Automated Quality Analysis of Software Engineering Method Models

Masud Fazal-Baqaie and Frank Kluthe
s-lab – Software Quality Lab, University of Paderborn, Zukunftsmeile 1, Paderborn, Germany

Keywords: Process Model Quality, Situational Method Engineering, Method Pattern, Static Analysis, OCL.

Abstract: Using a suitable software engineering method (SEM) for a specific project and following it properly is important for the resulting software quality. However, SEMs described in natural language are often ambiguous and lack automated guidance for the team members, causing impediments for the project. The model-based approach Method Engineering with Method Services and Method Patterns (MESP) allows to model enactable SEM models by composing pre-defined building blocks. Up to now, the quality of MESP models had to be checked manually which was tedious and error-prone at times. In this paper, we present an automated design-time quality analysis for MESP SEM models. In particular, our analysis allows to automatically evaluate generic quality characteristics relevant for all SEM models as well as specific quality requirements specified using MESP method patterns. We integrated the quality analysis framework into the MESP Workbench and our evaluation shows that the analysis is fast enough to provide timely feedback even for large SEM models.

1 INTRODUCTION

Using a suitable software engineering method (SEM) is important for the resulting software quality (Fitzgerald et al., 2003), hence SEMs need to be customized to the specific project, e.g., to the level of risk (Cockburn, 2000). In addition, their representation needs to be unambiguous and descriptive enough to be followed properly. For example, SEMs in natural language tend to be ambiguous, thus misunderstandings can lead to coordination overhead. In contrast, models created with formal process description languages describe the flow of activities in an unambiguous manner allowing for automation.

Method Engineering with Method Services and Method Patterns (MESP) (Fazal-Baqaie and Engels, 2016) is a situational method engineering approach (Henderson-Sellers et al., 2014) that supports the creation of such formal SEM models based on the specific project characteristics. Following the assembly-based paradigm (Brinkkemper, 1996), SEM models are composed based on pre-defined method services and method patterns. These models can be enacted with a workflow engine that supports the project team in applying the SEM. In addition, MESP introduces the notion of method patterns that describe quality requirements for a SEM model, e.g., that tasks with specific output have to be part of it.

As SEMs can comprise hundreds of tasks and work products (e.g. RUP (Kruchten, 1999)), models can become quite large. Thus, automated feedback about the quality of SEM models is desirable. This paper presents an automated quality analysis framework for the MESP approach. It makes the following contributions: (i) introduction of a set of general quality characteristics for MESP models, which are formally specified using the Object Constraint Language (OCL), (ii) translation of quality requirements specified with MESP method patterns into OCL, (iii) a framework to automatically analyze the quality of MESP models and its implementation and integration into the MESP workbench, and (iv) a performance evaluation of the analysis.

This paper is structured as follows: Section 2 explains the MESP approach and the structure of SEM models. Section 3 presents our automated quality analysis framework. Section 4 describes the results of our performance analysis. Section 5 discusses related work. Finally, Section 6 concludes the paper.

2 BACKGROUND

In the following, we provide a brief overview of the MESP approach and explain which part is supported by our quality analysis. Subsequently, we explain how SEM models are modeled and especially the use of method patterns.

Fazal-Baqaie, M. and Kluthe, F.
Automated Quality Analysis of Software Engineering Method Models.
DOI: 10.5220/0005743205270534
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 527-534
ISBN: 978-989-758-168-7

2.1 Overview of the MESP Approach

As illustrated in Figure 1, MESP differentiates three roles that are responsible for different tasks:

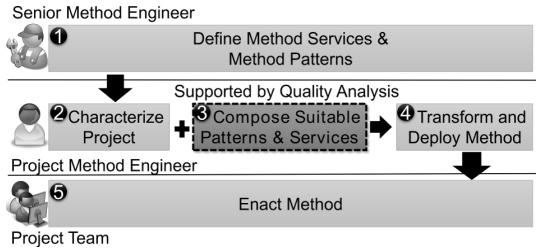

Figure 1: Overview of MESP, highlighting the task supported by the quality analysis.

