# Model Transformation Applications from Requirements Engineering Perspective

Sobhan Yassipour Tehrani, Kevin Lano
Department of Informatics, King's College London, London WC2R 2LS, U.K.
E-mail: {sobhan.yassipour_tehrani,kevin.lano}@kcl.ac.uk

***Abstract*—Requirements Engineering (RE) is an essential process in the development of effective software systems, and it is the basis for subsequent development processes. At present, the focus of Model Transformation (MT) is mainly on the specification and implementation stages. Transformations are not using engineering principles, which may not be an issue within a small project, but it will be problematic in large scale industry projects. One of the main reasons that hinders a systematic RE process to be used before starting the development could be the false assumption that it is a waste of time/cost and would delay the implementation. The goal of this paper is to evaluate model transformation technology from a requirements engineering process point of view. We identify techniques for the RE of MT, taking into account specific characteristics of different categories of model transformations.**

***Keywords- model transformations; requirements engineering; requirements engineering framework.***

## I. INTRODUCTION

Requirements engineering has been a relatively neglected aspect of model transformation development because the emphasis in transformation development has been upon specifications and implementations. The failure to explicitly identify requirements may result in developed transformations, which do not satisfy the needs of the users of the transformation. Problems may arise because implicitly-assumed requirements have not been explicitly stated; for instance, that a migration or refactoring transformation should preserve the semantics of its source model in the target model, or that a transformation is only required to operate on a restricted range of input models. Without thorough requirements elicitation, important requirements may be omitted from consideration, resulting in a developed transformation which fails to achieve its intended purpose.

We use the RE process model proposed by Kotonya and Sommerville [1] and adapt it according to our specific needs. This process model is widely accepted by researchers and professional experts. The following are the most important phases of RE, which have to be applied: domain analysis and requirements elicitation, evaluation and negotiation, specification and documentation, validation and verification.

In this paper we focus on the specification stage, which makes precise the informal requirements agreed with the stakeholders of the proposed development. By providing a comprehensive catalogue of model requirement types, this paper can help transformation developers to ensure that all requirements of a transformation are explicitly considered.

Section 3 gives a background on requirements engineering for model transformations as well as transformation semantics and its nature. We also identify how formalised requirements can be validated and can be used to guide the selection of design patterns for the development of the transformation. In Section 4 we examine some published requirements statements of model transformation to identify their gaps and subsequent consequences on the quality of the solutions. In Section 5 we give a case study to illustrate the benefits of systematic requirements engineering for model transformations.

## II. STATE OF THE ART

As Selic [2] argues, "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 [3][4][5] is focussed upon design and verification rather than upon requirements engineering. The work on requirements engineering in *trans*ML [3] is focussed upon functional requirements, and the use of abstract syntax rules to express them. Here, we consider a full range of functional and non-functional requirements and we use concrete syntax rules for the initial expression of functional requirements.

In order to trace the requirements into subsequent steps, *trans*ML defines a modelling language, which represents the requirements in the form of Systems Modeling Language (SysML) [6] diagrams. This would allows the transformer(s) to link requirements of a model transformation to its corresponding analysis and design models, code and other artifacts. Having a connection amongst different artifacts in the model transformation development process enables the transformer(s) to check the correctness and completeness of all requirements [7]. At present, transformations are not using engineering principles which may not be an issue within a small project, but it will be problematic in large scale industry projects. Jumping straight to an implementation language might be possible for simple transformations, however it would be problematic for large ones. Transformations should be constructed by applying engineering principles especially if they are to be used in an industry. Therefore, the development of the transformation's life-cycle should include other phases in addition to coding and testing, namely, requirements engineering process [3].

In this paper, we describe a requirements engineering process for transformations based on adaptions of the RE process model, and specialisations of RE techniques for transformations.

## III. REQUIREMENTS FOR MODEL TRANSFORMATIONS

Requirements for a software product are generally divided into two main categories: functional requirements, which identify what functional capabilities the system should provide,

 ISBN: 978-1-61208-438-1

and non-functional requirements, which identify quality characteristics expected from the developed system and restrictions upon the development process itself.

The functional requirements of a model transformation $\tau$:$S \rightarrow T$, which maps models of a source language $S$ to a target language $T$ are defined in terms of the effect of $\tau$ on model $m$ of $S$, and the relationship of the resulting model $n$ of $T$ to $m$. It is a characteristic of model transformations that such functional requirements are usually decomposed into a set of mapping requirements for different cases of structures and elements within $S$. In addition, assumptions about the input model should be identified as part of the functional requirements.

