Chapter 1
Experiences in Teaching Model Transformation with the QVT Language
Anna DEREZIŃSKA1
Institute of Computer Science, Warsaw University of Technology

Abstract. Model transformation is a fundamental concept of Model Driven Engineering (MDE). Different general purpose or domain specific languages are applied to model-to-model (m2m) transformation. Query View Transformation language (QVT) belongs to the OMG standards. Selected MDE issues are taught to graduate students in a module of an Advanced Methods for Software Engineering course (AMSE). The course is accompanied by laboratory classes, during which meta-models and model-to-model transformations in QVT are designed and practically applied. Despite limited time, simple tasks have been solved and successfully implemented by students. We report on QVT teaching and assessment, laboratory outline, task development, achievements and obstacles faced during several editions of the AMSE course.

Keywords. Model-Driven Engineering, MDE, Model transformation, QVT, Education

1. Introduction

Model transformations are essential activities within Model Driven Engineering (MDE) [1]. Different approaches and various languages are used for model transformations [2][3]. Query/View/Transformation (QVT) [4] is an OMG standard language for specifying model transformations. It contributes as one of core standards to the Model Driven Architecture (MDA). Despite several industrial applications [5] QVT is not widely used due to a lack of knowledge and partially due to a still unsatisfactory tool support. Deficiency of educated professionals is one of obstacles in further application of model-based approaches in practice.

Within teaching of MDE fundamentals there is much effort devoted to modeling with UML and to transformation of models into different forms, like program code, data base schemata, test scripts etc. Transformations to be applied are mainly prepared by developers of CASE tools, or similar experts. However, basing on this, further teaching of software modelling should cover the core technology with two main directions: metamodeling and model transformation, as stated by Bezivin [6].

A model transformation can be implemented in a general purpose language, e.g. Java, hence application of a known notation could have an advantage in education. In this paper, we focus on another approach, in which transformations are developed using a specialized language, namely QVT. It has been shown that despite a new

1 Corresponding Author. Institute of Computer Science, Nowowiejska 15/19, 00-665 Warsaw, Poland, mail: A.Derezinska@ii.pw.edu.pl
notation and limited time, students were able to practically develop simple QVT transformations.

Since 2008, a course of Advanced Methods for Software Engineering, in short AMSE, has been delivered in our Institute. The course is devoted to development of high quality software. Among others, it includes a module aimed at selected aspects of model engineering. Course modules are accompanied by laboratory classes. In a practical part of the MDE module, model transformations are developed by course participants. An approach based on the QVT language has been applied in the MDE laboratory since 2011.

This paper focuses on education of model transformation, with the stress on independent creation of meta-models as well as development and testing of model transformations written in the QVT language. We share some experiences in teaching general issues of meta-modeling and model transformation. We report about mastering practical skills in model transformation with QVT.

The paper is organized as follows. In the next Section, the QVT language is briefly introduced. We give basic information of the AMSE course in Section 3. Section 4 reports on laboratory classes with QVT and experiences gained. Assessment of other MDE issues is presented Section 5. We discuss related work in Section 6 and conclude the paper in Section 7.

2. Preliminaries of transformation with QVT

Query View Transformation Language (QVT) is a hybrid declarative/imperative language devoted to transformation purposes [4]. It is specified and maintained by OMG in cooperation with many organizations and companies. Its concise overview can be found in a Kurtev’s paper [7].

QVT is specified in conformance to the MOF (MetaObject Facility) standard [8], which is the OMG base specification in the model-driven engineering. MOF provides a meta-meta model that conforms to itself. It is also used as a reference model in various kinds of models and modelling approaches, including QVT transformations.

QVT architecture consists of a Core language, Relations language, and Operational mappings. QVT Core is a simple declarative language. Typically, it is not used independently but incorporated in fundamentals of other QVT facilities.

The Relations QVT language is used for declarative specification of the relationships between models. A transformation between models is described by a set of relations that must hold. Models conform to their model types, which are meta-models conforming to MOF.

