On using Sarkar Metrics to Evaluate the Modularity of Metamodels

Georg Hinkel[1] and Misha Strittmatter[2]
[1]*Software Engineering Division, FZI Research Center of Information Technologies, Karlsruhe, Germany*
[2]*Software Design & Quality Group, Karlsruhe Institute of Technology, Karlsruhe, Germany*

Keywords: Metamodel, Modularity, Metric.

Abstract: As model-driven engineering (MDE) gets applied for the development of larger systems, the quality assurance of model-driven artifacts gets more important. Here, metamodels are particularly important as many other artifacts depend on them. Existing approaches to measure the modularity of metamodels have not been validated for metamodels thoroughly. In this paper, we evaluate the usage of the metrics suggested by Sarkar et al. to automatically measure the modularity of metamodels with the goal of automated quality improvements. For this, we analyze the data from a previous controlled experiment on the perception of metamodel quality with 24 participants, including both students and academic professionals. From the results, we were able to statistically disprove even a slight correlation with perceived metamodel quality.

1 INTRODUCTION

Metamodels are a central artifact of model-driven engineering as many other artifacts depend on them. If a metamodel contains design flaws, then presumably all other artifacts have to compensate for them. It is therefore very important to detect such design flaws as early as possible.

In object-oriented programming, several approaches exist to detect design flaws and can be categorized into (anti-)patterns and metrics. Antipatterns are commonly used, for example as code smells. If an antipattern can be found in the code, there is a high defect probability and the smell may be avoidable through better design. On the other hand, metrics have been established to monitor the complexity of object-oriented design not captured by smells such as the depth of inheritance or lines of code.

In prior work (Hinkel et al., 2016b), we have identified modularity as a quality attribute of metamodels that has a significant influence on the perception of metamodel quality alongside correctness and completeness. While the latter are hard to measure, metrics exist in object-oriented design to measure modularity.

Metamodels essentially describe type systems just as object-oriented designs do using UML models. In fact, the differences between usual class diagrams and formal metamodels lies mostly in the degree of formalization and how the resulting models are used. Whereas UML models of object-oriented design are often used only for documentation or to generate code skeletons, metamodels are usually used to generate a multitude of artifacts such as serialization and editors. But like class diagrams, metamodels can be structured in packages, which makes it appealing to apply the same metrics to measure metamodel modularity as also used for class diagrams.

Metrics make it viable to automate fixing design flaws through design space exploration of possible semantics-preserving operations. Such an optimization system repeatedly alters the metamodel randomly in several places and outputs the version that scores best according to the metrics (or outputs all versions along the Pareto-front if multiple metrics are used). For modularity, this is practical as e.g. the module structure can be easily altered without changing the metamodels' semantics. For such an auto-tuner to produce meaningful results, the underlying metrics must have a clear and validated correlation to modularity. Otherwise, it is not clear that the outcome of the auto-tuner is better than the previous one. However, such a validation of correlations of metrics to quality attributes is hard as one has to consider consequences of metamodel design on all dependent artifacts. To the best of our knowledge, this rarely has been done before.

In the Neurorobotics-platform developed in the scope of the *Human Brain Project* (HBP), these dependent artifacts include not only editors, but also

Hinkel G. and Strittmatter M.
On using Sarkar Metrics to Evaluate the Modularity of Metamodels.
DOI: 10.5220/0006105502530260
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 253-260
ISBN: 978-989-758-210-3

an entire simulation platform where the connection between robots and neural networks is described in models (Hinkel et al., 2015; Hinkel et al., 2016a). As the HBP is designed for a total duration of ten years, it is likely that the metamodel will degrade unless extra effort is spent for its refactorings (Lehman, 1974; Lehman et al., 1997). For such refactorings, we aim to measure their success and potentially automate them.

Given the similarity of metamodels to object-oriented design, we think that metrics for object-oriented design are good starting points when trying to measure the quality of metamodels. In particular, the set of metrics developed by Sarkar (Sarkar et al., 2008) have been established to measure the quality of modularization. All of the metrics are scaled to have values between 0 and 1, where 0 always is the worst and 1 the best value. This handyness has given these metrics some popularity.

In this paper, we have picked the metrics by Sarkar and analyze whether they can be applied to metamodels. We analyze how the values for these metrics correlate with perceived metamodel quality and/or metamodel modularity.