The *senior method engineer* is responsible for formalizing reusable SEM model building blocks by defining method services and method patterns. The building blocks are stored in a repository and can be reused in arbitrary many SEM models.

The *project method engineer* is responsible for composing a customized SEM model with respect to a specific project. Based on the characterization of a project (❷), suitable method services and method patterns are retrieved from the repository and composed by specifying control and data flow between them (❸). Thereby, method services must not violate the chosen method patterns (Fazal-Baqaie et al., 2013). The automated quality analysis, which is introduced in Section 3, is performed during the highlighted task (❸). When the method composition is completed, the project method engineer invokes a transformation into a mainstream process description language and deploys the method model into a workflow engine (Fazal-Baqaie et al., 2014) (❹).

The *project team* is guided by the workflow engine, which enacts the SEM model (❺). It creates tasks and assigns them to the team and thereby ensures that the team follows the SEM model.

2.2 Running Example

In this section, we illustrate the building blocks of SEM models and their composition to a SEM model, in order to show how MESP methods are formalized. We also briefly discuss examples for quality issues and method pattern violations.

2.2.1 Defining Method Services and Method Patterns

The senior method engineer creates method services and method patterns and maintains them in the method repository. Figure 2 shows an example for a method service that we derived from tasks in the practices library of the Eclipse Process Framework[1]. On the top, it shows the textual description of the task that is used by the project team during enactment. On the bottom, it shows further information as part of the interface that is used to identify suitable method services and to compose them correctly. Here, the senior method engineer describes the relation to work products and roles, for which project context the method service is suitable (situational factors), and what kind of task it describes (categories).

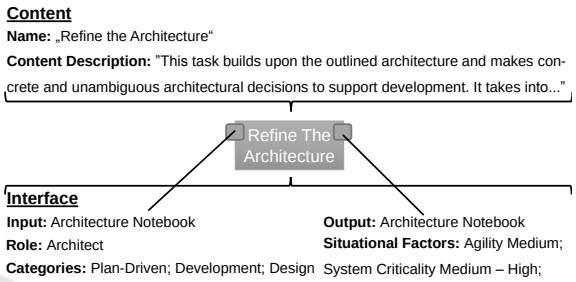

Figure 2: Illustration of an exemplary method service.

Method patterns are used as context-specific quality requirements for the SEM model contents (Fazal-Baqaie et al., 2013). To illustrate this, we use a sprint loop as an example. A sprint loop is a common construct in agile methodologies, e.g., Scrum (Schwaber and Sutherland, 2013). It prescribes a repetitive loop of fixed length, where planning activities are followed by implementation activities, which in turn are followed by reviewing activities. Figure 3 shows a partially composed method using a “Sprint Loop” method pattern. The figure also shows a sequence and three method services, which are not part of the method pattern and which we ignore for the moment. The method pattern contains three so called constrained scopes, denoted by dotted rectangles. Constrained scopes are connected by control flow, e.g., sequential or parallel flow, or, like in this case, an iteration (loop). Each constrained scope contains a constraint in the lower half (grey background) that needs to be fulfilled by the upper half (white background), which potentially hosts method services.

Constraints are formulated with a specialized domain-specific language that is part of the MESP meta-model. In this paper, we describe how these constraints are translated to OCL as part of our quality analysis in order to evaluate them automatically. For example, the constraint of the constrained scope in the center of Figure 3 describes that all method services (`All MS`) need to be of category “Development”

[1] http://epf.eclipse.org/wikis/epfpractices/index.htm

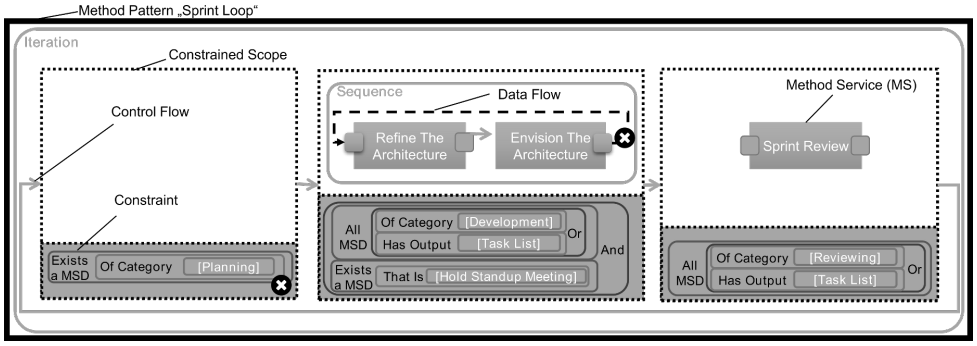

Figure 3: Partly composed MESP method with issues highlighted by our quality analysis.