The Operational QVT is an imperative language in which bidirectional transformations are to be written. Structure of QVTo programs is based on Operational mappings that can invoke other mappings.

In general, we can write operational transformations following an imperative approach, or combine relational transformations with imperative operations. It is typical that the same issue can be implemented in very different ways. Other advantages of QVT are its clarity and easiness of a transformation creation, which writing in traditional languages would take a longer time and more code lines. Moreover, modification of a transformation or its extension appears to be straightforward.
3. AMSE Course Outline

Main goals of the Advanced Methods for Software Engineering (AMSE) course are to: (i) make students aware of the need of assuring high quality software, (ii) comprehend by them selected methods of high quality software production, (iii) enhance skills in certain fields of design and implementation of high quality software.

The course syllabus covers software metrics, software refactoring including refactoring to design patterns; selected methods of software testing, e.g. testing with mock objects; aspect oriented programming, model-driven engineering, and design of formal specifications. The detailed subjects of the course may slightly vary in dependence of a course realization.

3.1. Course Participants and Course Credits

The course is attended by students that hold a bachelor degree of computer science, computer engineering, or an equivalent diploma. They are intended to get MSc degree in computer science. Prerequisites of participants are working knowledge of programming in an object-oriented language (such as C++, Java, C#) and basic knowledge of UML. During the first degree education, the students have attended a basic course on Software Engineering, including a project on modelling and design using the UML notation. Most of the students have developed a simple application based on transformation of UML models to code, e.g. Java. Apart from their educational background, almost all of the students have been working and gained some professional experience, mainly in programming using different general purpose languages.

The AMSE course is an elective course. Students have to select several such courses, according to their interests, specializations, or domains of their theses to be prepared. They are also obliged to collect a certain amount of ECTS (European Credit Transfer System) points after passing courses specified in the curricula.

The whole AMSE course takes 30 hours of lectures and 15 hours of laboratory classes with a supervisor. A student effort of the course is foreseen for about 120 hours, including lectures, tests, classes in laboratories, but also consultation, preparation of projects and self-study. This is counted as 4 ECTS points.

The MDE issues are only a part of the course, and this module can be assessed for 1.2 ECTS. The MDE unit contributes approximately to 36 hours of a student effort. It has been estimated as 8 hours of lectures and a final test, about 3 hours of a lecture preparation and consulting with a supervisor, about 3 hours preparation for the final test, and 4 obligatory hours in laboratory. The remaining 18 hours are aimed at maintaining laboratory tasks, in particular, making familiar with the subject and laboratory instructions, arranging a tool support in an own computer, realizing a project in laboratory and at home, completing a final report, etc.

3.2. MDE Part of the Course

Basic modeling concepts with meta-model ideas and examples are discussed during lectures on MDE. Intuitive and formal definitions of model concepts are presented. Students learn about relations between models and meta-models in a model hierarchy, such as representation and conformance. Different examples, based on simple UML and other IT areas are discussed. Students are provided with the basic knowledge about
MOF standard and selected meta-model examples from the UML specification. Different mechanisms of UML extension with and without meta-model modification are discussed. Principles of a profile creation and its application are illustrated with examples. The fundamentals of MDA with its viewpoints, platforms, and model levels are presented.

The rudimentary notions of model transformation are surveyed, together with basic transformation feature. Several approaches to transformation realization and transformation languages are examined and illustrated by simple examples [2][3]. The examples refer to relational and operational QVT, but also other approaches like model to text solutions, graph-based transformations, etc.

As for a selected transformation language, the most important features of the QVT Relations Language and Operational Mappings presented. Examples cover the basic language constructions, such as a transformation structure, operation mapping, when and where closures, declarations of objects, working on collections, passing of parameters, writing of helpers and query, usage of the self keyword, control statements and loops.

The final section of the MDE lectures is devoted to merging of models, treated as a special kind of transformations. Basic rules and common problems are reviewed and illustrated with examples of languages from Epsilon [9], which is a set of languages and eclipse-based tools supporting m2m transformation and other processing of models.

