On the Influence of Metamodel Design to Analyses and Transformations

Georg Hinkel[1] and Erik Burger[2]

[1] FZI Research Center of Information Technologies, Software Engineering Dept.
Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany, hinkel@fzi.de
[2] Karlsruhe Institute for Technology (KIT), Software Design & Quality Group (SDQ)
Am Fasanengarten 5, 76131 Karlsruhe, Germany, burger@kit.edu

Abstract. Metamodels are a central artifact of model-driven engineering. As they determine the structure of instance models, they are a foundation for other model-driven artifacts such as model transformations, code generators or model analyses. Therefore, the quality of metamodels is important for any model-driven process. However, the implications of metamodel design to other artifacts such as model analyses or model transformations has barely been looked at through empirical research. In this paper, we present an empirical study where we analyzed equivalent model analyses and transformations for 19 different metamodels of the same domain. The results indicate that metamodel design has a strong influence to model analysis in terms of code metrics but only little influence on model transformations targeting this metamodel.

1 Introduction

To aid the increased complexity of modern (software) systems, model-driven engineering (MDE) helps by raising the level of abstraction. Models can be used for analysis to conclude insights on the represented system or for transformation into other artifacts, such as models of other metamodels or code.

In the Neurorobotics-platform developed in the scope of the *Human Brain Project* (HBP), these dependent artifacts include not only editors, but also an entire simulation platform where the connection between robots and neural networks is described in models [1], [2]. As the HBP is designed for a total duration of ten years, it is likely that the metamodel will degrade unless extra effort is spent for its refactorings [3], [4]. For such refactorings, we aim to measure their success and potentially automate them.

A central artifact for the specification of model analyses or model transformations is the metamodel [5] as it defines the abstract syntax used by instances. As a consequence, metamodel design has a strong impact on the design of model analyses and model transformations, particularly because metamodel evolution usually implies an evolution of these artifacts [6]. Therefore, ensuring the quality of metamodels is very important.

As metamodels are formalizations of the domain they are describing, many design decisions for the creation of the metamodel are already determined by

the domain. Nevertheless, many decisions are left to the metamodel developer, opening a design space of metamodels, even for a fixed domain. The goal of our research is to evaluate the metamodels of this design space for their quality. This quality may be dependent on the particular domain, but we think there are also notions of it that are domain-independent such as subtleties in the inheritance and composition hierarchies.

There have been surprisingly few research activities on the actual implications of metamodel design to other artifacts: Does a concise metamodel imply concise analyses? Does a modular metamodel support modularity of model transformations? Does an understandable metamodel help to understand analyses? Often, it is difficult to answer these questions, because especially the quality of metamodels is not well captured by metrics. Further, as model analyses and transformations are laborious to create, they are hardly created for multiple semantically equivalent metamodels. This makes it very hard to reason on the influence of metamodeling design decisions to these artifacts: We usually do not know how transformations or analyses would look like if the metamodel was designed differently. Existing empirical studies neglect the different intention of transformations [7] or only look at correlations within the metamodel [8]–[10].

On the other hand, an automated quantitative measurement of quality is only the second step: For many purposes, it suffices to know whether the intuitive perception of quality correlates with code metrics of model transformations or analyses: To what degree matches the intuition of what is good or bad in a metamodel to what the metrics for later developed artifacts tell us.

In this paper, we present an empirical study to analyze and quantify the design impact of 19 different metamodels of the same domain to 38 equivalent model analyses (two for each metamodel) and 19 equivalent transformations that each transform into one of the metamodels from a common source model. We have reported correlations between the perceived quality or metric values of the metamodels and metric values of model analyses or transformations.

The remainder of this paper is structured as follows: Section 2 explains the experiment setup in more detail. Section 3 presents the resulting correlations. Section 4 discusses insights drawn from the results. Section 5 discusses threats to validity. Section 6 presents related work. Lastly, Section 7 describes future work before Section 8 concludes the paper.

2 Method

The method for our study is sketched in Figure 1.

