TCC@ STAF*, pages 47–51. Citeseer, 2014.

[7] Steffen Zschaler and Sobhan Yassipour Tehrani. Mapping FIXML to OO with Aspectual Code Generators. *TTC 2014*, page 52, 2014.

[8] Vahdat Abdelzad, Hamoud I Aljamaan, Opeyemi Adesina, Miguel Garzón, and Timothy Lethbridge. A Model-Driven Solution for Financial Data Representation Expressed in FIXML. In *TCC@ STAF*, pages 65–70, 2014.

[9] Géza Kulcsár, Erhan Leblebici, and Anthony Anjorin. A Solution to the FIXML Case Study Using Triple Graph Grammars and eMoflon. In *TCC@ STAF*, pages 71–75, 2014.

[10] Filip Krikava and Philippe Collet. Solving the TTC'14 FIXML Case Study with SIGMA. In *Proceedings of the 7th Transformation Tool Contest part of the Software Technologies: Applications and Foundations (STAF 2014) federation of conferences*, 2014.

[11] Horacio Hoyos, Jaime Chavarriaga, and Paola Gómez. Solving the FIXML Case Study Using Epsilon and Java. In *TCC@ STAF*, pages 87–92. Citeseer, 2014.

[12] Christopher Gerking and Christian Heinzemann. Solving the Movie Database Case with QVTo. In *TCC@ STAF*, pages 98–102. Citeseer, 2014.

[13] Gábor Szárnyas Oszkár Semeráth Benedek Izsó, Csaba Debreceni, and Ábel Hegedüs Zoltán Ujhelyi Gábor Bergmann. Movie Database Case: An EMF-INCQUERY Solution. *TTC 2014*, page 103, 2014.

[14] Antonio Moreno-Delgado and Francisco Durán. The Movie Database Case: Solutions using Maude and the Maude-based e-Motions tool. *TTC 2014*, page 116, 2014.

[15] Edgar Jakumeit. Solving the TTC 2014 Movie Database Case with GrGen. NET. In *TCC@ STAF*, pages 125–133. Citeseer, 2014.

[16] Hüseyin Ergin and Eugene Syriani. AToMPM Solution for the IMDB Case Study. In *TCC@ STAF*, pages 134–138, 2014.

[17] Tassilo Horn. Solving the TTC Movie Database Case with FunnyQT. In *TCC@ STAF*, pages 139–144. Citeseer, 2014.

[18] None. Duplicate paper.

[19] Kevin Lano and Sobhan Yassipour Tehrani. Solving the TTC 2014 Movie Database Case with UML-RSDS. In *TCC@ STAF*, pages 150–154. Citeseer, 2014.

[20] Pablo Inostroza and Tijs Van Der Storm. The TTC 2014 Movie Database Case: Rascal Solution. In *Transformation Tool Contest*, pages 155–159. CEUR, 2014.

[21] Yingfei Xiong, Hui Song, Zhenjiang Hu, and Masato Takeichi. Synchronizing concurrent model updates based on bidirectional transformation. *Software & Systems Modeling*, 12(1):89–104, 2013.

[22] Tassilo Horn. Transformation tool soccer worldcup (TTC 2014 live contest case). In *TCC@ STAF*. Citeseer, 2014.

[23] Sagar Sen, Naouel Moha, Vincent Mahé, Olivier Barais, Benoit Baudry, and Jean-Marc Jézéquel. Reusable model transformations. *Software & Systems Modeling*, 11(1):111–125, 2012.

[24] Alcino Cunha, Ana Garis, and Daniel Riesco. Translating between Alloy specifications and UML class diagrams annotated with OCL. *Software & Systems Modeling*, 14(1):5–25, 2015.

[25] Gehan MK Selim, Shige Wang, James R Cordy, and Juergen Dingel. Model transformations for migrating legacy deployment models in the automotive industry. *Software & Systems Modeling*, 14(1):365–381, 2015.

[26] Eugene Syriani and Hans Vangheluwe. A modular timed graph transformation language for simulation-based design. *Software & Systems Modeling*, 12(2):387–414, 2013.