The remainder of this paper is structured as follows: Section 2 analyzes the Sarkar metrics and presents adoptions for metamodels. Section 3 presents the setup of the empirical experiment that we use to validate these metrics to measure metamodel modularity. Section 4 presents the results from this experiment. Section 5 discusses threats to validity of the results. Finally, Section 6 discusses related work before Section 7 concludes the paper.

2 SARKAR METRICS TO MEASURE METAMODEL QUALITY

In this section, we analyze which Sarkar metrics can be used to measure metamodel modularity, but only from an applicability point of view. That is, many of the metrics need adjustments to be applied to metamodels or cannot be applied at all. We base this discussion on the Essential Meta Object Facility (EMOF) standard, especially on its implementation in Ecore, to describe the structure of metamodels.

An adaptation is necessary because the Sarkar metrics require an implementation of the object-oriented system under observation. They operate on an executable method specification that allows them to retrace how classes in the object-oriented design are used. Furthermore, they rely on interface concepts such as APIs that exist in many object-oriented programming languages but are omitted in many metametamodels. Our goal is to provide metrics to support the metamodel design process where no implementation in the form of transformations, analyses or other artifacts are available.

In the remainder of this section, we discuss the inheritance-based coupling metric *IC* in Section 2.1, the association-based coupling metrics *AC* in different variants in Section 2.2 and the size uniformity metrics *MU* and *CU* in Section 2.3. In Section 2.4, we discuss the (in-)applicability of the other metrics and present a new proposed metric to measure the degree of modularization.

2.1 Inheritance-based Coupling

One of the metrics by Sarkar et al. is the Inheritance-Based Intermodule Coupling *IC*. It measures inheritance-based coupling between packages based on three different rationales, represented by sub-metrics $IC_1 - IC_3$. The first metric IC_1 measures for a package p the fraction of other packages p' who are coupled to p by including a class that inherits from a class in p. Conversely, IC_2 measures the fraction of classes outside the package p that inherit from a class in p. The third component IC_3 measures the fraction of classes of p that have base classes in another package. The components are combined by simply taking the minimum value for each of the components for each package. A formal definition is given in Figure 1.

There, $\mathcal{C}$ defines the set of all classes, $\mathcal{P}$ defines the set of all packages and the predicates $C, Module, Par$ and $Chlds$ depict the classes of a package, the package of a class, the parent classes and the derived classes of a class. IC_1 and IC_2 are set to 1 if the metamodel only consists of a single package.

While all of the components for *IC* can be evaluated for metamodels as well, especially the component IC_3 yields a large problem. Many metamodels use a single base class to extract common functionality. An example for this is the support for stereotypes that can be implemented using a common base class `EStereotypeableObject` (Kramer et al., 2012), separated in its own module. However, using such an approach means immediately that the component IC_3 constantly equals zero. Therefore, we excluded the component IC_3 from the composite metric *IC*.

$$IC(p) = \min\{IC_1(p), IC_2(p)\}.$$

As in the proposal of Sarkar et al., inheritance-based coupling for an entire metamodel simply is the average inheritance-based intermodule coupling of its packages.

$$IC_1(p) = 1 - \frac{|\{p' \in \mathcal{P} | \exists_{d \in C(p')} \exists_{c \in C(p)} (c \in Chlds(d) \wedge p \neq p')\}|}{|\mathcal{P}| - 1}$$
$$IC_2(p) = 1 - \frac{|\{d \in C | \exists_{c \in C(p)} (c \in Chlds(d) \wedge p \neq Module(d))\}|}{|C| - |C(p)|}$$
$$IC_3(p) = 1 - \frac{|\{c \in C | \exists_{d \in Par(c)} (Module(d) \neq p)\}|}{|C(p)|}$$
$$IC(p) = \min\{IC_1(p), IC_2(p), IC_3(p)\}$$

Figure 1: Formal definition of inheritance-based coupling.

2.2 Association-based Coupling

Similar to the inheritance-based coupling, Sarkar also defines the association-based intermodule coupling *AC* based on the usage of classes from a different module in the public API of a class. Translating the public API to the set of features of a class, this can be applied to metamodels as well where a usage is defined as including a reference to the used class. Here, metamodels offer to distinguish between multiple types of associations. Unlike usual object-oriented platforms, metamodels draw a big difference between regular associations and composite references, in Ecore called containments. Thus, we compute three association-based coupling indices, one for associations only, one for composite references and lastly one for both of them together.