At first, a group of participants create metamodels on a common domain description. Then, we propose to let the participants review each others created metamodels. In parallel, transformation developers and analysis developers may develop artifacts based on the created metamodels. Here, it is very important that analyses and transformations are created consistently across metamodels, which is why all transformations and all analyses should be written by the same developers, ideally. Then, from each artifact, metamodels, analyses and trans-

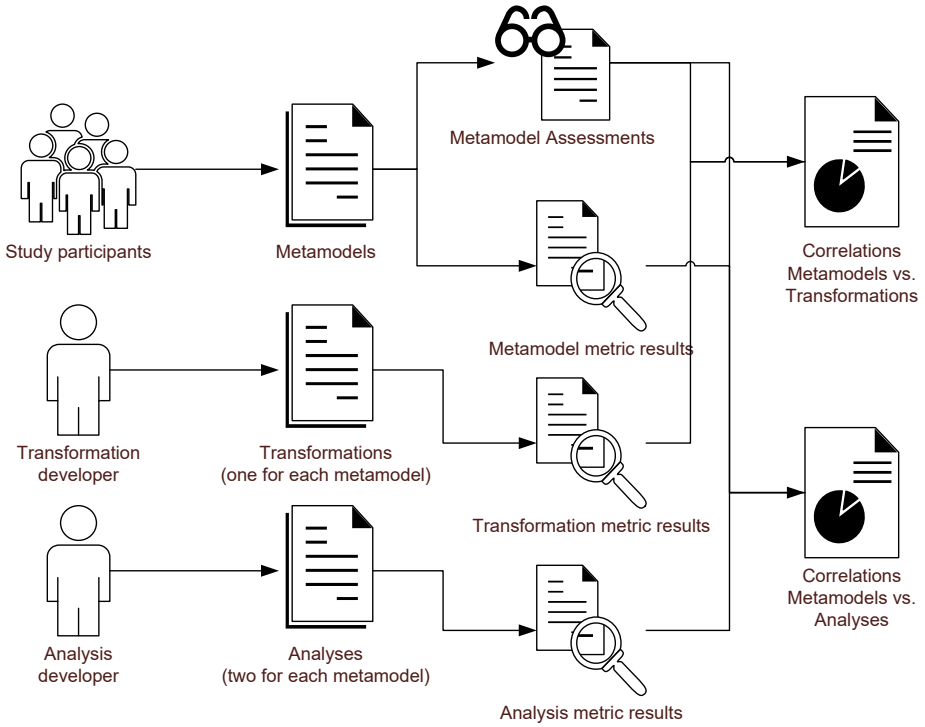


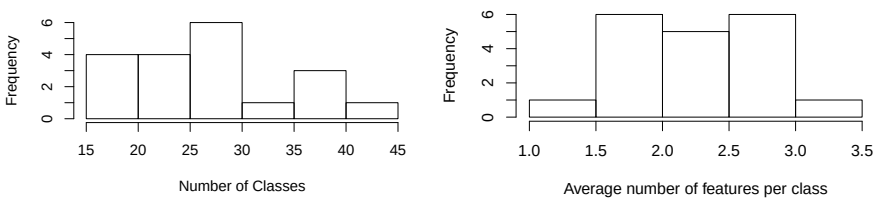
Fig. 1: Method overview for the empirical study

formations, several metrics are computed that are correlated in a last step. In the remainder of this section, we discuss each step in more detail.

2.1 Data Sources

In this section, the data sources of this experiment setup are described in detail.

Metamodels The 19 metamodels we use originate from students attending a practical course on model-driven engineering across multiple years. The students were working in groups of two to create an Ecore-metamodel based on a textual domain description of component-based software architectures. This domain description includes creating a component repository, assembling software architectures from components and deploying software architectures to resources. The domain description is inspired by Palladio [11] that uses similar models to predict non-functional properties of such systems. Palladio is taught in other courses so that several students already knew the underlying concepts. In the experiment, we asked them to create Ecore metamodels for these concepts. For that, the students used the tools provided by Eclipse.



(a) Number of classes (b) Average number of features

Fig. 2: Characteristics of the created metamodels

Due to space limitations, we cannot describe the 19 metamodels in detail. To still give an impression on the created metamodels, we depicted a histogram of the number of classes in Figure 2a. The metamodels contained between 15 and 45 classes with a peek between 25 and 30 classes. Further, we depicted the average number of features per class in Figure 2b, i.e. the number of attributes and references, including those defined in base classes.

We did not publish the metamodels as they originate from homework assignments of a practical course and unfortunately, we have not for a consent to publish them.

Peer-reviews We captured the human perception of metamodel quality for the metamodels in a controlled setting: We gave a questionnaire to participants of the last edition of that practical course and asked the students to review 7 metamodels of their colleagues, making sure that nobody had to review their own metamodel. The questionnaire that we used is publicly available online[3].

The review of the metamodels was done by students in a controlled experiment setting, though we allowed the students to collaborate on evaluating metamodels assigned to them. However, the low level of experience with modeling techniques means that the peer reviews rather reflect an intuitive feeling on the metamodels rather than expert opinions.