It can be observed in many published examples of model transformations that the initial descriptions of their intended functional behaviour is in terms of a concrete syntax for the source and target languages, which they operate upon. For instance in [8], the three key effects of the transformation are expressed in terms of rewritings of Unified Modeling Language (UML) class diagrams. In [9], the transformation effects are expressed by parallel rewritings of Petri Nets and statecharts. In general, specification of the intended functionality of the transformation in terms of concrete syntax rules is more natural and comprehensible for the stakeholders than is specification in terms of abstract syntax. However, this form of description has the disadvantage that it may be imprecise; there may be significant details of models, which have no representation in the 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.

Requirements may be functional or non-functional (e.g., concerned with the size of generated models, transformation efficiency or confluence). Another distinction, which is useful for transformations is between local and global requirements:

- Local requirements are concerned with localised parts of one or more models. Mapping requirements define when and how a part of one model should be mapped onto a part of another. Rewriting requirements dictate when and how a part of a model should be refactored/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. Invariants, assumptions and postconditions of a transformation usually apply at the entire model level.

Figure 1 shows a taxonomy of functional requirements for model transformations based on our experience of transformation requirements.

We have also created a taxonomy of the non-functional requirements that one has to consider during the RE process. Figure 2 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 [10].

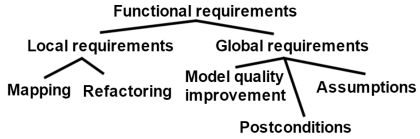


Figure. 1. A taxonomy of functional requirements

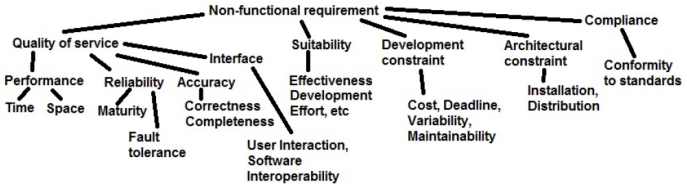


Figure. 2. A taxonomy of non-functional requirements for MT

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). 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.

Maturity and fault tolerance are a subset of reliability requirements for a transformation. Depending on its history and to the extent to which a transformation has been used, maturity requirements could be measured. Fault tolerance requirements 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.

As depicted in the above figure, the accuracy characteristic includes two sub-characteristics: correctness and completeness. Correctness requirements can be further divided into the following forms [11]:

- *Syntactic correctness*: a transformation $\tau$ is syntactically correct when a valid input model $m$ from source language $S$ is transformed to target language $T$, then (if it terminates) it produces a valid result, in terms of conformation to the $T$'s language constraints.
- *Termination*: a transformation $\tau$ will always terminate if applied to a valid $S$ model.
- *Confluence*: all result models produced by transformation $\tau$ from a single source model are isomorphic.
- *Model-level semantic preservation*: a transformation $\tau$ is preserved model-level semantically, if $m$ and $n$ have equivalent semantics under semantics-assigning maps $Sem_S$ on models of S and $Sem_T$ on models of T.

 ISBN: 978-1-61208-438-1 

- *Invariance*: some properties *Inv* should be preserved as true during the entire execution of transformation $\tau$ [11].

An additional accuracy property that can be considered is the existence of invertibility in a transformation $\sigma : T \rightarrow S$ which inverts the effect of $\tau$. Given a model $n$ derived from $m$ by $\tau$, $\sigma$ applied to $n$ produces a model $m\prime$ 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. A further property of 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. Under interface are requirements categories of User interaction (subdivided into usability and convenience) and software interoperability. Usability requirements can be decomposed into aspects, such as understandability, learnability and attractiveness [12]. Software interoperability can be decomposed into interoperability capabilities of the system with each intended environment and software system, with which it is expected to operate.

Based on [12], we define suitability 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 (developer time, computational resources, etc.). In [8] we identified the following subcharacteristics for the suitability quality characteristic of model transformation specifications: abstraction level, size, complexity, effectiveness and development effort.

Requirements of single transformations can be documented using the SysML notation adopted in [3], but with a wider range of requirement types represented. Use case diagrams can be used to describe the requirements of a system of transformations. Each use case represents an individual transformation which may be available as a service for external users, or which may be used internally within the system as a subtransformation of other transformations.

We have investigated a specific functional requirements taxonomy according to the characteristic of model transformations (Table I). All types of functional requirements for model transformations including: mapping, assumptions and postconditions requirements could be formalized as predicates or diagrams at the concrete and abstract syntax levels. Concrete syntax is often used at the early stages (RE stages) in the development cycle in order to validate the requirements by stakeholders since the concrete syntax level is more convenient, 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/semi-formal expression of the requirements, and the abstract syntax elements in the formalised versions.

TABLE I. 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 |

## IV. APPLICATION OF RE IN MT

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 precisely what is to be developed.