Like *IC*, the composite metric *AC* as defined by Sarkar et al. consists of three components AC_1, AC_2 and AC_3. Their definition is equivalent to the definition of IC_1, IC_2 and IC_3 except that they are using the predicate *Uses* instead of *Chlds* and *Par* that yields the set of used classes for a given class.

We adjust the metrics by altering the semantics of the usage predicate. The closest adoption of *AC* is to use the types of references as usages, but we also obtain the metric AC^{cmp} by limiting the usage to composite and container references and omitting all remaining non-composite cross-references.

This distinction is useful as composite references have a very different characteristics than cross-references in many meta-metamodels such as Ecore, which is widely used in the model-driven community. The largest difference probably is that composite references determine the lifecycle of referenced model elements. Container references are just the opposites of composite references and thus AC^{cmp} also takes these into account automatically.

Opposite references introduce a strong coupling between their declaring classes not only if they are containment references. If a reference is set for one of these classes, this implies that the opposite reference is set for the target value as well. Therefore, we have separated a third variant of the *AC* metric that only measures the association-based coupling introduced by opposite references AC^{op}.

2.3 Size Uniformity

The size-uniformity metrics *MU* and *CU* relate the mean size of modules and classes to the standard deviation and are defined as follows:

$$\{MU, CU\} = \frac{\mu_{\{p,c\}}}{\mu_{\{p,c\}} + \sigma_{\{p,c\}}}$$

Here, $\mu_{\{p,c\}}$ and $\sigma_{\{p,c\}}$ denote the mean value and standard deviation for the size of packages in terms of number of classes contained in a package (*MU*: μ_p, σ_p) or the size of classes in terms of number of methods or lines of code (*CU*: μ_c, σ_c). While the number of classes of a package can be measured for metamodels as well, the number of methods for a metamodel is usually meaningless since metamodels rather concentrate on the structural features, i.e. attributes and references. Also the lines of code for a class is not applicable since metamodels are often not defined in textual syntaxes.

Therefore, we adapt the uniformity for classes in that we take the number of structural features as they make up the essential parts of a model class, in our opinion.

2.4 Other Metrics

The remaining metrics defined by Sarkar et al. are not applicable for metamodels, at least not in an early stage of development when no subsequent artifact is available. They may be applicable if e.g. analyses or transformations based on this metamodel are taken into account. An overview of these metrics and an analysis whether they are applicable for metamodels is depicted in Table 1.

As we do not have an implementation to analyze, the metrics *MII*, *NC*, *NPII*, *SAVI*, *PPI* and *APIU* are

Table 1: Summary of the Sarkar metrics with original rationale (Sarkar et al., 2008) and analysis whether they are suited to measure metamodel modularity.

Metric	Rationale	Suited
MII	Is the intermodule method call traffic routed through APIs?	No
NC	To what extent are the non-API methods accessed by other modules?	No
BCFI	Does the fragile base class problem exist across the modules?	No
IC	To what extend are the modules coupled through inheritance?	**Yes**
NPII	To what extend does the implementation code in each class program to the public interfaces of the other classes?	No
AC	To what extend are modules coupled through association?	**Yes**
SAVI	To what extend do the classes directly access the state in other classes?	No
MU	To what extend are the modules different in size?	**Yes**
CU	To what extend are the classes different in size?	**Yes**
PPI	How much superfluous code exists in a plugin module?	No
APIU	Are the APIs of a module cohesive from the standpoint of similarity of purpose and to what extend are the clients of an API segmented?	No
CReuM	To what extend are the classes that are used together also grouped together in the same module?	No

not applicable for metamodels. Furthermore, *BCFI* is not applicable as the underlying problematic "Fragile Base Class Problem" is not possible if method contents are not considered. Likewise, we do not have any information what classes are used together as we want to apply the metrics already during the metamodel development. This makes the metric *CReuM* also not applicable.