Similar to previous experiments [9], [10], we asked students to evaluate the complexity, understandability, conciseness, modularity, consistency, completeness and changeability of the metamodels. Further, students were asked to estimate how easy it would be to create instances of this metamodel and how easy it would be to specify model transformations. Lastly, they should evaluate the overall quality. Complexity is asked as the degree in which a metamodel is not complex. Therefore, a high value for complexity means that the metamodel is simple. As the values obtained this way are basically intuitions, they are highly subjective.

[3] `https://sdqweb.ipd.kit.edu/wiki/Metamodel_Quality`

```
Root =>
  (from assemblyConnector in Root.Systems
      .FirstOrDefault().AssemblyConnectors
   where !assemblyConnector.From.Provided
      .Contains(assemblyConnector.ConnectedInterface) ||
     !assemblyConnector.To.Required
      .Contains(assemblyConnector.ConnectedInterface)
   select assemblyConnector).Distinct()
```

Listing 1: An exemplary analysis whether components are correctly connected

Analyses Based on the metamodels, we created two model analyses and a model transformation for each metamodel. The analyses compute violations to the following validity rules:

Correct Connection of Assemblies Assemblies, i.e. usages of components in a software architecture, must be connected on the same interface. This interface is required by the requiring component and provided by the providing component.

Correct Allocation of Assemblies Assemblies that are connected in the software architecture must be deployed either to the same resource container or there must be a link between the resource containers. Such a connection exists because the component of one assembly requires an interface provided by the component of the other assembly.

We are aware that validation constraints are only one particular kind of analyses. In the domain that we studied, they seem to us as a good compromise between simplicity (the analyses have to be implemented for each metamodel) and non-trivial complexity.

All model analyses are implemented C# in the query syntax, based on metamodel representations using NMF [12]. As argued previously by Akehurst and others [13], C# is equivalent to OCL. Therefore, we expect that the results for analyses written in the C# query syntax should generalize to OCL. However, the usage of C# allows us to use the code metric implementations from Visual Studio to correlate them to metamodel properties.

As two examples, we depicted exemplary analyses in Listings 1 and 2 (for MM15).

Both model analysis tasks are rather simple and therefore solvable in a few lines of code. The solutions for the query whether assemblies are correctly contained ranged from 6 to 25 lines of code. The solutions for the allocation query ranged from 17 to 62 lines of code.

All analyses and transformations were created by two students in order to reduce the influence of the individual analysis developer. After all analyses and transformations were written, all of them were refactored to minimize learning effects.

Transformations For every metamodel, we also implemented a model transformation from the much more detailed Palladio Component Model (PCM, the

```
Root =>
  (from connector in Root.Systems
     .FirstOrDefault().AssemblyConnectors
   from requiringAllocation in Root.Allocations
     .FirstOrDefault().AllocationContexts
   where requiringAllocation.AllocatedContext == connector.From
   from providingAllocation in Root.Allocations
     .FirstOrDefault().AllocationContexts
   where providingAllocation.AllocatedContext == connector.To &&
     providingAllocation.Container != requiringAllocation.Container &&
     !Root.Environments.FirstOrDefault().Links
       .Any(link =>
         link.Containers.Contains(providingAllocation.Container) &&
         link.Containers.Contains(requiringAllocation.Container))
   select connector).Distinct()
```

Listing 2: An exemplary analysis whether components are correctly allocated

metamodel used in Palladio) to that metamodel. PCM also describes component repositories, software architecture and deployment of component-based software architectures. However, PCM is a complex metamodel with more than 200 classes, as it supports the architecture-based quality prediction of component-based systems. If a developer simply wants to take a look at the software architecture and deployment, a transformation to a simpler metamodel is appropriate.

As we aleays create a transformation from PCM, we vary the target metamodel of the model transformation, not the source. The reason is that we assume that the impact of metamodel design to model transformations from the metamodel into another domain is quite similar to the impact that metamodel design has on model analyses.

All model transformations have been implemented using ATL [14], one of the most commonly used textual model transformation languages [15].

Because the created metamodels are mostly projections of PCM, many transformation rules are essentially simple copy rules with filters. For example, we depicted the transformation rule to transform assembly contexts from PCM to MM15 in Listing 3.

2.2 Metrics

For a quantitative analysis, we capture metrics for each of the involved artifacts – metamodels, analyses and transformations.

