A Bottom-Up Quality Model for QVTo

Christine M. Gerpheide* Ramon R.H. Schiffelers*† Alexander Serebrenik*
*Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
c.m.gerpheide@student.tue.nl, r.r.h.schiffelers@tue.nl, a.serebrenik@tue.nl

†ASML N.V.
De Run 6501, 5504 DR Veldhoven
The Netherlands
ramon.schiffelers@asml.com

***Abstract*—We investigate the notion of quality in QVT Operational Mappings (QVTo), one of the languages defined in the OMG standard on model-to-model transformations. We utilize a bottom-up approach, starting with a broad exploratory study including QVTo expert interviews, a review of existing material, and introspection. We then formalize QVTo transformation quality into a QVTo quality model, consisting of high-level quality goals, quality properties, and evaluation procedures. We validate the quality model by conducting a survey in which a broader group of QVTo developers rate each property on its importance to QVTo code quality. We find that although many quality properties recognized as important for QVTo do have counterparts in traditional languages, a number are specific to QVTo or model transformation languages. Additionally, a selection of QVTo best practices discovered are presented. The primary contribution of this paper is a QVTo quality model relevant to QVTo practitioners, while secondary contributions are a bottom-up approach to building a quality model and a validation approach leveraging developer perceptions to evaluate individual quality properties.**

I. Introduction

Model-driven engineering (MDE) can be used to develop highly-reliable software, offering benefits from analysis to code generation [1]. In MDE, models are created by domain experts and then transformed into other models or code using model transformations. One language for implementing model transformations is QVT Operational Mappings, a.k.a. QVTo, specified in the 2007 Object Management Group (OMG) standard for model-to-model transformation (MMT) languages [2].

As a standard, QVTo is used today in both academia and industry. In particular, ASML [3], the leading provider of complex lithography systems for the semiconductor industry, uses QVTo as their primary language for implementing MMTs. Currently ASML has tens of thousands of lines of QVTo code supporting activities from real-time schedule analysis to servo controller initialization during machine startup [4].

While for general purpose languages (GPLs) developers have built up many common notions to judge whether a piece of code is high- or low-quality, such shared best practices and quality indicators do not yet exist for QVTo. Given the many specific language features of QVTo, it is even unclear whether quality properties for traditional languages apply to QVTo at all. This lack of standardized and codified best practices has already been identified as one of the largest current challenges in assessing model transformation quality [5].

Therefore, in this research we investigate what code properties, best practices, and in general quality concerns currently exist for QVTo. To maximize the relevance of our research to industry, a focus on practitioners is maintained. We therefore explicitly adopt a *pragmatist* stance [6], which states that knowledge is judged by how useful it is for solving practical problems. This stance drives many of our design decisions as well as our validation strategy.

We begin with an introduction to QVTo and software quality in Section II. To guarantee that the notions of quality we identify are relevant for QVTo practitioners, we follow a *bottom-up approach*, described in Section III, starting with a broad exploratory study including expert interviews, a review of existing material, and introspection. We then formalize the exploratory study results into a quality model for QVTo transformations, presented along with a selection of best practices and difficulties in Section IV. The validation of our quality model is presented in Section V. Finally, we discuss additional related work in Section VI and conclusions and future work in Section VII.

The primary contribution of this paper is a QVTo quality model relevant to QVTo practitioners. Secondary contributions are first our bottom-up approach to building a quality model, and second, our validation approach, which we argue provides more convincing evidence than approaches from literature that the quality model is indeed useful for assessing QVTo transformation quality.

II. Preliminaries

A. QVT Operational Mappings

A QVTo transformation [2], [7] transforms models conforming to one or more metamodels. The primary constructs of a QVTo transformation are *mappings*, functions that map an input model element to an output element. Mappings

```
transformation UML2SysML(in source:UML, out target:SysML);
main() {
        // main: entry point of transformation
        source.rootObjects()[Class]->map Class2Block();
}
mapping UML::Class::Class2Block() : SysML::Block {
        // implicit population section start
        name := self.name;
}
```

Listing 1. Example QVTo transforming UML Classes to SysML Blocks

have three sections: an optional `init` section for object initialization, a `population` section for mapping input element fields to the output, and an optional `end` section for post-processing. In addition to mappings, there are also *helpers* and *queries*, which take an arbitrary input and return an output. To support more complex transformations, QVTo combines declarative concepts (e.g. OCL [8] for model traversal) with imperative concepts (e.g. `forEach`-loops). Furthermore, many reuse mechanisms are supported, such as mapping inheritance and storing oft-used mappings in libraries. A QVTo code example is shown in Listing 1.

B. Software Quality

Software quality can be approached from many perspectives [9], some of which we adopt *a priori*. First, we are primarily interested in *internal* quality, which means measuring quality by looking inside the software product (e.g. analyzing source code), rather than measuring the software during execution (e.g. testing) [10]. We also focus on *direct* measurements of transformation quality, which measure the transformation itself, rather than *indirect* measurements, which measure other artifacts involved in model transformation (e.g. models) to assess transformation quality [11]. We chose these perspectives because they are most likely to provide insights immediately useful to produce higher-quality transformations, our goal with the pragmatist stance.

Internal quality is often formalized in a *quality model*. According to ISO/IEC 25010 [12], the standard on system and software quality, a quality model contains high-level quality characteristics, quality attributes, and evaluation procedures. The standard also presents a quality model for software products, containing all characteristics and attributes relevant to software quality in general. The top-level quality characteristics defined there are depicted in Figure 1. Evaluation methods, however, are not provided in the standard product quality model, so it cannot be used directly to assess code quality. Furthermore, although the quality model is comprehensive, it is not *a priori* clear given the unique range of features and applications of QVTo that it is optimal for QVTo in practice.