[27] Mina Boström Nakićenović. An Agile Driven Architecture Modernization to a Model-Driven Development Solution. *International Journal on Advances in Software Volume 5, Number 3 & 4*, 2012.

[28] Louis M Rose, Dimitrios S Kolovos, Richard F Paige, and FA Polack. Model migration case for TTC 2010. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 1, 2010.

[29] Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel Briand. Environment modeling and simulation for automated testing of soft real-time embedded software. *Software & Systems Modeling*, 14(1):483–524, 2015.

[30] Pieter Van Gorp and Louis M Rose. The petri-nets to statecharts transformation case. *arXiv preprint arXiv:1312.0342*, 2013.

[31] Georg Hinkel, Thomas Goldschmidt, and Lucia Happe. An NMF Solution for the Flowgraphs case study at the TTC 2013. *Sixth Transformation Tool Contest (TTC 2013), ser. EPTCS*, 2013.

[32] Anthony Anjorin and Marius Lauder. A Solution to the Flowgraphs Case Study using Triple Graph Grammars and eMoflon. *arXiv preprint arXiv:1312.0348*, 2013.

[33] Georg Hinkel, Thomas Goldschmidt, and Lucia Happe. An NMF solution for the Petri Nets to State Charts case study at the TTC 2013. *arXiv preprint arXiv:1312.0596*, 2013.

[34] Tassilo Horn. Solving the TTC 2013 Flowgraphs Case with FunnyQT. In *TTC*, pages 57–68, 2013.

[35] Valerio Cosentino, Massimo Tisi, and Fabian Büttner. Analyzing Flowgraphs with ATL. *arXiv preprint arXiv:1312.0343*, 2013.

[36] Wietse Smid and Arend Rensink. Class diagram restructuring with GROOVE. *arXiv preprint arXiv:1312.0350*, 2013.

[37] Kevin Lano and S Kolahdouz Rahimi. Case study: Class diagram restructuring. *arXiv preprint arXiv:1309.0369*, 2013.

[38] Tassilo Horn. Solving the Class Diagram Restructuring Transformation Case with FunnyQT. *arXiv preprint arXiv:1312.0349*, 2013.

[39] Jesús Sánchez Cuadrado. Solving the Flowgraphs Case with Eclectic. *arXiv preprint arXiv:1312.0346*, 2013.

[40] Kevin Lano, S Kolahdouz-Rahimi, and Krikor Maroukian. Solving the Petri-Nets to statecharts transformation case with UML-RSDS. *arXiv preprint arXiv:1312.0352*, 2013.

[41] Tassilo Horn. Solving the Petri-Nets to Statecharts Transformation Case with FunnyQT. *arXiv preprint arXiv:1312.0351*, 2013.

[42] Benedek Izsó, Ábel Hegedüs, Gábor Bergmann, Ákos Horváth, and István Ráth. PN2SC Case Study: An EMF-IncQuery solution. *arXiv preprint arXiv:1312.0354*, 2013.

[43] Sven Peldszus, Géza Kulcsár, and Malte Lochau. A Solution to the Java Refactoring Case Study using eMoflon. In *8th Transformation Tool Contest*. CEUR, 2015.

[44] Benoit Combemale, Julien Deantoni, Olivier Barais, Arnaud Blouin, Erwan Bousse, Cédric Brun, Thomas Degueule, and Didier Vojtisek. A Solution to the TTC'15 Model Execution Case Using the GEMOC Studio. In *8th Transformation Tool Contest*. CEUR, 2015.

[45] Georg Hinkel. An NMF solution to the Java Refactoring Case at the TTC 2015. In *8th Transformation Tool Contest*. CEUR, 2015.

[46] Georg Hinkel and Lucia Happe. An NMF solution to the Train Benchmark Case at the TTC 2015. In *8th Transformation Tool Contest*. CEUR, 2015.

