Usage analysis of the object constraint language in model driven engineering

Noten, J.F.H.

Award date:
2017

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Software Engineering and Technology

Usage Analysis of the Object Constraint Language in Model Driven Engineering

Master Thesis

Jeroen Noten

Supervisors:
dr. Alexander Serebrenik
ir. Josh Mengerink
ir. Jelle Schühmacher
ir. Marc Hamilton

Committee:
dr. Alexander Serebrenik
ir. Josh Mengerink
dr. George Fletcher
dr. Yaping Luo

Eindhoven, April 2017

Abstract

In model driven engineering (MDE), meta-models and model transformations are important artifacts. The Object Constraint Language (OCL) is a language used to express constraints and operations on meta-models and to query models in model transformations.

Existing empirical studies of the OCL have been conducted on small collections of data. We collect a large dataset containing both open source and industrial OCL code consisting of 116475 OCL expressions. We perform an empirical usage study to see how developers are using OCL and what the differences are among the different sources of OCL code. We use several measures to analyze the OCL expressions: distribution of expressions among classes in a meta-model, complexity of expressions and frequency of constructs used. Our usage study is based on an earlier study on a smaller dataset. We perform a replication study and find limitations in the results of the original study as well as interesting new observations.

We also identify maintainability measures to be able to give an indication of the maintainability of an OCL expression. These include a set of OCL smells, a complexity measure and guidelines that improve maintainability.

Acknowledgements

I would like to thank everyone that have spent time and effort to help me finish this project and my entire studies. In particular, thanks to my TU/e supervisors Alexander Serebrenik and Josh Mengerink for their help, advice and feedback during this project, for co-authoring my first scientific publication and for bringing me in contact with Altran for my intership. Thanks to Jelle Schümacher from Altran for his advice and feedback during my internship period and to Marc Hamilton for taking over the supervision during Jelle's absence in the last weeks.

Thanks to George Fletcher and Yaping Luo for being part of my graduation committee and for their flexibility.

Finally, thanks to my family, friends, classmates and colleagues for their support throughout this project and/or other parts of my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Model-driven engineering (MDE) is a engineering methodology that aims to capture and exploit conceptual models of topics relevant to a specific problem domain. It is becoming more and more popular to use MDE to solve problems that arise when building large and complex software systems. Examples of industries that are already practicing MDE are automotive, banking and printing [24][22][35]. This adoption is foreseen to grow exponentially in the near future, e.g., due to the convergence of software development and business analysis [8].

The Object Modeling Group (OMG) is a consortium that defines standards for MDE. One of these standards is the Object Constraint Language. OMG utilizes the language in their specifications of other standards. For example, in the UML specification, many attribute derivations, operations and constraints are specified using the Object Constraint Language. Hence, the Object Constraint Language plays an important role within OMG based model-driven engineering initiatives.

In this chapter, we first give a short introduction to model-driven engineering in Section 1.1. In Section 1.2 we explain the role of the Object Constraint Language within MDE and argue why it is important to study the usage of the Object Constraint Language. To conclude this introduction, we formulate the problem description in Section 1.3 and state the research questions in Section 1.4.

Then, in the following chapters, we discuss related work (Chapter 2), describe our research methodologies (Chapter 3) and our data collection (Chapter 4) and discuss the results (Chapter 5). Finally, we conclude this thesis (Chapter 6) with a summary that answers the research questions and propose future work.

1.1 Model-Driven Engineering

Model-driven engineering (MDE) offers an approach to address problems that arise when building large complex software systems. With MDE, system abstraction is captured in *models*, instead of in source code. The OMG provides a standard called MOF (Meta-Object Facility), which provides a base for modeling languages (e.g. UML). A modeling language can be used to write models, which can then be used to automatically generate the source code for the software system. This is expected to increase maintainability, as a developer does not have to understand the implementation details in order to make a high level change in the system. Also, domain experts that are not software engineers can make changes to the system without knowledge of the underlying code.

An example of such a model is shown in Figure 1.1. In this figure, we have created a model that can be transformed into a form for a web or desktop application by means of model transformations.

Developers can design a *meta-model* to describe what modelers are able model. This meta-model describes the entities, such as the components, attributes and relations that can be used in a model. The example model of a Form in Figure 1.1 is *an instance of* the example meta-model in Figure 1.2.

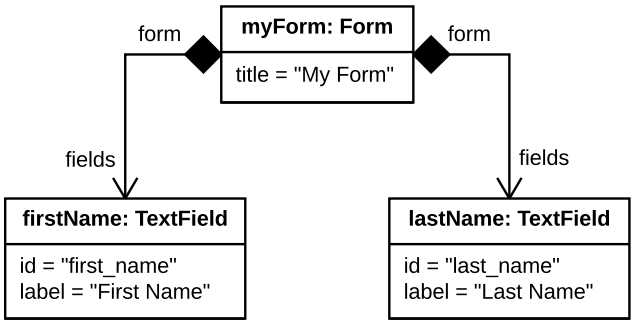

Figure 1.1: An example model of a Form.

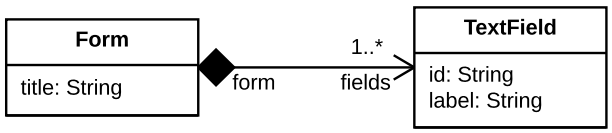

Figure 1.2: An example meta-model of a Form.

1.2 The Object Constraint Language

However, in many cases MOF only is not sufficient to impose all necessary restrictions and requirements on the models. For example, one might want to require that the title of a *Form* instance starts with a capital or that the *id*s of fields within one *Form* are unique. To overcome this limitation, the Object Constraint Language (OCL) was developed.

OCL is a textual language that can be used to write declarative expressions to impose restrictions on model attributes and relations. The language started as part of the UML standard, with the goal to overcome the limitations on specifying details of a system design. Nowadays, OCL is also used in model-driven engineering to set constraints on model attributes and relations and to query models. It is also used embedded within other languages, such as model transformation languages [4][2].

The following example of an OCL constraint describes that the title of a form must start with a capital.

```
context Form inv titleStartsWithCapital:
self.title.at(1).toUpperCase() = self.title.at(1)
```

An OCL expression has a *context*: the type of instances that *self* refers to. In this case, the context is *Form*. The keyword *inv* here indicates that this is an invariant (a boolean expression that must evaluate to true, otherwise the model is in an invalid state) and it is named *titleStartsWithCapital*. The rest of the OCL code is the expression itself and results in *true* or *false*. In this expression, we require that the title starts with a capital: the *at(1)* operation takes the first character from the title and the *toUpperCase()* makes that character uppercase (if it was not already in uppercase). Then, with the equality (=) operator, we check whether it equals the first character in the title. If so, the title starts with a capital and the expression evaluates to *true*. If not, the expression evaluates to *false*, which indicates that the model is invalid.

The second restriction that we want to describe in OCL is that within one form, all fields must have an unique *id*. One way to describe this in OCL is the following:

```
context TextField inv uniqueId:
TextField.allInstances()->forAll(
   f | f.form = self.form implies (f.id = self.id implies f = self)
)
```

This constraint is more difficult to understand: it states that for a given *TextField* (*self*), for all other *TextField*s *f*, it holds that when it is part of the same form, then, if the *id*s of *self* and *f* are equal, it is the same *TextField*. In other words: instances of *TextField*s within the same form cannot have equal *id*s.

However, there are many ways to describe this constraint. Another way is to describe this constraint in the context of the *Form*:

```
context Form inv fieldsHaveUniqueIds:
fields->forAll(f1, f2 | f1 <> f2 implies f1.id <> f2.id)
```

This is already easier to understand: it states that for a given form, for each two of its fields, when they are not the same, the *id*s are not equal. Note that we also omitted the *self* keyword here, as it is optional in OCL. This constraint can be simplified even further:

```
context Form inv fieldsHaveUniqueIds:
fields->isUnique(id)
```

This last example is clearly more readable and easier to understand than the other two: it is shorter and closer to the constraint when described in English. Therefore, it is more maintainable and in the end reduces costs [6]. This interests us to investigate how OCL is used in practice and how we can help developers to write more maintainable OCL code.

1.3 Problem Description

Constraints or queries written in OCL can become long and complex, and therefore have the risk of becoming hard to read, understand and maintain. As shown in the previous section, there can be more than one way to describe a certain constraint, the one more readable and maintainable than the other. The brevity and complexity of such an OCL specification is also related to the context of the meta-model it is in and the way the corresponding meta-model is structured. To solve this problem we plan to suggest guidelines for good OCL code and find measures that give an indication of the maintainability.

To be able to come up with such guidelines and measures, we will first look at how developers use OCL to describe constraints, queries, and transformations. We will look for patterns that are often used, and patterns that are rarely used. There may be constructs or patterns that developers do not know about but that may be used to improve readability and maintainability. We will also compare OCL code from different sources, e.g. differences between open source and industrial OCL code or between meta-models and model transformations. We will investigate if there is a notable difference in usage and how different sources can learn from each other. To be able to know which OCL code is more maintainable than others, we need to know what maintainability measures there are.

1.4 Research Questions

Summarizing, we define three research questions. As stated in Section 1.2, we are interested in the usage of OCL.

RQ1: How are developers using OCL?

This includes investigating the complexity of written OCL expressions, for what purposes OCL is being used and which patterns and constructs are used. To do so, we investigate projects from open source and industry.

However usage metrics do no directly say anything about the maintainability. To be able to give an indication of the maintainability of an OCL expression, we must define measures.

RQ2: What maintainability measures are there for OCL code?

Using these maintainability measures, we compare OCL code from different sources, e.g. differences between open source and industrial OCL code or between meta-models and model transformations.

RQ3: What are the differences in OCL usage among different sources?

Answers to the the first two questions enable answering the last one: to be able to say what the differences are between different kind of sources we need to know how OCL is used and to say which one is more maintainable, we need to know maintainability measures.

Chapter 2

Related Work

Related work can be roughly divided into three categories: studies that discuss refactoring OCL code to improve maintainability (Section 2.1), studies related to co-evolution (Section 2.2), and empirical usage studies of OCL code (Section 2.3).

2.1 Refactoring OCL Code

Quite some research has already been performed on refactoring OCL code in order to improve maintainability. Correa et al. [13][14][15][16][12] published a series of five papers in which they introduce a number of OCL smells and how to remove them using refactoring techniques. Occurrences of OCL smells are an indication of low maintainability, and therefore this series of papers is related to research question 2. The authors define an OCL smell by 'structures present in OCL expressions that might negatively affect the understandability or maintainability of OCL specifications'. The first publication in this series [13] presents five OCL smells: magic literal, *and* chain, long journey, rules exposure, duplicated code. A number of refactoring techniques to remove them and how to automate this are also described. The second paper [14] describes the same five smells plus seven additional ones: *implies* chain, redundancy, non-atomic rule, verbose expression, *forAll* chain, downcasting, type-related conditionals. They also describe more refactoring techniques in this second paper. In the third [15] and fourth [16] they perform an empirical study that investigates the impact of those OCL smells and refactorings on the understandability of OCL expressions. The results show that refactoring improve the understandability. The last one [12] is an extended version of [16] with more detailed descriptions. This paper presents the most comprehensive list of OCL smells, which can contribute to our RQ2: we can say that the number of OCL smells is one possible indication for the maintainability of an OCL expression. The paper lists the following OCL smells:

- *implies* chain, e.g. `A implies (B implies C)`
- *forAll* chain, e.g. `X->forAll(x1 | x1.Y->forAll(y1 | P(y1)))`
- verbose expression, e.g. `X->forAll(x1, x2 | x1 <> x2 implies x1.p <> x2.p)`
- long journey, e.g. `a.b.c.d`
- duplication, i.e. code clones
- downcasting, e.g. `x.oclAsType(Y).z`
- type related conditionls, e.g. `x.oclIsKindOf(A) then X else if x.oclIsKindOf(B) then Y else Z endif`

It is remarkable that in their final paper about this topic, they reduced the number of smells from 12 to only 7. Apparently, not all smells that they introduced in previous were important enough to include in this final set of OCL smells, however the exact reason is not stated in their latest paper.

In [32], Reimann et al. criticized the refactorings from Correa et al. and presented a catalogue of 28 OCL refactorings grouped in to four categories: renamings, removals/materializations, extractions/inlinings, separations/merges. For each refactoring, they provide a list of steps to follow in order to perform the refactoring. This paper is indirectly related to research question 2: if maintainability measures that we find are applied and indicate that a certain OCL expression has a low maintainability, this paper presents techniques to improve maintainability using refactoring. The same holds for the following two papers.

In [9], Cabot et al. presented refactoring techniques for OCL constraints. They divide the refactorings in two categories: (1) refactoring to an equivalent OCL expression while keeping the context the same and (2) refactoring to an alternative representation of an OCL constraint by changing the context. They also demonstrate a tool that can automatically generate all possible context changes for an OCL constraint.

2.2 Co-evolution

An important aspect of maintainability is evolvability: the ease of further developing a software element (definition taken from [29]). When multiple kind of artifacts must evolve at the same time because of dependencies, this is called co-evolution. This is often the case with meta-models and OCL code: a change in the meta-model often has impact on the OCL code as well. Studies about this topic investigate how to keep OCL code in sync when the meta-model changes. The main problem here is that the tool used to maintain mete-models cannot always know in which way the OCL code should be changed.

Proposed solutions to this problem can basically be divided into two categories. The first is using templates to write OCL constraints to make sure that the tool used to maintain meta-models knows how a piece of OCL code should be changed when a change to the meta-model is introduced [17][18][19]. This results in fully automatic co-evolving OCL code, but developers are limited in what OCL expressions the can write. The second solution is a semi-automatic co-evolution of OCL code and is implemented by letting the developer choose from a couple of alternatives [28][26].

Marković et al. [30] described how to take into account OCL constraints when applying refactorings to UML diagrams. They list 15 common UML refactorings and formalize the necessary changes of the attached constraints.

As evolvability contributes to maintainability, these papers are related to research question 2.

2.3 OCL Usage

Finally, there are a couple of studies that have investigated how developers actually use OCL. These are most similar to our research.

In 2014, *S. Ali et al.* published *Insights on the Use of OCL in Diverse Industrial Applications* [5], in which they investigate what the actual use cases for OCL are in industrial applications. They perform six industrial case studies, in which OCL is used. The use cases include model-based testing, safety certification, and automated product configuration. They also claim that a subset of OCL should be sufficient for these applications.

A very relevant study is performed by *Cadavid et al.* In 2011, they published about tooling to perform detailed analysis of OCL source code from various types of sources [10]. This paper mainly focuses on the hurdles to overcome while gathering and analyzing OCL code. In 2015, they performed a detailed study into the usage of OCL in 16 sample meta-models from different sources (OMG, academic research and industrial). Several metrics were presented, including the usage

frequency of the different OCL constructs, the complexity of the constraints and the distribution of constraints over classes [11].

Both papers are investigating how OCL is used by developers, and are therefore directly related to research question 1. The paper of *Cadavid et al.* also helps in answering research question 1 by providing techniques for measuring OCL usage. The authors also analyses OCL from open source as well as from industry, which contributes to our research question 3.

Chapter 3

Methodology

In this chapter, we describe our plan to answer our various questions. In Section 3.1, we describe our methodology for RQ1, in Section 3.2 the methodology for RQ2, and in Section 3.3 the methodology for RQ3.