Overall, the lectures give a broad overview of transformation approaches and languages, while selected notions are discussed in more details. An important part is devoted to QVT, although it is presented as one of many alternatives and compared to other languages.

Apart from theoretical knowledge, the course is aimed particularly at the development of advanced practical skills. The following educational goals have been appointed to be undertaken in the MDE part of the course:
A. comprehension of prepared meta-models,
B. modification (e.g. extension, adaptation) of a given meta-model according to certain guidelines,
C. evaluation of a model conformance to a given meta-model,
D. design and creation of a meta-model based on a given domain description,
E. creation of a meta-model derived from a representative set of models,
F. comprehension of transformations prepared by other parties,
G. usage of transformations in model-driven engineering,
H. evaluation of transformations in conformance to meta-models of an input/output model,
I. modification of a given transformation,
J. design and implementation of a transformation that completes a specified task using source and target models conforming to the same or different meta-models,
K. inspection and testing of a given transformation, design of representative models to test a transformation in regards to meta-models of source and target models.

The number of participants in each course edition was not very big, ranging from 11 to 27 (Table 1.), therefore, it was a chance of an active involvement of students also during lectures. They asked questions, compared different approaches, discussed and solved simple examples. However, those activities referred only to a part of students...
who were more interested in the subject. Nevertheless, all students were obligated to attend laboratory classes during which above goals have been taken on.

Table 1. Evaluation results of MDE part

<table>
<thead>
<tr>
<th>Year</th>
<th>Number of participants</th>
<th>Laboratory tasks</th>
<th>Final test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean</td>
<td>Standard dev.</td>
</tr>
<tr>
<td>2011</td>
<td>13</td>
<td>7.9</td>
<td>1.9</td>
</tr>
<tr>
<td>2012</td>
<td>14</td>
<td>8.3</td>
<td>2.7</td>
</tr>
<tr>
<td>2013</td>
<td>26</td>
<td>9.3</td>
<td>0.4</td>
</tr>
<tr>
<td>2014</td>
<td>21</td>
<td>6.8</td>
<td>1.9</td>
</tr>
<tr>
<td>2015</td>
<td>11</td>
<td>8.9</td>
<td>2.5</td>
</tr>
<tr>
<td>2017</td>
<td>24</td>
<td>8.6</td>
<td>1.8</td>
</tr>
</tbody>
</table>

2011-2017 | 109 | 8.3 | 1.7 | 8.3 | 4.1 | 2.1 | 3.7 |

4. Laboratory on Model Transformation

One of the general goals of the laboratory classes was demonstrating of the advantages and disadvantages of a transformational approach. We focused on an illustration and independent realization of a model-to-model transformation process.

In laboratory tasks, operational QVT was used. In general, it was simpler to comprehend than relational QVT, as students had been more used to write programs in imperative languages and scarcely in declarative ones.

Prerequisites of model transformation were comprehension of meta-modeling concepts. Therefore, most of laboratory classes started with an introductory test revising meta-modeling concepts and putting them into praxis. Three kinds of main activities were performed during laboratory classes:

- Introductory test on meta-modeling
- Tutorial - preparation of a given meta-model, a given transformation, and representative test models
- Independent realization of a whole m2m process for a given problem.

4.1. Laboratory Environment Setup

Considering tools for MDE, and especially transformation in QVT, availability of tools is much smaller than for UML. Starting in 2010, we selected a set of the following most mature tools: UML ModelTransformation Tool (UML-QVT) v.0.8, SmartQVT v.0.22, Eclipse M2M, QVTParser, and Borland Together. The tools were compared taking into account the following aspects: supported transformation languages, formats of model saving, software license, application type (stand-alone, plug-in), year of an origin, an organization/company managing it, a project web page, a framework or an operating system. Moreover, after performing a set of experiments with the mentioned tools, they were subjectively assessed by a user in two categories: project maturity and GUI friendliness. Evaluation results of the consecutive tools mentioned above, except Borland Together, are the following: project maturity (3, 6, 5, 4) and GUI friendliness (3, 6, 4, 7), where an 1-10 scale was assumed.