Exploring quality in MDE, van Amstel, van den Brand, and Nguyen [13] defined a set of metrics to evaluate model transformation quality, including QVTo transformations. These metrics included size metrics, function complexity metrics, modularity metrics, inheritance metrics, dependency metrics, consistency metrics, input/output metrics, and QVTo-specific metrics. However, because these metrics were proposed based on language features and prior research for GPLs, it is not evident that this set of metrics is useful to assess QVTo quality. Van Amstel [11] later provided an evaluation of these metrics with respect to maintainability for ATL and ASF+SDF. There, an average of three experts rated thirteen code samples on seven quality characteristics. The expert ratings were then correlated with metric measurements. For ATL no significant correlations were found, and for ASF+SDF, while some significant correlations were observed, the high correlations between metrics made it difficult to assess the impact of individual metrics. Identifying individual metrics is particularly important because having too many metrics is one of the reasons metrics programs fail in practice [14], [15]. Therefore, although quality models have been proposed that are applicable at least in theory to QVTo, there is little empirical evidence that these models represent a useful notion of QVTo quality. We call these approaches starting exclusively from theory *top-down* approaches.

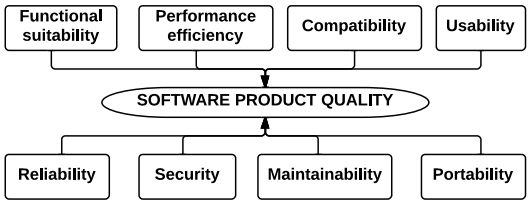

Fig. 1. Quality characteristics for software products in ISO/IEC 25010

III. Approach

To identify the most important aspects for QVTo quality, we follow a bottom-up approach based in grounded theory [16]. This way, the most *pressing* concerns in QVTo quality are captured, which are most important according to our pragmatist stance. Furthermore, utilizing a bottom-up approach already provides high empirical validity [17], particularly valuable in domains where validation is difficult. Our approach consists of two phases. The first is a broad exploratory study to gather substantial support for what affects quality in QVTo. The second phase is formalizing the information gathered into cohesive format which can be used to assess QVTo transformation quality.

A. Exploratory Study

The exploratory study focuses on gathering *qualitative* data. Since producing high-quality code heavily depends on developer actions, namely, how developers write code, qualitative data can provide the richest insights into the complex reasoning processes that developers use [16]. Since qualitative methods are sometimes considered "fuzzier" than quantitative methods in computer science [6], care was taken in selecting methods and conducting the study, leveraging in particular the extensive experience with qualitative methods from social science research. To guarantee that the exploratory study produces generalizable data and builds upon previous research and experience, we utilize a triangulation approach: expert interviews, a review of existing material, and introspection.

1) Expert interviews: Four semi-structured interviews of QVTo experts at ASML were conducted. By gathering data directly from developers, we gain access to potentially years of best practices and learning involved with becoming an experienced programmer. Additionally, by speaking with developers directly, we can quickly obtain insights into which aspects of QVTo are most important for quality today. As with all interview research, the data gathered in this component of the exploratory phase may be very specific to the individuals interviewed. Mitigating this bias while still leveraging rich

developer experience is a primary motivation for our triangulation approach.

State-of-the-art techniques for interviews were followed, in particular recommendations from Seaman [16] and Hove and Anda [18]. Each interview lasted 45-60 minutes and was audio recorded and later transcribed. A one-page interview guide (available in [19]) was used by the interviewer to help direct the interview, containing questions about development processes, typical bugs and refactorings, development guidelines, differences in developing QVTo versus other languages, as well as poor practices exhibited by those learning QVTo. To assess the interview format, a mock interview was performed with an independent developer before-hand.

Examples of information gathered during the interviews are that common refactorings include converting imperative-style programming to declarative (since the latter is more readable as well as faster in execution), that a transformation is easiest to work with if its structure mirrors the hierarchy of either the input or output metamodel (but not both), and that unit testing is used extensively by the team but there does not exist any measure of test coverage.

2) Existing material review: To leverage the vast amount of knowledge on code quality, an extensive review of existing materials was performed. This review included not only literature, but also other sources such as online forums. The latter are particularly important since the amount of scientific literature on QVTo is still limited and much insight can instead be gained from online sources.

For literature, a *systematic literature review* [20] was performed with search terms (including synonyms thereof) "best practices", "quality", "metrics", "limitations", "difficulties", "tools", and "testing", in combination with technology terms "QVTo", "QVT", "ATL", "model transformation", "transformation language", "Java", and "code". ATL and Java were chosen since they are also prominently used for model transformation. The technology terms also served as exclusion criteria for literature found, where search queries already returning a great deal of literature for technologies closest to our focus terms (e.g. "QVTo" or "model transformation") were not pursued for the more distant technologies (e.g. "Java" and "code"). Rich online sources reviewed were the forum for the Eclipse QVTo implementation [21] and the publicly-available comments from the Transformation Tool Contest [22].

Examples of data gathered during this review are that mixing notations (e.g. text, graphical) can reduce understandability, and that a number of metric sets have been proposed in model transformation literature (e.g. [23], [24])

3) Introspection: The third component of the exploratory study was learning QVTo by the first author. Here, a series of QVTo tutorials and examples was followed (e.g. [1]). During the learning process all aspects related to quality were recorded, such as difficulties and realizations about better ways to implement certain functionalities. Since it is important for software engineer interviewers to be knowledgeable of the domain at hand [18], this component was performed before the interview component of the triangulation approach.

An example of data gathered during introspection is that a novice tends to create large `init` sections inside mappings to initialize the output elements, since that is similar to GPL programming, as opposed to using the more concise implementation with a `population` section.

B. Formalizing the Quality Model

With the qualitative data from exploratory study, we perform what is known in classic theory generation as the *constant comparison method* [16]. All *key points* related to transformation quality are extracted, approximately one sentence per point. Each point is *tagged* (or *coded* [16]) with keywords describing why it is relevant for quality, e.g., "conciseness", matching the wording used in the raw data as closely as possible. Then an iterative approach is applied where similar points are grouped together, while always maintaining traceability links back to the original sources. The set of quality tags are then reduced by combining closely related tags, e.g., "readability" was combined with "understandability".

Until this point, to avoid researcher bias, it was not assumed that a formal quality model would be the result of our investigation of QVTo quality. It became clear however that most of our key points and tags could be effectively formalized within a quality model. The quality tags became the *quality goals* of our model, representing the primary quality concerns for QVTo transformations today. We then translated each point into a short phrase. These phrases comprise the *quality properties* of the QVTo quality model. To give an indication of whether the property helps or hurts quality, we also assigned a *directionality* to the property. Directions were chosen so that every property generally increases transformation quality. To accommodate for quality goal tradeoffs, however, each quality goal associated with the property was also marked with a direction. For example, "Using mappings instead of helpers increases understandability, but can hurt performance due to the additional tracing added by the engine" was converted to property "More mappings than helpers" with goals "Understandability (+)" and "Performance (-)". We use the term property rather than attribute because attributes generally do not have directionality [12].