3 EXPERIMENT SETUP

To evaluate the goodness-of-fit of the presented Sarkar metrics to measure metamodel modularity, we used the data collected from a previous controlled experiment on metamodel quality perception (Hinkel et al., 2016b). In this experiment, participants were asked to manually asses the quality of metamodels created by peers. The material – domain descriptions, assessments and created metamodels – are publicly available online[1]. Due to space limitations, we therefore only replicate a very short description of the experiment.

The 24 participants created metamodels for two domains. Each domain was described in a text and the participants were asked to design a metamodel according to it. The participants consisted of professional researchers as well as students from a practical course on MDE. They were randomly assigned to the domains, ensuring a balance between the domains.

The first domain concerned user interfaces of mobile applications. Participants were asked to create a metamodel that would be able to capture designs of the user interface of mobile applications so that these user interface descriptions could later be used platform-independently. The participants created the metamodel according to a domain description in natural language from scratch. We refer to creating the metamodel of this mobile applications domain as the *Mobiles* scenario.

The second domain was business process modeling. Here, the participants were given a truncated metamodel of the Business Process Modeling Language and Notation (BPMN) (The Object Management Group, 2011) where the packages containing conversations and collaborations had been removed. The task for the participants was to reproduce the missing part of the metamodel according to a textual description of the requirements for the collaborations and conversations. We refer to this evolution task as the *BPMN* scenario in the remainder of this paper.

To evaluate our adoptions of the Sarkar metrics to measure the quality of metamodel modularity, we correlated the metric results with the manual modularity assessments and applied an analysis of variance. That is, we try to statistically prove that metric results and metamodel modularity are connected.

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

4 RESULTS

We correlated the manual quality assessments with the metric results for the metamodels created by the experiment participants. The discussion of the results is split into three sections, one for each of the scenarios and a third for discussion.

4.1 Mobiles

The results correlating the metric results against manually assessed metamodel quality perceptions are depicted in Table 2. To get a quicker overview, we have printed strong correlations ($|\rho| > 0.5$) in bold. For the metric AC^{op}, no correlations are shown as the metric values do not have a variance, i.e. no metamodel contained opposite references across package boundaries.

Table 2: Correlations of metric results to quality attribute assessments in the Mobiles and BPMN scenario. Strong correlations ($|\rho| > 0.5$) are printed in bold.

	Mobiles		BPMN	
	Quality	**Modularity**	**Quality**	**Modularity**
IC	-0.28	-0.43	0.08	0.45
AC	-0.21	-0.46	0.12	0.48
AC^{cmp}	-0.20	-0.45	0.06	**0.59**
$AC(op)$	–	–	0.23	**0.64**
MU	-0.24	**-0.76**	0.38	-0.32
CU	**0.73**	0.35	0.16	0.35

A surprising result is that all of the coupling-based Sarkar metrics have a negative correlation with modularity, among many other negative correlations to other quality attributes. This is due to the fact that these metrics only measure the quality of a modularization, but not its degree. In particular, metamodels with only one package get the maximum score of 1 for inheritance- and association-based coupling indices as there are no inheritance or association relations to other packages. However, such a metamodel is perceived as not modular as there is no modularization involved.

Using the Fisher transformation based on 14 observations and applying the Bonferroni method to control the family-wise error-rate, we can reject the null-hypothesis that the true correlation of a given metric with modularity is at least 0.3 on a 95% confidence level when the correlation is lower than -0.39. Besides *CU*, this is the case for all Sarkar metrics. For *MU*, we can even reject this hypothesis at a 99.9% confidence level.

The module uniformity metric *MU* shows the strongest negative correlations not only to modularity but also to changeability and transformation creation. The reason for this is the same as for the coupling-based metrics: Those participants that were not aware of the benefits of a good modularization often also failed in other aspects and therefore created metamodels that are hard to read. But unlike the coupling metrics where a value of 1 can also be achieved through a high quality modularization, it is highly unlikely for metamodel developers to create perfectly balanced modules, in particular because many metamodels contain modules that are only used to give a structure but do not contain any classes.

Despite it only being a corner case, the case of lacking modularization is very important. The reason is the automated refactoring, we envisioned in the introduction. Such an approach requires the underlying metrics to be robust against lacking modularization. Otherwise, the obtained results will always be monolithic metamodels, i.e. all classes put into a single package.

