Evaluating Student Work in Modelling Courses

Richard F. Paige, Dimitrios S. Kolovos, Fiona A.C. Polack, and Louis M. Rose

Department of Computer Science, University of York, York, YO10 5GH, UK
{richard.paige, dimitris.kolovos, fiona.polack, louis.rose}@york.ac.uk

Abstract. In any curriculum or course that teaches or emphasises the use of modelling, a critical concern is *evaluating* the modelling artefacts that students produce, leading to the production of *feedback*. Students need to be able to use this feedback to improve their understanding and to become better, more mature modellers. In this position paper, we discuss the challenges and importance of evaluating modelling (and related) artefacts and providing precise and reproducible feedback. We also discuss some possible principles and criteria for supporting this critical part of how we teach modelling.

1 Introduction

A recurring problem in Computer Science and Software Engineering education is *how we evaluate the artefacts that students produce*. When a student takes an examination, evaluation is usually provided against a form of rubric or marking scheme that captures characteristics of possible solutions (both good and bad), allocating marks accordingly. However, when a student submits an assignment, project, or artefact, evaluation is more challenging – there are many degrees of freedom in the students' submissions, and defining a clear and precise rubric is a significant challenge. As such, providing *precise and reproducible evaluations* of student submissions is challenging. A software engineering curriculum that includes modelling causes particular problems.

Whether modelling is taught as a stand-alone software engineering topic (e.g., in an advanced course covering modelling, metamodelling and model transformation), or as a key component in a software or systems engineering course (e.g., where modelling is used in solving specific problems), modelling artefacts are produced by students, and instructors are required to evaluate them.

A key difficulty with designing and delivering a modelling course is to identify *precise* criteria for evaluating students' modelling artefacts which, when applied, lead to repeatable and reproducible results. Modelling artefacts may include example models, metamodels, well-formedness constraints, model transformations, code generators, model management workflows, graphical or textual editors, and more. The artefacts may be supplemented with documents – e.g., justifying the design decisions made, explaining the risks associated with specific design choices, motivating the context in which the artefacts may be used. For such a diverse set of artefacts, identifying precise criteria for evaluation is difficult. Ensuring that criteria are applied systematically is even more challenging.

1.1 Precision and repeatability

Why is it important for modelling evaluation criteria to be precise and repeatable? Precision is important so that clear feedback can be provided, to help students to understand feedback and to integrated it with their learning. Precise criteria are *operational*: they can be applied by students to assess their progress and to track their understanding.

Precise criteria are also an effective tool for *closing the loop*. Students tend to use feedback to challenge and discuss the evaluation of their work, helping instructors to identify and correct any errors or inconsistencies in grading and feedback. Whilst this may sound like an invitation for students to complain about the quality and accuracy of their feedback, it is an important principle: if we expect students to learn from feedback, why shouldn't instructors learn from feedback on what they provide, too?

Repeatability of evaluation criteria is essential to support team teaching of modelling, allowing evaluation to be shared between team members, and so that the overall soundness of evaluation results can be validated. Team teaching of modelling in itself can have significant benefits for students (e.g., appreciating that models can be read and used differently by different people), but this is a topic for a different paper.

1.2 Purpose of the paper

This paper attempts to illustrate the challenges associated with identifying precise and repeatable criteria for evaluating modelling artefacts, and gives some indication of the principles and criteria that the authors have used in teaching. The criteria for evaluation that we describe have been developed over a number of years, in courses that teach modelling as a specialist subject, and also courses that use modelling to enable engineering of systems.

2 Related Work

The issue of evaluating modelling artefacts has seen past study, though not specifically in the computing education literature; the focus has been on evaluation of models and modelling languages in industrial application.

Early work by Howatt [1] identified criteria for evaluating software languages (not specifically modelling languages). Criteria identified include existence of a formal definition of the language; user-friendliness; a language implementation following sound software engineering principles; language support for building the intended class of application. The criteria are very high-level, and in some cases imprecisely defined (what constitutes a user-friendly language?), and are thus difficult to operationalise.

Mohagheghi and Haugen [2] present an important work which identifies criteria for evaluating domain-specific modelling (DSM) solutions, as well as reviewing some of the related work in the area. The criteria for evaluation include

the usability of the DSM solution; the "fit" between a DSM solution and the requirements of a problem (e.g., in terms of support for necessary abstractions); the DSM solution's support for necessary engineering activities (e.g., code generation); the evolvability of the DSM solution; and the minimalism of the solution.

Barisic at al. [3] propose a user interface-focused validation approach for (domain-specific modelling) languages that combines usability evaluation processes and DSL development. The reported work is at a conceptual stage, and focuses on analysing types of usability evaluation methods in terms of their suitability for application within software language engineering.