Quality properties with insufficient support from our triangulation approach were then discarded. Specifically, properties with fewer than two original sources (for example being mentioned by only a single paper) were removed from the model. Properties which were only proposed in literature with no validation were also removed. This latter step is required by our pragmatist stance to avoid including properties for which there is still no evidence of usefulness in practice. To complete the model, evaluation procedures are added for each property. An elaboration of our approach is given in [19].

IV. Results

In Table I the QVTo quality model resulting from the exploratory study is presented. It consists of 37 quality properties and 4 quality goals, namely Functionality, Understandability,

Performance, and Maintainability. Although the list of properties may seem large, our validation approach (Section V) distinguishes their relative importance. The properties have been classified as specific to QVTo, MMT, or neither, in which case they are also applicable to GPLs. The proposed classification is based on the traceability links from our quality model, therefore reflecting why the property is in the model. For example, "Deletion uses trashbin pattern" is classified QVTo-specific because it relates to the QVTo implementation used by the interviewed developers. According to our classification scheme, 13 properties are QVTo-specific, 4 are MMT-specific, and 20 are applicable to GPLs.

The quality properties have been further organized in Table I according to their nature. Two of the properties are presentation-related, since they focus on style and are unrelated to transformation behavior. Nine of the properties concern the high-level architecture of a set of transformations modules in a project. Four properties are related to the current QVTo engine implementation, and therefore may differ between implementations. For example, "Few queries with side-effects" is implementation-dependent because in the Eclipse QVTo implementation queries are implemented as helpers, therefore allowing side-effects, despite this not being allowed by the QVTo specification. The remaining 22 properties can be considered quality properties local to a transformation, since they are specific to a how a single transformation has been written. This categorization helps determine when and where each property could be leveraged to improve transformation quality. For example, if a developer has the opportunity to modify a transformation but is not able to re-architecture his or her project, then the transformation-local properties may be most useful. As another example, if a new version of the QVTo engine is released, it may be necessary to reevaluate the implementation-dependent properties. The high proportion of transformation-local properties is likely a result of our approach, since existing quality models also have high proportions of properties local to a single module or class (e.g. [13], [23]) and the developers interviewed work also most frequently on modifying single transformations. We do not consider the effect of our approach on the properties a threat to validity, however, since our approach was designed to capture the most pertinent concerns in QVTo quality. Instead, this suggests that the most useful model to assess QVTo quality contains a variety of properties but with an emphasis on transformation-local concerns.

It is clear from the quality model that understandability and maintainability were the most ubiquitous (though not necessarily most important) quality goals for QVTo practitioners. This is expected, however, due to our focus on internal quality and that understandability and maintainability are more complex goals to describe than performance or functionality, therefore requiring more properties. Furthermore, although performance may be most related to external quality, it is still incorporated in our model because it was clear from our exploratory study that any description of QVTo quality without a consideration for performance would be incomplete.

Some properties in the model are of particular interest. For example, "Little imperative programming" may seem unintuitive, since QVTo was specifically designed to include imperative constructs. However, our exploratory study suggests that although these constructs are indeed useful for creating more complex transformations, where the alternative would be using a black-box containing Java code, they should be minimized where possible. In the experience of one of the developers interviewed, "eighty percent of transformations can be written in a purely declarative fashion, making them much easier to understand and work with later on". Property "Mappings only used when tracing needed" is also notable, since although mappings are a core concept of QVTo and avoiding them decreases understandability, it emerged as more important to improve performance, therefore serving as an example where a quality model not considering execution performance may be less useful for developers.

That the model was built bottom-up can also be seen. For example, the property "Small `init` sections" is included, while for `end` sections, "Few `end` sections" is included. This reflects that according to our exploratory study, the *amount* of code inside `init` sections affects quality, whereas for `end` sections it was only suggested that the *frequency* of use may affect quality. Recall also that no direct attempt was made to cover QVTo language features in the exploratory study, so the QVTo-specific properties included the model are there because they emerged as relevant to quality during the triangulation approach.

A. Model Scope

Because our approach was not restricted to specific types of QVTo transformations, we assert that our model is relevant to all strata of the model transformation taxonomy proposed by Mens and Van Gorp [26] that are applicable to QVTo. Specifically, the quality model can be used to assess quality of QVTo transformations with one or multiple source and/or target models. Since QVTo supports source and target models having different metamodels, the quality model can be used on both endogenous and exogenous QVTo transformations. Similarly, the model can be used to assess both horizontal transformations (i.e. performing refinements or abstractions) and vertical transformations (i.e. which do not change the level of abstraction). The experts interviewed also worked regularly with all of these transformation types. However, we do not claim that the quality model is applicable to MMT languages other than QVTo without further study.

We also place our model in the context of the transformation *intents* proposed by Amrani et al. [27]. Since QVTo could be theoretically used for each of the 17 most common intents identified there, the quality model proposed here is also applicable to transformations with those intents. Of the 17 intents, however, the experts interviewed had less experience in working with transformations addressing the "approximation" and "model generation" intents. Due to the importance of the interview data in constructing our quality model, we therefore