[47] Filip Křikava. Solving the TTC'15 Train Benchmark Case Study with SIGMA. In *Proceedings of the 8th Transformation Tool Contest*, 2015.

[48] Gábor Szárnyas, Márton Búr, and István Ráth. Train Benchmark Case: an EMF-INCQUERY Solution. In *8th Transformation Tool Contest*. CEUR, 2015.

[49] Stefan Lindel and Albert Zündorf. The SDMLib solution to the Model Execution Case for TTC2015. In *Proceedings of the 8th Transformation Tool Contest*, 2015.

[50] Olaf Gunkel, Matthias Schmidt, and Albert Zündorf. The SDMLib solution to the Java Refactoring case for TTC2015. In *Proceedings of the 8th Transformation Tool Contest*, 2015.

[51] Tassilo Horn. Solving the TTC Train Benchmark Case with FunnyQT. In *Proceedings of the 8th Transformation Tool Contest*, 2015.

[52] Tassilo Horn. Solving the TTC Model Execution Case with FunnyQT. In *Proceedings of the 8th Transformation Tool Contest*, 2015.

[53] Tassilo Horn. Solving the TTC Java Refactoring Case with FunnyQT. In *Proceedings of the 8th Transformation Tool Contest*, 2015.

[54] Christoff Bürger. fUML Activity Diagrams with RAG-controlled Rewriting: A RACR Solution of The TTC 2015 Model Execution Case. In *8th Transformation Tool Contest*, volume 1524, pages 27–36. CEUR Workshop Proceedings (CEUR-WS. org), 2015.

[55] Dániel Stein, Gábor Szárnyas, and István Ráth. Java Refactoring Case: a VIATRA Solution.

[56] Dennis Wagelaar. The ATL/EMFTVM Solution to the Train Benchmark Case for TTC2015.

[57] Gérard Paligot, Nicolas Petitprez, and Martin Monperrus. TTC'2015 Case: Refactoring Java Programs using Spoon. In *Transformation Tool Contest*, 2015.

[58] Marlon Dumas. Case study: BPMN to BPEL model transformation. In *5th International Workshop on Graph-Based Tools–GraBaTs*, 2009.

[59] Hessa Abdulrahman A Alfraihi Kevin Lano, Sobhan Yassipour Tehrani. Improving the Application of Agile Model-based Development: Experiences from Case Studies. volume 14, pages 5–25. Springer, 2015.

[60] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy: A challenging model transformation. In *International Conference on Model Driven Engineering Languages and Systems*, pages 436–450. Springer, 2007.

[61] Jordi Cabot, Robert Clarisó, Esther Guerra, and Juan de Lara. A UML/OCL framework for the analysis of graph transformation rules. *Software & Systems Modeling*, 9(3):335–357, 2010.

[62] Greg Freeman, Don Batory, and Greg Lavender. Lifting transformational models of product lines: A case study. In *International Conference on Theory and Practice of Model Transformations*, pages 16–30. Springer, 2008.

[63] Zef Hemel, Lennart CL Kats, Danny M Groenewegen, and Eelco Visser. Code generation by model transformation: a case study in transformation modularity. *Software & Systems Modeling*, 9(3):375–402, 2010.

[64] Yasser A Khan and Mohamed El-Attar. Using model transformation to refactor use case models based on antipatterns. *Information Systems Frontiers*, 18(1):171–204, 2016.

[65] Alexander Hück, Jean Utke, and Christian Bischof. Source Transformation of C++ Codes for Compatibility with Operator Overloading. *Procedia Computer Science*, 80:1485–1496, 2016.

[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.

[67] Viktoria Ovchinnikova and Erika Asnina. The algorithm of transformation from UML sequence diagrams to the Topological Functioning Model. In *Evaluation of Novel Approaches to Software Engineering (ENASE), 2015 International Conference on*, pages 377–384. IEEE, 2015.

[68] Slaviša Marković and Thomas Baar. Semantics of OCL specified with QVT. *Software & Systems Modeling*, 7(4):399–422, 2008.