This comparison showed that at that time, the best tools for QVT transformation were associated with the Eclipse development platform. During a preliminary course, transformation tasks with the voluntary students were performed using SmartQVT. However, this project was abandoned and some members of the development team
jointed the Eclipse M2M community. The Eclipse M2M engine was also used in the Borland Together.

Therefore, since 2011 we have been using Eclipse with the Eclipse Modelling Framework and Operational QVT SDK during the regular AMSE laboratory classes. The Eclipse versions followed the currently available environment, starting with Helios in 2011, until Mars. We kept on with the M2M changes, introducing slight updates to an installation and usage instruction delivered to students, if necessary. Although conceptually the main flow of the process used during tutorials and further student tasks has remained the same.

In teaching of basic modeling with UML, it is often posed a dilemma about using professional but heavyweight tools versus simple ones. Application of a simple tool, especially during preliminary courses, gives an advantage not to waste too much attention on the technological issues and concentrate on the main concepts. Here, advanced students were more willing to work with commonly used tools, or based on common, familiar interfaces. They tend to prefer professional tools, which are likely to be faced at their work, or at least learning of skills that have a direct impact on their current or future job.

4.2. Laboratory Workflow

Laboratory classes consist of two meetings lasting two hours each. Before the first meeting, all lectures concerning MDE subjects are delivered. Moreover, students are supplied with materials explaining laboratory tutorials, a way of working with meta-models and QVT transformations in a selected environment. During the first meeting, optionally after a short introductory test, a tutorial is realized.

Later, proper tasks to be prepared are distributed. One or two transformations are to be designed, implemented, and tested by each student. Transformations are based on the meta-model used during the tutorial and/or other specified meta-models. The tasks are similar, but as a rule different for each student.

Students have one or two weeks before the next laboratory meeting. In this time, they can prepare their solutions using their computers (an instruction how to prepare and use the environment is given in the laboratory materials). In the meantime, they can also work in the laboratory in their spare time without a supervisor.

During the second laboratory meeting, students have time to complete and improve their solutions. They present their meta-models, runnable transformations, tests of models, and transformed models to a supervisor. They are asked to improve inconsistencies or incompleteness in transformations, if any detected. It is very often, that it is necessary to create more complete test models to test a transformation. For example, models used for testing are intended to cover all element, artifacts and relations, of meta-models of concern.

After presentation and discussion of results, all prepared artifacts with the final documentation are provided to a supervisor to a final evaluation.

4.3. Laboratory Instruction

A laboratory instruction includes detailed guidelines how to use a tool in the laboratory and how to install it in own computers. Moreover, a detailed tutorial is also given, covering building of a given meta-model, installing it as a plug-in to the environment, verifying whether the meta-model extends the environment, creating a model consistent
with the meta-model, development of a given transformation, and running of the transformation on a test model. In the tutorial, all steps and all selected options, included data, possible messages, etc. are exactly specified.

Apart from the instruction, during the tutorial realization (Sec. 4.4) students were assisted by a supervisor and cooperated with each other to clarify any doubts.

Students were also provided with a concise QVT description. It was based on the QVT specification but concentrated on the most important language features. Students could also make use of some examples that showed typical application of the language.

4.4. Laboratory Tutorial and Tasks

Students have to complete an MDE process staring from creating a meta-model, writing a transformation working on the meta-model, building test models and testing the transformation. During the tutorial phase, a meta-model and a transformation given in the course materials are used. The main goal of this first laboratory meeting is to make them familiar with the process realization, tool support, practical application of MDE concepts, running of transformations and preparing of test models.

The tutorial task was based on a simple class meta-model. A package can contain other packages and classes. A class has a name, and other characteristics. It can contain other classes, attributes, and methods. A simple tutorial transformation changes names of classes. It works and creates an output model, but on purpose it does not preserve a structure of aggregations among packages and classes. With such a basic example, students learn to apply different models to test transformations and reveal their weaknesses.