Quality property[1]	Quality goals	Evaluation procedure	Applicability
		Presentation	
Detailed comments throughout code	U+	Comment/LOC ratio	GPL
Formatting conventions followed	U+	# Violations of a coding standard	GPL
		High-level architecture	
Few blackboxes	U+, M+	# Blockboxes	QVTo
Few configuration properties	U+, P-	# Configuration properties	QVTo
Few dependencies on other modules	U+, M+	Module fan-out	GPL
Few input/output models	U+, M+	# Input/output models and metamodels	MMT
Few intermediate properties	U+	# Intermediate properties	QVTo
Low code duplication with other modules	M+	# Instances where at least five lines repeated	GPL
Pre- and post-conditions specified	F+, M+	Presence of formal specification	MMT
Small interfaces to other modules	U+, M+	Function fan-out to other modules	GPL
Small transformation size	U+	Module LOC	GPL
		Transformation local	
Confluence satisfied	F+	Proof of confluence	MMT
Few dependencies between functions	U+, M+	Function fan-out within module	GPL
Few `end` sections	U+	# `end` sections	QVTo
Few mapping arguments	P+, U+	# Arguments per mapping	GPL
Few nested `if` statements	U+	Nesting depth	GPL
Few `when` and `where` clauses	U+	# `when` and `where` clauses	QVTo
High test coverage	F+, M+	Test code coverage	GPL
High usage of design patterns	U+	Comparison of patterns used to a pattern catalog	GPL
Inheritance usage matches metamodel	U+	# Abstract classes in common with metamodel	MMT
Interdependent functions near each other	U+, M+	Custom semantic similarity measure	GPL
Little dead code	U+, M+	# Unused LOC	GPL
Little imperative programming	U+, M+	# `forEach` loops	QVTo
Little overloading	U+	# Instances of overloaded functions	GPL
Low code duplication within module	U+, M+	# Instances where at least two lines repeated	GPL
Low syntactic complexity	U+	Syntactic complexity measure [25]	GPL
Minimal reassignment of objects	F+	# Instances when object assigned to multiple sets	QVTo
More mappings than helpers	U+, P-	Ratio # mappings to # helpers	QVTo
More queries than helpers	U+, P+	Ratio # helpers to # queries	QVTo
Short function chains	U+	Length of chains	GPL
Small function size	U+	Function LOC	GPL
Small `init` sections	U+	LOC inside `init` sections	QVTo
Termination checked	F+	Proof of termination	GPL
		Implementation-dependent	
Few queries with side-effects	U+	# Queries with side-effects	QVTo
Deletion uses trashbin pattern	P+	# Instances where trashbin pattern not used	QVTo
Low execution time	P+	Execution speed with typical model	GPL
Mappings only used when tracing needed	U-, P+	# Instances when mapping used but not tracing	QVTo

TABLE I
QVTo QUALITY MODEL RESULTING FROM THE EXPLORATORY STUDY. CONTAINS QUALITY PROPERTIES, QUALITY GOALS (F, U, P, M FOR FUNCTIONALITY, UNDERSTANDABILITY, PERFORMANCE, AND MAINTAINABILITY, RESPECTIVELY), AND EVALUATION PROCEDURES.

we consider this a threat to validity when using our quality model to describe QVTo transformations with those intents.

B. Similarities to other models

The model most closely related to our QVTo quality model is the set of QVTo quality metrics proposed by [13], introduced in Section II-B. Of our 37 quality properties, approximately one third have corresponding metrics in their metric set (which coincidentally also contained 37 entries). Some of the similar metrics are nearly identical (e.g. metric "# Overloaded mappings" and our property "Little overloading") while others are similar but not equivalent (e.g. metric "# Trace resolution calls" and property "Mappings only used when tracing needed"). These similarities appear in each of the metric categories identified by [13], suggesting that our synthesis approach also succeeds in covering a similar range of concerns as a top-down approach.

The similar metrics, however, comprise the more straightforward properties from our model (e.g. "Few input/output models"). Our more complex properties (e.g. "Deletion uses trashbin pattern", "Interdependent functions near each other"), on the other hand, do not have counterparts in the metric set. In some cases, a metric may seem similar to a property, but as a result of the bottom-up approach, the property is sub-

[1]"Function" refers to a mapping, query, or helper. "LOC" is lines of code. "Module" refers to a transformation or a library. "Function chain" refers to a chain of function calls. "Confluence" is a property of declarative languages where output is independent of execution order. *Tracing* internally links input and output elements. Clauses with `when` and `where` specify pre- and post-conditions on mappings. *Intermediate properties* store transient data during execution, and *configuration properties* have global scope.

stantially more nuanced. For example, while the set from [13] includes the metric "# Abstract mappings", our model includes the property "Inheritance usage matches metamodel" because according to our study it was not simply the number of abstract mappings that affected quality, but the extent to which they represent the abstract classes in a metamodel. The properties in our quality model are also in general more concise than the metrics. For example, five individual metrics are proposed for number of unused mappings, helpers, parameters, and local variables, whereas our quality model simply includes "Little dead code", which in our model is evaluated by measuring lines of unused code. Conciseness is valuable since it again reduces the risk of overwhelming practitioners with too many metrics, as mentioned in Section II-B.

Notably, a number of properties identified as important during our exploratory study are not present in [13]. For example, "Little imperative programming" was recognized in each of our developer interviews in addition to some literature as important for maintainability and understandability, but has no counterpart in the metrics from [13]. Our model also includes more QVTo-specific properties (e.g. "Few `end` sections"). These may be excluded from [13], however, because there are simply too many language-specific constructs to add a property for each construct. This difficulty to distinguish the most important features of new domain is in our opinion a fundamental problem with top-down approaches.

In Van Amstel's quality model for MMT languages in general [11] (cf. in Section II-B), a number of additional quality attributes are present. Some are similar to our quality model, e.g., "Depth of inheritance tree" proposed for ATL which is closer to our "Inheritance usage matches a metamodel" property. However, due to the top-down approach, Van Amstel's metrics [11] describe relatively shallow properties compared to the rich data incorporated in our quality properties.

Kapová et al. [23] presented a set of maintainability metrics for QVTr. There a combination of "automated" and "manual" metrics were proposed. The automated metrics, like those from [13] and [11], are simple, including "# Local variables" and "# `when` predicates" (the latter of which corresponds directly with our property "Few `when` and `where` clauses"). The manual metrics, however, contained "Similarity of relations",[2] "# Relations that follow a design pattern", and "Type cut through source/target metamodel". The last in particular is similar in essence to our "Inheritance usage matches a metamodel" property, since it measures the match between metamodel elements and the elements addressed by relations. The metric is presented however only in the context of increasing metamodel coverage (chosen as a quality goal by the the authors), rather than contributing to understandability like our property. Because this property was voiced as important by multiple interviewees for transformation readability and understandability, its explicit link to understandability is a valuable addition to our model.

Therefore, while there is some overlap between top-down models and our quality properties, we advocate the use of bottom-up approaches in future quality research to obtain the data most relevant in practice.