[69] Ulf Eliasson, Rogardt Heldal, Jonn Lantz, and Christian Berger. Agile model-driven engineering in mechatronic systems-an industrial case study. In *International Conference on Model Driven Engineering Languages and Systems*, pages 433–449. Springer, 2014.

[70] Slaviša Marković and Thomas Baar. Refactoring OCL annotated UML class diagrams. In *International Conference on Model Driven Engineering Languages and Systems*, pages 280–294. Springer, 2005.

[71] Alexandre Correa and Cláudia Werner. Refactoring object constraint language specifications. *Software & Systems Modeling*, 6(2):113–138, 2007.

[72] Ashalatha Nayak and Debasis Samanta. Synthesis of test scenarios using UML activity diagrams. *Software & Systems Modeling*, 10(1):63–89, 2011.

[73] Victoria Torres, Pau Giner, and Vicente Pelechano. Developing BP-driven web applications through the use of MDE techniques. *Software & Systems Modeling*, 11(4):609–631, 2012.

[74] Zoltn Petres, Pter Baranyi, Pter Korondi, and Hideki Hashimoto. Trajectory tracking by TP model transformation: case study of a benchmark problem. *IEEE Transactions on Industrial Electronics*, 54(3):1654–1663, 2007.

[75] Chihab eddine Mokaddem, Houari Sahraoui, and Eugene Syriani. Towards Rule-Based Detection of Design Patterns in Model Transformations. In *International Conference on System Analysis and Modeling*, pages 211–225. Springer, 2016.

[76] István Dávid, István Ráth, and Dániel Varró. Streaming model transformations by complex event processing. In *International Conference on Model Driven Engineering Languages and Systems*, pages 68–83. Springer, 2014.

[77] Valeria De Castro, Juan M Vara, and Esperanza Marcos. Model Transformation for Service-Oriented Web Applications Development. In *MDWE*, 2007.

[78] Jesús Sánchez Cuadrado, Esther Guerra, and Juan de Lara. A component model for model transformations. *IEEE Transactions on Software Engineering*, 40(11):1042–1060, 2014.

[79] Ken Vanherpen, Joachim Denil, Hans Vangheluwe, and Paul De Meulenaere. Model transformations for round-trip engineering in control deployment co-design. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 55–62. Society for Computer Simulation International, 2015.

[80] Veronica Andrea Bollati, Juan Manuel Vara, Alvaro Jimenez, and Esperanza Marcos. Applying MDE to the (semi-) automatic development of model transformations. *Information and Software Technology*, 55(4):699–718, 2013.

[81] Tassilo Horn. Model migration with GReTL for TTC 2010 case. pages 10–03. Citeseer, 2010.

[82] Louis M Rose, Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. Migrating activity diagrams with Epsilon Flock. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 30, 2010.

[83] Elina Kalnina, Audris Kalnins, Janis Iraids, Agris Sostaks, and Edgars Celms. Model migration with MOLA. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 38, 2010.

[84] Andreas Koch, Ruben Jubeh, and Albert Zündorf. UML 1. 4 to 2.1 activity diagram model migration with Fujaba TTC 2010 case study.

[85] Sebastian Buchwald and Edgar Jakumeit. A GrGen .NET solution of the model migration case for the Transformation Tool Contest 2010. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 61, 2010.

[86] Markus Herrmannsdoerfer. Migrating UML activity models with COPE. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 72, 2010.

[87] Bernhard Schätz. UML model migration with PETE. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 85, 2010.

[88] Antonio Cicchetti, Bart Mayers, and Manuel Wimmer. Abstract and concrete syntax migration of instance models. In *TTC: Transformation Tool Contest, Satellite workshop to TOOLS 2010*, 2010.

[89] Amir Hossein Ghamarian Maarten de Mol and Arend Rensink Eduardo Zambon. Solving the Topology Analysis Case Study with GROOVE. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 119.

[90] Peter Backes and Jan Reineke. Abstract topology analysis of the join phase of the merge protocol for TTC 2010. *Proceedings of Transformation Tool Contest (TTC 2010), Malaga, Spain, 1–2 July*, 2010.