3.1 RQ1: How are developers using OCL?

Since we want to know how OCL is being used in real projects, we conduct a survey research (according to the classification of Easterbrook et al. [20]). In their paper about selecting empirical methods, Easterbrook et al. [20] state that although a survey research is 'most closely associated with the use of questionnaires for data collection', 'the defining characteristic of survey research is the selection of a representative sample from a well-defined population, and the data analysis techniques used to generalize from that sample to the population'. In our case, the population consists of OCL code written by developers. So we will gather a sample data collection of OCL code and investigate how it is used. Therefore, we take the following steps:

- Gather data
- Define metrics and run them on the gathered data
- Analyze the results in order to answer RQ1

The sources of the data that we will use are real-life projects. This kind of data collection is classified by Singer et al. [34] as 'Independent Techniques' and in particular 'Static Analysis of a System'. Advantages are that the source code is usually readily available and contains a very large amount of information ready to be mined. Disadvantages are that extracting useful information often requires a lot of effort: parsers and other analysis tools are required. Such technologies are not always mature.

To collect data, we can look at multiple sources. We need to include these different sources to be able to answer research question 3. The source include open source OCL and commercial/industrial OCL. Also, OCL is used for different purposes. The OCL specification [3] lists the following:

- as a query language,
- to specify invariants on classes and types in the class model,
- to specify type invariant for Stereotypes,
- to describe pre- and post conditions on Operations and Methods,
- to describe Guards,
- to specify target (sets) for messages and actions,

- to specify constraints on operations, and
- to specify derivation rules for attributes for any expression over a UML model.

Note that the examples in the introduction demonstrated the second purpose: to specify invariants on classes and types in the class model.

OCL is also heavily used within model-to-model transformations (such as QVT [4]) and model-to-text transformations (such as Acceleo [2]) to query the model.

So we have to search for different types of places where OCL is used as well. When analyzing the OCL expressions, we have to keep in mind that there are different purposes. So we have to keep track of the purpose of each OCL expression and we will distinguish between these purposes when we apply usage measures.

3.1.1 OCL in the Eclipse Modeling Framework

Because our focus is on the usage in model-driven engineering, we will look at OCL used within tooling that supports model-driven engineering. We have chosen to focus on Eclipse-based MDE technologies, due to the fact that the Eclipse Modeling Project (`http://eclipse.org/modeling`) arguably fosters the most active open-source MDE community [27].

Ecore

At the core of the Eclipse Modeling Framework (EMF) is the Ecore meta-model. Ecore is an implementation of MOF and it is used to describe meta-models in EMF. Ecore meta-models are persisted in files with the extension `.ecore`. Ecore meta-models can be annotated with OCL in two ways: (1) OCL expressions within the `.ecore` file itself and (2) OCL expressions in a separate `.ocl` file that is linked to the `.ecore` file. As showed in the introduction, an OCL expression must always have a *context*. In Ecore, there are three different types of contexts an OCL expression can be written in: classifiers (i.e. classes, data types or enumerations), operations or properties. For each context, different types of expressions are allowed:

- classifier context:
  - `inv` to define an invariant; used to prevent modelers to model invalid instances of the meta-model
  - `def` to describe an additional feature; often used as helper for other OCL expressions to prevent duplication
- operation context:
  - `body` to define the operation body
  - `pre` to define a precondition on the operation
  - `post` to define a postcondition on the operation
- property context:
  - `init` to specify the initial value for the property
  - `der` to specify a derivation

Acceleo

Apart from Ecore, the EMF ecosystem consists of a number of other technologies, such as tools for model transformations. We choose to focus on one of these additional technologies in our usage study: Acceleo, a template based model-to-text transformation language and one of the most used EMF technologies [27]. It can for example be used to write transformations that can be applied to transform models into executable source code. Acceleo templates are stored in `.mtl` files and is an implementation of the OMG's Model-to-text specification [2]. In these MTL templates, OCL is mainly used to query the model that is transformed into text.

3.1.2 Collecting Open Source OCL Code

GitHub as Platform for OCL Data Collection

Several collections of open-source MDE projects are publicly available. For OCL, Jordi Cabot (who is also an author of the work about OCL refactorings [9]) has compiled the OCL repository[1]. However, this collection is relatively small (105 `.ocl` files and 2 `.ecore` files). Also, it does not have a consistent directory structure and way of persisting data, which hinders automated analysis. Other examples of meta-model collections include MDE Forge[2] and ReMoDD[3]. However, none of the meta-models at MDE Forge contain any OCL expression, and the ReMoDD collection contains only 81 meta-modeling related artifacts.

To create a more representative, and up-to-date data set we mine public GitHub repositories. We chose GitHub, as it is the largest source of open-source in-development software systems. Moreover, previous studies into the usage of modeling-related technologies [23, 27] have also used the GitHub data. Finally, by focussing on GitHub we ensure that our data set includes the aforementioned OCL repository of Jordi Cabot.

An alternative way of collecting data might have been using the Google search. Indeed, Google can provide access to meta-models stored on smaller sites, e.g., personal sites of researchers or their projects. However, we have observed that the indexing of GitHub files by Google is far from complete.

GitHub Search

GitHub provides advanced search features, allowing one to look for different artifacts (e.g., commits, code files or wiki entries) containing or not containing given search strings, created during a given time period and owned by given users. Intuitively, we would like to search GitHub for all repositories that contain `.ocl` files, `.mtl` files or OCL annotated `.ecore` files.

On difficulty is that GitHub requires at least one search term to be included *in addition to* the requirement that a file has a given extension, i.e. one cannot identify every file with a particular extension. Kolovos et al. [27] also faced this limitation. To mitigate this problem the authors constructed search terms that provided the largest number of relevant results. For `.ocl` they for example included the term *context*. However, this query does not match `.ocl` files without the term *context*. Using the query *extension:ocl NOT context*, we conclude that 999 code results would have been missed using the method of Kolovos et al.

Hence, we suggest an alternative approach to query construction that does not suffer from the limitations of the approach of Kolovos et al. As the search string we take *the negation of a search term that matches no `.ocl` files*. This negated term, by definition, matches all `.ocl` files. In our case the search term "foofoo" returned no results for files with the `.ocl` extension. (i.e. the query *extension:ocl foofoo* returned no results). Hence, the query *extension:ocl NOT foofoo* yields all files with the `.ocl` extension. Note that the extension `.ocl` is in some cases used for something that is not related to the Object Constraint Language, however these files will be filtered out in a later stage.

Next, we create a query that matches `.ecore` files containing OCL constraints. To enable the use of embedded OCL constraints in a `.ecore` file, the `.ecore` file should contain an annotation with the value "`http://www.eclipse.org/emf/2002/Ecore/OCL`"[4]. This value is persisted verbatim, and we use it to search for `.ecore` files containing OCL.

Finally, we want to query all `.mtl` files. However, it turns out that the vast majority of `.mtl` files on GitHub are not Acceleo templates but so called material files for 3D modeling software. Therefore, in this case, we do actually include a search term that always occurs in an Acceleo template: *module*.

Hence, we look for the "code" results of the following queries:

[1] https://github.com/jcabot/ocl-repository
[2] http://www.mdeforge.org
[3] http://www.cs.colostate.edu/remodd/v1
[4] `http://help.eclipse.org`

- extension:ocl NOT foofoo
- extension:ecore "`http://www.eclipse.org/emf/2002/Ecore/OCL`"
- extension:mtl module

`.ocl` and `.mtl` files often requires one or more corresponding `.ecore` files to be parsed and `.ecore` files also can reference other `.ecore` files. To ensure that all our data is processable, we need to also obtain these `.ecore` files. On GitHub, every *code match* belongs to a file in a repository. Rather than identifying single files, we download the full repository of the files identified by our queries, and identify the files required for parsing OCL offline.

The next limitation of GitHub search is that only 1000 results are retrieved. We circumvent this by incrementally modifying our search query: we exclude repositories that we have already been found in previous iterations (using *-repo:[user]/[repo]*). For example, if in the first iteration we find only code results from the repositories *eclipse/ocl* and *eclipse/ecore*, the query for our next iteration will be *extension:ecore* `http://www.eclipse.org/emf/2002/Ecore/OCL` *-repo:eclipse/ocl -repo:eclipse/ecore*. We repeat this procedure as long as results are retrieved. Finally, all excluded repositories form the list of relevant repositories.

Because we need to add many `-repo:[user]/[repo]` statements to the search query, we encounter another limitation: the search query length limit. This is only a problem if the number of code results is still greater than 1000 when the query length limit is hit. In the next chapter we show that the number of results is less than 1000 when we hit the query length limit. In that case, we simply add all repositories from the search results to the list of repositories.

Our data collection of OCL expression on GitHub is published at MSR17 (The 14th International Conference on Mining Software Repositories).

3.1.3 Industrial OCL Code

To be able to perform a more complete analysis, we also include industrial/commercial OCL code. Data from this kind of source is not as easy to collect as open source data. However, during the collaboration with Altran, we were able to analyze OCL code from an MDE project that utilizes the Eclipse Modeling Framework and makes use of OCL.

3.1.4 Collecting usage statistics

When we have our data, we calculate statistics to get a view of how developers are using OCL in practice. The results of these measurements will help in answering the other research questions.

In order to follow a systematic way of working, we replicate the study of Cadavid et al. [11], who analyzed the practical usage of OCL in 37 meta-models. We reuse the original methodology and apply it to our data set of OCL expressions in 485 meta-models. According to the definition of Shull et al. [33], we perform an exact replication, and in particular, a dependent replication. This means that we follow same procedure as in the original paper and only change one variable: in this case the artifacts (i.e. projects using OCL) on which the methods are applied. This ensures that when results deviate, we can be pretty sure that it is caused by the one variable (the data) that we changed, and not by e.g. the followed procedure.

Our replication of the original paper by Cadavid et al. consists of the following metrics:

- Proportions of OCL expressions in subsets of classes: shows how balanced the distribution of OCL expressions over classes in a meta-model.
- Complexity of OCL expressions: the number of distinct properties referenced in an OCL expression.
- Frequency of OCL constructs: how often is each OCL construct used.
- Frequency of OCL operations: how often is each operation called.

We utilize the Eclipse Modeling Framework to do these measurements.

Besides answering our research question, this replication study also results in either a gain in confidence in the results of the study of Cadavid et al. (for similar results) or understanding its scope (for different results).

3.2 RQ2: What maintainability measures are there for OCL code?

We use two approaches to find answers to this question: searching relevant literature for maintainability measures and deriving maintainability guidelines from usage metrics.

We will search using Google Scholar with the query *intitle:"object constraint language" OR intitle:"OCL" "readability" OR "understandability" OR "maintainability" OR "evolvability" OR "evolution" OR "usage"* to find all papers that have *object constraint language* or *OCL* in the title and use one of the keywords that are related to maintainability. Using manual selection, we filter out the papers that do not contribute to the research question.

We use Google Scholar because of its coverage. It searches through many other libraries, which eliminates the need to manually search various libraries. We deselect patents and citations, because they do not result in new papers to include. We already described the results of this literature study in the previous chapter.

Besides searching for maintainability measures in literature, we also derive maintainability guidelines from our usage metrics. If we observe an interesting pattern in our metrics, we derive a guideline. Note that this assumes that the reality is good, so we have to be aware of this.

3.3 RQ3: What are the differences in OCL usage among different sources?

We compare the usage of OCL from different sources and identify the differences. We make the following comparisons:

- Open source and industrial OCL expressions
- Meta-models and model-to-text transformations
- Expressions defined in `.ecore` and `.ocl` files.

We perform statistical t-tests to find the 95% confidence interval for these differences to make sure that our results are statistical sound.

Chapter 4

Data Collection

As described in previous chapter, we collect data in the form of OCL code from two sources: open source OCL code from GitHub and commercial OCL code from an industrial MDE project.

4.1 GitHub Data Collection

The OCL expressions in our open source data collection can be divided in two categories:

- OCL expressions used in annotations on a meta-model: located in `.ocl` and `.ecore` files
- OCL expressions used in model-to-text transformations: located in `.mtl` files

Due to the different nature of these kind of files, and the need to process them in a different way, we split the GitHub search process into these two categories. For the OCL expressions in the first category, we wrote a paper that is accepted for the Data Showcase track for the MSR (Mining Software Repositories) conference 2017 [31].

4.1.1 GitHub Search

We follow our GitHub search methodology as described in the previous chapter and run the following three queries:

- extension:ocl NOT foofoo
- extension:ecore "`http://www.eclipse.org/emf/2002/Ecore/OCL`"
- extension:mtl module

In March 2017 the first two queries produced 6237 and 1045 hits, respectively. The third query was ran in April 2017 and produced 7383 hits.

During the search process in which we incrementally exclude repositories, as described in previous chapter, we did reach the query length limit. However, this did not lead to problems because at the point when we reached the limit, the number of search results was less than 1000, so we added all remaining repositories that occurred in the search results to our result set. (Note that this limitation is crucial for scalability of this search strategy. For bigger result sets, this limitation will be a problem, so this strategy is only applicable when the result set is small enough.)

As a result of this search process, we have three lists of repositories, one for each of the three search queries. We merge the first two, because `.ocl` and `.ecore` files contain the same kind of OCL expressions: annotations on a meta-model. This results in 519 repositories containing OCL in `.ocl`/`.ecore` files and 459 repositories containing OCL in `.mtl` files.

4.1.2 Downloading and Stripping the Repositories

Using a Python script, we download the 519 relevant repositories. Next, we remove all files and empty directories other than files ending in `.ocl`, `.ecore` and `.mtl`, as we only want to keep those for our data set. This results in a collection of of 6258 `.ocl` files, 21188 `.ecore` files and 13541 `.mtl` files. In order to keep the files parsable we preserve the original file names and the directory structure. This is important, because import statements that reference dependencies are hardcoded and often refer to other directories.

4.1.3 Identifying unique files

We have observed that there are many duplicate files both in the same repository as well as across repositories. This happens for example when files or directories are copied or when dependencies are included. To prevent bias in the usage statistics, we only want to include unique files.

Hence, we first identify duplicates using MD5 hashing of the files (i.e. their content). In the collection of 6258 `.ocl` and 21188 `.ecore` files, only 11706 files are unique (spread over the 519 repositories). In the collection of 13541 `.mtl` files, only 6783 files are unique (spread over the 459 repositories).

4.2 Industrial Data Collection

Our collection of industrial OCL code consists of 7 `.ecore` meta-models and 191 `.mtl` model-to-text transformations. This industrial dataset is very small compared to our open source dataset. Ideally we would have more industrial sources, but industrial projects are more difficult to collect compared to open source projects, due to the confidential nature of these industrial projects. We got access to only one industrial project.

4.3 Parsing

4.3.1 Parsing Meta-Model Files

Using the Eclipse Modeling Framework, we parse all unique `.ocl` and `.ecore` files and store them as abstract syntax trees (ASTs) in XMI format conforming to the OCL Pivot Meta Model [36]. We successfully parse 8947 files (76%) resulting in 8947 AST files. The remaining 2759 files resulted in parse errors, due to e.g. the extension `.ocl` being used for technologies not related to the Object Constraint Language, missing references, or syntax errors. Of the 519 repositories, 274 contained no parsable OCL constraints at all. Since we are only interested in files with (parsable) OCL expressions, we exclude AST files with no parsable OCL expressions. This step resulted 504 AST files containing 9188 OCL expressions, derived from 245 ($519 - 274$) repositories.