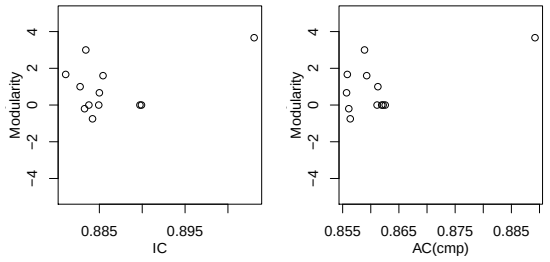
Interestingly, the class uniformity metric *CU* has strong correlations to a range of quality attributes, but not to modularity as one might have expected. A uniform class design correlated strongly with consistency, completeness, correctness, instance creation and ultimately also overall quality. A potential reason is that in object-oriented code, many bad smells such as god classes manifest in single classes having far more members than others, so one may suspect causality here. While we agree that *CU* can be a suitable metric for consistency, we think that the correlations to completeness and correctness are rather introduced by the fact that the most complete and correct metamodels were created presumably by the most experienced participants that also had an eye on consistency.

4.2 BPMN

In this section, we validate the applicability of the metrics in the BPMN scenario. Despite the fact that the participants have only evaluated manual extensions, the metric results were taken from the complete metamodels, also taking into account the larger part of the metamodel that had not been changed. While this means that the metric values may not be compared across scenarios, the influence on correlations is limited. Furthermore, we do think that this better represents an evolution scenario which is more common than creating a metamodel from scratch.

Besides, it is also not trivial to identify the relevant subset of a metamodel that should be evaluated. Even

though a major part of the metamodel was not modified by the participants of the experiment, the created extension had references and inheritance relations to the rest of the metamodel such that this could not be ignored easily by the metrics.



(a) Inheritance-based coupling (b) Association-based coupling

Figure 2: Coupling metrics plotted against the perceived modularity.

We can see that the inheritance- and association-based coupling metrics correlate with modularity, but this correlation is not so strong and for both *IC* and *AC*, the correlation coefficient is below 0.5. Especially the association-based coupling has a stronger correlation to consistency than to modularity, although we get a stronger correlation to modularity if we limit the coupling to containment references. However, this still gives worse results than restricting the association to opposite references. The correlation of AC^{cmp} to modularity is significant with $p = 0.045$ in an ANOVA, but does not withstand a correction. A similar ANOVA for *IC* yields a p-value of $p = 0.14$ so that this correlation is not even significant on the 10%-level.

The results for inheritance-based and containment-based coupling are depicted in Figure 2. As one can see, most metamodels were in a small range of metric values achieved for the inheritance-based coupling. However, the one metamodel that received a much higher score was also perceived as most modular.

The best results have been achieved by restricting the association-based coupling to opposite references with a correlation coefficient of $\rho = 0.64$ and a p-value of $p = 0.024$. However, the samples showed only a very small variance as only two metamodels had introduced new opposite references, so the sample size is too small to produce reliable results.

The class size uniformity correlates strongly with conciseness but in the BPMN scenario had a negative correlation with consistency. This means that this metric apparently cannot be used to measure consistency, as suggested from the Mobiles scenario. Likewise, the correlation to conciseness is not confirmed by the Mobiles scenario.

4.3 Discussion

The metrics by Sarkar et al. are only meant to measure the quality of modularization but not the degree to which a system is modularized. In particular, many of the metrics, in particular the ones that we adopted for metamodels as well, yield best results when no modularization is applied at all. The metrics *IC* and *AC* are set to the maximum and presumably best value 1 in case no package structure has been applied to the system or in our case the metamodel.

A possible conclusion for metamodel developers could be that the best modularization can be achieved simply by putting all classes of a metamodel into a single package and thus forgetting about packages at all. There are several examples of larger metamodels used in both industry and academia that seem to have adopted this idea as they consist of exactly one package but this way, developers have to know the entire metamodel before they can do anything. These examples include the UML metamodel used by Eclipse and many component models such as Kevoree (Fouquet et al., 2014) or SOFA 2 (Bureš et al., 2006).

On the other hand, if a metamodel consists only of a single package, developers are *aware* that they have to understand the entire metamodel before they can do anything. This may be better than a poor modularization where developers may get the impression that they can neglect some packages which in the end turns out as wrong, because of complex dependencies between packages. Therefore, the goal of developers must be a balance between the degree of modularization and its quality.