(`Of Category`) or alternatively (`Or`) produce the output work product “Task List” (`Has Output`). Additionally (`And`), at least one method service (`Exists a MS`) must be “Hold Standup Meeting”.

2.2.2 Composing Suitable Patterns and Services

The project method engineer chooses suitable method patterns and method services and composes the SEM model for his project. Figure 3 shows a partially composed SEM model. It contains a method pattern and three method services. Also, the control flow was refined with a “Sequence” element that denotes that “Envision the Architecture” is executed after “Refine the Architecture”. These two method services were also connected with data flow (dotted arrow) that denotes that the output of “Envision the Architecture” is used by “Refine the Architecture”.

To ensure that the method is enactable, control and data flow must be consistent. In addition, all constraints of the used method patterns have to be fulfilled. Looking at the example in Figure 3, as the first constrained scope is empty, the respective constraint is not fulfilled as denoted by an ✪. In addition, the control and data flow between “Envision the Architecture” and “Refine the Architecture” is contradicting and marked with an ✪.

2.3 Overview of the MESP Meta-Model

While the meta-model is hidden to the users of MESP, our quality analysis is build on top of it, because OCL constraints are formulated with respect to meta-models. Figure 4 shows an excerpt of the main classes of the MESP meta-model. The left side shows classes that are created and maintained by the senior method engineer and stored in the method repository. As illustrated, `Method Patterns` and `Method Service` contain a defined `Interface` that references `Role`, `Situational Factor Value`, etc. to characterize the pattern or service (cf. Figure 2).

The right shows classes that are part of a composed MESP SEM model and which are thus used by the project method engineer. A SEM model contains a `Process` that contains `Activity` elements. These can be control flow elements like sequences and parallel flows (classes not depicted), references to method services (`Method Service Descriptor`), or references to method patterns (`Method Pattern Descriptor`). The data flow between `Method Service Descriptor` elements is modeled using `Activity Input Mappings` elements.

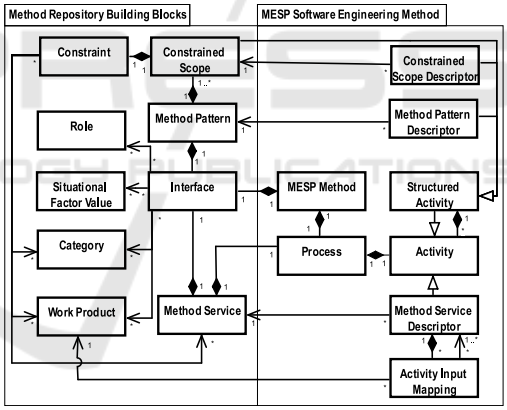

Figure 4: An overview of relevant MESP meta-model classes.

3 AUTOMATED QUALITY ANALYSIS

In this section, we present our automated quality analysis that can be used by the method engineer during the composition. We first discuss the requirements for an automated quality analysis framework. Then we provide an overview of how the analysis framework evaluates constraints on a SEM model. Afterward, we describe the general quality characteristics and their formalization with OCL. Thereafter, we explain the on-the-fly translation of method pattern

constraints into OCL constraints.

3.1 Requirements for the Analysis

The overall goal of the automated quality analysis is to support the project method engineer to "Compose Suitable Patterns & Services" (cf. Section 2.1) by reporting quality issues and method pattern violations. More specifically, we have identified the following requirements for an automated quality analysis framework:

R1 Partial Analysis of Models: In order to provide early feedback, the analysis shall be applicable to incomplete SEM models and restrictable to single regions of the model.

R2 Traceability of Issues: In order to help in fixing quality issues, the elements of a SEM model that cause quality issues shall be reported.