4.3.2 Parsing Model-to-Text Transformation Files

Of the 6783 unique `.mtl` files, we successfully parse 2634 (39%) of them. The other files resulted in parse errors. Many of the unparsable `.mtl` files are related to other file types, some have missing dependencies or syntax errors. Of the 459 repositories containing `.mtl` files, 385 (84%) repositories contain at least one parsable `.mtl` file.

In Meta-Models files, OCL expressions are easily separated from the meta-models, because they are really additions to the meta-model. However, in `.mtl` files, this separation is less clear. Let us first describe how such an `.mtl` file looks like.

An `.mtl` file contains templates and/or queries. Templates can be used to generate output files or as sub-template in other templates. Queries can be used as helper functions in templates or other queries. An `.mtl` file can import other `.mtl` files to reference templates and queries in other files.

Templates

Templates are functions that take parameters and output a string. A very basic example, taken from the MOF Model To Text Transformation Language specification [2] is the following template that transforms a model class into a Java class. It is represented as a piece of text with placeholders for data to be extracted from models.

```
[template public classToJava(c : Class)]
class [c.name/]
{
    // Constructor
    [c.name/]()
    {
    }
}
[/template]
```

These placeholders can contain OCL expressions. Besides simple placeholders, logical blocks `if`, `let`, and `for`, blocks that indicate `file` storage and invocations of sub-templates and queries are also possible. All these blocks make use of OCL, e.g. to define variables, for loop guards, and for conditionals. So we identify the following use cases for OCL in templates:

- in placeholders that are of type String
- in argument lists for query invocations
- in template invocations in the arguments list, and as *each*, *before* and *after* expressions
- as guard for a template
- in a `for` block as guard, collection expression and as *each*, *before* and *after* expressions
- in an `if` block as guard
- in a `let` block as variable expression
- in a `file` block as expression for the filename

In the next chapter, we will measure the usage in these use cases for OCL in `.mtl` files.

Queries

Queries are simply OCL expressions with a name and parameters. The following is a basic example of an Acceleo query.

```
[query public nameSpace(n : String) : String =
 n.replaceAll('::', '.')
/]
```

This query is named *nameSpace*, is declared *public* (which means that it can also be used by other `.mtl` files that depend on this file), accepts a *String* and returns the same string with all double colons replaced with a dot.

The extraction of OCL here is easier: we take the expression and add it to our OCL expressions to analyze.

	Open-source		Industrial	**Total**
	.ecore	.ocl	.ecore	
Files	428	76	7	**511**
Expressions	7554	1619	73	**9246**
- Constraints	5333	1317	58	**6708**
- Operations	1828	231	24	**2083**
- Properties	393	71	11	**475**

Table 4.1: Overview of OCL expressions in our meta-model data collection

	Open-source	Industrial	**Total**
Files	2634	191	**2825**
Expressions	94089	13140	**107229**
- Template Expression	29278	2531	**31809**
- Template Invocation Argument	20442	3891	**24333**
- Query Invocation Argument	10725	3389	**14114**
- If Expression	12543	437	**12980**
- Query Expression	7304	1152	**8456**
- For Iter	6127	684	**6811**
- File URL	2075	148	**2223**
- Template Post Condition	1626	507	**2133**
- File Charset	1736	148	**1884**
- Template Guard	1153	0	**1153**
- For Each	809	245	**1054**
- For Guard	153	0	**153**
- For Before	85	8	**93**
- For After	31	0	**31**
- Template Invocation After	1	0	**1**
- Template Invocation Before	1	0	**1**

Table 4.2: Overview of OCL expressions in our model-to-text transformation data collection

4.4 Overview

Table 4.1 gives an overview of our collection of 9246 OCL expressions in 511 meta-models from open-source and industry. The OCL expressions are subdivided in constraints (including pre- and post-conditions), operations and properties defined using OCL. We use these categories when applying our metrics, so that we can compare usage for different types.

Table 4.2 gives an overview of our collection of 107229 OCL expressions in 2825 model-to-text transformations from open-source and industry. The OCL expressions are subdivided in categories as described in section 4.3.2.

4.5 Limitations

The fact that our industrial data collection is small limits the possibilities for statistical comparison. We have to keep in mind that any comparison between our open source and industrial data cannot be generalized to all industrial OCL usage.

Several threats to validity have been introduced by our decision to *use GitHub*. The "peril of mining GitHub" [25] most relevant for our work is that "many active projects do not conduct all their software development on GitHub". To mitigate this threat, as a future work we plan to extend the data set with additional sources of data, such as SourceForge, OMG documents, and scientific articles.

The limitations of the *search functionality of GitHub* [1] also induce several threats to validity.

The search functionality of GitHub only allows searching of the main branch in repositories, i.e. our search query might miss files [7]. However, our data is less likely to contain experimental files, giving a more accurate representation of finished products. Similarly, files might be missed due to project forks being excluded by default from the GitHub search. While, in general, this is beneficial as it reduces noise in the data, it is also possible that forks contain new data as well, which we then miss.

Moreover, only files smaller than 384 KB are searchable. This means that the search misses repositories in which *all* `.ocl` and `.ecore` files exceed 384KB (note that we download full repositories that we identified, potentially including files bigger than 384KB). To estimate the number of `.ecore` and `.ocl` files larger than 384KB, we investigate the repositories that we included in the data set. We conclude that of all `.ecore` and `.ocl` files in the repositories, 3% is bigger than 384 KB. We therefore expect the impact of this threat to be limited.

Another limitation of the search pertains to very big repositories: GitHub search covers only repositories with fewer than 500,000 files. This may cause us to miss files in very large repositories.

Chapter 5

Results

As described in our methodology for research question 1 (*How are developers using OCL?*), we replicate the study of Cadavid et al. [11]. They analyzed the practical usage of OCL in 37 meta-models. We reuse the original methodology and apply it to our data set of 9246 OCL expressions in 511 meta-models and 107229 OCL expressions in 2825 model-transformations.

5.1 Proportions of OCL expressions in subsets of classes

5.1.1 Replication Study

To answer their research question '*How balanced is the distribution of OCL invariants definitions in the domain structure?*', they measured for each meta-model the distribution of OCL expressions over the classes in the meta-model (remind that each OCL expression in a meta-model has a context that is a class or belongs to a class). This shows for example whether OCL expressions are equally scattered over the meta-model, or if they tend to concentrate on a subset of the structure. Note that this metric can only be applied to OCL in meta-models, because in model transformations, OCL expressions are not attached to a class.

They conclude that there their collection of meta-models '*comprises both balanced and unbalanced meta-models*'. We replicated their measurements with our data set of open source and industrial meta-models. The results are shown in Table A.1 in Appendix A. Note that our table is very long because of the size of our data set (504 open source meta-models and 7 industrial meta-models) and the fact that there is a row in the table for each meta-model. Each row in the table displays the cumulated proportion of invariants defined on each percentile of classes. As in the original paper, the table is sorted from the most balanced meta-model to the most unbalanced. Although the authors did not explain how to sort the meta-models, we use the Gini index [21] to sort the meta-models from balanced to unbalanced. We choose this measure of statistical dispersion, because it measures the inequality among values of a frequency distribution. In our case, this translates to a value that indicates the distribution of OCL expressions over the classes in the meta-model.

We observe the same as the authors of the original paper: our collection of meta-model '*comprises both balanced and unbalanced meta-models*'. This indicates a gain in confidence of the original study and contributes to our research question 1 as well. However, this is not a very interesting conclusion; we will go a step further in the next subsection.

5.1.2 Comparison Between Different Sources

Let us plot the Gini indexes; see Figure 5.1 for a violin plot with Gini indexes per source: open source `.ocl` files, open source `.ecore` files and industrial `.ecore` files (note that we do not have `.ocl` files from industry). It is interesting to see that in the meta-models with separate `.ocl` files, the expressions are much more balanced. To be precise, the Gini index for `.ocl` files is on average

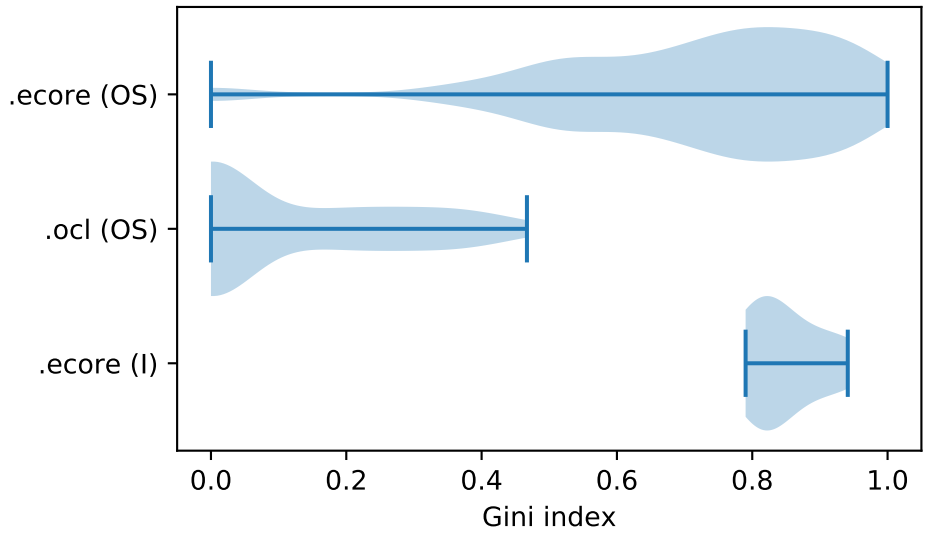

Figure 5.1: A violin plot with Gini indexes of the distribution of OCL expressions over classes in meta-models for each source

0.50 greater than `.ecore` with a 95% confidence interval of $[0.55, 0.65]$, computed with a t-test. A reason for this could be that when OCL expressions are distributed over most classes, developers see the need to use a separate `.ocl` file to keep track. This might suggest a guideline to define OCL expressions in a separate `.ocl` file when the expressions are scattered over the meta-model.

The meta-models from industry are all not very balanced. Note that this is not necessary good or bad, it only means that most business logic is concentrated on a subset of the classes in the meta-models. This contributes to research question 3.

To conclude this section: we successfully replicated this part of of the study by Cadivid et al., which indicates a gain in confidence of the original study. We also observed that expressions defined in `.ocl` are more scattered over the meta-model than expressions defined in `.ecore` files itself.

5.2 Complexity of OCL expressions

5.2.1 Replication Study

Cadavid et al. defined complexity of an OCL expression as the number of distinct properties that is referenced in the expression. We can calculate this using EMF by traversing the abstract syntax tree of an OCL expression and in each node of type `PropertyCall`, we get the referred property and add it to the set of referred properties. The size of this set is the complexity of the expression.

Cadavid et al. plotted the complexities of all OCL expressions using a boxplot per meta-model and mentioned a few observations:

- In 81% of the meta-models in their collection, the complexity of an OCL expression is 8 or less.
- In 39% of the meta-models, the complexity is even 4 or less.
- Among all expressions, 87.62% have a complexity of 4 or below.
- The most complex OCL expression has complexity 38.

Complexity	# of OCL expressions
0	1498
1	2688
2	2338
3	1369
4	594
5	249
6	188
7	100
8	68
9	27
10	12
11	15
12	10
13	40
14	68
36	2

Table 5.1: Complexities of OCL expressions in meta-models

They conclude that there are strong variations in invariants complexities from one meta-model to the other, even if most of them define simple invariants.

We first replicate these bloxplots using our own collection of 504 meta-models and see to what extent we can replicate these statistics and conclusions. Due to the large number of meta-models, to keep the diagrams readable, we are forced to divide the bloxplots over multiple figures. See the figures in Appendix B.1. To replicate the statistics we need exact numbers.

Table 5.1 shows the number of OCL expressions for each complexity. Table 5.2 shows the complexities of the most complex OCL expression per meta-model and the number of meta-models with that maximum complexity.

With this data, we can replicate the statistics mentioned by Cadavid et al. and compare them to the original values.

- In 95% of the meta-models in our collection, the complexity of an OCL expression is 8 or less. This is even more than the original percentage of 81%.
- In 79% of the meta-models, the complexity is even 4 or less, which is a much higher percentage than the 39% mentioned in the original paper.
- Among all expressions, 91.59% have a complexity of 4 or below, which is also slightly more than the 87.62% from the original study.
- The most complex OCL expression has complexity 36, which is slightly less than the original 38.

All these statistics indicate that on average, our large data collection contains less complex OCL constraints than the smaller collection of Cadavid et al. (Unfortunately, we cannot perform a statistical comparison due to the absence of exact data in the original paper.) To review their conclusion: it is indeed the case that there are strong variations in complexities from one meta-model to the other, even if most of them define simple expressions.

5.2.2 Comparison Between Different Sources

To give a total overview of the complexities in all analyzed OCL expressions in our collection of meta-models, we plot all expressions in a boxplot, see Figure 5.2. For the outliers, darker points

Maximum complexity	# of meta-models
0	30
1	93
2	125
3	105
4	49
5	27
6	31
7	13
8	10
9	5
10	6
12	7
14	4
36	2

Table 5.2: Complexities of the most complex OCL expression per meta-model and the number of meta-models with that maximum complexity

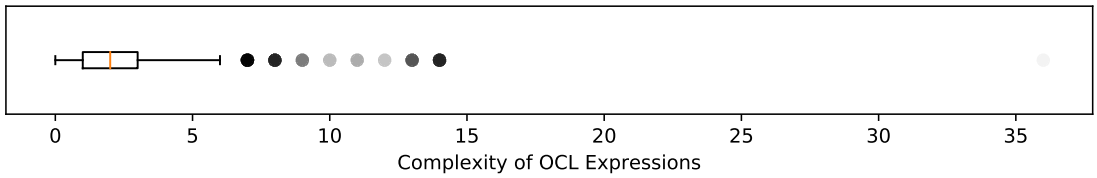

Figure 5.2: Boxplot with the complexities of all OCL expressions in our collection of **meta-models**.

indicate multiple overlapping points, in a range from 2 (the lightest outlier) to 100 (the darkest outlier).

Let us now see how this compares to the 107229 OCL expressions in our collection of model-to-text transformation files. We have plotted the complexities of these expressions in figure 5.3.

This shows that OCL expressions in model-to-text transformations are simpler than OCL expressions in meta-models. To be more specific: 97.9% has a complexity of 2 or less, and 99.6% has a complexity 4 or less, against 79% of the OCL expressions in meta-models. This makes sense: meta-models should contain domain logic, model-to-text transformations should not. This might suggest a guideline to keep the complexity in transformations low (e.g. 2 or less) and extract complex expressions to the meta-model or queries.

If we compare our industrial and open-source OCL expressions, we see that the industrial OCL expressions have a on average a lower complexity than the open-source OCL expressions. See Figure 5.4. In particular, expressions in industrial model transformations are very simple:

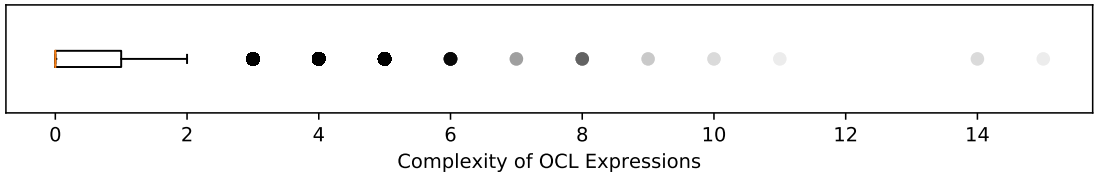

Figure 5.3: Boxplot with the complexities of all OCL expressions in our collection of **model-to-text transformations**.