5 THREATS TO VALIDITY

The internal threats to validity described in the original experiment description (Hinkel et al., 2016b) also apply when using the collected data to validate metamodel metrics. We do not repeat them here due to space limitations.

However, a threat to validity arises as we computed the metric values in the BPMN scenario based on the entire metamodels whereas the participants were asked to assess the quality specifically of the user extensions. Additionally to the problems of alternative approaches we mentioned before, we think that the threat to the validity is acceptable since correlations coefficients do not change under linear transformations.

We are correlating the metrics results with perceived modularity in order to utilize the wisdom of our study participants. However, metrics are most

valuable if they find the subtle flaws that humans do not perceive in order to raise awareness that there might be something wrong. Furthermore, the experience of our experiment participants, especially the students, may be insufficient.

6 RELATED WORK

The Sarkar metrics have been already validated in large-scale software systems (Sarkar et al., 2008). This validation showed that randomly introduced design flaws could be detected by decreasing metric values. The goal in our validation, however, is to compare entirely different design alternatives.

Related work in the context of metamodel quality consists mostly of adoptions of metrics for UML class diagrams and object-oriented design. However, to the best of our knowledge, the characterization of metamodel quality has not yet been approached through the perception of modeling experts.

Bertoa et al. (Bertoa and Vallecillo, 2010) present a rich collection of quality attributes for metamodels. However, as it is not the scope of their work, they do not give any information how to quantify the attributes.

Ma et al. (Ma et al., 2013) present a quality model for metamodels. By transferring metrics from object-oriented models and weighting them, they provide composite metrics to quantify quality properties. They calculate these metrics for several versions of the UML metamodel. However, they do not provide a correlation between their metrics and quality.

López et al. propose a tool and language to check for properties of metamodels (López-Fernández et al., 2014). In their paper, they also provide a catalog of negative properties, categorized in design flaws, best practices, naming conventions and metrics. They check for breaches of fixed thresholds for the same metrics, but both their catalog and also these thresholds stem from conventions and experience and are not empirically validated.

Williams et al. applied a variety of size metrics onto a big collection of metamodels (Williams et al., 2013). However, they did not draw any conclusions with regards to quality.

Di Rocco et al. also applied metrics onto a large set of metamodels (Di Rocco et al., 2014). Besides size metrics, they also feature the number of isolated metaclasses and the number of concrete immediately featureless metaclasses. Based on the characteristics they draw conclusions about general characteristics of metamodels. However, to the best of our knowledge, they did not correlate the metric results to any quality attributes.

Leitner et al. propose complexity metrics for domain models of the software product line field as well as feature models (Leitner et al., 2012). However, domain models are not as constrained by their metamodels as it is the case with feature models. The authors argue, that the complexity of both, feature and domain models, influences the overall quality of the model, but especially usability and maintainability. They show the applicability of their metrics, but do not validate the influence between the metrics and quality.

Vanderfeesten et al. investigated quality and designed metrics for business process models (Vanderfeesten et al., 2007). Some of them can be applied to metamodels or even graphs in general. The metrics are validated by assessing the relation between metric results and error occurrences and manual quality assessments (Mendling and Neumann, 2007; Mendling et al., 2007; Sánchez-González et al., 2010; Vanderfeesten et al., 2008). However, it is subject of further research to investigate how these metrics can be adapted to metamodels.

7 CONCLUSION AND OUTLOOK

The results of this paper suggest that the metrics established to measure the quality of modularization in software systems alone may be misleading. From the few metrics suggested by Sarkar et al., many were not applicable to metamodels as they require an existing implementation and the remaining metrics partially favor monolithic metamodels over properly modularized ones. As a consequence, no significant correlations between these metrics and the manually assessed modularity of metamodels could be observed. Particularly in the Mobiles scenario, we were even able to statistically disprove even a slight correlation of 0.3 between the metric values and the perceived metamodel, which makes the metrics practically useless for the purpose of predicting how the modularity of a given metamodel is perceived.

This insight raises the question whether there are there other metrics that correlate with the perception of metamodel quality. An answer to this question will improve the understanding on how the modularity of metamodels is perceived.