[91] Christian Heinzemann, Julian Suck, Ruben Jubeh, and Albert Zündorf. Topology analysis of car platoons merge with Fujabart & TimedStoryCharts - TTC2010 case study. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 134, 2010.

[92] Enrico Biermann, Claudia Ermel, and Stefan Jurack. Modeling the Ecore to GenModel Transformation with EMF Henshin. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 153, 2010.

[93] Jens von Pilgrim. Ecore2GenModel with Mitra and GEF3D. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 166, 2010.

[94] Ábel Hegedüs, Zoltán Ujhelyi, Gábor Bergmann, and Ákos Horváth. Ecore to Genmodel case study solution using the Viatra2 framework. *Transformation Tool Contest 2010 1-2 July 2010, Malaga, Spain*, page 187, 2010.

[95] F Taghizadeh and SR Taghizadeh. A Graph Transformation-Based Approach for Applying MDA to SOA. In *2009 Fourth International Conference on Frontier of Computer Science and Technology*, pages 446–451. IEEE, 2009.

[96] Michel dos Santos Soares and Jos Vrancken. A metamodeling approach to transform UML 2.0 sequence diagrams to Petri nets. In *Proceedings of the IASTED International Conference on Software Engineering*, pages 159–164, 2008.

[97] Eman Saleh, AMR Kamel, and Aly Fahmy. A model driven engineering design approach for developing multi-platform user interfaces. *WSEAS Transactions on Computers*, 9(5):536–545, 2010.

[98] Leon Urbas and Falk Doherr. autoHMI: a model driven software engineering approach for HMIs in process industries. In *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, volume 3, pages 627–631. IEEE, 2011.

[99] Manuel Wimmer. A semi-automatic approach for bridging DSMLs with UML. *International Journal of Web Information Systems*, 5(3):372–404, 2009.

[100] Amogh Kavimandan and Aniruddha Gokhale. Automated Middleware QoS Configuration Techniques using Model Transformations. In *EDOC Conference Workshop, 2007. EDOC'07. Eleventh International IEEE*, pages 20–27. IEEE, 2007.

[101] Emanuel Montero Reyno and José Á Carsí Cubel. Automatic prototyping in model-driven game development. *Computers in Entertainment (CIE)*, 7(2):29, 2009.

[102] Andrea Schauerhuber, Manuel Wimmer, Elisabeth Kapsammer, Wieland Schwinger, and Werner Retschitzegger. Bridging webml to model-driven engineering: from document type definitions to meta object facility. *IET software*, 1(3):81–97, 2007.

[103] Ashley Sterritt and Vinny Cahill. Customisable model transformations based on non-functional requirements. In *2008 IEEE Congress on Services-Part I*, pages 329–336. IEEE, 2008.

[104] Ahmed Harbouche, Mohammed Erradi, and Aicha Mokhtari. Deriving Multi-Agent System Behavior. *International Journal of Software Engineering and Its Applications*, 7(4):137–156, 2013.

[105] Soon-Kyeong Kim, Toby Myers, Marc-Florian Wendland, and Peter A Lindsay. Execution of natural language requirements using

State Machines synthesised from Behavior Trees. *Journal of Systems and Software*, 85(11):2652–2664, 2012.

[106] Carsten Amelunxen and A Schurr. Formalising model transformation rules for UML/MOF 2. *IET software*, 2(3):204–222, 2008.

[107] Zhibin Yang, Kai Hu, Dianfu Ma, Jean-Paul Bodeveix, Lei Pi, and Jean-Pierre Talpin. From AADL to timed abstract state machines: A verified model transformation. *Journal of Systems and Software*, 93:42–68, 2014.

[108] Mathias Funk, Er Nyßen, and Horst Lichter. From UML to ANSI-C-An Eclipse-Based Code Generation Framework. In *Proceedings of 3rd International Conference on Software and Data Technologies (ICSOFT)*. Citeseer, 2008.