R3 Categorization of Issues: In order to help in assessing quality issues, the analysis shall classify quality issues. Most importantly, critical (preventing prevent SEM models from being enacted) and non-critical issues shall be distinguished.

R4 Extensibility of the Analysis Framework: In order to implement further quality checks easily, the analysis framework shall be extensible.

R5 High Performance of the Analysis: In order to be used frequently throughout the composition of a SEM model, the analysis shall not take longer than a couple of seconds.

3.2 Overview of the Quality Analysis Framework

We have created a quality analysis framework to implement the quality analysis. We integrated this framework into the existing MESP workbench of the MESP approach (Fazal-Baqaie and Engels, 2016; Fazal-Baqaie et al., 2014). It is based on the Eclipse Modeling Framework (Steinberg et al., 2009) that allows to automatically check conformance of models with their meta-models. This allows to ensure that all elements referenced from the method repository exist, a quality characteristic we call "Reference Completeness" (see Section 3.3). However, other quality characteristics and the fulfillment of constraints described in method patterns cannot be checked out of the box.

Figure 5 provides an overview of our extensible quality analysis framework that is based on the Object Constraint Language (OCL) (OMG, 2014). The analysis consists of two parts. First, the SEM model is evaluated against generic quality constraints derived from general quality characteristics (1.). This will be explained in the following Section 3.3. Second, the SEM model is evaluated against MESP method pattern constraints that are extracted from the model and transformed into equivalent OCL expressions (2.). This will be explained in Section 3.4. For the evaluation of OCL constraints, our framework reuses the Eclipse OCL Component. Together with the responsible model elements, the detected issues and pattern violations are passed to the Eclipse Problems View component and then are presented to the project method engineer.

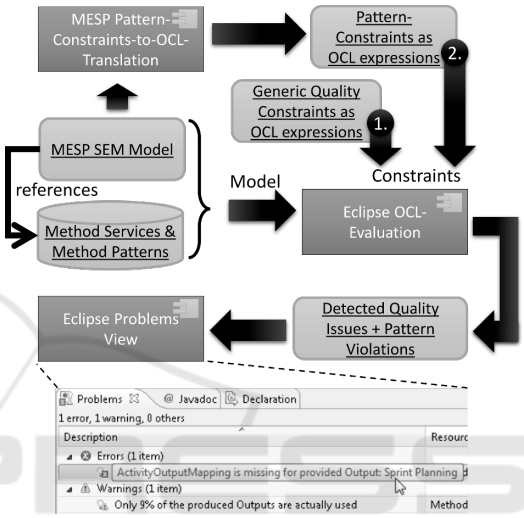

Figure 5: Quality analysis framework of MESP.

3.3 Formalization and Analysis of General Quality Characteristics

We defined several quality characteristics for MESP-based SEM models based on the categories proposed in (Harmsen, 1997) and considering additional literature from the method engineering and process model domains. As shown in Table 1, we have currently implemented a basic set of 4 critical and 2 non-critical quality characteristics (cf. requirement R3) to demonstrate the approach and the extensibility of the framework (cf. requirement R4). While our defined characteristics cover all essential aspects to ensure the enactment of SEM models, providing a "complete" set of quality characteristics is not in scope of this paper. Instead, we focused on creating a quality analysis framework, which makes the integration of further quality characteristics fairly easy.

For each of the quality characteristics we derived an OCL constraint. Every time the user invokes the analysis, the SEM model is checked against all of these general quality constraints (see 1. in Figure 5) and the quality issues are reported. In this section,

Table 1: Implemented Quality Characteristics.

Critical Quality Characteristics	
Reference Completeness	All referenced elements exist in the method repository.
Input/Output Completeness	For all required input work products of method service descriptors the necessary data flow is specified.
Flow Consistency	The control flow order of method service descriptors does not contradict the specified data flow.
Input/Output Consistency	The specified data flow is consistent to the referenced method services.
Non-Critical Quality Characteristics	
Work Product Efficiency	The percentage of produced, but unused outputs of service descriptors.
Structural Soundness	The percentage of empty, thus unnecessary control flow constructs.

we use "Flow Consistency" as an example to exemplify the derivation of OCL constraints. The MESP SEM model illustrated in Figure 3 violates the quality characteristic, because the method service descriptor "Refine the Architecture" needs an input work product from "Envision the Architecture", which is executed afterward. In other words, it is required to check whether an input of a method service descriptor X (here "Refine the Architecture") is created by another method service descriptor Y (here "Envision the Architecture"), where Y is not a predecessor of X. This would signify a contradiction between the control flow order (X before Y) and the data flow (X needs input from Y).