Requirements engineering for model transformations involves specialised techniques and approaches because transformations (i) have highly complex behaviour, involving nondeterministic application of rules and inspection/ construction of complex model data, (ii) are often high-integrity and business-critical systems with strong requirements for reliability and correctness.

Transformations do not usually involve much user interaction, but may have security requirements if they process secure data. Correctness requirements which are specific to transformations, due to their characteristic execution as a series of rewrite rule applications, with the order of these applications not algorithmically determined, are: (i) confluence (that the output models produced by the transformation are equivalent, regardless of the rule application orders), (ii) termination (regardless of the execution order), (iii) to achieve specified properties of the target model, regardless of the execution order which is referred to as semantic correctness.

The source and target languages of a transformation may be precisely specified by metamodels, whereas 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

 ISBN: 978-1-61208-438-1 

input models for a given mapping strategy to be well-defined. 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 and the application of one rule to a model may enable further rule applications which were not originally enabled. 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.

### A. Application of RE Techniques for MT

A large number of requirements elicitation techniques have been devised. Through the analysis of surveys and case studies, we have identified the following adaption of RE techniques for MT.

The following techniques are the most suitable RE techniques to use during the requirements elicitation stage, which have been adapted according to the nature of model transformation technology.

Structured interviews: in this technique the requirements engineer asks stakeholders specific prepared questions about the domain and the system. The requirements engineer needs to define appropriate questions which help to identify issues of scope and product (output model) requirements, similar to that of unstructured interviews. 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.

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.

Scenario analysis: in this approach the requirements engineer formulates detailed scenarios/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 [3]. 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.

Regarding the requirements evaluation and negotiation stage, 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 Knowledge Acquisition in automated specification (KAOS) or SySML can be used to decompose requirements into sub-goals. A formal modelling notation such as Object Constraint Language (OCL) or state machines/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. For instance, in a refinement transformation, the semantics of the source and target model have to be equivalent as the primary requirement and to have a traceability feature as a secondary requirement. Also, 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. Several RE techniques exist which could be applicable to the transformation of RE during the requirements specification phase in which business goals are represented in terms of functional and non-functional requirements. In the following Table 2, requirements have been categorised according to the type of the transformation.

TABLE II. REQUIREMENTS PRIORITY FOR DIFFERENT TRANSFORMATIONS

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

Techniques for requirements specification and documentation stage include: UML and OCL, structured natural language, and formal modelling languages. At the initial stages of requirements elicitation and analysis, the intended effect of a transformation is often expressed by sketches or diagrams using the concrete grammar of the source and target languages concerned (if such grammars exist), or by node and line graphs if there is no concrete grammar. A benefit of concrete grammar rules is that they are directly understandable by stakeholders with knowledge of the source and target language notations. They are also independent of specific MT languages or technologies. Concrete grammar diagrams can be made more precise during requirements formalisation, or refined into abstract grammar rules. An informal mapping/refactoring requirement of the form of

> "*For each instance e of entity type E, that satisfies condition Cond, establish Pred* "

can be formalised as a use case postcondition such as:

 ISBN: 978-1-61208-438-1 

E::
Cond′ ⇒ Pred′

where *Cond′* formalises Cond, and *Pred′* formalises Pred.

For requirements verification and validation stage, the formalised rules can 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, it can be validated with the stakeholders to confirm that it does indeed accurately express the stakeholders intentions and needs for the system. The formalised requirements of a transformation $\tau$: $S \rightarrow T$ can also be verified to check that they are consistent; the functional requirements must be mutually consistent. The assumptions and invariant of $\tau$, and the language constraints of $S$ must be jointly consistent. The invariant and postconditions of $\tau$, and the language constraints of $T$ must be jointly consistent. 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 [13], [14]. Model-level semantics 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 [11]. Formalised requirements in temporal logic could then be checked for particular implementations using model-checking techniques, as in [15].

## V. RE PROCESS ON REFACTORING TRANSFORMATION

Refactoring is a type of model transformation. 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" [16]. We describe an example [17] of an in-place endogenous transformation which refactors class diagrams to improve their quality by removing redundant feature declarations. Figure 3 shows the metamodel of the source/target language of this transformation.

In this section, we are going to apply RE on a refactoring [18] transformation case study. The properties for this type of transformation are: endogenous, model-to-model, many-to-many (source to target model), horizontal, semantics preservation, explicit control/rule application scoping, rule iteration, traceable and that it is a unidirectional transformation. The following general requirements for refactoring transformations should be satisfied:

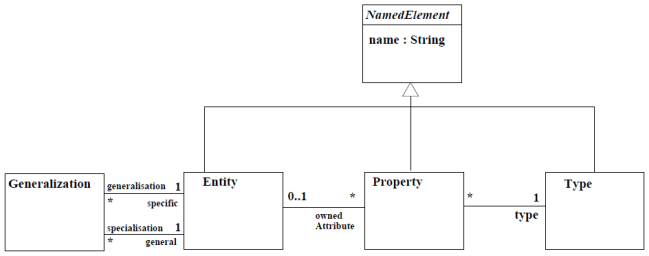


Figure. 3. Class diagram metamodel [18]

- Functionality: suitability, accuracy, interoperability, security, functionality compliance
- 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

*1) 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 by 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.