Metamodels Although many studies have reported metrics for metamodels, very few of them are empirically validated (cf. Section 6), making the selection of metrics to some degree arbitrary. In this work, we use the adapted Sarkar metrics for inheritance-based coupling (IC), association-based coupling (AC), association-based coupling from opposite references (AC_{op}), association-based coupling of composite references (AC_{cmp}) and the class uniformity (CU) [10][4]

[4] In [10], we also introduced an adapted version of module uniformity (MU) but we discarded this metric as it showed major weaknesses.

```
rule AssemblyContext2AssemblyContext {
 from
  pcmAssemblyContext : PCM!AssemblyContext
 to
  assemblyContext : MM15!AssemblyContext(
   name <- pcmAssemblyContext.entityName,
   instantiatedComponent <- pcmAssemblyContext.
        encapsulatedComponent__AssemblyContext,
   provided <- pcmAssemblyContext.encapsulatedComponent__AssemblyContext
     .providedRoles_InterfaceProvidingEntity
     ->select(role | role.oclIsKindOf(PCM!OperationProvidedRole))
     ->collect(role | role.providedInterface__OperationProvidedRole),
   required <- pcmAssemblyContext.encapsulatedComponent__AssemblyContext
     .requiredRoles_InterfaceRequiringEntity
     ->select(role | role.oclIsKindOf(PCM!OperationRequiredRole))
     ->collect(role | role.requiredInterface__OperationRequiredRole)
  )
}
```

Listing 3: An exemplary transformation rule from PCM

and some basic metrics such as the number of classes (TNC), the number of features (NF, attributes and references of a class, including inherited ones) and the depth of inheritance (DIT).

Analyses For the analyses, we rely on widely accepted metrics: cyclomatic complexity, class coupling and lines of code. Additionally, we use the *Maintainability index* [16], a composite metric based on the Halstead effort, the lines of code and the cyclomatic complexity. We use the implementations of these metrics that ship with Visual Studio as the analyses are specified in C#[5].

Transformations For the ATL transformations, we use the metrics defined by van Amstel [17]. However, some of these metrics rather measure the coding style of the transformation developer such as the number of unused rules. Though these metrics are useful for transformation development, we do not expect a correlation of this metric to metamodel design. Therefore, we concentrate only on the following metrics:

- The number of transformation rules
- The number of rule inheritance trees
- The mean number of bindings per rule
- The mean cyclomatic complexity of helpers

These metrics are chosen because a correlation of them with metamodel quality seems reasonable and the metrics are sufficiently generic. We did not account for metrics such as the number of abstract transformation rules or the depth of rule inheritance trees simply because the model transformations that we used

[5] In contrast to [16], Visual Studio rescales the maintainability index to fit into the value range of 0 to 100.

as inputs hardly used abstract transformation rules and therefore, correlations are rather random. However, we included the number of inheritance trees such where transformation rules that share a common base rule are counted as one.

2.3 Analysis

For the analysis, we rely on a statistical tool: Pearson correlation indices. Because these correlation indices require the random variables to be normal distributed, we check this using Quartile-Quartile-plots (QQ-plots). QQ-plots print the quartiles of a sample distribution against the quartiles of a normal distribution. If the points appear on a line, the assumption of a normal distribution for the sample data is reasonable.

Due to the low number of data points in our experiments, we do not control for family-wise error rates. As a consequence, our results are not statistically clear but only indicate trends. More empirical studies are necessary to confirm the results of this paper to be sure on a good level of confidence.

3 Results

We divide the presentation of results into four parts: Correlations of model transformation metrics and model analysis metrics to metamodel perception and metamodel metrics. Due to space limitations, we only depict selected correlations. However, the raw metric values, scripts used to obtain the correlations and spreadsheets with all correlations are available online[6].

3.1 Correlations between Metamodel Perception and Model Transformation Metrics

The correlations between the perceived quality evaluations from the students and metrics for the model transformations are depicted in Table 1.

Very interesting is the fact that there are no significant correlations for Transformation Creation with any of the considered transformation metrics (there are also no significant correlations with any of the metrics not reported in Table 1). We have two interpretations for this fact. Either, the impact of metamodel design to model transformations is not well captured by the selected metrics or it is not easy to estimate how good a metamodel is suited for the creation of model transformations.

The strongest correlation between perceived metamodel quality and metrics of the model transformations is the connection between complexity and the mean cyclomatic complexity of helpers: A simple metamodel is correlated with a low cyclomatic complexity of helpers, can be expressed with slightly less transformation rules that contain more bindings.