3.4 Formalization and Analysis of Method Pattern Constraints

As explained in Section 2.2, constraints in method patterns are formulated using a structured language that is designed to be used by senior and project method engineers and which is part of the MESP approach. Translating method pattern constraints to OCL allows to reuse the existing Eclipse OCL Component to evaluate constraints against the SEM model. The translation works as follows (see Figure 5): Based on the pattern references in the SEM model, the corresponding constraints from method repository are extracted and translated into equivalent OCL constraints. Then these are checked against the SEM model. While the general quality constraints presented in the previous section are defined statically, these pattern-related constraints are created on-the-fly during each run of the analysis. In addition, pattern-related constraints are evaluated only against the respective constrained scope descriptor, while general quality constraints are evaluated against all elements in the scope of the analysis, so usually the whole SEM model.

We illustrate the translation of method pattern constraints using the constraint depicted in the middle constrained scope of Figure 3. The internal representation of this constraint is shown in Figure 6. Underneath each object the corresponding concrete syntax element is presented. As depicted, some information is stored as attributes of the objects, e.g., whether negation was used. As part of the analysis, the Pattern-Constraint-to-OCL-Translation component translates the presented pattern constraint into the two OCL constraints in Figure 7. We marked the constraint language constructs of MESP in Figure 6 and the corresponding OCL statements in Figure 7 with the same number. As shown, based on the quantifier of the statement, i.e., `OrStatement` or `AtomicStatement`, a "`forAll`" or "`exists`" construct is used in OCL (1., 6.). The body of this construct then depends on the type of the term, i.e., `CategoryTerm` (2.), `WorkProductTerm` (4.), or `IdentityTerm` (7.). If a term was negated, "`exist`" would be replaced by "`not exist`" (2.,4.,6.). From the referenced `Category` (3.), `Work Product` (5.), and `Method Service` (8.) the "Name" attribute is used to create the respective search expressions.

The two OCL constraints are evaluated against the respective constrained scope descriptors using the Eclipse OCL Component. The result of the overall constraint consisting of the two `ConjunctivePartialConstraint` is then computed using a Java "AND"-Statement. The evaluation of our example, the pattern-related constraint of the middle constraint scope in Figure 3 would evaluate to false: With the assumption that both referenced method services have the "Development" category, the upper conjunctive partial constraint would be fulfilled, however, as there is no "Hold Standup Meeting" method service referenced in the constrained scope, the lower conjunctive partial constraint B is not fulfilled. Thus, the overall evaluation result is also negative. Similar to the discussed example, all possible method pattern constraints can be translated to OCL and consequently evaluated with our quality analysis.

3.5 Fulfillment of Stated Requirements

After we explained the quality analysis, we revisit the requirements described in Section 3.1 and discuss their fulfillment.

R1: The analysis is accessible via the Eclipse Validation menu and can be invoked on the whole `Process`, but also on single `Activities` (cf. Figure 4). It runs also on SEM models that are only partially completed. Thus, R1 is fulfilled.