Furthermore, a draft of another meta-model was given that describes a university domain familiar to students. For example, many persons constitute a crowd, a person could be a student, a student attends a university, a person has documents, student has index with marks, etc. Based on this a participant had to build a meta-model that covered at least all necessary elements and relations to complete a given task, such as:

- Transform a person with valid documents and of a given age into a student associated with a certain university and provided by a student identification number.
- Transform each student fulfilling given formal requirements and having a desired average mark of all courses into a graduate student with a bachelor degree.

4.5. Laboratory Challenges

Students were encouraged to complain and assess the tool support during the whole process. Most complaints were focused on preparation and application of meta-models, partially because of an insufficient tool support in verification and detection of errors in meta-models. Errors of this kind were sometimes manifested only during preparation or running a transformation. In some cases a model creation crashed giving no reasons why. Any visible mistakes have been discovered in a meta-model, hence the fastest solution was to rewrite it once again. One the other hand, students observed that preparation of a proper meta-model is an activity performed once, or rarely, whereas transformations can be written many times for a given meta-model.

Transformation programs were not very long, but support for identification of errors was also not sufficient. In QVT it is allowed to write some functions on
collections that has no sense. A programmer is not warned about it although types of collections are defined. The basic criteria of test case generation were coverage of meta-model concepts, contract specification of transformations, and coverage of transformation code. Students could observe insufficiency of use only ad-hoc test models instead of a set of representative models covering all concepts of meta-models. However, development of comprehensive test models can lead to many test cases or complex test models in case of a complicated meta-model. Students complained about a lack of support for creating and running unit tests for QVT.

QVT has a flexible syntax, in which typical notations originated from different commonly used programming languages are accepted. For example: comparison operator as "=" or "==", checking of inequality with "<>" or "!="., comments limited by /*..*/, //…, or --, etc. These are details, but altogether have made the language easy to use by students who are not spending much time to study it but try to write code based on representative examples and provided basic language principles. This is true under an assumption that those students have already been familiar with general purpose languages like C, C++, Java or C#, have been acquainted with the SQL notation, and even have tackled some specification notations, like OCL, Alloy, or functional languages as Haskell, etc.

An important observation is that running of a transformation is similar to running another program, which is a well-known activity for students. They noticed a strength of the language and pointed out that problems originated from different domains can be solved with the same transformational notation. They were also aware of potential opportunities for adopting transformations to their professional needs.

Students were often positively surprised that a concise and meaningful transformation code can accomplish required tasks. After having evaluated correct transformation code of their tasks, some students made the code refactoring finding solutions that were well designed and more compact.

5. Assessment of the MDE Module of the Course

There are four main sources of student contributions used in an assessment of the MDE part of the course:

1. Student activity during lectures and tutorials (0-2 points)
2. Introductory test during laboratory classes (0-2 points)
3. Tasks prepared, run and documented in labs (8-9 points)
4. Tasks solved during final tests (4-8 points per task)

Given points are used for calculation of a final course score, whereas there are 60 points for the whole course.

Before starting a laboratory class, an introductory short tests can be performed. Tests of this kind have a low impact on the overall evaluation, as they were intended to revise basic concepts of meta-modelling before starting their practical utilization. Two exemplary tasks are given below.

A. Draw an exemplary meta-model to which conforms a given model A (Figure 1).

B. For a given meta-model X (Figure 2) describe a set of models that conform to it. Does model B conform to X? Justify the answer.

In the task A, although a possible solution is conceptually very similar to those discussed during lectures, the first attempt to design a meta-model caused several
problems. In general, it is much more straightforward to use meta-classes for named elements and graph nodes, e.g. classes, their attributes, than for unnamed relations.

![Figure 1. Model A](image)

In the final tests, tasks referring to the MDE part have covered two main areas. One category of tasks deal with different forms of models and meta-model design, e.g.:

- creation of a meta-model, while having a model expressed as a meta-model instance,
- verification of a model conformance to a given meta-model,
- finding of inconsistencies, missing elements in a meta-model; designing an extension of a given meta-model.