There is limited published research on approaches that can be used for evaluating modelling artefacts within the teaching context. Hints of criteria can be found in work by Demuth et al.[4], which presents the results of an experiment to assess whether modelling and coding skills are correlated. An interesting criterion relates to the use of *design patterns* in evaluating the quality of modelling (and, in this case, code) artefacts – in our experience, also, design patterns are a key characteristic to examine when evaluating student modelling artefacts.

School	**Course**	**level**	**Assessment**	**Evaluation**
UTEP [9]	MDSD	PG	Project, exam	Lessons learned, quality
Rennes [6]	MDE	PG	Quiz, Homework	Quiz mark scheme, Presentation
MDH [7]	MDE	UG	Practical project, essay	Metamodels, discussion of limits
Helsinki [8]	Intensive MDE	UG	Course portfolio	Quality of artefacts
Harvey Mudd [10]	DSLs	UG	Project, demo, write-up	Q&A essay, justification
Kings College [11]	Software Architecture	PG	Exam	Specified mark scheme
Edinburgh [12]	Software Engineering	UG	Exam	Specified mark scheme
York [5]	MDE	PG	Implementation	Justification, Working Application

Table 1. Modelling and modelling-related courses with assessment types and evaluation criteria [UG = undergraduate, PG = postgraduate].

There are numerous modelling and modelling-related courses with (different quantities and qualities of) publicly accessible material from around the world. We examined eight courses, from Bilkent (Turkey), Rennes (France), Mälardalen (Sweden), Helsinki (Finland), Harvey Mudd (USA), Kings College (UK), Edin-

burgh, (UK) and University of York (UK). The course details are summarised in Table 1; URLs, where publicly available, are in the bibliography.

In general, the *requirements* for what to submit (e.g., descriptions of metamodels, transformations, execution results, a presentation) are precise, but the *criteria* against which those artefacts are evaluated are not. In the courses considered, the evaluation criteria for modelling artefacts submitted as part of an assignment, project, or closed examination, are typically very high level, a general marking scheme against which solutions are judged; statements such as "the instructor will evaluate the quality of the artefacts" are common.

3 Principles and Criteria

In this section we synthesise principles and criteria – derived from our experience, from the literature, and from an analysis of the modelling-related courses – for how we evaluate student work, and how we provide feedback to students. Our criteria, in particular, are derived from our experience in delivering a Masters-level course on Model-Driven Engineering.

3.1 Principles for evaluating student work

We suggest the following principles for evaluating student assignments, projects, demonstrations, reports, code, and modelling artefacts, and for constructing and delivering feedback to students on the quality of their work.

1. *Reproducibility in marking and feedback.* The criteria used for evaluating work should lead to reproducible results. Reproducibility means both that the same piece of work would receive the same mark and similar feedback if marked by different markers, and that an intelligent student should be able to use the feedback to (re)produce an answer that is close to optimal. Reproducibility is critical in helping students to appreciate how well they have understood topics, and how model answers differ from what they have produced – that is, showing a student how and where they need to improve their understanding. Where evaluation is reproducible, learning can be developed and reinforced by repeated exercises, so that students and staff can monitor improvement in understanding. Reproducibility is also important for quality control (e.g., for assessing standards of teaching and learning at institutions), for helping instructors to improve their skills, and for team teaching (for instance, where one instructor sets an examination or project, but a colleague marks it).
2. *Evidence of Justification.* In assessing modelling activities, it is likely that students will deliver a range of solutions, depending on their interpretation of the question. It is possible that all the solutions are valid (syntactically and as answers to the question). To be able to give relevant, reproducible feedback, it is essential that the assessment asks students to justify and explain, clearly and precisely, the decisions that they took in arriving at their

modelling solution. It is insufficient for students to simply deliver artefacts (models, metamodels, transformations, etc.); the thinking that went in to the production of the artefact needs to be clearly delimited. As trainee software engineers, students need to learn to critically evaluate and explain their own solutions, so evaluation should focus on assessing the quality of the justification or explanation: this might comprise a systematic report of how the modelling solution meets the set question; clear, justified consideration of alternatives and decisions; and, perhaps, explanation and mitigation of identified risks inherent to the chosen solutions. Evaluation of justification needs to be supported by evaluation of *validity*, which we discuss next.