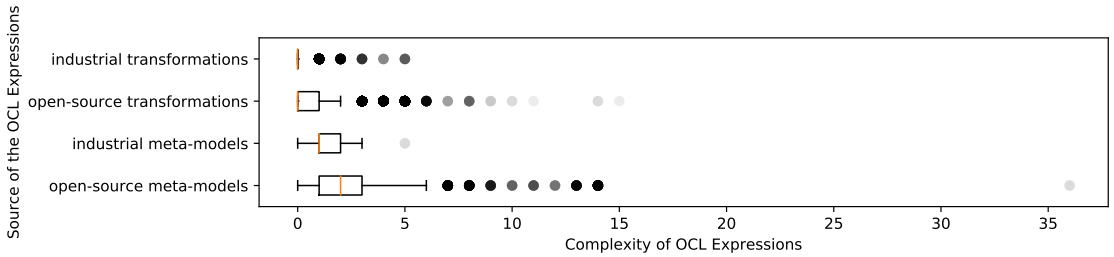

Figure 5.4: Complexity of OCL expressions per source.

only the outliers in the boxplot show a complexity greater than 0. To be precise: 94% of the expressions in our dataset of industrial model transformations have a complexity of 0.

5.2.3 Comparison Between Different Types

We now compare complexities between the different types of OCL expressions as listed in Table 4.1 and Table 4.2. See Figure 5.5 with a boxplot of complexities of OCL expressions in meta-models for each type. See Figure 5.6 with boxplots of complexities of OCL expressions in model-to-text transformations per type.

Different Expression Types in Meta-models

The boxplots for *Constraint* and *Property* expressions in meta-models are equal if we ignore outliers. However, there are relatively more outliers for the *Constraint* expressions than for *Property* expressions (4.7% versus 2.7%) and with a higher maximum (14 versus 10), which also results in a higher mean (2.4 versus 1.9) and variance (5.1 versus 2.8) for the *Constraint* expressions. So we conclude that although *Constraint* and *Property* expressions have the same complexity median, on average *Constraint* expressions are more complex and vary more in complexity than *Property* expressions. For *Operation* expressions, the median and mean are lower (median = 1, mean = 1.3) than for the other two expression types. However, because of two outliers with a complexity of 36, the variance (= 3.1) is higher is between the variance of the other two expression types.

Different Expression Types in Model-to-text transformations

Among the expression types of the model-to-text transformations, *TemplateExpression* has the greatest mean (1.0). It makes sense that this mean is 1.0: *TemplateExpresions* are OCL expressions that result in *String* and are directly placed in the template, and this is often just a reference to a property on the model, which results in a complexity of 1. The *QueryExpression* has the greatest variance in complexity (1.2), however the mean is lower than the *TemplateExpression* (0.6) (the *TemplateExpression* has a mean that is on average 0.51 higher than *QueryExpression*, with a confidence interval of $[0.41, 0.61]$). This is remarkable: *QueryExpression*s are basically helper functions used to extract complex or duplicate expressions in order to keep templates clean. However, on average they are less complex than *TemplateExpression*s. Other types that are on average more complex than *QueryExpression*s are *ForIter* (the set to iterate over in for block), *ForGuard* (the guard in a for block), *TemplateGuard* (the guard in a template block) and *IfExpression* (the condition in an if statement).

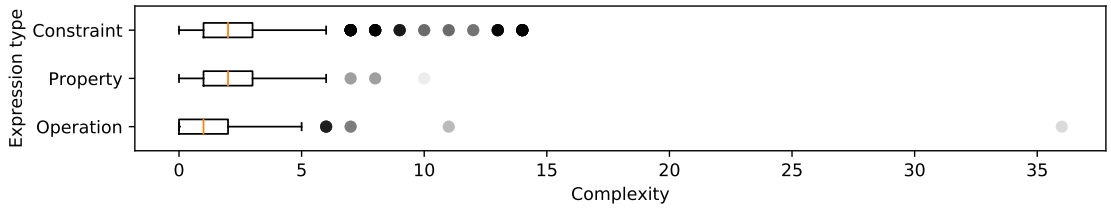

Figure 5.5: Complexity of OCL expressions in **meta-models** per type.

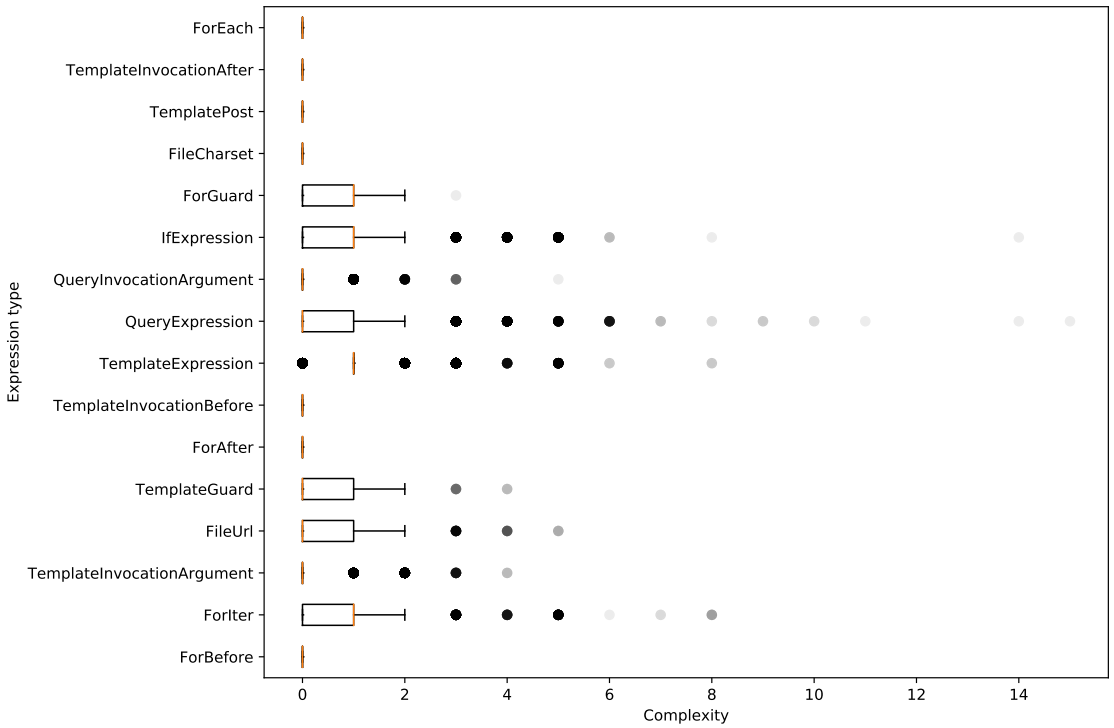

Figure 5.6: Complexity of OCL expressions in **model-to-text transformations** per type.

5.3 Frequency of OCL Constructs and Operations

5.3.1 Replication Study

Frequency of OCL Constructs

In order to replicate the last part of the usage study of Cadavid et al. [11], we count the frequency of each OCL construct in all meta-models. Figure 5.7 shows these frequencies. Refer to Table 5.3 for the exact numbers.

These results are in line with the original results by Cadavid et al. [11], e.g. the set of the 6 of most used constructs is the same. Also, the chart looks similar, which indicates a gain in confidence of the original study. However, we cannot measure the differences precisely due to the lack of exact numbers in the original study. They present some statistics we can compare though:

- The 10 constructs *OperationCallExp*, *VariableExp*, *PropertyCallExp Type*, *Iterator*, *CollectionLiteral*, *EnumLiteral*, *BooleanLiteral*, *If* and *IntegerLiteral* capture 98.6% of the constructs present in their data collection.
- 96.9% of all OCL expressions in their data collection rely only on these 10 constructs (so

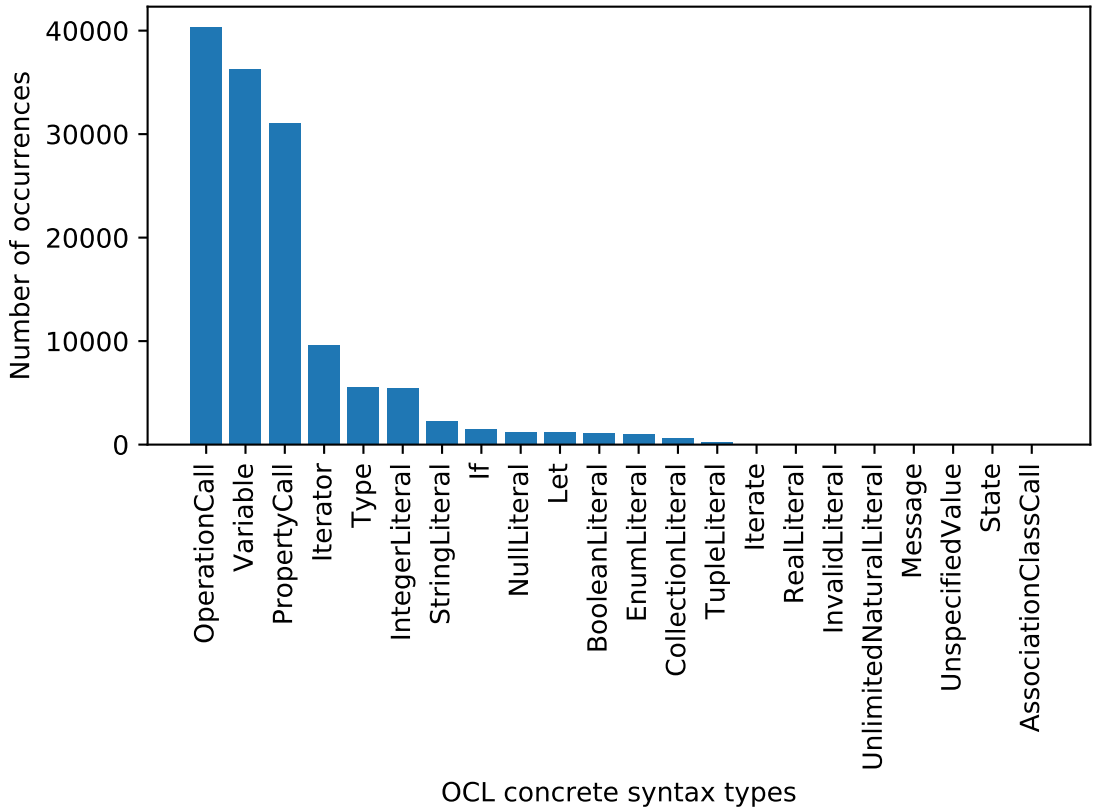

Figure 5.7: Frequency of OCL constructs in analyzed expressions.

only 3.1% make some use of the remaining expression types). This means that 96.9 % of the OCL expressions are expressed with 45.5% of the OCL (10 constructs out of 22 concrete expression types).

We replicate these statistics with our own data collection:

- The 10 constructs *OperationCallExp*, *VariableExp*, *PropertyCallExp Type*, *Iterator*, *CollectionLiteral*, *EnumLiteral*, *BooleanLiteral*, *If* and *IntegerLiteral* capture 95.6% of the constructs present in our data collection. This is comparable to the 98.6% in the original paper, although our percentage is slightly less.
- 68.3% of all OCL expressions in their data collection rely only on these 10 constructs. This is a much lower percentage than the 96.9% mentioned in the original paper.

This last comparison is an important observation: from the original paper, one might conclude that since 96.9% of all expressions only use the 10 mentioned constructs, it is not very important to support the other 12 constructs in for example tooling or teaching. However, our larger data collection indicate that this percentage is much lower.

Frequency of OCL Operations

According to Cadavid et al. 'one particular type of expression, *OperationCall*, deserves special attention since due to its nature of an expression used to invoke an operation, and different operations are called among the matched occurrences'. It is also the most used OCL construct. Figure 5.8 shows the resulting frequencies of the referred operation of the *OperationCall* constructs

Construct	Frequency
OperationCall	40300
Variable	36275
PropertyCall	31024
Iterator	9605
Type	5584
IntegerLiteral	5475
StringLiteral	2265
If	1495
NullLiteral	1245
Let	1217
BooleanLiteral	1107
EnumLiteral	1000
CollectionLiteral	629
TupleLiteral	271
Iterate	89
RealLiteral	82
InvalidLiteral	37
UnlimitedNaturalLiteral	19
Message	0
AssociationClassCall	0
State	0
UnspecifiedValue	0

Table 5.3: Frequency of OCL constructs in analyzed expressions

in our data collection. We only show the top 25 operations, to make the chart readable. Table 5.4 shows the exact numbers.

This chart is not in line with the original results. We mention a few notable differences: the most used operation in the original paper is not even in our top 25. Furthermore, our second most used operation 'and' is at place 21 in the original paper. These differences imply that the scope of the conclusions from this data in the original paper is limited.

5.3.2 Comparison Between Different Sources

To compare the construct usage frequency among different sources, we add the constructs used in the model-to-text transformations (note that the data in the previous subsection is from meta-models only) and we make a distinction between open-source and industrial usage. This makes four categories, which we present in a stacked bar chart in Figure 5.9. The values are normalized by the total amount of constructs in the category (i.e. all bars of each color add up to 1).

It is remarkable that in this diagram, the *Variable* expression has the highest frequency, instead of the *OperationCall* expression. This is due to the fact that model-to-text transformations have a much higher usage of the *Variable* construct and a much lower usage of the *OperationCall* construct.

Another expression that has a much higher usage in model-to-text transformations than in meta-models is the *StringLiteral* construct. This makes sense because templates output strings.

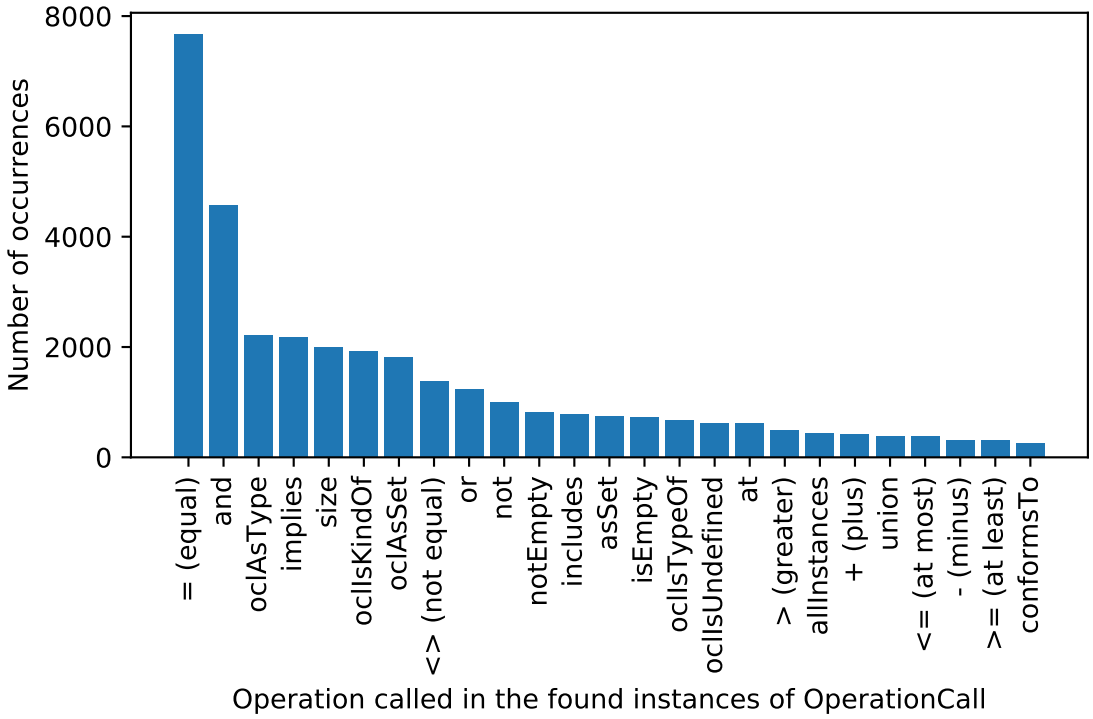

Figure 5.8: Frequency of top 25 of operations called in occurrences of *OperationCall*.