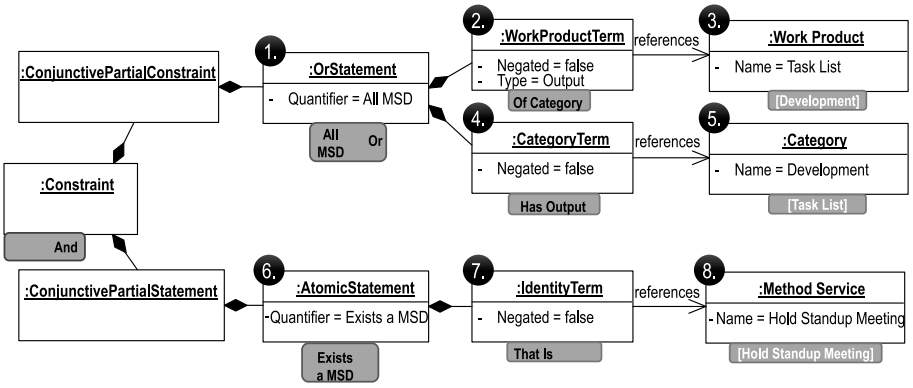

Figure 6: The object model representation of a MESP Constraint.

```
1 -- ConjunctivePartialConstraint A
2 context ConstrainedScopeDescriptor
3 getNested()->
4 select(oclIsTypeOf(MethodServiceDescriptor))->
5   forAll(msd |                                        --(1.)
6     msd.methodService.interface.categories->          --(2.)
7    exist(category | category.name                     --(2.)
8       = 'Development')                                --(3.)
9         or                                            --(1.)
10    msd.methodService.interface.mandatoryOutput->     --(4.)
11    exist(workProduct | workProduct.name              --(4.)
12      = 'Task List')                                  --(5.)
13  )
14
15 -- ConjunctivePartialConstraint B
16 context ConstrainedScopeDescriptor
17 getNested()->
18 select(oclIsTypeOf(MethodServiceDescriptor))->
19   exists( msd |                                       --(6.)
20     msd.methodService.name                           --(7.)
21     = 'Hold Standup Meeting'                         --(8.)
22   )
```

Figure 7: Generated Method Pattern Constraints.

R2: In our analysis quality issues are presented to the user in the Eclipse Problems View (cf. Figure 5). If the user double-clicks on an issue in the Problems View, he jumps to the causing element. In addition, the elements that cause quality issues are highlighted in the modeling editors. Thus, R2 is fulfilled.

R3: In the Problems View, critical issues are presented as errors and non-critical issues as warnings (cf. Figure 5). Thus R3 is fulfilled.

R4: The quality analysis framework is not limited to a specific set of quality characteristics, because additional quality characteristics can always be added by extending the OCL quality constraints. In Section 3.3, we explained the basic set of quality constraints that we have already provided using this mechanism. Thus, R4 is fulfilled.

R5: As shown in the evaluation (see Section 4), the analysis is fast enough for realistically sized SEM models, thus R5 is fulfilled.

4 PERFORMANCE EVALUATION

We fully implemented and integrated the quality analysis into the existing MESP workbench. In this section, we discuss the runtime performance based on the implementation of realistically sized SEM models in order to determine the fulfillment of requirement R5.

4.1 Evaluation Approach

For the evaluation, we composed SEM models from a method repository derived from the practices library of the Eclipse Process Framework[2]. We prepared SEM models focusing on the parameters that impact the runtime of the analysis (see Table 2). For example, the number of detected issues has no noticeable effect on the runtime and thus was omitted.

We created three sets of models. We created a set “A” of SEM models to investigate the analysis of general quality characteristics that is influenced mainly by the number of method service descriptors and input mappings (data flow specifications). In addition, we created a set “B” of models to investigate the influence of the method pattern analysis on the runtime (cf. Section 3.4). Here, the influencing parameters are the number of constrained scopes, the number of partial constraints (that are evaluated individually by the OCL Component), and the number of method service descriptors per constrained scope. Both sets exceed the complexity of big well-known SEMs like RUP and represent an upper bound for realistically sized SEM models. Set “C” of SEM models contains models that are unrealistically big and aims primarily at investigating the boundaries of the analysis.

[2]http://epf.eclipse.org/wikis/epfpractices/index.htm

4.2 Results and Interpretation

Table 2 shows an excerpt of the results of our runtime evaluation. For each model, we took the average of 10 runs, although the measurements deviated only in fractions of a second. The evaluation shows that realistically sized models (sets A and B) are analyzed in a few seconds. Thereby, the analysis of method pattern constraints seems to have a minor influence on the runtime compared to the evaluation of the general quality characteristics (cf. A3 to B2/B3 and C1 to C2). The analysis of the unrealistically big SEM models (set C), which are about 4 times bigger, takes about 30 seconds (cf. C1/C2).