[2]QVTr *relations* are similar to QVTo mappings

C. Conformance to ISO/IEC 25010

As an international standard, ISO/IEC 25010 represents a broad consensus for how to describe software quality. It is therefore valuable to show that our QVTo quality model conforms to the software product quality model. According to the standard conformance can be demonstrated either by using the standard models directly or by showing traceability links between the standard model and a tailored model. We demonstrate conformance using the latter method, showing the links from the software product quality model to our QVTo quality model. First we map our quality goals to the quality characteristics of the standard: Functionality is mapped to Functional suitability, Performance to Performance efficiency, Understandability to Usability, and Maintainability to Maintainability. In each case the reason for using a different name in our model is that these terms were more natural for developers. Notably, our model excludes the characteristics Compatibility, Reliability, and Security, and Portability. They are excluded since, according to our exploratory study, they are lesser concerns in QVTo development at this time. Finally, our quality properties can each be mapped to a quality attribute in the standard quality model by removing the directionality.

D. Best Practices

As a result of not restricting the exploratory study to building a quality model from the beginning, some data from the study can be formalized better as best practices than as quantifiable quality properties. Here we present a selection of best practices gleaned in particular from the expert interviews, constructed by performing the constant comparison approach using just the interview data. Although not validated in Section V and highly specific to the experts' experiences, these can be used as a starting point for other QVTo practitioners to formalize their own best practices. They also provide an example of the rich data gathered using a bottom-up approach.

First, navigation over models should be separated from mappings as much as possible, ideally by creating a query library for each metamodel. Second, `init` sections are appropriate only when objects need to be *explicitly* constructed, e.g., selecting the concrete type of an object from the abstract type. Third, when multiple objects must be generated and assigned from one input object, a `constructor` should be used instead of the assignment operator. Finally, if performing incremental transformations with large input models, the traces can be utilized to avoid unnecessary regeneration of the target model.

E. Difficulties

In addition to best practices, many current issues were identified which can serve as areas for future work in QVTo research. First, lack of documentation as well as lack of (non-buggy) tooling were both identified as major problems. Eclipse

editor support could be improved, for instance preventing accidental reassignment of objects. There are also many limitations in the debugger, such as poor support for chains of transformations, so developers often default to using `log` statements instead. Testing frameworks are still considered immature, and in particular no measure of test coverage exists. Finally, there are also inconsistencies between the QVTo specification and Eclipse implementation used by many developers, for example side-effects being allowed in queries, blackboxes not being supported when running a transformation standalone, and not supporting deep recursive calls.

V. Validation

Although the bottom-up approach guarantees initial support for each property, validation is still required for confirmation and to show generalizability. Before introducing our validation approach, however, we discuss the drawbacks of the approach used in the most closely-related work (e.g. by [11] and [28]). There, transformation quality models were validated by conducting a survey where experts rated code samples on quality goals, which were later correlated with the measured quality metrics. In addition to the conclusions being drawn from this validation approach being generally weak, it is clearest to see its drawbacks by simulating it on our quality model.

We observe that in this method, experts are required to make snap judgements based on a visual sample of the code. This method therefore first strongly biases judgements towards visual properties, such as properties related to readability. Therefore properties related to other quality goals (e.g. "Termination checked") could incorrectly receive lower correlations. Second, properties which cannot be deduced visually at all (e.g. "Low execution time") or cannot be deduced *quickly* (e.g. "Few dependencies to other modules") would likely go unnoticed by the reviewer. Furthermore, there are simply so many properties that it is not possible to control for each individually, making it impossible to evaluate the relative importance of the properties; to do so would require preparation of an extremely large set of code samples for experts to review, which is infeasible within reasonable time constraints. This effect would in particular lead to lower correlations for less-common properties (e.g. "Small `init` sections"). Hence, a large portion of our quality model would likely go unvalidated if we were to use this method, regardless of whether the properties are important for QVTo transformation quality.

We propose a different validation approach for our quality model. With our assumption that developers themselves are one of the best resources to determine what high-quality code is, we validate our quality model by performing a survey of developer perceptions of each quality property. Using perceptions is based on the pragmatist stance, directly leveraging practitioner experience. By addressing each property individually, we assess the model on a very fine-grained level.

The validation survey consisted of an introduction to the research, instructions, a field for the developer to describe their experience with QVTo, definitions of each quality goal, and finally, a question regarding each quality property. Each

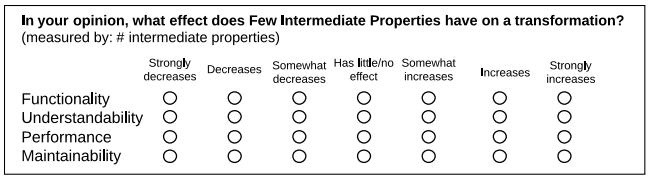

Fig. 2. Example question from validation survey

property question was a compound Likert-style [29] question asking about the relationship between the property and the four quality goals. An example question is shown in Figure 2. All property questions were worded similarly, since although acquiescence bias (where respondents passively indicate responses which agree with the question) is a drawback of Likert scales, it is more important to have this bias consistently throughout the survey [30]. After each question was a field for comments, and for less-known QVTo or programming terms a short explanation was provided. The survey instructions stressed that the developer should answer the questions in terms of typical QVTo transformations and should reflect their own experiences. A mock survey was carried out with an independent developer to test the survey clarity and completion time. The survey was distributed online by posting on the QVTo forum and the QVTo developer mailing list and the link was sent directly to the ASML QVTo developers.

Fifteen respondents filled out the survey, including the original four experts interviewed during the exploratory study and the two remaining ASML QVTo developers. The rest were from the general public, including three from industry, five graduate-level students, as well as one of the four primary committers of the Eclipse QVTo implementation. Each respondent had significant experience in QVTo.

The survey results are summarized in Table II. To measure overall agreement, we calculated Kendall's coefficient of concordance W [31], a measure of respondent agreement where 1 implies perfect agreement. For our response set, $W = 0.42$ (p-value$<$.01),[3] which can represent weak overall concordance [32], implying that at least some properties may have high disagreement. To better interpret the coefficient, we examined the interquartile ranges for each quality property/-goal pair, where an interquartile range of one represents 50% of developers having a responses which neighbor each other on the Likert scale. We find that 80% of pairs had an interquartile range of one or less, which we interpret as high agreement for most properties.