Operation	Frequency
=	7676
and	4575
oclAsType	2215
implies	2182
size	1998
oclIsKindOf	1921
oclAsSet	1825
<>	1390
or	1240
not	996
notEmpty	815
includes	774
asSet	754
isEmpty	726
oclIsTypeOf	667
oclIsUndefined	627
at	613
>	485
allInstances	445
+	411
union	388
<=	384
−	314
>=	309
conformsTo	264

Table 5.4: Frequency of top 25 of operations called in occurrences of *OperationCall*

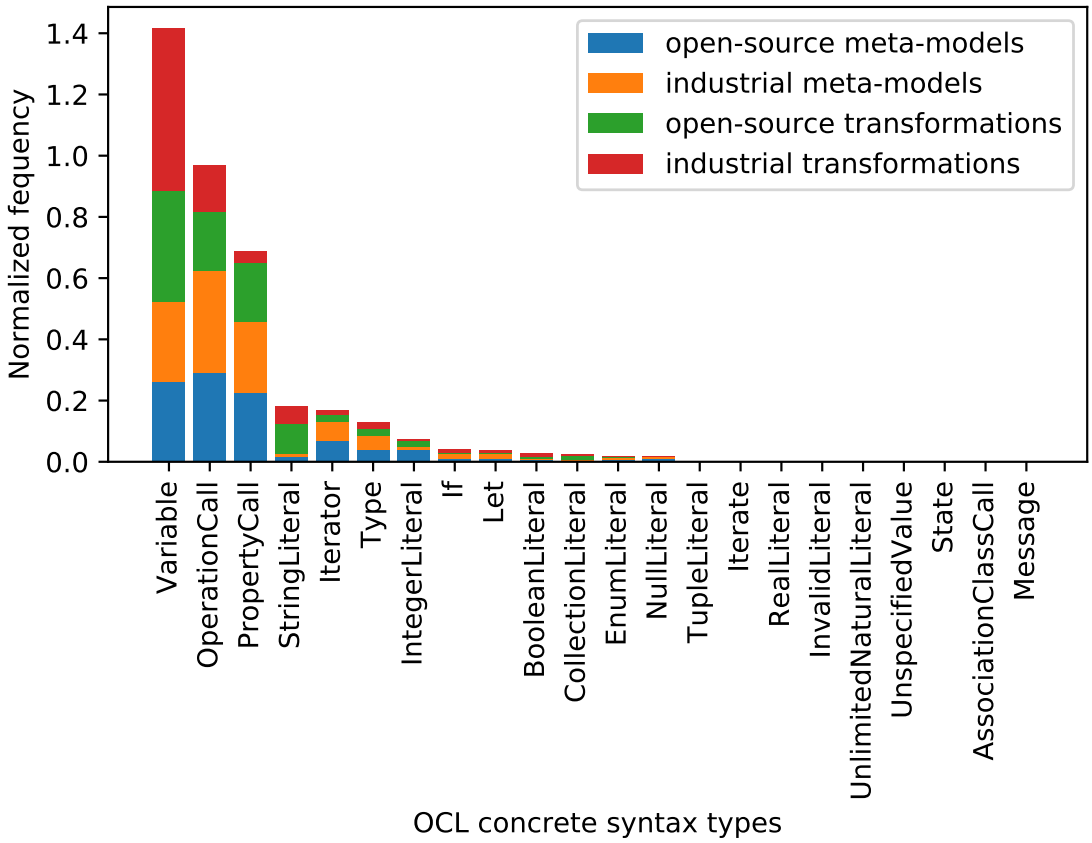

Figure 5.9: Normalized frequency of OCL constructs in analyzed expressions per source.

Chapter 6

Conclusions

In this last chapter, we summarize the answers to the research questions and discuss how others can continue this study in the Future Work section.

6.1 Answers to Research Questions

6.1.1 RQ1: How are developers using OCL?

In Chapter 4, we described our data collection of OCL expressions. The open source part of this data collection is also published at MSR17 (The 14th International Conference on Mining Software Repositories).

We performed several usage measurements on this data collection and in Chapter 5, we extensively discussed the results these measurements. We mention the most interesting results:

- Embedding OCL expressions in `.ecore` files is more common than OCL expressions in a separate `.ocl` file: 82% of the OCL expressions in our meta-model data collection are defined in `.ecore` files.

- OCL expressions that are defined in separate `.ocl` files, are generally more balanced over the meta-model, i.e. a greater percentage of the meta-classes in have OCL expressions.

- OCL expressions in model-to-text transformations are generally less complex than OCL expressions in meta-models: 99.6% of the expressions in transformations in our data collection have a complexity of 4 or less. For meta-models, this percentage is 79%. In the industrial transformations in our dataset, 94% of the expressions have a complexity of 0.

- In model-to-text transformations, OCL expressions of type *TemplateExpresion* are generally more complex than *QueryExpression*s, however *QueryExpression*s have more outliers in terms of complexity.

- Cadavid et al. name 10 OCL constructs that capture 98.6% of the constructs present in their data collection. They also state that 96.9% of all OCL expressions in their data collection rely only on these 10 constructs. However, we found that in our data collection, only 68.3% of the OCL expressions (in meta-models) rely exclusively on these 10 constructs.

- According to our measurements *and* the measurements of Cadavid et al., the *OperationCall* construct is the most used construct in OCL expressions defined in meta-models. However, we observed that this does not hold when also taking OCL expressions in model-to-text transformations into account: then the *Variable* construct type has the highest frequency.

6.1.2 RQ2: What maintainability measures are there for OCL code?

We have found answers to this question mainly in literature. In Chapter 2, we discussed a number of relevant papers, including a series of papers by Correa et al. [13][14][15][16][12] in which they present a list of OCL smells and associated refactorings to remove the smells. We proposed that these smells can be used as maintainability measures: a high number of occurrences of smells indicate low maintainability.

Another maintainability measure that we found in literature is related to complexity. Cadavid et al. [11] defined complexity of an OCL expression as the number of distinct properties referenced in the expression. This can also be used as maintainability measure: high complexity indicate low maintainability.

Besides these measures from literature, we also deduced a number of suggestions for maintainability guidelines during our usage study:

- Define OCL expressions in a separate `.ocl` file when the expressions are scattered over the meta-model.
- Keep the complexity in transformations low (e.g. 2 or less) and extract complex expressions to the meta-model or queries.

6.1.3 RQ3: What are the differences in OCL usage among different sources?

During our usage measurements, we studied differences between open source and industrial usage of OCL and differences between OCL usage in meta-models and in model-to-text transformations. For meta-models, we also compared expressions defined in `.ocl` with expressions defined in `.ecore`. Since this question has some overlap with RQ1, we already mentioned some differences in OCL usage among different sources in section 6.1.1. We repeat them in short:

- OCL expressions that are defined in `.ocl` files, are generally more balanced.
- OCL expressions in model-to-text transformations are generally less complex than OCL expressions in meta-models.
- In meta-models, *OperationCall* is the most frequent used construct construct is the most used construct; in transformations the *Variable* construct type has the highest frequency.

Other interesting differences:

- In general, industrial OCL expressions are less complex than open source OCL expressions, especially OCL expressions in industrial transformations are very simple: 94% of the expressions in our dataset of industrial model transformations have a complexity of 0.
- The *StringLiteral* OCL construct has a much higher usage in model-to-text transformations than in meta-models. This is logical, because templates are meant to produce strings.

6.2 Future Work

As future work, we consider extending our open source data set with OCL expressions and corresponding meta-models and transformations from other sources such as Google Code, SourceForge and data sources used by existing research into the OCL. It would also be nice to have a larger industrial dataset.

We have performed a number of measurements on our dataset. However, advanced measurements that require more complicated analysis such as pattern matching are outside the scope of this thesis, so we propose this for future work. Examples are automatic detection of OCL smells or finding recurring patterns in OCL expressions. We have published a dataset of open source OCL

expressions that can be used to perform future work. This makes it easier for other researchers to analyze OCL usage as they do not have to compose their own data collection.

Another future study could be an evaluation of the maintainability measures and guidelines that we proposed. I would be interesting to know to which extent these measures and guidelines indeed indicate or improve maintainability. Such a study could also be used to make these measures more concrete and quantitative.

Bibliography

[1] GitHub help — searching code. https://help.github.com/articles/searching-code/. Accessed: 2017-03-14. 17

[2] MOF Model to Text Transformation Language. http://www.omg.org/spec/MOFM2T/1.0/. Accessed: 2017-04-14. 2, 10, 16

[3] Object Constraint Language. http://www.omg.org/spec/OCL/2.4/. Accessed: 2017-04-13. 9

[4] Query/View/Transformation Specification. http://www.omg.org/spec/QVT/1.3/. Accessed: 2017-04-26. 2, 10

[5] Shaukat Ali, Tao Yue, Muhammad Zohaib Iqbal, and Rajwinder Kaur Panesar-Walawege. Insights on the use of ocl in diverse industrial applications. In *International Conference on System Analysis and Modeling*, pages 223–238. Springer, 2014. 6

[6] T. Bakota, P. Hegeds, G. Ladnyi, P. Krtvlyesi, R. Ferenc, and T. Gyimthy. A cost model based on software maintainability. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 316–325, Sept 2012. 3

[7] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. Germán, and Premkumar T. Devanbu. The promises and perils of mining git. In *MSR*, pages 1–10, 2009. 18

[8] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012. 1

[9] Jordi Cabot and Ernest Teniente. Transformation techniques for ocl constraints. *Science of Computer Programming*, 68(3):179–195, 2007. 6, 11

[10] Juan Cadavid, Benoit Baudry, and Benoit Combemale. Empirical evaluation of the conjunct use of mof and ocl. In *Experiences and Empirical Studies in Software Modelling (EESSMod 2011)*. CEUR, 2011. 6

[11] Juan José Cadavid, Benoit Combemale, and Benoit Baudry. An analysis of metamodeling practices for MOF and OCL. *Computer Languages, Systems & Structures*, 41:42–65, 2015. 7, 12, 19, 24, 32

[12] A Correa, C Werner, and M Barros. Refactoring to improve the understandability of specifications written in object constraint language. *IET software*, 3(2):69–90, 2009. 5, 32

[13] Alexandre Correa and Cláudia Werner. Applying refactoring techniques to uml/ocl models. In *International Conference on the Unified Modeling Language*, pages 173–187. Springer, 2004. 5, 32

[14] Alexandre Correa and Cláudia Werner. Refactoring object constraint language specifications. *Software & Systems Modeling*, 6(2):113–138, 2007. 5, 32

[15] Alexandre Correa, Cláudia Werner, and Márcio Barros. An empirical study of the impact of ocl smells and refactorings on the understandability of ocl specifications. In *International Conference on Model Driven Engineering Languages and Systems*, pages 76–90. Springer, 2007. 5, 32

[16] Alexandre Correa, Cláudia Werner, and Márcio Barros. Enhancing the understandability of ocl specifications. In *Proc. Brazilian Symp. Software Engineering, Joao Pessoa, Brazil*, pages 22–38, 2007. 5, 32

[17] Andreas Demuth, Roberto E Lopez-Herrejon, and Alexander Egyed. Automatically generating and adapting model constraints to support co-evolution of design models. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*, pages 302–305. IEEE, 2012. 6

[18] Andreas Demuth, Roberto E Lopez-Herrejon, and Alexander Egyed. Supporting the co-evolution of metamodels and constraints through incremental constraint management. In *International Conference on Model Driven Engineering Languages and Systems*, pages 287–303. Springer, 2013. 6

[19] Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. Dealing with the coupled evolution of metamodels and model-to-text transformations. *Alfonso Pierantonio (co-chair) Universita degli Studi dellAquila (Italy) Bernhard Schätz (co-chair) fortiss GmbH (Germany)*, page 22, 2014. 6

[20] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008. 9

[21] CW Gini. Variability and mutability, contribution to the study of statistical distribution and relaitons. *Studi Economico-Giuricici della R*, 1912. 19

[22] Wolfgang Haberl, Markus Herrmannsdoerfer, Stefan Kugele, Michael Tautschnig, and Martin Wechs. Seamless model-driven development put into practice. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 18–32. Springer, 2010. 1

[23] Regina Hebig, Truong Ho Quang, Michel RV Chaudron, Gregorio Robles, and Miguel Angel Fernandez. The quest for open source projects that use UML: mining GitHub. In *MODELS*, pages 173–183. ACM, 2016. 11

[24] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. Automatability of coupled evolution of metamodels and models in practice. *Model Driven Engineering Languages and Systems*, pages 645–659, 2008. 1

[25] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela Damian. The promises and perils of mining GitHub. In *MSR*, pages 92–101, 2014. 17

[26] Djamel Eddine Khelladi, Regina Hebig, Reda Bendraou, Jacques Robin, and Marie-Pierre Gervais. Metamodel and constraints co-evolution: A semi automatic maintenance of ocl constraints. In *International Conference on Software Reuse*, pages 333–349. Springer, 2016. 6

[27] Dimitrios S Kolovos, Nicholas Drivalos Matragkas, Ioannis Korkontzelos, Sophia Ananiadou, and Richard F Paige. Assessing the use of eclipse mde technologies in open-source software projects. In *OSS4MDE@ MoDELS*, pages 20–29, 2015. 10, 11

[28] Angelika Kusel, Juergen Etzlstorfer, Elisabeth Kapsammer, Werner Retschitzegger, Johannes Schoenboeck, Wieland Schwinger, and Manuel Wimmer. Systematic co-evolution of ocl expressions. *11th APCCM*, 27:30, 2015. 6

[29] Mika V. Mäntylä and Casper Lassenius. Subjective evaluation of software evolvability using code smells: An empirical study. *Empirical Software Engineering*, 11(3):395–431, 2006. 6

[30] Slaviša Marković and Thomas Baar. Refactoring ocl annotated uml class diagrams. *Software and Systems Modeling*, 7(1):25–47, 2008. 6

[31] Jeroen Noten, Josh G M Mengerink, and Alexander Serebrenik. A data set of ocl expressions on github. In *Proceedings of the 14th International Conference on Mining Software Repositories*, 2017. 14

[32] Jan Reimann, Claas Wilke, Birgit Demuth, Michael Muck, and Uwe Aßmann. Tool supported ocl refactoring catalogue. In *Proceedings of the 12th Workshop on OCL and Textual Modelling*, pages 7–12. ACM, 2012. 6

[33] Forrest J Shull, Jeffrey C Carver, Sira Vegas, and Natalia Juristo. The role of replications in empirical software engineering. *Empirical Software Engineering*, 13(2):211–218, 2008. 12

[34] Janice Singer, Susan E Sim, and Timothy C Lethbridge. Software engineering data collection for field studies. In *Guide to Advanced Empirical Software Engineering*, pages 9–34. Springer, 2008. 9

[35] J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79–85, May 2014. 1

[36] Edward D. Willink. Aligning OCL with UML. *ECEASST*, 44, 2011. 15

Appendix A

Proportions of OCL expressions in subsets of classes

This table shows the proportions of OCL expressions in subsets of classes. Each row in the table displays the cumulated proportion of invariants defined on each percentile of classes. Example: in the first row, 29% of the OCL expressions are defined in 30% of the classes, 36% of the OCL expressions are defined in 35% of the classes, etc. In meta-models where the number of classes is not big enough to calculate a subset with a given percentile, a dash ('-') is given.