ACKNOWLEDGEMENTS

This research has received funding from the European Union Horizon 2020 Future and Emerging Technolo-

gies Programme (H2020-EU.1.2.FET) under grant agreement no. 720270 (Human Brain Project SGA-I) and the Helmholtz Association of German Research Centers.

REFERENCES

Bertoa, M. F. and Vallecillo, A. (2010). Quality attributes for software metamodels. In *Proceedings of the 13th TOOLS Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2010)*.

Bureš, T., Hnetynka, P., and Plášil, F. (2006). Sofa 2.0: Balancing advanced features in a hierarchical component model. In *Proceedings of the fourth International Conference on Software Engineering Research, Management and Applications*, pages 40–48. IEEE.

Di Rocco, J., Di Ruscio, D., Iovino, L., and Pierantonio, A. (2014). Mining metrics for understanding metamodel characteristics. In *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, MiSE 2014, pages 55–60, New York, NY, USA. ACM.

Fouquet, F., Nain, G., Morin, B., Daubert, E., Barais, O., Plouzeau, N., and Jézéquel, J.-M. (2014). Kevoree Modeling Framework (KMF): Efficient modeling techniques for runtime use. Technical report, SnT-University of Luxembourg.

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

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

Hinkel, G., Kramer, M., Burger, E., Strittmatter, M., and Happe, L. (2016b). An Empirical Study on the Perception of Metamodel Quality. In *Proceedings of the 4th International Conference on Model-driven Engineering and Software Development (MODELSWARD)*. Scitepress.

Kramer, M. E., Durdik, Z., Hauck, M., Henss, J., Küster, M., Merkle, P., and Rentschler, A. (2012). Extending the Palladio Component Model using Profiles and Stereotypes. In Becker, S., Happe, J., Koziolek, A., and Reussner, R., editors, *Palladio Days 2012 Proceedings (appeared as technical report)*, Karlsruhe Reports in Informatics ; 2012,21, pages 7–15, Karlsruhe. KIT, Faculty of Informatics.

Lehman, M., Ramil, J., Wernick, P., Perry, D., and Turski, W. (1997). Metrics and laws of software evolution-the nineties view. In *Software Metrics Symposium, 1997. Proceedings., Fourth International*, pages 20–32.

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

Leitner, A., Weiß, R., and Kreiner, C. (2012). Analyzing the complexity of domain model representations. In *Proceedings of the 19th International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*, pages 242–248.

López-Fernández, J. J., Guerra, E., and de Lara, J. (2014). Assessing the quality of meta-models. In *Proceedings of the 11th Workshop on Model Driven Engineering, Verification and Validation (MoDeVVa)*, page 3.

Ma, Z., He, X., and Liu, C. (2013). Assessing the quality of metamodels. *Frontiers of Computer Science*, 7(4):558–570.

Mendling, J. and Neumann, G. (2007). Error metrics for business process models. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, pages 53–56.

Mendling, J., Neumann, G., and Van Der Aalst, W. (2007). Understanding the occurrence of errors in process models based on metrics. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, pages 113–130. Springer.

Sánchez-González, L., García, F., Mendling, J., Ruiz, F., and Piattini, M. (2010). Prediction of business process model quality based on structural metrics. In *Conceptual Modeling–ER 2010*, pages 458–463. Springer.

Sarkar, S., Kak, A. C., and Rama, G. M. (2008). Metrics for measuring the quality of modularization of large-scale object-oriented software. *Software Engineering, IEEE Transactions on*, 34(5):700–720.

The Object Management Group (2011). Business process model and notation 2.0. http://www.bpmn.org/.

Vanderfeesten, I., Cardoso, J., Mendling, J., Reijers, H. A., and van der Aalst, W. (2007). Quality metrics for business process models. *BPM and Workflow handbook*, 144.

Vanderfeesten, I., Reijers, H. A., Mendling, J., van der Aalst, W. M., and Cardoso, J. (2008). On a quest for good process models: the cross-connectivity metric. In *Advanced Information Systems Engineering*, pages 480–494. Springer.

Williams, J. R., Zolotas, A., Matragkas, N. D., Rose, L. M., Kolovos, D. S., Paige, R. F., and Polack, F. A. (2013). What do metamodels really look like? In *Proceedings of the first international Workshop on Experiences and Empirical Studies in Software Modelling (EESSMod)*, pages 55–60.