Second type of tasks concerns creation and inspection of a transformation written in different notations, mainly QVT, for example:

- for a given meta-model and transformation, explain the meaning of particular statements/procedures in the context of meta-model elements,
- extend a given transformation with an additional helper or mapping that is used in the transformation,
- complete a given transformation, in which missing statements are denoted by code gaps,
- write a simple transformation using a given meta-model(s).

Examples of tasks to be solved during a final test are given below:

A) For a given meta-model $K1$ (Figure 4) and model Job (Figure 3) specify:
   - all relations between model elements and corresponding meta-model components,
   - all missing elements of the meta-model
   - other inconsistencies of the meta-model

   Improve the given meta-model, assuming the model Job is correct.

B) Extend the improved meta-model (result of point A) in order to specify with named stereotypes such elements as Employer, position, define_Rate.

C) Develop a transformation of models that conforms to the extended meta-model (result of point B). The transformation adds a suffix ‘S’ to names of model elements that have stereotypes. The transformation has to be written in the relational or operational QVT. All statements have to be commented.
During a final test, students are allowed to use additional reference materials, lecture notes, especially manuals specifying syntax and semantics of the QVT language. They are not allowed to communicate with each other or to surf on the Internet.

At the beginning of the MDE classes, typical problems faced by students correspond to comprehension of fundamental meta-modeling ideas, an application of meta-models in a conscious manner, ability of discriminating between different levels of concepts, e.g. of current models, meta-models, meta-meta-models. Those problems were partially overcome after laboratory classes.

The following errors in the domain of meta-modeling encountered in final tests:
- mixing of concepts from a model and meta-model level (17%),
- incomplete identification of conformance relations between model elements and corresponding meta-model entities (17%),
- detection of not all missing elements of meta-models (36%),
- correction of meta-model mistakes in a wrong way (34%),
- incorrect incorporation of a stereotype into a meta-model (20%)

The number denotes a percentage of tests in which a mistake of this kind encountered. The following types of errors in QVT transformations were observed in final tests:
- improper reference to meta-elements (10%),
- fault in domain of an operation (10%),
- missing part of a transformation, e.g. mapping, handler (10%),
- lack of a transformation code (10%),

Results of the MDE part of the course are summarized in Table 1. The results are given for the laboratory and final tests separately and normalized to 10 points maximum. In general, scores of both contributions highly depended on a student...
attitude towards the subject. In case of the laboratory task, another crucial factor was a student background.

Most of the students who completed laboratory tasks with at least satisfactory notes (i.e. obtained above half of points from the possible maximum number) coped fairly well with tasks from a final test. However, it was much easier to solve a problem, when an executable result could be visualized as an output model. Some students have difficulties in applying their knowledge in an abstract manner. Moreover, some tests, e.g. in 2015, were written before finalizing laboratory classes, which resulted in lower test results.

6. Related Work

There are many references to teaching of object-oriented analysis and design of systems with the UML notation, e.g. [10][11], as this is a fundamental subject in the software engineering area. Therefore, selection of tools to be used in education is also discussed mainly in the context of UML modelling and design, as well as application of build-in transformations in MDE [12]. Some experiences are also reported on advanced courses covering various aspects of Model-Driven Development [12]-[18]. However, there are no many reports on educational issues of the direct model transformation. Transformations are often applied in scripting languages, as PHP [18] or general purpose languages, as Java [19]. To the best of our knowledge few of them have reported teaching experiences in developing model transformations in specialized languages [12][12], QVT in particular [5].