Blue rows indicate that expressions are defined in `.ocl` files instead of in the `.ecore` files itself. Yellow rows indicate industrial meta-models instead of open source.

The table is sorted from the most balanced meta-model to the most unbalanced, using the gini-index.

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
05906e	-	-	0.25	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
5ac57f	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
e24fce	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
fee311	-	-	-	0.33	0.33	0.33	0.33	0.33	0.33	0.67	0.67	0.67	0.67	0.67	0.67	0.67
1edb4e	-	-	0.25	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
e4797e	-	0.17	0.17	0.17	0.33	0.33	0.33	0.33	0.5	0.5	0.5	0.67	0.67	0.67	0.67	0.83
fcb043	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
f31c6c	-	-	0.25	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
bfd457	-	-	0.25	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
ce741d	-	-	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0
a1f7c3	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
386194	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
e56d27	-	-	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0
fb1b5b	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
d3a0b9	-	-	-	0.33	0.33	0.33	0.33	0.33	0.33	0.67	0.67	0.67	0.67	0.67	0.67	0.67
37a22e	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
20a2fe	-	-	0.25	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
6f3a56	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
3dde70	-	-	0.2	0.2	0.2	0.4	0.4	0.4	0.4	0.4	0.6	0.6	0.6	0.8	0.8	0.8
3b5242	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
1c86f7	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
58b4ad	-	0.11	0.11	0.22	0.22	0.33	0.33	0.44	0.44	0.44	0.56	0.56	0.67	0.67	0.78	0.78
942518	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
a73fdb	-	-	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0
1e7e75	-	-	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0
400cbb	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
6acf8f	-	-	-	0.33	0.33	0.33	0.33	0.33	0.33	0.67	0.67	0.67	0.67	0.67	0.67	0.67
432529	-	-	0.2	0.2	0.2	0.4	0.4	0.4	0.4	0.4	0.6	0.6	0.6	0.8	0.8	0.8
f9a3d8	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
6f02bd	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
861c18	-	-	0.25	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
f8eca5	-	-	0.25	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
5ac005	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
434921	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
168a46	0.07	0.07	0.14	0.21	0.29	0.29	0.36	0.43	0.43	0.5	0.57	0.57	0.64	0.71	0.71	0.79
06dfb0	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
55d536	-	-	0.25	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
c370a3	-	-	0.2	0.2	0.2	0.4	0.4	0.4	0.4	0.4	0.6	0.6	0.6	0.8	0.8	0.8
50ee06	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
cd07e6	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
608171	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
790154	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
cc6e8c	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
cfd2a7	-	-	0.2	0.2	0.2	0.4	0.4	0.4	0.4	0.4	0.6	0.6	0.6	0.8	0.8	0.8
e6c675	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
d1ac39	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
8ce256	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
a984d3	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
2d6bf1	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
f802bb	-	-	-	0.33	0.33	0.33	0.33	0.33	0.33	0.67	0.67	0.67	0.67	0.67	0.67	0.67
69bf1d	-	-	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0
9879f9	-	-	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0
af6323	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
671094	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
f25169	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
59a79d	-	-	-	0.33	0.33	0.33	0.33	0.33	0.33	0.67	0.67	0.67	0.67	0.67	0.67	0.67
842d9e	-	-	-	-	-	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0
356749	-	-	-	0.33	0.33	0.33	0.33	0.33	0.33	0.67	0.67	0.67	0.67	0.67	0.67	0.67
f1ba3f	-	-	-	-	-	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.59	0.59	1.0	1.0
8b02fb	-	-	-	-	-	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	1.0	1.0
93b0ec	-	-	-	0.4	0.4	0.4	0.4	0.4	0.4	0.8	0.8	0.8	0.8	0.8	0.8	0.8
24b557	-	-	0.4	0.4	0.4	0.4	0.4	0.6	0.6	0.6	0.6	0.6	0.8	0.8	0.8	0.8
e0d00b	-	-	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0
c5b1b8	-	-	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0
474c2e	-	-	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0
7aa8bc	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75	0.75	0.75
b0ee00	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75	0.75	0.75
f446ee	-	-	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
d041c5	-	-	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0
5ebfee	-	-	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	1.0	1.0	1.0
7626db	-	-	-	0.46	0.46	0.46	0.46	0.46	0.46	0.85	0.85	0.85	0.85	0.85	0.85	0.85
14f5e0	-	0.33	0.33	0.33	0.56	0.56	0.56	0.56	0.67	0.67	0.67	0.78	0.78	0.78	0.78	0.89
c71573	-	0.2	0.2	0.35	0.35	0.5	0.5	0.62	0.62	0.62	0.75	0.75	0.82	0.82	0.9	0.9
65f711	-	-	-	-	-	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	1.0	1.0
75088d	-	-	-	-	-	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	1.0	1.0
beba17	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.88	0.88	0.88	0.88	0.88	0.88	0.88
59a1b2	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	0.88	0.88	0.88	0.88	0.88	0.88	0.88
9c746c	-	0.27	0.27	0.27	0.55	0.55	0.55	0.55	0.73	0.73	0.73	0.82	0.82	0.82	0.82	0.91
2adf93	-	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.88	0.88	0.88	0.88	1.0
57348f	-	-	-	0.62	0.62	0.62	0.62	0.62	0.62	0.85	0.85	0.85	0.85	0.85	0.85	0.85
e72bfe	-	-	-	0.53	0.53	0.53	0.53	0.53	0.53	0.95	0.95	0.95	0.95	0.95	0.95	0.95
e1eeac	-	-	-	0.53	0.53	0.53	0.53	0.53	0.53	0.95	0.95	0.95	0.95	0.95	0.95	0.95
e29684	-	-	-	0.53	0.53	0.53	0.53	0.53	0.53	0.95	0.95	0.95	0.95	0.95	0.95	0.95
420279	-	-	0.38	0.38	0.38	0.66	0.66	0.66	0.66	0.66	0.83	0.83	0.83	0.97	0.97	0.97
d79d72	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0
fa13ca	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0
547ea1	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c61a7e	-	-	-	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0
43a2ad	-	0.37	0.37	0.37	0.57	0.57	0.57	0.57	0.73	0.73	0.73	0.87	0.87	0.87	0.87	0.97
9e3b54	0.19	0.19	0.3	0.41	0.41	0.52	0.63	0.63	0.7	0.7	0.78	0.85	0.85	0.93	0.96	0.96
3882d2	0.26	0.26	0.41	0.41	0.54	0.61	0.61	0.68	0.68	0.74	0.79	0.79	0.84	0.84	0.89	0.93
8b9f86	0.16	0.27	0.36	0.44	0.51	0.56	0.61	0.67	0.72	0.76	0.79	0.82	0.86	0.89	0.91	0.93
cc9705	0.29	0.29	0.43	0.43	0.55	0.62	0.62	0.69	0.69	0.75	0.8	0.8	0.85	0.85	0.89	0.93
a5a3c0	0.29	0.29	0.43	0.43	0.55	0.62	0.62	0.69	0.69	0.75	0.8	0.8	0.85	0.85	0.89	0.93
303602	0.29	0.29	0.43	0.43	0.55	0.62	0.62	0.69	0.69	0.75	0.8	0.8	0.85	0.85	0.89	0.93
0d2d6f	-	0.33	0.33	0.33	0.5	0.5	0.5	0.67	0.67	0.83	0.83	0.83	1.0	1.0	1.0	1.0
3b43ce	-	-	-	0.57	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0
891aed	-	0.42	0.42	0.42	0.55	0.55	0.55	0.67	0.67	0.79	0.79	0.79	0.91	0.91	0.91	1.0
60cdbe	-	-	-	0.57	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0
63bc8e	0.12	0.2	0.33	0.4	0.47	0.53	0.58	0.68	0.73	0.78	0.82	0.85	0.92	0.95	0.97	0.98
a31965	-	0.5	0.5	0.5	0.64	0.64	0.64	0.64	0.75	0.75	0.75	0.86	0.86	0.86	0.86	0.93
921e47	-	-	0.43	0.43	0.43	0.43	0.43	0.86	0.86	0.86	0.86	0.86	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
78564a	-	0.27	0.27	0.45	0.45	0.45	0.64	0.64	0.82	0.82	0.82	0.91	0.91	1.0	1.0	1.0
6abbef	-	-	0.43	0.43	0.43	0.71	0.71	0.71	0.71	0.71	0.86	0.86	0.86	1.0	1.0	1.0
08bbb6	-	-	-	0.6	0.6	0.6	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b652e6	-	-	-	0.6	0.6	0.6	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d7a375	-	-	0.43	0.43	0.43	0.71	0.71	0.71	0.71	0.71	0.86	0.86	0.86	1.0	1.0	1.0
6e7841	-	-	0.55	0.55	0.55	0.73	0.73	0.73	0.73	0.73	0.82	0.82	0.82	0.91	0.91	0.91
755f4f	-	-	0.43	0.43	0.43	0.71	0.71	0.71	0.71	0.71	0.86	0.86	0.86	1.0	1.0	1.0
5e32f1	-	-	0.33	0.33	0.33	0.67	0.67	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0
c2a6c3	-	0.46	0.46	0.46	0.64	0.64	0.64	0.64	0.79	0.79	0.79	0.89	0.89	0.89	0.89	0.96
d7d75d	-	0.33	0.33	0.56	0.56	0.56	0.67	0.67	0.78	0.78	0.78	0.89	0.89	1.0	1.0	1.0
5ffd15	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7fca14	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0
89aa2e	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3032f8	-	-	-	0.67	0.67	0.67	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0505bf	-	-	0.6	0.6	0.6	0.6	0.6	0.8	0.8	0.8	0.8	0.8	1.0	1.0	1.0	1.0
429392	-	0.39	0.39	0.39	0.73	0.73	0.73	0.73	0.83	0.83	0.83	0.93	0.93	0.93	0.93	0.98
b5a136	0.17	0.17	0.33	0.33	0.5	0.5	0.67	0.67	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3aaa04	-	0.31	0.31	0.54	0.54	0.69	0.69	0.77	0.77	0.77	0.85	0.85	0.92	0.92	1.0	1.0
9d0a52	0.24	0.41	0.49	0.59	0.61	0.67	0.7	0.76	0.77	0.79	0.81	0.83	0.86	0.87	0.9	0.91
b6d06f	-	-	-	0.71	0.71	0.71	0.71	0.71	0.71	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ea0df1	-	-	0.4	0.4	0.4	0.8	0.8	0.8	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0
33277a	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
90284a	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
985f79	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
985c0a	-	0.2	0.4	0.4	0.4	0.6	0.8	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d97e1a	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3324a9	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a41bb6	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
fc9b1a	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8870b8	-	-	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0
d573a0	-	0.4	0.4	0.4	0.7	0.7	0.7	0.7	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0
679482	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
20b359	-	-	-	0.75	0.75	0.75	0.75	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a8586f	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
ed8379	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
06f36a	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9e37c5	-	0.43	0.43	0.43	0.71	0.71	0.71	0.71	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0
1ff647	-	-	-	0.75	0.75	0.75	0.75	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7efb07	-	0.33	0.33	0.33	0.67	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
f0a4a1	-	0.25	0.25	0.5	0.5	0.5	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
237de4	-	-	0.67	0.67	0.67	0.67	0.67	0.83	0.83	0.83	0.83	0.83	1.0	1.0	1.0	1.0
bb2ac7	-	0.43	0.43	0.43	0.71	0.71	0.71	0.71	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0
c45fb8	-	0.43	0.43	0.43	0.71	0.71	0.71	0.71	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0
284e98	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9ab1a3	-	-	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c3479b	-	0.43	0.43	0.43	0.71	0.71	0.71	0.71	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0
c3187a	0.22	0.33	0.33	0.44	0.56	0.56	0.67	0.78	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8b1663	0.22	0.33	0.33	0.44	0.56	0.56	0.67	0.78	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0
364517	0.29	0.38	0.38	0.48	0.57	0.57	0.67	0.76	0.86	0.95	0.95	1.0	1.0	1.0	1.0	1.0
2be475	-	0.38	0.38	0.62	0.62	0.62	0.75	0.75	0.88	0.88	0.88	1.0	1.0	1.0	1.0	1.0
3e432d	-	-	0.5	0.5	0.5	0.83	0.83	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0
ee5055	-	-	0.5	0.5	0.5	0.83	0.83	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0
7edf47	-	-	0.5	0.5	0.5	0.83	0.83	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0
8f21be	-	-	-	0.8	0.8	0.8	0.8	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4819d1	-	-	0.5	0.5	0.5	0.83	0.83	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0
0cabc8	-	-	-	0.8	0.8	0.8	0.8	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4627a1	-	-	0.5	0.5	0.5	0.83	0.83	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0
191726	-	-	0.5	0.5	0.5	0.83	0.83	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0
c45381	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b1e7fa	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5cbd9f	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5da2d0	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
f30347	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8677a8	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5a6382	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
bb7278	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4bef9e	-	-	0.57	0.57	0.57	0.57	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
bb8926	-	0.44	0.44	0.44	0.83	0.83	0.83	0.83	0.89	0.89	0.89	0.94	0.94	0.94	0.94	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
fd8ad9	-	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.88	0.88	0.88	1.0	1.0	1.0	1.0	1.0
7df832	-	0.5	0.5	0.5	0.75	0.75	0.75	0.75	0.88	0.88	0.88	1.0	1.0	1.0	1.0	1.0
86e5b6	0.18	0.33	0.42	0.52	0.64	0.7	0.76	0.82	0.85	0.88	0.91	0.94	0.97	1.0	1.0	1.0
a4c6af	-	-	-	0.82	0.82	0.82	0.82	0.82	0.82	1.0	1.0	1.0	1.0	1.0	1.0	1.0
95b292	-	-	-	0.82	0.82	0.82	0.82	0.82	0.82	1.0	1.0	1.0	1.0	1.0	1.0	1.0
16f312	-	-	-	0.82	0.82	0.82	0.82	0.82	0.82	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e51001	-	-	-	0.82	0.82	0.82	0.82	0.82	0.82	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e81ff1	-	-	0.6	0.6	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1ba33a	0.3	0.3	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0
74ab7d	-	-	0.6	0.6	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c5eeaf	-	0.5	0.5	0.5	0.8	0.8	0.8	0.8	0.9	0.9	0.9	0.95	0.95	0.95	0.95	1.0
57e764	0.3	0.3	0.5	0.5	0.7	0.7	0.8	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0
375974	-	-	0.67	0.67	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
795b42	-	-	0.75	0.75	0.75	0.75	0.75	0.92	0.92	0.92	0.92	0.92	1.0	1.0	1.0	1.0
9675ab	-	-	0.67	0.67	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
64c07f	-	0.5	0.5	0.5	0.75	0.75	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d84a57	-	-	0.75	0.75	0.75	0.75	0.75	0.92	0.92	0.92	0.92	0.92	1.0	1.0	1.0	1.0
740ce2	-	0.29	0.57	0.57	0.57	0.71	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c3a6db	0.33	0.33	0.56	0.56	0.67	0.67	0.78	0.78	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3daedc	0.33	0.33	0.56	0.56	0.67	0.67	0.78	0.78	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5502bb	-	-	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2952bb	-	-	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6eb718	0.33	0.33	0.5	0.5	0.67	0.67	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
49b589	0.14	0.29	0.43	0.57	0.57	0.71	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8fd722	0.14	0.29	0.43	0.57	0.57	0.71	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
607a6d	0.25	0.4	0.53	0.62	0.69	0.75	0.81	0.85	0.89	0.93	0.95	0.97	0.99	1.0	1.0	1.0
2ae7a0	0.25	0.4	0.53	0.62	0.69	0.75	0.81	0.85	0.89	0.93	0.95	0.97	0.99	1.0	1.0	1.0
98a6b5	0.25	0.4	0.53	0.62	0.69	0.75	0.81	0.85	0.89	0.93	0.95	0.97	0.99	1.0	1.0	1.0
68cf14	0.25	0.4	0.53	0.62	0.69	0.75	0.81	0.85	0.89	0.93	0.95	0.97	0.99	1.0	1.0	1.0
b4a03e	-	0.45	0.45	0.45	0.91	0.91	0.91	0.91	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3895ce	-	0.4	0.4	0.6	0.6	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e86319	-	-	0.75	0.75	0.75	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6c4873	-	-	0.75	0.75	0.75	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0e32ae	-	0.5	0.5	0.5	0.88	0.88	0.88	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
8d221b	0.31	0.46	0.46	0.57	0.69	0.8	0.86	0.86	0.91	0.94	0.97	1.0	1.0	1.0	1.0	1.0
ebebde	0.31	0.46	0.46	0.57	0.69	0.8	0.86	0.86	0.91	0.94	0.97	1.0	1.0	1.0	1.0	1.0
3e5395	0.22	0.41	0.41	0.56	0.72	0.81	0.91	0.91	0.94	0.97	1.0	1.0	1.0	1.0	1.0	1.0
7a91b0	0.22	0.41	0.41	0.56	0.72	0.81	0.91	0.91	0.94	0.97	1.0	1.0	1.0	1.0	1.0	1.0
991c6f	0.21	0.39	0.39	0.58	0.73	0.82	0.91	0.91	0.94	0.97	1.0	1.0	1.0	1.0	1.0	1.0
6b9a94	-	-	0.77	0.77	0.77	0.77	0.77	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8bcefc	0.68	0.68	0.72	0.72	0.76	0.76	0.8	0.8	0.84	0.88	0.88	0.92	0.92	0.96	0.96	1.0
59ec7a	0.17	0.43	0.52	0.65	0.7	0.78	0.83	0.91	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ae912f	0.17	0.43	0.52	0.65	0.7	0.78	0.83	0.91	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0
86e714	0.18	0.36	0.45	0.55	0.73	0.82	0.91	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c164f1	0.19	0.44	0.56	0.62	0.69	0.78	0.88	0.94	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e196d2	-	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
f53810	-	0.67	0.67	0.67	0.83	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c5ab8d	-	0.5	0.5	0.5	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ef989d	-	-	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
42adb1	-	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0f1d7a	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7f90e3	-	-	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
447d52	-	-	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
90812d	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8cad76	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5e27d8	-	0.71	0.75	0.75	0.75	0.79	0.83	0.83	0.83	0.88	0.92	0.92	0.92	0.96	1.0	1.0
6d946e	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3ffefa	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3f768e	-	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6e0cea	0.2	0.4	0.4	0.6	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b2fedc	-	0.4	0.4	0.8	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6ae009	-	-	0.86	0.86	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
84cf46	-	-	0.86	0.86	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e464f7	-	-	0.86	0.86	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0391df	0.21	0.43	0.61	0.75	0.75	0.82	0.89	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d0d4b9	0.35	0.54	0.66	0.71	0.75	0.81	0.85	0.87	0.89	0.93	0.96	0.98	1.0	1.0	1.0	1.0
6b107c	0.35	0.54	0.66	0.71	0.75	0.81	0.85	0.87	0.89	0.93	0.96	0.98	1.0	1.0	1.0	1.0
79d072	0.25	0.25	0.5	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
9c0b82	0.25	0.25	0.5	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