[6] `https://sdqweb.ipd.kit.edu/mediawiki-sdq-extern/images/a/a0/ECMFA2018Results.zip`

	# Transformation rules	# Rule Inheritance Trees	Mean # Bindings	Mean Helper CC
Complexity	−0.16	−0.10	0.39	−0.43
Understandability	0.20	0.27	0.38	−0.06
Conciseness	0.16	0.14	0.25	−0.17
Modularity	0.34	0.44	0.03	0.08
Consistency	0.07	0.04	0.13	0.12
Completeness	−0.36	−0.20	0.17	0.33
Correctness	−0.08	−0.02	−0.05	0.26
Changeability	0.07	0.14	0.13	0.10
Instance Creation	0.00	−0.01	0.11	−0.13
Transformation Creation	0.24	0.31	−0.16	−0.11
Overall Quality	0.08	0.19	0.08	0.14

Table 1: Correlations between model transformation metrics and perceived metamodel quality

3.2 Correlations between Metamodel Perception and Model Analysis Metrics

The correlations between metamodel perception and model analysis metrics are depicted in Table 2. In this table, we have printed correlations with an absolute value higher than 0.5 in bold. For 19 observations, the threshold for a significant (i.e. significantly larger than 0) correlation is 0.53 for a one-sided test at confidence level 99% and 0.58 for the two-sided test. As mentioned above, we do not take the family-wise error-rate into account and therefore, the results only indicate trends.

We can see strong correlations (meaning that $|\varrho| > 0.5$) from the metamodel correctness to all of the selected code metrics for both analyses with the only exception for class coupling in the allocation analysis. The reason for this correlation is that metamodels perceived as incorrect often lack important navigation links. Therefore, analyses based on these metamodels often use naming conventions instead of references.

Furthermore, we have strong correlations between the overall quality and the maintainability index. This correlation is very interesting as the maintainability index is a very aggregated metric, similar to the overall quality of a metamodel being an aggregate of the quality attributes. The fact that there is a correlation between the two of them confirms the hypothesis that better metamodels make model analysis easier, even though this connection is hard to grasp, at least in terms of other quality attributes or code metrics, besides correctness.

	Class Coupling		Lines of Code		Maintainability Index		Cyclomatic Complexity	
Analysis	Alloc.	Conn.	Alloc.	Conn.	Alloc.	Conn.	Alloc.	Conn.
Complexity	0.04	0.03	0.06	−0.17	0.09	0.10	−0.22	−0.16
Understandability	−0.05	0.03	0.07	−0.04	0.11	0.08	−0.23	−0.03
Conciseness	0.20	−0.10	0.02	−0.22	0.07	0.27	−0.14	−0.21
Modularity	−0.14	−0.03	0.03	−0.01	0.09	0.07	−0.14	−0.01
Consistency	−0.25	−0.27	−0.34	−0.29	0.40	0.30	−0.36	−0.28
Completeness	−0.46	−0.08	−0.18	−0.14	0.28	0.12	−0.28	−0.13
Correctness	−0.32	**−0.54**	**−0.54**	**−0.58**	**0.59**	**0.62**	**0.51**	**−0.57**
Changeability	−0.23	−0.16	−0.24	−0.20	0.35	0.18	−0.35	−0.19
Instance Creation	−0.08	−0.05	−0.11	−0.23	0.21	0.20	−0.25	−0.22
Transf. Creation	−0.33	−0.42	−0.13	−0.36	0.33	0.36	−0.41	−0.36
Overall Quality	−0.37	0.43	−0.43	−0.48	**0.52**	**0.51**	**−0.50**	−0.47

Table 2: Correlations between model analysis metrics and perceived metamodel quality. The analyses are abbreviated with *Conn* for the analysis for *Correct Connection of Assemblies* and *Alloc* for the analysis *Correct Allocation of Assemblies*.

3.3 Correlations between Metamodel Metrics and Model Transformation Metrics

The correlations between the metrics from metamodels and model transformations are depicted in Table 3. Again, we have printed correlations with $|\varrho| > 0.5$ in bold.

	# Transf. rules	Mean # Bindings	Mean Helper CC
TNC	−0.07	−0.12	**0.82**
DIT	0.04	0.17	0.36
NF	0.09	−0.16	**−0.83**
AC	−0.40	−0.11	**0.50**
AC_{cmp}	−0.31	−0.29	0.43
AC_{op}	−0.28	−0.05	**0.51**
CU	−0.24	0.01	**−0.51**
IC	−0.41	−0.28	−0.03

Table 3: Correlations between model transformation metrics and metamodel metrics

Very interestingly, the cyclomatic complexity of the helpers correlates strongly with the total number of classes, but strongly negative with the number of features. This is due to the different implementations of parameter types in the

metamodels: While some metamodels implemented these types as enumerations (allowed by the domain description) while other implemented every enumeration literal as a type. Because these types do not have any features, also the average number of feature is lower for these metamodels. In the transformation, the helper implementation for the enumerations is a pre-initialized map from which the corresponding enumeration entry can be obtained. This causes a very low cyclomatic complexity whereas the implementation for metamodels with explicit type classes requires a more elaborated control flow.