[109] Valeria De Castro, Juan Manuel Vara Mesa, Elisa Herrmann, and Esperanza Marcos. From Real Computational Independent Models to Information System Models: An MDE Approach. In *4th International Workshop on Model-Driven Web Engineering (MDWE 2008)*, 2009.

[110] Óscar Pastor. Generating User Interfaces From Conceptual Models: A Model-Transformation Based Approach. In *Computer-Aided Design of User Interfaces V*, pages 1–14. Springer, 2007.

[111] João Pedro Santos, Ana Moreira, João Araújo, and Miguel Goulão. Increasing Quality in Scenario Modelling with Model-Driven Development. In *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pages 204–209. IEEE, 2010.

[112] Milan Milanović, Dragan Gašević, Adrian Giurca, Gerd Wagner, Sergey Lukichev, and Vladan Devedžić. Model transformations to bridge concrete and abstract syntax of web rule languages. *Computer Science and Information Systems*, 6(2):47–85, 2009.

[113] Oscar Díaz, Gorka Puente, Javier Luis Cánovas Izquierdo, and Jesús García Molina. Harvesting models from web 2.0 databases. *Software & Systems Modeling*, 12(1):15–34, 2013.

[114] Andreas Fuhr, Tassilo Horn, Volker Riediger, and Andreas Winter. Model-driven software migration into service-oriented architectures. *Computer Science-Research and Development*, 28(1):65–84, 2013.

[115] S Mbarki and M Erramdani. Model-driven transformations: From analysis to MVC 2 web model. *International Review on Computers and Software (I. RE. CO. S.)*, 4(5):612–620, 2009.

[116] Thomas Buchmann, Bernhard Westfechtel, and Sabine Winetzhammer. ModGraph-A Transformation Engine for EMF Model Transformations. In *ICSOFT (2)*, pages 212–219, 2011.

[117] Nikolaos Spanoudakis and Pavlos Moraitis. Modular JADE agents design and implementation using ASEME. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 2, pages 221–228. IEEE, 2010.

[118] Manuel Wimmer, Angelika Kusel, Johannes Schönböck, Werner Retschitzegger, Wieland Schwinger, and Gerti Kappel. On using inplace transformations for model co-evolution. In *Proceedings 2nd Int. Workshop Model Transformation with ATL*, volume 711, pages 65–78, 2010.

[119] Li Dan. QVT based model transformation from sequence diagram to CSP. In *Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on*, pages 349–354. IEEE, 2010.

[120] Joachim Denil, Pieter J Mosterman, and Hans Vangheluwe. Rule-based model transformation for, and in simulink. In *Proceedings*

of the Symposium on Theory of Modeling & Simulation-DEVS Integrative, page 4. Society for Computer Simulation International, 2014.

[121] István Madari, László Angyal, and László Lengyel. Traceability-based incremental model synchronization. *WSEAS Transactions on Computers*, 8(10):1691–1700, 2009.

[122] Gunter Mussbacher, Jörg Kienzle, and Daniel Amyot. Transformation of aspect-oriented requirements specifications for reactive systems into aspect-oriented design specifications. In *Model-Driven Requirements Engineering Workshop (MoDRE), 2011*, pages 39–47. IEEE, 2011.

[123] Fábio Levy Siqueira and Paulo Sérgio Muniz Silva. Transforming an enterprise model into a use case model in business process systems. *Journal of Systems and Software*, 96:152–171, 2014.

[124] Fabio Levy Siqueira, Paulo Sergio Muniz Silva, and Paulo Sérgio Muniz Silva. Transforming an enterprise model into a use case model using existing heuristics. In *Model-Driven Requirements Engineering Workshop (MoDRE), 2011*, pages 21–30. IEEE, 2011.

[125] Álvaro Jiménez, David Granada, VA Bollati, and Juan M Vara. Using ATL to support model-driven development of RubyTL model transformations. In *3rd International workshop on model transformation with ATL (MtATL2011), Zürich, Switzerland*, pages 35–48. Citeseer, 2011.