bee195	-	0.57	0.57	0.57	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
63f476	0.28	0.45	0.58	0.69	0.78	0.83	0.88	0.94	0.99	1.0	1.0	1.0	1.0	1.0	1.0	1.0
fbe9b2	-	-	0.75	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
579a27	0.29	0.29	0.57	0.57	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
315c84	0.18	0.45	0.55	0.64	0.73	0.91	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7efcc6	0.18	0.35	0.53	0.71	0.82	0.94	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c7bca7	0.18	0.35	0.53	0.71	0.82	0.94	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
eca9a7	0.29	0.29	0.57	0.71	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
f27d2d	0.25	0.45	0.6	0.72	0.78	0.85	0.92	0.97	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
353a5b	0.33	0.5	0.6	0.67	0.75	0.81	0.88	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ce703b	-	0.5	0.5	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c93579	0.33	0.5	0.5	0.67	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2f6f47	0.33	0.5	0.5	0.67	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0e04d1	-	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
dfa805	-	0.6	0.6	0.6	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
93357f	-	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5a1dcd	-	0.62	0.62	0.62	0.88	0.88	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a8e2df	-	0.5	0.5	0.88	0.88	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3c397d	0.19	0.49	0.59	0.7	0.86	0.89	0.92	0.97	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3b3f75	-	0.42	0.42	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8ba4e4	-	0.42	0.42	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d4ea1c	-	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3ad601	-	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
512f7c	-	0.47	0.73	0.73	0.73	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
587d34	0.5	0.5	0.67	0.67	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
392e2a	-	0.43	0.43	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
690556	0.33	0.44	0.63	0.78	0.81	0.89	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d870b8	0.33	0.56	0.67	0.67	0.78	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c87223	-	0.39	0.78	0.78	0.78	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
08d7e2	0.29	0.57	0.64	0.71	0.79	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
274b0c	0.29	0.43	0.57	0.71	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6c7417	0.29	0.43	0.57	0.71	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
089c96	0.27	0.45	0.64	0.73	0.82	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
d37f10	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c7d32b	-	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
220278	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
73b3b5	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
134bff	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
fb8c68	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7ba57d	-	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
44e5dc	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e64d58	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4af2c4	0.25	0.5	0.5	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1c43ef	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ccb36e	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a5c2a0	-	0.75	0.75	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e89224	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
47a964	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b983ec	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1f057b	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e5b574	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
241dc8	-	0.43	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
38e73c	-	0.43	0.86	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9a5fa9	0.25	0.54	0.64	0.82	0.86	0.93	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0d47b1	0.25	0.54	0.64	0.82	0.86	0.93	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
041b4d	-	0.67	0.67	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ac030d	0.29	0.57	0.71	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b2b4a6	0.46	0.69	0.69	0.77	0.85	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b48cd3	0.33	0.56	0.7	0.81	0.89	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6f1669	0.37	0.59	0.71	0.78	0.86	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
aa5eaf	-	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0e94da	-	0.5	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7814f6	0.38	0.56	0.75	0.88	0.88	0.94	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d12cf1	0.37	0.54	0.68	0.79	0.88	0.95	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
f15507	0.44	0.56	0.68	0.78	0.86	0.92	0.98	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2d85f8	0.5	0.62	0.62	0.75	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c8abcb	0.27	0.58	0.69	0.85	0.88	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
7b0b1f	0.27	0.58	0.69	0.85	0.88	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
70d6d7	0.36	0.56	0.71	0.81	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3c5f87	0.36	0.56	0.71	0.81	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2ab2cc	0.25	0.5	0.67	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9fcdd7	0.46	0.46	0.77	0.77	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
39f8fb	0.36	0.57	0.71	0.79	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
524baa	0.43	0.61	0.7	0.83	0.87	0.91	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8334c7	0.4	0.55	0.7	0.81	0.89	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2c1716	0.4	0.55	0.7	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ebae40	0.4	0.55	0.7	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8af39d	0.3	0.5	0.8	0.85	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7e4ab1	0.25	0.58	0.75	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
22de60	0.43	0.57	0.71	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
cc0d5f	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2e1a0e	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b35e99	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
df6be2	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
af1196	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
17e0f9	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7069f2	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
13a1ac	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a1e3a6	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
005629	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
691b9f	-	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d190eb	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
97164d	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
bc6b69	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
465a53	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3cb769	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
04d590	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2ff934	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
eeb8ea	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4848f5	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ea803b	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
b5758c	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6e23d0	-	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d5d718	-	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
45a44b	0.52	0.71	0.79	0.83	0.85	0.9	0.92	0.94	0.96	0.98	1.0	1.0	1.0	1.0	1.0	1.0
2b39d9	0.56	0.56	0.78	0.78	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
aa80e9	0.33	0.67	0.78	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
be136b	0.32	0.53	0.74	0.95	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d53199	-	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3e69e7	-	0.83	0.83	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
011e7d	0.38	0.5	0.75	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5781ee	0.56	0.67	0.78	0.78	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
f1d23c	0.4	0.7	0.8	0.9	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
84b2c0	0.4	0.7	0.8	0.9	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0b316d	0.4	0.7	0.8	0.9	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a30fc6	0.38	0.54	0.77	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
13c0b2	0.33	0.5	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6c2401	-	0.6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ee7d1f	0.38	0.62	0.75	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2dc504	-	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
84ce9d	0.5	0.64	0.79	0.86	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b88569	0.46	0.62	0.85	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
39fc80	0.55	0.68	0.81	0.87	0.9	0.97	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b1ec70	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7ad24e	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a51f2e	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
fb92a4	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
daa764	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
df01b4	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
fec56c	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
545bf6	0.4	0.6	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
573858	0.25	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d88361	0.49	0.62	0.82	0.88	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d34d20	0.49	0.62	0.82	0.88	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
bae200	0.49	0.68	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
da57f8	-	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b80926	-	0.86	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
521fbd	0.43	0.71	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6f225e	0.33	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6e4108	0.4	0.7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2dbdc4	0.33	0.56	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9ea811	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a9a7bd	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0f7bb9	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c8ba96	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
95e892	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a4a9bc	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
57dff2	0.55	0.73	0.82	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4ebbbd	0.55	0.73	0.82	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a9414e	0.5	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10d71a	0.42	0.68	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
678c4b	0.4	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2d093d	0.38	0.71	0.95	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
602b41	0.5	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
af7875	0.57	0.78	0.85	0.93	0.97	0.99	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b795a9	0.57	0.78	0.85	0.93	0.97	0.99	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2e64f6	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c59c6b	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e0e671	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
472a14	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
bdb576	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
96596f	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1e5b60	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
206dff	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
173656	0.59	0.76	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a87654	0.5	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1b5657	0.46	0.77	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a515f9	0.6	0.78	0.88	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2d94c0	0.6	0.78	0.88	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
7fc328	0.5	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10599c	0.5	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
173f76	0.6	0.87	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b9e784	0.58	0.92	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
24224b	0.61	0.87	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
21fb27	0.58	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6a4536	0.71	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ebfc79	0.57	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3fb904	0.62	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ca57fa	0.74	0.84	0.94	0.98	0.99	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
998462	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8ddb8a	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4cc7b0	0.58	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d8489c	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c22d9a	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3c7cf3	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1b8f27	-	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b92c08	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9da57a	0.62	0.85	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c6324f	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
02266c	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
61e53c	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
94e1d7	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9f1ff7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3253fd	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ee21d0	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9d1020	0.7	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
423fd9	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
cc7a88	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d470a9	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c22f58	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d1dc7a	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
55d51d	0.67	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
801043	0.69	0.87	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
900d46	0.69	0.87	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
dc3efc	0.71	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0cb205	0.72	0.87	0.97	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a9c43d	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
406983	0.7	0.89	0.97	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
534432	0.64	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e364ea	0.64	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
83b3b4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1c0aeb	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d0a73d	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d44a4b	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
769e8d	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d71657	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
06a7d4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8b7245	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d84669	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c0f732	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5a0f20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d14f48	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
84acd3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
535adb	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ed800f	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5bb26e	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ec76fa	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
01669b	0.81	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
925bbe	0.81	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6d28ec	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7ac881	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
254569	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
be3aba	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4598d9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
cf7d39	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
6d710f	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ee75d6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%
fd2839	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
689de7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
5fcfb4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c9658e	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
57b89b	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
860ef9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
919220	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
457f23	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e71f4d	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0d4079	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
c710b9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
faf918	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
62bc6a	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d8743b	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0d9510	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
33d522	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2d71da	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60fb42	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
055809	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
e19e35	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
bbe0b2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3c7826	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
520341	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b6b024	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
b7f46a	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a0df90	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
08d65f	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
64721d	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
a6cb59	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
bf1c8c	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
d2d3b6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
36999c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8b4467	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
e5cb7a	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Meta-model	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%