3. *Internal and external validity.* Evaluation of validity complements evaluation of the justification of decisions: a student may produce a good critical report, but might submit a model that is either syntactically invalid or does not answer the set question. Here, we identify principles of *internal* and *external* validity. Internal validity evaluates whether the model artefacts delivered by the student are well-formed. The measure of well-formedness depends on the context: we might be looking for models that conform to their metamodel, or a metamodel that loads with EMF, or a submitted model transformation that compiles and executes using the stated model management tools. Internal validity is thus about developing skill with modelling notations and memes, and feedback needs to inform and guide students to develop modelling expertise. External validity evaluates whether the submitted modelling solution satisfies the functional and non-functional requirements stated or implied in the set question. This aspect, of solutions that meet the requirements, is fairly easy for students to understand when presented in the software engineering context of modelling. Students typically expect to be evaluated on the validity of their modelling but, as noted, evaluations based only on only on validity are likely to be less meaningful and supportive of learning that evaluations that also consider quality and relevance of decisions and justifications.
4. *Necessity and sufficiency.* One of the most challenging aspects of learning to model effectively is understanding what to include in models, and what to exclude. A principle of evaluating student work and providing feedback is, therefore, to emphasise necessity and sufficiency of artefacts. *Necessity* is related to and supported by justification: by encouraging students to justify the concepts included in their models, we encourage them to develop the mindset of explaining the necessity of what they have produced for solving the problem. *Sufficiency* is more difficult to evaluate, but is related to validity: by encouraging students to assess internal and external validity, we encourage them to develop the mindset of explaining the sufficiency of what they have produced for solving the problem.
5. *Style is not unimportant.* The first four principles have focused on assessing modelling artefacts objectively. However, subjective aspects are also important. Stylistic issues in modelling need to be taught and evaluated. Stylistic issues may be the difference between, for example, creating a modelling language that satisfies its requirements, and creating a modelling language that

is useful and usable (e.g., because it comes with tools that engineers want to use).

6. *Approximation.* A software modelling problem rarely has a perfect solution; often there are many solutions. Very often there are partial solutions that address some significant challenges of the problem – these, while incomplete, may demonstrate significant potential, as well as some good understanding of the problem and how to apply knowledge. The way in which we evaluate student work needs to acknowledge that students may create incomplete solutions; grading schemes must support this, and feedback must be constructed to helps students learn how to transform their partial solution to one that addresses all the requirements.
7. *Practicalities.* Students of software engineering are understandably keen to solve problems that they encounter in practice. Whilst it is unrealistic to expect university assessments to fully address real-world software engineering problems, evaluation criteria should promote good software engineering practice. For example, modelling (like any form of design) is emergent, and usually follows a non-deterministic and heuristic process [13]. As such iterative refinement, and hence evaluation, of artefacts should be encouraged. Similarly, evaluation should reward and encourage engineering of artefacts in a manner that simplifies testing, or facilitates evolution.

3.2 Criteria for evaluation

We now describe criteria that we propose as useful for evaluating student modelling artefacts. Such criteria would be at the basis of any feedback given to students. The criteria are derived to be consistent with the principles described above, and to support learning and the development of experience through assessment. We separate the criteria into three categories: external, internal, and stylistic.

External criteria External criteria involve engagement with external stakeholders (e.g., domain experts, customers). In all cases, *artefact* may refer to any modelling artefact required by an assessment (model, metamodel, transformation, etc.).

- *Fitness for purpose* assesses whether the artefact is an accurate (adequate) representation of whatever domain is of interest. The assessment task would be written in terms of the concepts (e.g., model or metamodel elements) that need to be expressible and the structures that relate them. In terms of evaluation of student work, we should assess whether there are concepts missing from the artefact, whether there are missing "sentences" or "structures" that should have been expressible (because the domain supports them), and whether invalid sentences or structures are disallowed by some means (e.g., well-formedness constraints).

- *Operationality* assesses whether the artefact enables required operations (such as code generators) to be implemented against it. For evaluation purposes, we should assess whether the operations that are produced address the problem requirements, as well as their non-functional characteristics (e.g., that a code generator is incremental).
- *Acceptability* assesses whether the artefacts are presented to domain experts or users in a way that is acceptable (e.g., in terms of chosen icons, colours, annotations, etc.). For evaluation purposes, if students are asked to produce an editor or concrete syntax for a modelling language, evaluation needs to consider whether they have chosen a suitable style of presentation, and justified it.

Internal criteria Internal criteria do not involve external stakeholders (e.g., domain experts, end-users), but are used to evaluate whether an artefact has been constructed correctly. Typical software engineering criteria can be applied.

- *Patterns:* has the artefact been constructed using suitable, proven and trusted patterns (e.g., composite/opposite references for a metamodel)?
- *Naming:* has the artefact made use of appropriate naming conventions (e.g., a standard style for naming meta-elements in a metamodel)?
- *Implemented:* does the artefact load in a tool (e.g., does the model load in Eclipse EMF)?
- *Necessary:* is the design and implementation valid? Is each element of the artefact either a requirement (domain concept), an implementation (needed because of the chosen modelling platform), or a design concept (connecting requirements to implementations)?
- *Justification:* have suitable justifications been given for the decisions made in the modelling process?
- *Redundancy:* is any duplication or redundancy in the artefact justifiable, and justified?
- *Habitable:* following [14], are artefacts constructed in a way that facilitates change?