Table 2: Results of the runtime evaluation of the analysis framework.

ID	SDs	IMs	CSs	PCs	D/S	t
A1	500	-	-	-	-	0.1
A2	500	1000	-	-	-	2.0
A3	500	2000	-	-	-	3.4
B2	500	2000	5	15	100	3.9
B3	500	2000	20	60	25	4.0
C1	2000	4000	-	-	-	31
C2	2000	4000	20	60	100	34

SDs: Service Descriptors **IMs**: Input Mappings **CSs**: Constrained Scopes **PCs**: Partial Constraints **D/S**: Descriptors per Scope **t**: Runtime (sec)

The evaluation shows that the runtime performance of the analysis fulfills the requirement R5 and allows its continuous use during method composition. For the evaluation, we considered the runtime of the analysis of complete SEM models. However, the user has the choice to run the analysis on parts of the model to get specific results that are computed more quickly.

Regarding threats to validity, there is the risk that the SEM models that we created are not representative. However, we do not consider this to be very likely, because of the variety of characteristics we investigated in our experiments.

5 RELATED WORK

There are two directions of related work: Firstly, approaches to use enactable process description languages to represent software engineering methods. Models of these approaches can be enacted with workflow engines that support the project team in following the SEM. Secondly, approaches for situational method engineering that focus on the modeling of SEMs with respect to project characteristics.

5.1 Process Description Languages

BPMN (OMG, 2011) and BPEL (OASIS, 2007) are established languages to modeling processes for business workflow management (Aalst and Hee, 2002). For the domain of SEMs, there is no such established enactable process description language (Bendraou et al., 2010).

Formalizing and automatically checking quality characteristics has been done for various other languages, e.g. Little-JIL (Chen et al., 2008) or UML acitivity diagrams (Khaluf et al., 2011). Yet, these languages do not support assembly-based method engineering and the method patterns of MESP.

The de facto standard description language for SEMs is SPEM (OMG, 2008). It offers some limited method engineering capability (Rougemaille et al., 2009), but SPEM models are not enactable. Bendraou et al. (Bendraou et al., 2007) and Ellner et al. (Ellner et al., 2012) propose extensions to SPEM and tooling to model and enact their models. However, they offer at most basic quality assurance support and do not support a notion similar to method patterns, while their evaluation is one of our main goals.

5.2 Situational Method Engineering

There is a broad range of situational method engineering approaches (Henderson-Sellers et al., 2014), we focus on approaches with formal modeling languages and tool support here. Until the beginning of the last decade, several tool-supported method engineering approaches were proposed, but did not gain acceptance and are not available for current software platforms. Brinkkemper et al. proposed the method engineering tool Demacrone (Brinkkemper et al., 1998; Harmsen, 1997) that used a set-based formalism and offered method model consistency rules formalized in first order predicate logic. They defined five quality categories that we re-use for our quality characteristics.

To the best of our knowledge there is only one approach that was proposed lately. The assembly-based approach is build with a Eclipse-based modeling platform called MOSKitt (Cervera et al., 2011) and specifically focuses on deriving a suitable CASE environment to use with the SEM. Similar to our approach, method fragments are defined and annotated with meta-information that can be facilitated during method assembly to find suitable fragments. However, a notion similar to method patterns is not supported and quality analysis is not described as part of their approach.

6 CONCLUSIONS

In this paper, we presented an automated quality analysis for models composed with the approach Method Engineering with Method Services and Method Patterns (MESP). The composed models are formal representations of software engineering methods (SEMs) used by a project team to create software in the context of a specific project. Our extensible approach allows to check MESP SEM models against statically defined quality characteristics formalized with the Object Constraint Language (OCL) and we demonstrate that by offering a basic set of these constraints. In addition, based on an on-the-fly translation to OCL, it supports the analysis against method pattern constraints, a notion to define quality requirements introduced by the MESP approach. Our analysis runs on partial models and it traces quality issues back to their cause. We provide a quality analysis framework that is integrated into the MESP workbench and our performance evaluation shows that the quality analysis runs fast enough to be used in practice.