In general, presentation-related properties had the highest agreement. The GPL-applicable properties also had higher agreement in general than the MMT- or QVTo- specific properties. The pair which was least agreed-upon was "Few dependencies on other modules"/Maintainability. By reading the survey comments, we find that while some developers consider dependencies bad for quality, others consider the alternative to be higher code duplication, in which case more

[3]Calculated in R with correction for ties using the `irr` package command `kendall(df,correct=TRUE)`

	Validation			
Quality property	**Functionality**	**Understandability**	**Performance**	**Maintainability**
Detailed comments throughout code	0 (0, .5)	2 (2, 3)	0 (0, 0)	2 (1, 2.5)
Formatting conventions followed	0 (0, 0)	2 (1.5, 2)	0 (0, 0)	2 (1, 2)
Few blackboxes	0 (-1, 0)	1 (.25, 2)	0 (-1, 0)	2 (1, 2)
Few configuration properties	0 (0, .75)	0 (0, 1)	0 (0, 0)	0 (0, 1)
Few dependencies on other modules	0 (0, .5)	1 (0, 1.5)	0 (0, 1)	0 (-2, 2)
Few input/output models	0 (-1, 0)	1 (1, 1.5)	0 (0, 1)	1 (1, 2)
Few intermediate properties	-.5 (-1, 0)	0 (-1, .75)	0 (0, 0)	0 (-.75, 1)
Low code duplication with other modules	0 (0, .5)	1 (1, 2)	0 (0, 0)	2 (1.5, 2)
Pre- and post-conditions specified	0 (0, 1)	1 (1, 2)	0 (0, 0)	1 (.5, 2)
Small interfaces to other modules	0 (0, 0)	1 (.25, 2)	0 (0, 0)	1.5 (.25, 2)
Small transformation size	0 (0, 0)	2 (1, 2)	0 (-1, 1)	1 (1, 2)
Confluence satisfied	0 (0, 1.75)	0 (-.75, 0)	0 (0, 0)	0 (0, 1.75)
Few dependencies between functions	0 (0, 0)	1 (0, 1)	0 (0, 0)	1 (0, 1)
Few `end` sections	0 (0, 0)	0 (0, 1)	0 (0, 0)	0 (0, 1)
Few mapping arguments	0 (0, 0)	1 (-.5, 2)	0 (0, 1)	0 (-1, 1.5)
Few nested `if` statements	0 (-.5, 0)	1 (1, 2)	0 (0, 0)	1 (1, 2)
Few `when` and `where` clauses	0 (-.75, 0)	.5 (0, 1)	0 (-.75, .75)	0 (0, 1)
High test coverage	2 (0, 2)	0 (0, .5)	0 (0, 0)	1 (0, 1)
High usage of design patterns	~~1 (0, 1)~~	1 (1, 2)	0 (0, 1)	1 (1, 2)
Inheritance usage matches metamodel	.5 (0, 1)	.5 (0, 2)	0 (0, 0)	0 (-.75, 1.75)
Interdependent functions near each other	0 (0, 0)	2 (1, 2)	0 (0, 0)	1 (1, 2)
Little dead code	0 (0, 0)	2 (1.5, 2)	0 (0, 1)	2 (1.5, 2)
Little imperative programming	0 (-.5, 0)	1 (0, 1.5)	0 (-.5, .5)	1 (.5, 1.5)
Little overloading	0 (-1, 0)	-1 (-1, 1)	0 (0, .5)	-1 (-1, 1)
Low code duplication within module	0 (0, 0)	1 (1, 2)	0 (0, 0)	2 (2, 2)
Low syntactic complexity	0 (-1, 0)	1 (1, 1)	0 (0, 0)	1 (1, 1)
Minimal reassignment of objects	0 (-.5, 1)	0 (-1, 2)	~~1 (0, 1)~~	0 (-1, 2)
More mappings than helpers	0 (0, .5)	0 (0, 1)	0 (0, 0)	0 (0, .75)
More queries than helpers	0 (0, 0)	0 (0, 1)	0 (0, 0)	0 (0, 1)
Short function chains	0 (0, 0)	1 (0, 1.5)	0 (0, 1)	1 (0, 1.5)
Small function size	0 (-.5, .5)	2 (1.5, 2)	0 (-1, 0.5)	2 (1, 2)
Small `init` sections	0 (0, 0)	1 (0, 1)	0 (0, 0)	1 (0, 1)
Termination checked	0 (0, 2)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Deletion uses trashbin pattern	0 (0, 0)	-1 (-1, 0)	1.5 (0, 2.75)	0 (-.75, 0)
Few queries with side-effects	0 (-.1, 0)	2 (1, 2)	0 (0, 0)	1 (1, 2)
Low execution time	0 (0, 0)	0 (0, 0)	3 (2, 3)	0 (0, 0)
Mappings only used when tracing needed	0 (0, 0)	0 (-1, 0)	~~1 (0, 2)~~	0 (-1, 0)

TABLE II

VALIDATION RESULTS. CONTAINS EACH QUALITY PROPERTY WITH RESPECTIVE VALIDATION RESPONSES SHOWN IN THE FORMAT [MEDIAN] ([25TH PERCENTILE], [75TH PERCENTILE]). RESPONSES FROM "STRONGLY DECREASES" TO "STRONGLY INCREASES" ARE ENCODED ON THE INTERVAL [-3,3]. PROPERTIES ARE ORGANIZED AS IN TABLE I.

dependencies are preferred.

Comparing the ASML team to the public, $W = 0.44$ and $W = 0.32$ respectively, so there is more agreement within the ASML team than amongst members of the public. This is most likely because the general public do not have a common context in which they use QVTo, and therefore their opinions about the best way to write QVTo differ. The median answers for "Termination checked" and "Confluence satisfied" were also considerably higher for the ASML team than for the public. This we explain by differences in academic background (two ASML team members have PhDs in formal computer science topics, whereas members of the public did not have formal computer science backgrounds).