[126] Thomas Buchmann and Felix Schwägerl. Using Meta-code Generation to Realize Higher-order Model Transformations. In *ICSOFT*, pages 536–541, 2013.

[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*

Model-Driven Engineering, Verification and Validation, page 3. ACM, 2009.

[128] Karim Dahman, François Charoy, and Claude Godart. Generation of component based architecture from business processes: model driven engineering for SOA. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 155–162. IEEE, 2010.

[129] Olaf Muliawan, Pieter Van Gorp, Anne Keller, and Dirk Janssens. Executing a standard compliant transformation model on a non-standard platform. In *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on*, pages 151–160. IEEE, 2008.

[130] Xiaofeng Cui. Co-design of the business and software architectures: A systems engineering and model-driven method. In *Software Engineering Artificial Intelligence Networking and Parallel/Distributed Computing (SNPD), 2010 11th ACIS International Conference on*, pages 209–214. IEEE, 2010.

[131] Antinisca Di Marco and Stefano Pace. Model-driven approach to Agilla agent generation. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1482–1487. IEEE, 2013.

[132] Federico Ciccozzi, Antonio Cicchetti, Toni Siljamäki, and Jenis Kavadiya. Automating test cases generation: From xtUML system models to QML test models. In *Proceedings of the 7th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pages 9–16. ACM, 2010.

[133] DA Meedeniya and Indika Perera. Model based software design: Tool support for scripting in immersive environments. In *2013 IEEE 8th International Conference on Industrial and Information Systems*, pages 248–253. IEEE, 2013.

[134] Tudor Gîrba, Jean-Marie Favre, and Stéphane Ducasse. Using meta-model transformation to model software evolution. *Electronic Notes in Theoretical Computer Science*, 137(3):57–64, 2005.

[135] Friedrich Steimann. Constraint-based model refactoring. In *International Conference on Model Driven Engineering Languages and Systems*, pages 440–454. Springer, 2011.

[136] Eugene Syriani and Hüseyin Ergin. Operational semantics of UML activity diagram: An application in project management. In *Model-Driven Requirements Engineering Workshop (MoDRE), 2012 IEEE*, pages 1–8. IEEE, 2012.

[137] Jim Steel and Robin Drogemuller. Domain-specific model transformation in building quantity take-off. In *International Conference on Model Driven Engineering Languages and Systems*, pages 198–212. Springer, 2011.

[138] Hui Song, Gang Huang, Franck Chauvel, Wei Zhang, Yanchun Sun, Weizhong Shao, and Hong Mei. Instant and incremental QVT transformation for runtime models. In *International Conference on Model Driven Engineering Languages and Systems*, pages 273–288. Springer, 2011.

[139] Mirco Kuhlmann and Martin Gogolla. From UML and OCL to relational logic and back. In *International Conference on Model Driven Engineering Languages and Systems*, pages 415–431. Springer, 2012.

[140] Mathieu Acher, Philippe Lahire, Sabine Moisan, and Jean-Paul Rigault. Tackling high variability in video surveillance systems through a model transformation approach. In *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*, pages 44–49. IEEE computer society, 2009.

[141] Gerd Kainz, Christian Buckl, Stephan Sommer, and Alois Knoll. Model-to-metamodel transformation for the development of component-based systems. In *International Conference on Model Driven Engineering Languages and Systems*, pages 391–405. Springer, 2010.

[142] Tirdad Rahmani, Daniel Oberle, and Marco Dahms. An adjustable transformation from owl to ecore. In *International Conference on Model Driven Engineering Languages and Systems*, pages 243–257. Springer, 2010.

[143] António Miguel Rosado da Cruz and João Pascoal Faria. A metamodel-based approach for automatic user interface generation. In *International Conference on Model Driven Engineering Languages and Systems*, pages 256–270. Springer, 2010.

[144] None. Duplicate paper.

[145] Juan de Lara, Esther Guerra, Artur Boronat, Reiko Heckel, and Paolo Torrini. Domain-specific discrete event modelling and simulation using graph transformation. *Software & Systems Modeling*, 13(1):209–238, 2014.