Table A.1: Proportion of OCL expressions in subsets of classes.

Appendix B

Complexity of OCL Expressions

B.1 Complexity of OCL Expressions in Meta-Models

These figures display the distribution of complexities of OCL expressions in each meta-model. The suffix of each labels indicates whether the expressions are defined in `.ocl` or in `.ecore`; the prefix *(I)* indicates the industrial meta-models.

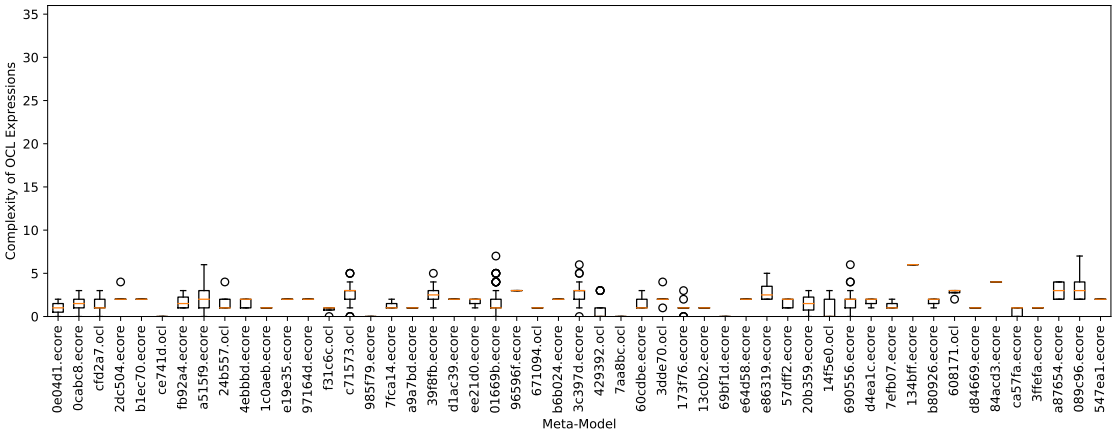

Figure B.1: Complexities of OCL expressions in meta-models (part 1)

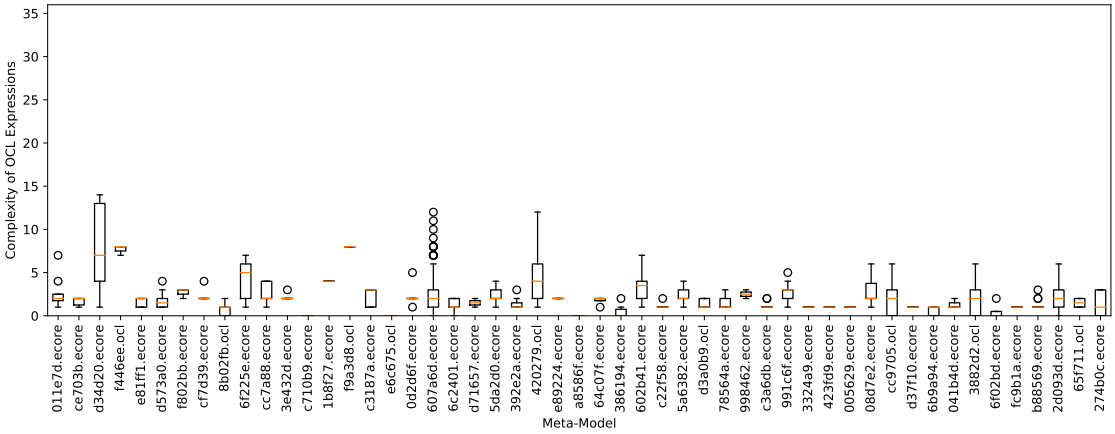

Figure B.2: Complexities of OCL expressions in meta-models (part 2)

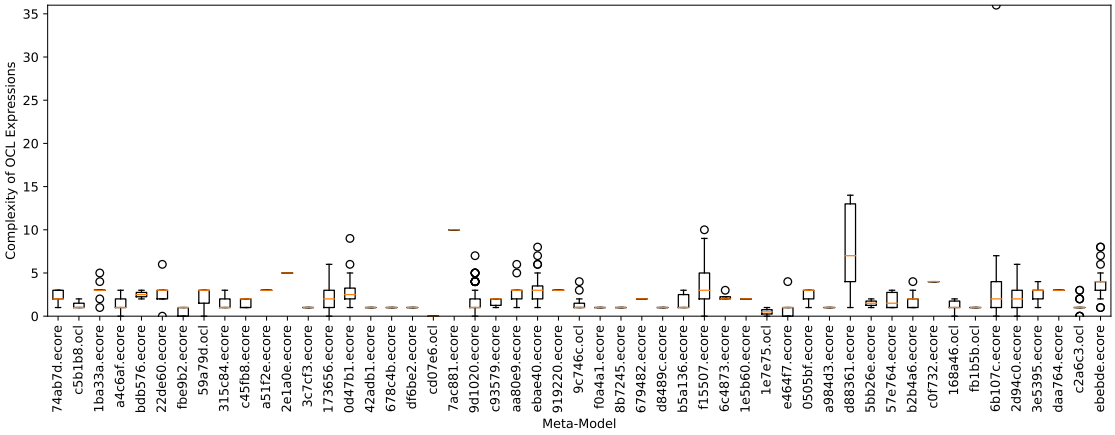

Figure B.3: Complexities of OCL expressions in meta-models (part 3)

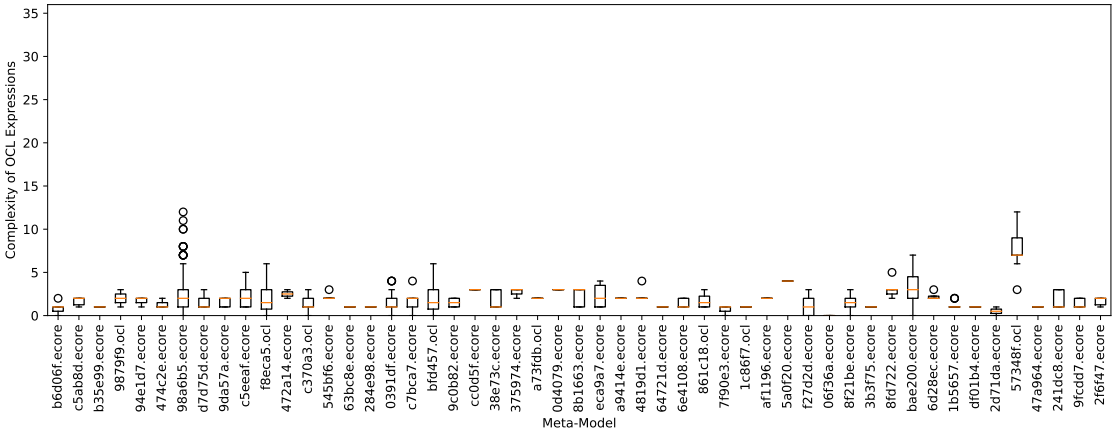

Figure B.4: Complexities of OCL expressions in meta-models (part 4)

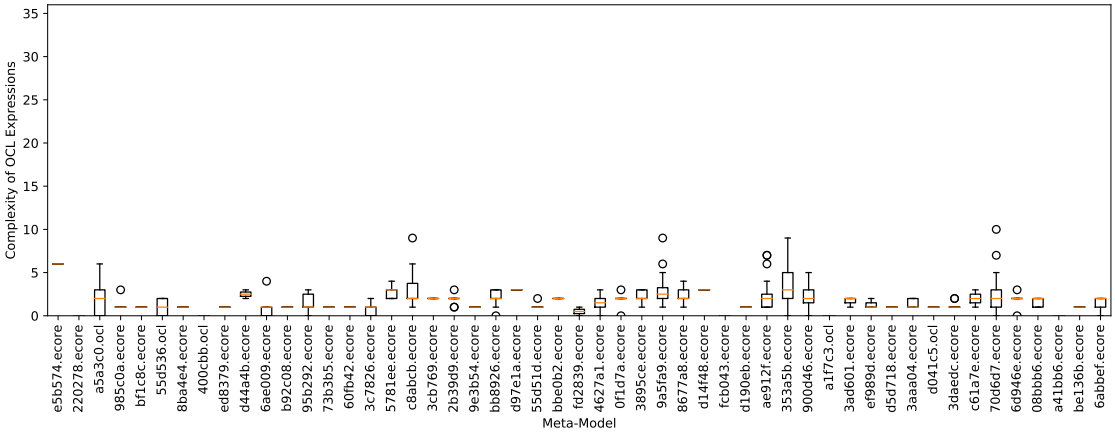

Figure B.5: Complexities of OCL expressions in meta-models (part 5)

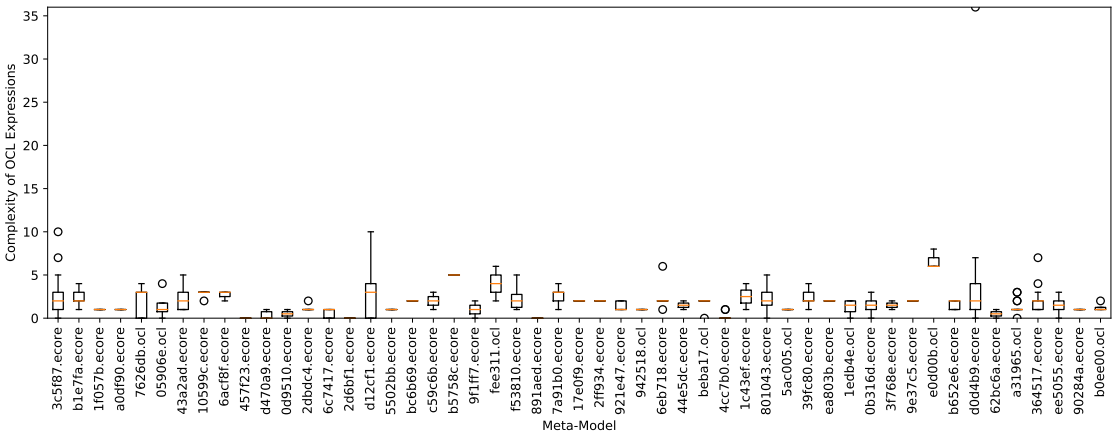

Figure B.6: Complexities of OCL expressions in meta-models (part 6)

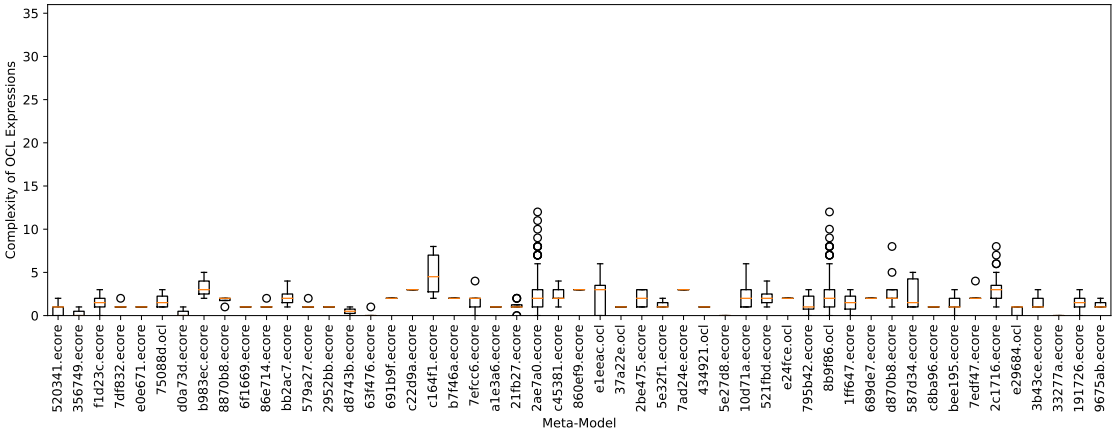

Figure B.7: Complexities of OCL expressions in meta-models (part 7)

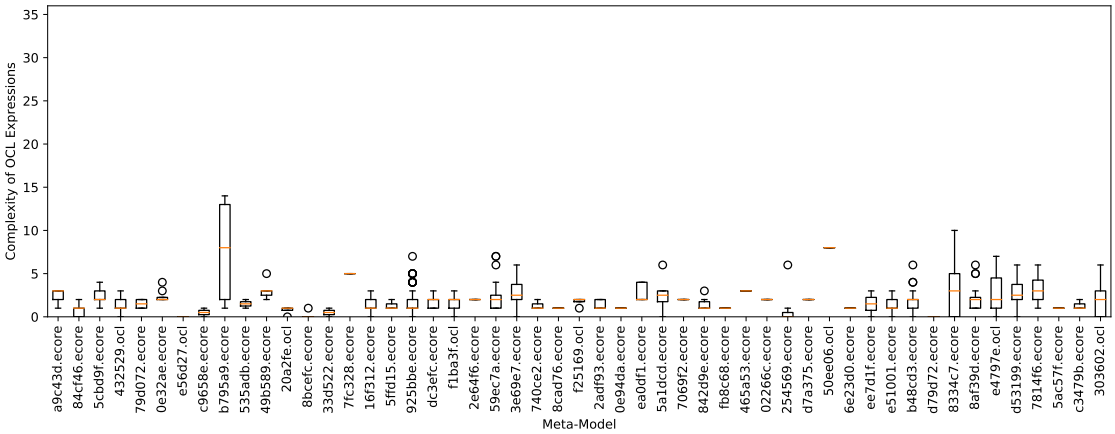

Figure B.8: Complexities of OCL expressions in meta-models (part 8)

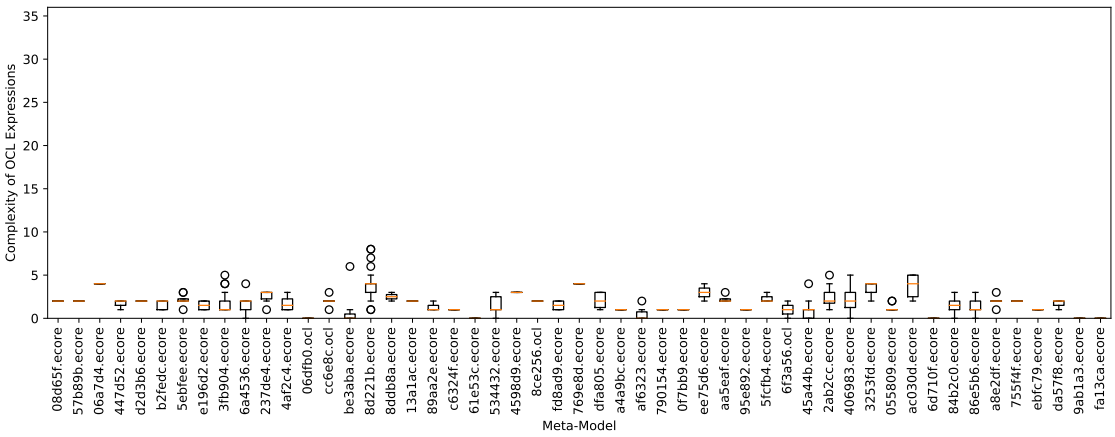

Figure B.9: Complexities of OCL expressions in meta-models (part 9)

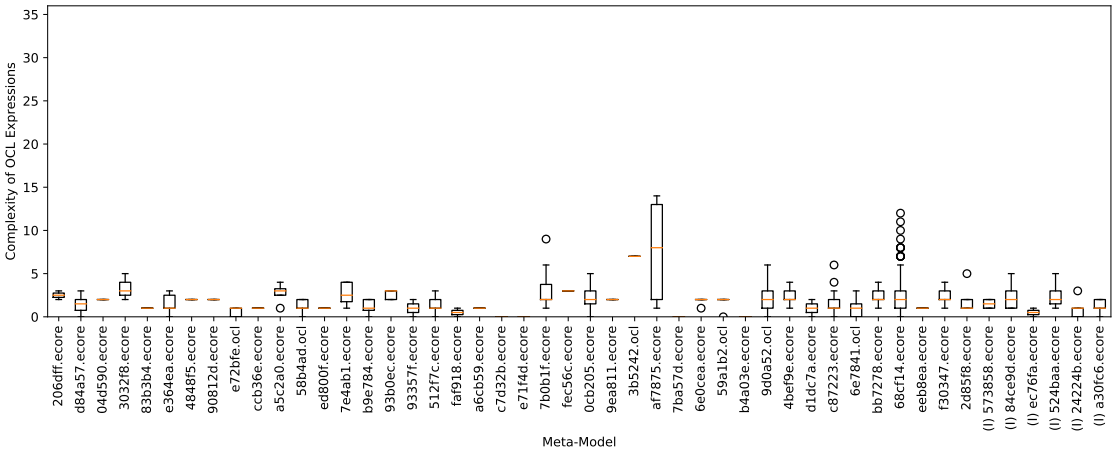

Figure B.10: Complexities of OCL expressions in meta-models (part 10)