To see the relationship between individual survey respondent answers, we calculated the pairwise correlations between responses, shown in Figure 3. Respondents 1 through 6 were the members of the ASML team. Correlations were calculated using the parametric statistic Spearman's ρ , but were similar to Kendall's τ. It is clear that some respondents had much more similar responses than others, for example respondent 12 correlated highly with 14 and 15. Looking deeper, this high correlation came largely from GPL-applicable properties, and these three respondents in fact reported having the least experience in QVTo. So, the correlation may be explained because these respondents are answering the survey questions based more on prior knowledge of code quality from GPLs than QVTo experience.

Next, we consider which property/goal pairs are perceived as having the strongest impact on quality by comparing median responses. The pair with the highest median was "Low execution time"/Performance followed by properties "Little dead code", "High test coverage", and "Few blackboxes", among others. Four pairs yielded negative medians, indicating that developers perceive these to decrease quality rather than in-

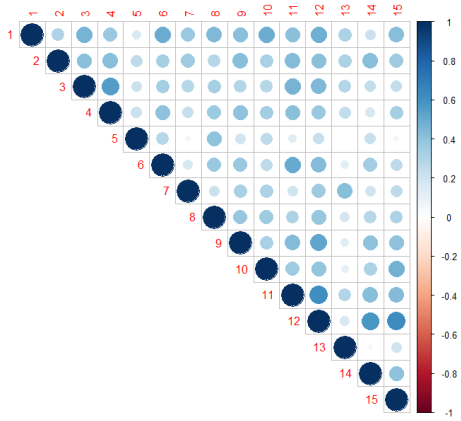

Fig. 3. Pairwise correlations of responses between respondents

crease it. For one of these, "Deletion uses a trashbin pattern",[4] a tradeoff is clearly acknowledged by the respondents where it decreases understandability but increases performance.

Finally, and arguably most importantly, we define two criteria which must be met by a property/goal pair before we consider it validated by the survey. First, the median answer must be at least "Somewhat increases". Second, at least 75% of the responses must have been at least "Has little/no effect". The results which satisfied these criteria are shaded gray in Table II. Of our original 37 properties, 26 are therefore validated for being important for at least one quality goal. Of those validated, nine were considered MMT- or QVTo-specific.

A sensitivity analysis was also performed where each respondent was removed one at a time and the two criteria were rechecked. The three pairs which did not satisfy the criteria in every round are struck through in Table I, suggesting that these pairs in particular may benefit from additional validation. There are also four pairs which satisfied the criteria only when one respondent was removed. These pairs are displayed in boxes. Since these pairs also appear highly sensitive to the specific set of respondents used and because these pairs are all MMT- or QVTo-specific, they also make interesting candidates for additional validation techniques.

A. Threats to Validity

Our approach is not without limitations. First, the quality model is not a complete picture of QVTo quality, since it only contains the properties representing the most important issues at the time of our research. Second, a large portion of our approach is based on perceptions, which are inevitably biased. So, even though we build our quality model with no prior conception of what quality in QVTo should mean, developer perceptions can nonetheless be based on previous impressions of software quality. Third, the validation survey format could be improved. It was noted by some respondents that the strict question format makes some answers obvious while others feel oversimplified. These feelings could hurt response validity or cause others to opt not to take the survey. Finally, although our validation method is more likely than other methods to lead to fine-grained results already useful to practitioners, both our quality model and validation results are susceptible to overfitting due to small sample sizes. For example, ASML may have a particular development style which is not shared by other QVTo developers, providing non-representative interview data. Also, because QVTo is still a young language, many of the QVTo developers today are the same as those creating the QVTo implementations and tooling, and therefore may have different quality needs than future QVTo developers. This overfitting, in fact a common concession in the pragmatist stance [6], is the primary motivation of using a triangulation approach. Nonetheless, conducting additional interviews with developers from other domains and backgrounds is an important step for future validation.

[4]In the "trashbin pattern" objects are assigned to a parent object before deletion.

VI. Related Work

Since the amount of work published on software quality is vast, we only mention the work most closely related to this research. This work was also incorporated in the exploratory study used to construct our quality model.

Ferenc, Hegedűs and Gyimóthy [10] provide an introduction to software quality models with respect to maintainability. Syriani and Gray [5] enumerate the challenges in model transformation quality and propose two directions for research: 1) cataloging transformation design patterns; and 2) identifying quality criteria for transformations, including quantitative metrics. Therefore [5] serves as motivation for our research and we directly build on the second research area they identified.

Like our research, other research has also addressed model transformation quality. However, we classify these as top-down approaches because the authors construct their notions of quality exclusively from theory and related work. In addition to the work described in Section II and MMT quality models discussed in Section IV-B, Vignaga [24] proposed a set of ATL metrics and Kapová et al. [23] defined metrics for the declarative transformation language QVTr, though in neither case was an empirical validation performed. These approaches heavily influenced our research, first by showing that new metrics can be defined for model transformations, and second, these works motivated us to pursue a bottom-up approach to build a quality model useful for practitioners. Using a bottom-up approach is further motivated by Hall and Fenton's [14] recommendations for creating successful metrics programs: Among their eleven recommendations were transparency, usefulness, developer participation, feedback, and a goal-oriented approach; all of which were heavily incorporated in designing our approach.

Moody [17] provides a rich overview of techniques to evaluate quality models, pointing out that a surprisingly low proportion of quality models proposed in literature are validated. Each technique is related explicitly to philosophical stances. A demonstration of one of these evaluation techniques is provided by Moody in his assessment of data model quality [15]. There, a set of data model metrics was proposed.

To evaluate the metrics, action research was performed where the metric set was applied in multiple development projects over the course of two years, refining the metrics iteratively. Of the original 29, only 5 remained at the final iteration. Metrics were removed primarily because it was unclear to practitioners how they were useful for quality assessment. The result was therefore a concise set of metrics which were validated to be useful for assessing data model quality in practice. Although action research has not been used to validate our quality model, we consider it a promising direction for future research.