In the future, we want to formalize further quality constraints, e.g., the degree to which method patterns are used in a SEM model.

REFERENCES

Aalst, W. M. P. and Hee, K. M. (2002). *Workflow management: Models, methods, and systems*. MIT Press.

Bendraou, R. et al. (2007). Definition of an Executable SPEM 2.0. In *14th Asia-Pacific Softw. Eng. Conf.*, pages 390–397. IEEE.

Bendraou, R. et al. (2010). A Comparison of Six UML-Based Languages for Software Process Modeling. *IEEE Trans. on Soft. Eng.*, 36(5):662–675.

Brinkkemper, S. (1996). Method engineering: engineering of information systems development methods and tools. *Information & Softw. Technology*, 38(4):275–280.

Brinkkemper, S., Saeki, M., and Harmsen, A. F. (1998). Assembly Techniques for Method Engineering. In Pernici, B. and Thanos, C., editors, *Proc. of the 10th Int. Conf. on Advanced information systems engineering*, volume 1413 of *LNCS*, pages 381–400. Springer.

Cervera, M. et al. (2011). Turning Method Engineering Support into Reality. In Ralyté, J., Mirbel, I., and Deneckère, R., editors, *IFIP Advances in Information and Communication Technology*, volume 351, pages 138–152. Springer.

Chen, B. et al. (2008). Analyzing medical processes. In *Proceedings of the 30th Int. Conf. on Software Engineering*, pages 623–632. ACM.

Cockburn, A. (2000). Selecting a project's methodology. *IEEE Software*, 17(4):64–71.

Ellner, R. et al. (2012). An Integrated Tool Chain for Software Process Modeling and Execution. In Störrle et al., editors, *Joint Proc. of the 8th European Conf. on Modelling Foundations and Applications*, pages 73–82. Technical University of Denmark.

Fazal-Baqaie, M. and Engels, G. (to appear in 2016). Managing Software Processes Evolution by Assembly-Based Method Engineering with MESP. In Kuhrmann et al., editors, *Managing Software Process Evolution*. Springer.

Fazal-Baqaie, M., Gerth, C., and Engels, G. (2014). Breathing life into situational software engineering methods. In Jedlitschka et al., editors, *Proc. of the 15th Int. Conf. of Product Focused Software Process Improvement*, volume 8892, pages 281–284. Springer.

Fazal-Baqaie, M., Luckey, M., and Engels, G. (2013). Assembly-Based Method Engineering with Method Patterns. In Wagner, S. and Lichter, H., editors, *Proc. of the Software Engineering 2013*, volume 215 of *LNI*, pages 435–444. GI.

Fitzgerald, B., Russo, N. L., and O'Kane, T. (2003). Software development method tailoring at Motorola. *Communications of the ACM*, 46(4):64–70.

Harmsen, A. F. (1997). *Situational method engineering*. Moret Ernst & Young.

Henderson-Sellers, B. et al. (2014). *Situational Method Engineering*. Springer.

Khaluf, L., Gerth, C., and Engels, G. (2011). Pattern-based modeling and formalizing of business process quality constraints. In Mouratidis, H. and Rolland, C., editors, *23rd Int. Conf. on Advanced Information Systems Engineering*, volume 6741 of *LNCS*, pages 521–535. Springer.

Kruchten, P. (1999). *The rational unified process: An introduction*. Object technology series. Addison-Wesley.

OASIS (2007). Web Services Business Process Execution Language.

OMG (2008). Software & Systems Process Engineering Metamodel Specification (SPEM).

OMG (2011). Business Process Model and Notation.

OMG (2014). Object Constraint Language.

Rougemaille, S. et al. (2009). Methodology Fragments Definition in SPEM for Designing Adaptive Methodology: A First Step. In Luck, M. and Gomez-Sanz, J. J., editors, *Proc. of the 9th int. workshop on Agent-oriented software engineering*, volume 5386 of *LNCS*, pages 74–85. Springer.

Schwaber, K. and Sutherland, J. (2013). The Scrum Guide.

Steinberg, D. et al. (2009). *EMF: Eclipse Modeling Framework*. Addison-Wesley.