Another interesting, though not so strong correlation is between the Sarkar coupling metrics AC or IC and the number of transformation rules, especially because the correlation is negative: A metamodel with a higher coupling can be transformed to with fewer rules. The inheritance-based coupling IC also correlates with coefficient 0.43 with the average number of output pattern elements of a transformation rule. Indeed, some of the metamodels are tightly coupled such that the conceptual separation between a component repository, a system and its deployment becomes meaningless as the classes are very intertwined. In such a case, a single transformation rule handles the transformation of all these concepts together.

However, fewer but larger rules also means that the connection between input and output model elements gets blurred. Moreover, rules also act as a unit for reuse, in particular for superimposition. This point of view also explains the positive, yet not so strong correlation of perceived metamodel modularity and the number of transformation rules and moreover the number of rule inheritance trees.

3.4 Correlations between Metamodel Metrics and Model Analysis Metrics

The correlations between metamodel metrics and model analysis metrics are depicted in Table 4.

The only correlation with $|\varrho| > 0.5$ between metamodel metrics and metrics of the model analyses is the negative correlation between the DIT of the metamodel and lines of code in the analysis: A more extensive usage of inheritance goes along with a reduction of the lines of code. This correlation is reasonable because the increased usage of inheritance enables a unified transformation of model elements. However, the correlation is only strong in the allocation analysis and much weaker in the connection analysis.

4 Discussion

The fact that the manual metamodel assessments were done by students instead of more trained experts in modeling technologies means that we essentially measure how intuitive the quality of model analyses and transformations can be guessed from the metamodel. The fact that we did not see very strong and significant correlations in many cases is therefore not surprising.

	Class Coupling		Lines of Code		Maintainability Index		Cyclomatic Complexity	
Analysis	Alloc.	Conn.	Alloc.	Conn.	Alloc.	Conn.	Alloc.	Conn.
TNC	0.00	0.28	−0.11	0.28	−0.02	−0.21	0.29	0.18
DIT	−0.23	−0.20	**−0.55**	−0.28	0.47	0.33	−0.27	−0.27
NF	−0.32	−0.41	−0.05	−0.32	0.14	0.23	−0.34	−0.24
AC	−0.14	0.24	−0.23	0.14	0.24	−0.21	0.17	−0.17
AC_{cmp}	−0.08	0.34	−0.23	0.19	0.16	−0.28	0.21	−0.05
AC_{op}	0.11	0.35	−0.15	0.10	0.12	−0.14	0.12	−0.05
CU	0.05	−0.18	−0.01	−0.23	0.06	0.14	−0.23	−0.15
IC	0.32	0.30	0.46	0.27	−0.45	−0.32	0.28	0.33

Table 4: Correlations between model analysis metrics and metamodel metrics. The analyses are abbreviated like in Table 2.

Furthermore, the correlations that we found often had explanations that are very specific to the domain: The fact that complexity influenced helpers more than transformation rules[7] is presumably specific to our case of a transformation between similar component models.

As shown for object-oriented design already [18], a higher *DIT* value eventually correlates with a fault probability, the fact that the *DIT* correlated with shorter (in terms of lines of code) analyses is therefore probably due to the experiment setup.

However, there are also connections that we think are to some degree independent of our scenario:

- The fact that a more accurate (correct) metamodel simplifies navigation through the model which in turn improves all code metrics of an analysis is a connection that we think holds independent from our study. However, we will have to repeat the study for other domains and other types of analysis to verify this.
- Using an enumeration in case there is no additional information need simplifies the analysis. In addition to the correlations, we think that a key advantage here is that using an enumeration makes it very explicit that constants are used, meanwhile this is not clear for classes that have no features: It is unclear whether they should be treated as singletons or not.
- A high coupling in the metamodel makes model transformation rules in transformations targeting that metamodel more complex as the target model elements are very intertwined. Whether the Sarkar metrics are a suitable choice to control this coupling will have to be double-checked.

Another very interesting finding is that while there is no model transformation metric that correlates strongly with the *perception* of any high-level quality

[7] The metric set by van Amstel does not include a metric to measure the complexity of model transformation rules, so we might have seen results if we had a proper metric.

attribute, we have a strong correlation even between the most abstract overall metamodel quality and very abstract code metrics such as the Maintainability Index. This may indicate that whereas code metrics do a relatively good job in capturing the complexity of code, this is not as much the case for model transformations. Thus, we require more elaborate model transformation metrics to better capture and thus predict the quality of a model transformation.