VII. Conclusion

In this paper, a quality model was presented for QVTo transformations. Due to our bottom-up approach, the QVTo quality model presented here captures the aspects most relevant for QVTo quality in practice today. The model was validated by conducting a survey of developer perceptions of each property. Of the 37 quality properties included in our model, 26 were considered validated, of which 9 are specific to MMT or QVTo, showing that quality models created for other languages will not cover some of the most important properties to assess QVTo quality. Moreover, although the quality model presented here can only but inform quality models targeting other languages, our approach for constructing the model and validation could be applied to any software quality model. Future work includes performing additional validation of the quality model, for example that used by [11], conducting additional interviews, and assessing the appropriateness of the quality evaluation procedures. Furthermore, while the quality properties we have identified, have been formulated to express directionality, e.g., "High test coverage" or "Small function size", we consider as another direction of future work determining thresholds for those metrics [33] and developing appropriate aggregation techniques, allowing one to lift the quality assessment to larger units [34], [35]. Finally, future work also includes developing tooling which leverages the quality model to assist developers in creating higher-quality QVTo transformations.

References

[1] T. Stahl and M. Voelter, *Model-driven software development*. John Wiley & Sons Chichester, 2006.

[2] OMG, "MOF 2.0 Query/View/Transformation Spec. V1.1," 2011.

[3] ASML N.V. [Online]. Available: http://www.asml.com

[4] R. R. Schiffelers, W. Alberts, and J. P. Voeten, "Model-based specification, analysis and synthesis of servo controllers for lithoscanners," in *Int. Workshop on Multi-Paradigm Modeling*. ACM, 2012, pp. 55–60.

[5] E. Syriani and J. Gray, "Challenges for addressing quality factors in model transformation," in *ICST*. IEEE, 2012, pp. 929–937.

[6] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to advanced empirical softw. eng.* Springer, 2008, pp. 285–311.

[7] P. J. Barendrecht, "Modeling transformations using QVT Operational Mappings," Master's thesis, Technische Universiteit Eindhoven, 2010, accessed 2014/4/1. [Online]. Available: http://redpanda.nl/BEP_P.J.Barendrecht.pdf

[8] OMG, "Object Constraint Language," 2012.

[9] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target," *IEEE Software*, vol. 13, no. 1, pp. 12–21, 1996.

[10] R. Ferenc, P. Hegedűs, and T. Gyimóthy, "Software product quality models," in *Evolving Software Systems*, T. Mens, A. Serebrenik, and A. Cleve, Eds. Springer Berlin Heidelberg, 2014, pp. 65–100.

[11] M. F. van Amstel, "Assessing and improving the quality of model transformations," Ph.D. dissertation, Technische Universiteit Eindhoven, 2012.

[12] ISO/IEC 25010, "Systems and software quality requirements and evaluation (SQuaRE) – System and software quality models." Geneva, Switzerland: ISO, 2011.

[13] M. F. van Amstel, M. G. J. van den Brand, and P. H. Nguyen, "Metrics for model transformations," in *BENEVOL*, 2010.

[14] T. Hall and N. Fenton, "Implementing effective software metrics programs," *IEEE Software*, vol. 14, no. 2, pp. 55–65, 1997.

[15] D. L. Moody, "Measuring the quality of data models: an empirical evaluation of the use of quality metrics in practice." in *ECIS*, 2003, pp. 1337–1352.

[16] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE TSE*, vol. 25, no. 4, pp. 557–572, 1999.

[17] D. L. Moody, "Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions," *Data & Knowledge Engineering*, vol. 55, no. 3, pp. 243–276, 2005.

[18] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *METRICS*. IEEE, 2005, pp. 10–23.

[19] C. M. Gerpheide, "Assessing and improving quality in QVTo model transformations," Master's thesis, Technische Universiteit Eindhoven, 2014 (forthcoming).

[20] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

[21] (2014, Jan.) Eclipse community forum QVT-OML. [Online]. Available: http://www.eclipse.org/forums/index.php/f/244

[22] (2013, Dec.) Transformation tool contest. [Online]. Available: http://www.transformation-tool-contest.eu

[23] L. Kapová, T. Goldschmidt, S. Becker, and J. Henss, "Evaluating maintainability with code metrics for model-to-model transformations," in *QoSA*, ser. LNCS, vol. 6093. Springer, 2010, pp. 151–166.

[24] A. Vignaga, "Metrics for measuring ATL model transformations," *Dept. of Computer Science, Universidad de Chile, Tech. Report*, 2009.

[25] S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya, and P. Van Gorp, "Evaluation of model transformation approaches for model refactoring," *Sci. Comput. Program.*, vol. 85, pp. 5–40, Jun. 2014.

[26] T. Mens and P. Van Gorp, "A taxonomy of model transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, pp. 125–142, 2006.

[27] M. Amrani, J. Dingel, L. Lambers, L. Lúcio, R. Salay, G. Selim, E. Syriani, and M. Wimmer, "Towards a model transformation intent catalog," in *Proceedings of the First Workshop on the Analysis of Model Transformations*. ACM, 2012, pp. 3–8.

[28] S. Lehrig, "Assessing the quality of model-to-model transformations based on scenarios," Ph.D. dissertation, MSc Thesis, University of Paderborn, Zukunftsmeile 1, 2012.

[29] R. Johns, "Likert items and scales," *Survey Question Bank: Methods Fact Sheet*, vol. 1, 2010.

[30] J. J. Barnette, "Effects of stem and likert response option reversals on survey internal consistency: If you feel the need, there is a better alternative to using those negatively worded stems," *Educational and Psychological Measurement*, vol. 60, no. 3, pp. 361–370, 2000.

[31] A. P. Field, "Kendall's coefficient of concordance," *Encyclopedia of Statistics in Behavioral Science*, 2005.

[32] G. Vidmar and N. Rode, "Visualising concordance," *Computational Statistics*, vol. 22, no. 4, pp. 499–509, 2007.

[33] P. Oliveira, M. T. Valente, and F. P. Lima, "Extracting relative thresholds for source code metrics," in *CSMR-WCRE*, S. Demeyer, D. Binkley, and F. Ricca, Eds. IEEE, 2014, pp. 254–263.

[34] B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, "By no means: A study on aggregating software metrics," in *2nd International Workshop on Emerging Trends in Software Metrics*, ser. WETSoM. ACM, 2011, pp. 23–26.

[35] K. Mordal, N. Anquetil, J. Laval, A. Serebrenik, B. Vasilescu, and S. Ducasse, "Software quality metrics aggregation in industry," *Journal of Software: Evolution and Process*, vol. 25, no. 10, pp. 1117–1135, 2013.