[146] Kelly Garcés, Juan M Vara, Frédéric Jouault, and Esperanza Marcos. Adapting transformations to metamodel changes via external transformation composition. *Software & Systems Modeling*, 13(2):789–806, 2014.

[147] Javier Luis Cánovas Izquierdo and Jesús García Molina. Extracting models from source code in software modernization. *Software & Systems Modeling*, 13(2):713–734, 2014.

[148] Haiping Zha, Wil MP van der Aalst, Jianmin Wang, Lijie Wen, and Jiaguang Sun. Verifying workflow processes: a transformation-based approach. *Software & Systems Modeling*, 10(2):253–264, 2011.

[149] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On challenges of model transformation from UML to Alloy. *Software & Systems Modeling*, 9(1):69–86, 2010.

[150] Frank Hilken, Lars Hamann, and Martin Gogolla. Transformation of UML and OCL models into filmstrip models. In *International Conference on Theory and Practice of Model Transformations*, pages 170–185. Springer, 2014.

[151] Frank Hermann, Susann Gottmann, Nico Nachtigall, Hartmut Ehrig, Benjamin Braatz, Gianluigi Morelli, Alain Pierre, Thomas Engel, and Claudia Ermel. Triple graph grammars in the large for translating satellite procedures. In *International Conference on Theory and Practice of Model Transformations*, pages 122–137. Springer, 2014.

[152] Jeroen Van den Bos and Tijs Van Der Storm. Domain-specific optimization in digital forensics. In *International Conference on Theory and Practice of Model Transformations*, pages 121–136. Springer, 2012.

[153] Dennis Wagelaar, Ludovico Iovino, Davide Di Ruscio, and Alfonso Pierantonio. Translational semantics of a co-evolution specific language with the EMF transformation virtual machine. In *International Conference on Theory and Practice of Model Transformations*, pages 192–207. Springer, 2012.

[154] Jácome Cunha, João P Fernandes, Jorge Mendes, Hugo Pacheco, and João Saraiva. Bidirectional transformation of model-driven spreadsheets. In *International Conference on Theory and Practice of Model Transformations*, pages 105–120. Springer, 2012.

[155] Eugene Syriani and Hans Vangheluwe. Programmed graph rewriting with time for simulation-based design. In *International Conference on Theory and Practice of Model Transformations*, pages 91–106. Springer, 2008.

[156] Ricardo Pérez-Castillo, Ignacio García-Rodríguez De Guzmán, and Mario Piattini. Implementing business process recovery patterns through QVT transformations. In *International Conference on Theory and Practice of Model Transformations*, pages 168–183. Springer, 2010.

[157] Roy Grønmo and Birger Møller-Pedersen. From sequence diagrams to state machines by graph transformation. In *International Conference on Theory and Practice of Model Transformations*, pages 93–107. Springer, 2010.

[158] MF Van Amstel, Mark GJ van den Brand, Z Protić, and Tom Verhoeff. Transforming process algebra models into UML state machines: Bridging a semantic gap? In *International Conference on Theory and Practice of Model Transformations*, pages 61–75. Springer, 2008.

[159] Jácome Cunha, João Saraiva, and Joost Visser. From spreadsheets to relational databases and back. In *Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, pages 179–188. ACM, 2009.

[160] Mathias Kleiner, Marcos Didonet Del Fabro, and Davi De Queiroz Santos. Transformation as search. In *European Conference on Modelling Foundations and Applications*, pages 54–69. Springer, 2013.

[161] Tao Yue and Shaukat Ali. Bridging the gap between requirements and aspect state machines to support non-functional testing: industrial case studies. In *European Conference on Modelling Foundations and Applications*, pages 133–145. Springer, 2012.

[162] Simon Schwichtenberg, Christian Gerth, Zille Huma, and Gregor Engels. Normalizing heterogeneous service description models with generated QVT transformations. In *European Conference on*

Modelling Foundations and Applications, pages 180–195. Springer, 2014.