*2) Evaluation and negotiation for Refactoring:* At this point we should ask whether these scenarios are complete and if they cover all intended cases of the required refactor-

 ISBN: 978-1-61208-438-1

ings. 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.

*3) Requirements formalisation for Refactoring:* To formalise the functional requirements, we express the three scenarios in the abstract grammar of the language. Rule1: 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.

*4) Validation and 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*. The efficiency requirements can be verified by executing the prototype transformation on realistic test cases of increasing size.

## VI. Conclusion and Future Work

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. In future work, we will construct tool support for recording and tracing transformation requirements, which will help to ensure that developers systematically consider all necessary requirements and that these are all formalised, validated and verified correctly.

We are currently carrying out research into improving the requirements engineering process in model transformation. We will investigate formal languages to express the requirements, as formalised rules can be checked for internal correctness properties, such as definedness and determinacy, which should hold for meaningful rules. Temporal logic can be used to define the specialised characteristics of particular transformation and to define transformation requirements in a formal but language-independent manner languages as model transformation systems necessarily involve a notion of time. Finally, we will be evaluating large case studies in order to compare results with and without RE process.

## References


[1] I. Sommerville and G. Kotonya, Requirements engineering: processes and techniques. John Wiley & Sons, Inc., 1998.

[2] B. Selic, "What will it take? a view on adoption of model-based methods in practice," Software & Systems Modeling, vol. 11, no. 4, 2012, pp. 513–526.

[3] E. Guerra, J. De Lara, D. S. Kolovos, R. F. Paige, and O. M. dos Santos, "transml: A family of languages to model model transformations," in Model Driven Engineering Languages and Systems. Springer, 2010, pp. 106–120.

[4] K. Lano and S. Kolahdouz-Rahimi, "Model-driven development of model transformations," in Theory and practice of model transformations. Springer, 2011, pp. 47–61.

[5] K. Lano and S. Rahimi, "Constraint-based specification of model transformations," Journal of Systems and Software, vol. 86, no. 2, 2013, pp. 412–436.

[6] S. Friedenthal, A. Moore, and R. Steiner, A practical guide to SysML: the systems modeling language. Morgan Kaufmann, 2014.

[7] T. Yue, L. C. Briand, and Y. Labiche, "A systematic review of transformation approaches between user requirements and analysis models," Requirements Engineering, vol. 16, no. 2, 2011, pp. 75–99.

[8] S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya, and P. Van Gorp, "Evaluation of model transformation approaches for model refactoring," Science of Computer Programming, vol. 85, 2014, pp. 5–40.

[9] P. Van Gorp and L. M. Rose, "The petri-nets to statecharts transformation case," arXiv preprint arXiv:1312.0342, 2013.

[10] I. Iso, "Iec 25010: 2011,," Systems and Software EngineeringSystems and Software Quality Requirements and Evaluation (SQuaRE)System and Software Quality Models, 2011.

[11] K. Lano, S. Kolahdouz-Rahimi, and T. Clark, "Comparing verification techniques for model transformations," in Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation. ACM, 2012, pp. 23–28.

[12] I. O. F. S. E. Commission et al., "Software engineering–product quality–part 1: Quality model," ISO/IEC, vol. 9126, 2001, p. 2001.

[13] K. Anastasakis, B. Bordbar, and J. M. Küster, "Analysis of model transformations via alloy," in Proceedings of the 4th MoDeVVa workshop Model-Driven Engineering, Verification and Validation, 2007, pp. 47–56.

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

[15] S. Yassipour Tehrani and K. Lano, "Temporal logic specification and analysis for model transformations," in Verification of Model Transformations, VOLT 2015, 2015.

[16] C. Ermel, H. Ehrig, and K. Ehrig, "Refactoring of model transformations," Electronic Communications of the EASST, vol. 18, 2009.

[17] K. Lano and S. K. Rahimi, "Case study: Class diagram restructuring," in Proceedings Sixth Transformation Tool Contest, TTC 2013, Budapest, Hungary, 19-20 June, 2013., 2013, pp. 8–15.

[18] S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya, and P. Van Gorp, "Evaluation of model transformation approaches for model refactoring," Science of Computer Programming, vol. 85, 2014, pp. 5–40.


 ISBN: 978-1-61208-438-1 