Batory &Azanza discussed problems in using Eclipse Modeling Tools in MDE education, both in respect to installation and tool usage [19]. A version of Eclipse was eventually posted that had all tools installed. It was assessed that only 25% of the upper-division undergraduate class got their task right, 50% had mediocre results, and the remaining gave up the task. After an additional time supported with tutorial help, 80% of students completed right the task. Based on these experiences they developed a new MDELite tool that intends to overcome those inconveniences [20]. There are no enough details to thoroughly compare this case to the ours, and the approach was focused on different subjects, namely relational databases. During our laboratories, we also used Eclipse tools. Students were supported by very detailed instructions prepared by other students and a supervisor. In general, we do not have faced problems with a software installation. The main problems were comprehension of basic concepts and recognizing of any benefits of a transformation usage. Postgraduate students that participated in our courses might be more experienced, including their work in IT area. The EMF environment was also used during the MDE course presented in [14].

An interactive approach to an MDE process combined with teaching of program translation techniques was presented in [12]. The whole course required a much bigger student effort than presented in this paper, hence transformation tools were prepared by course participants. It was emphasized the necessity of practical application of presented methods and trained skills. Although reporting about the profitable and prosperous course, the authors mentioned unpopularity of MDE among students.

A similar practical approach was described by Schmidt et al [18]. An MDE course was focused on creation a simple tool for code generation from UML models. No domain languages, such as transformation ones, are considered, as the whole process try to use tools already known to students.
A whole comprehensive MDE course is also discussed in [21], although no details about teaching QVT are given. The authors faced difficulties with concept comprehension, meta-modeling in particular, using new languages (QVT including), and insufficient tool support.

Brosch et al. reported their experience in delivering an advanced modeling course [16]. This course partially covered the similar subjects as the MDE part of AMSE, but the whole course was devoted to model engineering, therefore, more class units covered meta-modeling, and transformations were discussed with more details. Students completed laboratory tasks, during which they created meta-models with Eclipse Modeling Framework (EMF), defined constraints in OCL, developed an ATL transformation, and performed code generation to SQL and Java. In general, the authors were satisfied with their results and have similar experiences to ours, although complained about immature MDE tools. However, neither during the lectures nor in practical tasks they did use any type of QVT.

A common problem in modeling evaluation is an assessment of a model quality. It is partially done by a manual inspection of a supervisor, which is subjective and laborious. Application of an executable modeling language with continuous validation helped to solve this problem in a modeling course [17]. However, using a model transformation, students can directly run their solutions and observe the results. The main validation problems were moved over to the creation of representative test models. Testing of transformations is surveyed in [23].

A quality model of a QVT transformation is presented in [5]. Authors discussed a set of best practices, quality metrics, as well as a tool support for the model transformations in QVT, but also AMT (with the similar constructs to QVT) and Java (the mostly used for model transformation). The quality model is an interesting proposal, especially for big industrial transformations. In small transformations developed during an introductory course, only a subset of the quality properties could be applied. Testing of transformation can be improved by the QVTo code coverage support that was implemented for Eclipse [5]. It would be promising to incorporate these approaches into the future AMSE courses.

It is characteristic that novice developers tend to apply language patterns known from other programming languages, which results sometimes in good but sometimes in improper solutions. Similar was reported in [5], where reproducing of GPL program structure caused using a large init sections inside a QVT mapping, instead of a population section.

7. Conclusions

In this paper, experiences of introducing model transformation with QVT into an advanced software engineering course are presented. Basing on student knowledge of other programming languages and an extended tutorial, it was possible to successfully complete tasks with a new language in relatively short time. One of the most important obstacles in education of MDE issues was convincing students of any benefits of the transformation approach.

Teaching of transformation in general, illustrated with QVT applications in particular, could be thoroughly extended with more systematic and tool supported specification, verification, and design patterns. However, a still limited usage of transformations and their specialized languages in an industrial praxis made students
reluctant to a deeper study on these subjects, similarly to education about model-driven development or formal specifications. There is also an open issue whether it is more suitable to use a domain language, like QVT for a model transformation, or a general purpose language like Java.

Acknowledgments. The author would like to thank P. M. Zajączkowski for preparing a preliminary version of the instruction for the QVT laboratory within the AMSE course. Preparation of lecture presentations for the course was partially supported by EEA Norway grant in project of Education System Development Foundation.

References