However, there is also another interpretation to this correlation, namely that the quality of a metamodel simply has more implications to artifacts that consume instances of it (such as model analyses) than to artifacts that produce these instances (such as model transformations targeting this metamodel).

Interestingly, if we have a look at the correlations between metamodel *metrics* and metrics for model transformations and model analyses, we see a different picture: While there is only one stronger correlation of the metamodels to code, we see plenty of them to model transformation metrics. We think that this is because the metrics of model transformations and metamodels are somehow closer together. A possible reason for this is that existing metamodel metrics are as bad in characterizing metamodel quality as model transformation metrics are for characterizing model transformation quality.

The results (the correlations) from this study will have to be reproduced at least in another domain and for different types of analysis for the results to have a more general applicability. Independently from the obtained correlations, they do not imply causality. Therefore, the presented paper only is a starting point to study the connections between metamodels and other artifacts in more depth.

5 Threats to Validity

As usual, we divide the threats to validity into internal and external validity.

5.1 Internal Validity

The participants in the study did not know how we wanted to analyze the peer-reviews from the metamodels, nor did they know what analyses or transformations we wanted to create based on these metamodels. Therefore, we can exclude a subject effect. For the correlation of metrics, such a subject effect is also clear.

We collected the metamodel reviews in a single session such that we can exclude an influence of histories, maturation or mortality. For the creation of the metamodels, these threats do not apply. The assignment of students to their review assignments was done by random such that we can exclude a subject effect from ourselves.

However, not all metamodels have been evaluated by all of the students as evaluating a metamodel is also quite time-consuming. Further, we may have faced an instrumentation or sequencing effect as the students evaluated the metamodels sequentially. However, we allowed the students to revoke edit their reviews after they had started reviewing other metamodels to mitigate such an effect.

Furthermore, the internal validity only affects the correlations of perceived metamodel quality to the other metrics.

5.2 External Validity

All metamodels that we have analyzed come from the same domain. Further, all analyses are in principle only two different analyses that essentially both simple validation constraints. Therefore, we cannot claim that the results are generalizable (cf. Section 4) for all domains and all analyses, but we think that the results may be a good indicator for future research.

6 Related Work

To the best of our knowledge, the study by Di Rocco et al. [7] was the only one yet to connect model transformations with metamodel metrics by means of empirical research. However, their study ignores the different purpose of the considered transformations. Furthermore, the relation of the analyzed metrics to quality of metamodels or model transformations is unclear as the study was only limited to correlations between metrics.

Most related work in the context of metamodel quality consists of adoptions of metrics for UML class diagrams and object-oriented design . In prior work, we have shown that this adoption is sometimes misleading as in the case of the Sarkar metrics [10]. Others work very well [19]. However, these studies did not consider model transformations or analysis.

Di Rocco et al. applied metrics onto a large set of metamodels [8]. Besides size metrics, they also feature the number of isolated metaclasses and the number of concrete immediately featureless metaclasses. Based on the characteristics they draw conclusions about general characteristics of metamodels. However, to the best of our knowledge, they did not correlate the metric results to any quality attributes.

A more elaborate analysis of related work in the context of measuring metamodel quality can be found in our previous work [10], [19]. We do not repeat it here due to space limitations.

7 Future Work

In the current form of the experiment, we were only able to relate the quality of metamodels to static properties of model analyses and transformations. However, we are particularly interested also in the dynamic properties such as runtime for an example model and response times to incremental updates. For this, we only need to run the transformations and analyses incrementally. For this, we plan to use NMF Expressions[8] to incrementalize the analyses and NMF Synchronizations to run the transformations incrementally [21].

Furthermore, though our results look promising, we require more data points, in particular for more domains. Therefore, we aim to repeat the study in a different domain.

[8] See [20] for a usage example

Lastly, it would be interesting to what degree the results of this study were different if we used expert opinions instead of student assessments for the metamodel quality.

8 Conclusion

In this paper, we have presented an empirical study to evaluate the influence of metamodel design to other artifacts, in particular model analyses and model transformations. We see this study as a starting point for empirical research in this direction in order to gain a better characterization of this connection and to use insights from such empirical study to improve the metamodel design process.

Besides metamodel metrics, we used intuitive metamodel assessments done by students to calculate correlations. Here, we see that the perception hardly correlates with model transformation metrics, but there are strong correlations with code metrics.

From the correlations between metamodel metrics and metrics of model analyses or model transformations, we were able to detect several correlations that we think apply independent of our experiment setup: More correct metamodels makes model analysis easier and a high coupling of the metamodels causes fewer but more complex rules in model transformations.