Style criteria Assuming that a set of modelling artefacts satisfies external criteria, we may want to evaluate the extent to which artefacts are (somewhat subjectively) "good", rather than just solutions that "work". For example, a solution that "works" but is not of good style might receive a high mark overall (because it works) but a low mark for any component assigned to style.

- *Appropriateness:* does the artefact make appropriate use of MDE languages and tools (e.g., use of Ecore)?
- *Declarative vs Operational:* given the problem being solved, is the artefact (e.g., a transformation) expressed in a suitably declarative or operational style?
- *Idiomatic:* are idioms used appropriately (e.g., dynamic output sections in a model-to-text transformation, rather than static print expressions).

– *Metamodelling features:* are metamodelling features used appropriately (e.g., reference navigation, containment, generalisation)?
– *Understandable by peers and instructors:* can other students and instructors understand both the artefact and its rationale?

4 Conclusions and Further Work

We have described a number of principles for evaluating student modelling work, emphasising Reproducibility, the use of justification, validity, necessary and sufficient construction of artefacts, and approximations. From these principles, we have described criteria for evaluating student modelling artefacts, ranging from assessment of fitness for purpose, through to stylistic criteria related to understandability. The weighting of criteria that are useful for a modelling course will depend on the maturity and experience of students, the style of assessment (e.g., exam, code review, presentation), and the emphasis that the instructor wants to put on different aspects of modelling – for example, external validation may be considered to be more important for one cohort of students than another.

We are using these criteria and these principles in our MDE course (taught to MSc students) at York. We emphasise the production of working artefacts (models, metamodels, constraints, transformations and code generators) supported by tools. To support the evaluation and feedback process, we have moved to *rubric-based* evaluation, where both students and instructors have explicit criteria-based evaluation processes that lead to relevant and consistent feedback. This, combined with both formative and summative assessment, has led to a course where students are able to develop experience by practising what they have learned, and instructors can assess whether students are able to apply the feedback they have received, all within one course.

Acknowledgements This research was part supported by the EU, through the MONDO FP7 STREP project (grant #611125).

References

1. J. Howatt. A project-based approach to programming language evolution. `academic.luther.edu/~howaja01/v/lang.pdf`, 2001.
2. Parastoo Mohagheghi and Øystein Haugen. Evaluating domain-specific modelling solutions. In *Advances in Conceptual Modeling - Applications and Challenges, ER 2010 Workshops ACM-L, CMLSA, CMS, DE@ER, FP-UML, SeCoGIS, WISM, Vancouver, BC, Canada, November 1-4, 2010. Proceedings*, pages 212–221, 2010.
3. Ankica Barisic, Vasco Amaral, and Miguel Goulão. Usability evaluation of domain-specific languages. In *8th International Conference on the Quality of Information and Communications Technology, QUATIC 2012, Lisbon, Portugal, 2-6 September 2012, Proceedings*, pages 342–347, 2012.

4. Birgit Demuth, Sebastian Götz, Harry M. Sneed, and Uwe Schmidt. Evaluation of students' modeling and programming skills. In *Proceedings of the Educators' Symposium co-located with ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013), Miami, USA, September 30th, 2013.*, 2013.
5. Course: Model-Driven Engineering (York). `https://www.cs.york.ac.uk/postgraduate/modules/mode.html`, last accessed, June 2015.
6. Course: Model-Driven Engineering of Embedded Software (Rennes). `http://master.irisa.fr/index.php/en/parcoursprog-en/parcours-rennes-1-en?id=203:mde-en&catid=108:modules-en`, last accessed, June 2015.
7. Course: Model-Driven Engineering (MDH). `http://www.idt.mdh.se/kurser/dva411/`, last accessed, June 2015.
8. Course: Intensive Course on Model-Driven Engineering (Helsinki). `https://www.cs.helsinki.fi/en/courses/582706/2013/k/k/1`, last accessed, June 2015.
9. Course: Model-Driven Software Development (Bilkent). `http://www.cs.bilkent.edu.tr/~bedir/CS587-MDSD/`, last accessed, June 2015.
10. Course: Domain-Specific Languages (Harvey Mudd College). `http://www.cs.hmc.edu/courses/2014/fall/cs111/`, last accessed, June 2015.
11. Course: Software Design and Architecture (Kings). `http://www.kcl.ac.uk/nms/depts/informatics/study/current/handbook/progs/modules/7CCSMDAS.aspx`, last accessed, June 2015.
12. Course: Software Engineering with Objects and Components (Edinburgh). `http://www.drps.ed.ac.uk/14-15/dpt/cxinfr10056.htm`, last accessed, June 2015.
13. Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, 2004.
14. Richard P. Gabriel. *Patterns of Software*. Oxford University Press, 1996.