In general, the correlations between perceived metamodel quality and code metrics for the model analyses were stronger than the correlations between perceived metamodel quality and transformations targeting this metamodel. On the contrary, metamodel metrics seem to correlate stronger to model transformation metrics than they do to code metrics of the model analyses.

Because we had not enough data points to perform a family-wise error correction, these results have to be seen as preliminary and we look forward to repeat this study and confirm the results in future research.

Acknowledgements

We would like to thank all students that participated in our study as well as Frederik Petersen and Lennart Henseler who helped us creating the model transformations and analyses.

This research has received funding from the European Union Horizon 2020 Future and Emerging Technologies Programme (H2020-EU.1.2.FET) under grant agreement no. 720270 (Human Brain Project SGA-I).

References

[1] G. Hinkel, H. Groenda, L. Vannucci, O. Denninger, N. Cauli, and S. Ulbrich, "A Domain-Specific Language (DSL) for Integrating Neuronal Networks in Robot Control," in *2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*, 2015.

[2] G. Hinkel, H. Groenda, S. Krach, L. Vannucci, O. Denninger, N. Cauli, S. Ulbrich, A. Roennau, E. Falotico, M.-O. Gewaltig, A. Knoll, R. Dillmann, C. Laschi, and R. Reussner, "A Framework for Coupled Simulations of Robots and Spiking Neuronal Networks," *Journal of Intelligent & Robotics Systems*, 2016.

[3] M. M. Lehman, *Programs, cities, students: Limits to growth? (Inaugural lecture - Imperial College of Science and Technology ; 1974)*. Imperial College of Science and Technology, University of London, 1974.

[4] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski, "Metrics and laws of software evolution-the nineties view," in *Software Metrics Symposium, 1997. Proceedings., Fourth International*, 1997, pp. 20–32.

[5] D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, p. 25, 2006.

[6] D. Di Ruscio, L. Iovino, and A. Pierantonio, "Evolutionary togetherness: How to manage coupled evolution in metamodeling ecosystems," in *International Conference on Graph Transformation*, Springer, 2012, pp. 20–37.

[7] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Mining correlations of atl model transformation and metamodel metrics," in *Proceedings of the Seventh International Workshop on Modeling in Software Engineering*, IEEE Press, 2015, pp. 54–59.

[8] ——, "Mining metrics for understanding metamodel characteristics," in *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, ser. MiSE 2014, ACM, 2014, pp. 55–60.

[9] G. Hinkel, M. Kramer, E. Burger, M. Strittmatter, and L. Happe, "An Empirical Study on the Perception of Metamodel Quality," in *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, 2016, pp. 145–152.

[10] G. Hinkel and M. Strittmatter, "On Using Sarkar Metrics to Evaluate the Modularity of Metamodels," in *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development*, 2017.

[11] R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann, *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016, 408 pp.

[12] G. Hinkel, "NMF: A Modeling Framework for the .NET Platform," Karlsruhe Institute of Technology, Tech. Rep., 2016.

[13] D. H. Akehurst, W. G. J. Howells, M. Scheidgen, and K. D. McDonald-Maier, "C# 3.0 makes ocl redundant," *Electronic Communications of the EASST*, vol. 9, 2008.

[14] F. Jouault and I. Kurtev, "Transforming models with ATL," in *Satellite Events at the MoDELS 2005 Conference*, Springer, 2006, pp. 128–138.

[15] J. Troya and A. Vallecillo, "A Rewriting Logic Semantics for ATL," *Journal of Object Technology*, vol. 10, no. 5, pp. 1–29, 2011.

[16] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability," in *Software Maintenance, 1992. Proceerdings., Conference on*, IEEE, 1992, pp. 337–344.

[17] M. van Amstel and M. van den Brand, "Using metrics for assessing the quality of atl model transformations," in *Proceedings of the Third International Workshop on Model Transformation with ATL (MtATL 2011)*, vol. 742, 2011, pp. 20–34.

[18] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771–789, 2006.

[19] G. Hinkel and M. Strittmatter, "Predicting the Perceived Modularity of MOF-based Metamodels," in *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development*, 2018.

[20] G. Hinkel and L. Happe, "An NMF Solution to the TTC Train Benchmark Case," in *Proceedings of the 8th Transformation Tool Contest, a part of the Software Technologies: Applications and Foundations (STAF 2015) federation of conferences*, ser. CEUR Workshop Proceedings, vol. 1524, CEUR-WS.org, 2015, pp. 142–146.

[21] G. Hinkel and E. Burger, "Change Propagation and Bidirectionality in Internal Transformation DSLs," *Software & Systems Modeling*, 2017.