Concern Metrics for Modularity-Oriented Modernizations

Chapter · August 2019

DOI: 10.1007/978-3-030-29238-6_16

Bruno Santos
Universidade Federal de São Carlos
8 PUBLICATIONS **45** CITATIONS
SEE PROFILE

Daniel Gustavo San Martín Santibáñez
Universidad Católica del Norte (Chile)
17 PUBLICATIONS **80** CITATIONS
SEE PROFILE

Raphael Honda
4 PUBLICATIONS **18** CITATIONS
SEE PROFILE

Valter Vieira de Camargo
Universidade Federal de São Carlos
25 PUBLICATIONS **123** CITATIONS
SEE PROFILE

Concern Metrics for Modularity-Oriented Modernizations

Bruno Santos$^{1[0000-0002-0075-801X]}$, Daniel San Martín$^{1[0000-0001-5274-0148]}$, Raphael Honda^{1}, and Valter Vieira de Camargo$^{1[0000-0002-6439-4649]}$

Universidade Federal de São Carlos, São Carlos, SP, Brazil
{bruno.santos,daniel.santibanez,raphael.honda,valtervcamargo}@ufscar.br

Abstract. A known problem in legacy systems is the presence of crosscutting concerns in their architecture hampering and increasing the maintenance costs. A possible solution for that is the conduction of modularity-oriented modernization aiming at restructuring the legacy software system in order to improve its artifacts (source code, database, others) quality. Architecture-Driven Modernization (ADM) is focused on modernizing legacy software systems by using the concepts of model-driven architecture, software reengineering and standard metamodels. Knowledge Discovery Metamodel (KDM) is a standard from ADM that is able to represent software systems by means of KDM instances that are going to be modernized. An intrinsic part of the modernization process is to measure both KDM versions, the legacy and the modernized one. The measurement of legacy KDM instances enables the software engineer to quantify the existing problems while the measurement of the modernized one enables to verify whether the problems have been solved or not. Structured Metrics Metamodel (SMM) is a metamodel that can be used to specify metrics to be applied on KDM instances. However, even though most of the well known metrics are supported by SMM, there is no study that investigate how SMM could be used to specify concern metrics. We present how SMM can be used to specify concern metrics and a tool to support this measurement process, enabling the conduction of modularity-oriented modernizations to help ensuring the quality of the modernization process. Furthermore, we also discuss some challenges to be overcome involving the quality measurement process in ADM blueprint.

Keywords: Software Modernization · Metrics · Architecture-Driven Modernization · Structured Metrics Metamodel · Knowledge Discovery Metamodel.

1 Introduction

Software systems become legacy when their maintenance and evolution cost increasingly rise to unbearable levels, but they still deliver great and valuable benefits for companies [12]. An alternative to the aforementioned problem is software reengineering. Nowadays, we have a more updated methodology to deal with software reengineering that is Architecture-Driven Modernization (ADM), where the software goes through a reengineering process by using models as can be seen in Model Driven Architecture (MDA). ADM was started in 2003 by the ADM-Task Force (ADMTF), an OMG initiative to use MDA concepts in the legacy software modernization context. The ADM's main goals are to provide a set of standard metamodels to support software modernization and to support software engineers while performing modernization projects by means of enabling the creation of reusable and interoperable modernization tools (MTs) [3]. MTs support the automation of at least one of the modernization tasks by following ADM concepts [10].

Among the metamodels proposed by OMG are both Knowledge Discovery Metamodel (KDM), and the Structured Metrics Metamodel (SMM). By Means of KDM it is possible to represent all software artifacts present in a legacy system, such as source code, graphic user interface (GUI), databases, configurations files, architecture and higher level abstractions [7]. On the other hand, The SMM Metamodel can be used to represent both metrics and metrics results and its advantage is to facilitate the reuse of metrics definitions, tool interoperability and sharing of metrics results that helps in the process of software quality assurance while using models/metamodels instances. KDM and SMM are platform and language independent so modernization tools that consider them can be reused in different modernization projects.

In a modernization process, normally, there are the "as-is" and the "to-be" software, e.g. the legacy and the target software. Performing measurements is important to plan and estimate how to deal with the modernizations refactorings. The Aspect-Oriented Programming (AOP) is an existing alternative when the goal is to conduct a modernization process to deal with crosscutting concern modularization. Crosscutting concerns are software concerns which could be scattered and tangled over the legacy software code, like persistence, logging, security and others. Thus, this research deals with metrics and a measurement approach to collect data about crosscutting concerns in a modernization process that uses ADM approach. To do this, KDM models need to be annotated/marked by concern mining tools, like Crosscutting Concern KDM (CCKDM) tool [8]. The main goal of this research is to measure the legacy software searching for relevant information about the diffusion of the crosscutting concerns and also to measure the target software, to investigate if the target software quality is in conformance with the planning.

As a motivation, we can mention the scarcity of approaches for measuring KDM models using SMM models [4], that is, the usage of existing metrics translated to be used in ADM context by means of SMM. Another motivation was the feasibility study of the definition and computation of concern measures in SMM instances to be applied in KDM instances.

To achieve the main goal of this research, we used the approach of Santos et al. that enables the representation of aspect-oriented systems in KDM models [11]. Once we could represent both the legacy and the target software in AOP we used concern metrics to measure them and collect important information to the software engineer. This process is performed with help of a modernization tool that is able to measure KDM and/or KDM-AO (Aspect-Oriented KDM) models and generate metrics results. To perform the measurements, we used two known metrics; Concern Diffusion over Operations (CDO) and Concern Diffusion over Components (CDC) [9].

An important step of the approach here presented is the implementation of a parameterization system for the users in order to set which crosscutting concerns are going to be measured, for instance: Persistence, Logging, Security, and others. Another important point is to understand how these concerns had been annotated by the previously used concern mining tool, since a KDM model could be marked/annotated with tags that might look like "concern=persistence" or "cc=persistence", thus, this parameters need to be informed [8]. All the process, measurements and tools used in this paper are detailed in the following sections.

Summing up, the goal of this paper is to study the feasibility of the concern measurements in the ADM context and to define a tool support to apply concern measures over KDM Metamodel instances. To achieve this goal, we performed the following activities: Reuse an approach that could represent AOP in KDM models, define a SMM Metrics Library that represents concern measures, and implement a tool for applying SMM metrics in KDM and aspect-oriented KDM (KDM-AO) instances.

The main contribution of this paper is a research about concern measurement and its feasibility in the ADM context, a SMM Concern Metrics Library, and a tool that applies the quality metrics approach in KDM and KDM-AO instances.

This paper is organized as follow: Section 2 describes the background about ADM and its metamodels (SMM and KDM). In Section 3, Modularity-Oriented Modernization is presented. Section 4 describes the specification Concern Metrics for ADM. Section 5 shows the Tool Support. Section 6 presents an Execution Engine and Metrics Library case study. Section 7 presents related works. Finally, Section 8 presents the conclusion about this research.

2 Architecture-Driven Modernization

Architecture-Driven Modernization (ADM) is an approach where the software goes through a reengineering process using models. The main ADM goal is to develop a set of standardized metamodels to represent all the information existing in a modernization process. KDM and SMM are two of a set of standard metamodels of ADM.

According to the OMG the most important artifact provided by ADM is the KDM metamodel, which is a multipurpose standard metamodel that represents all aspects of the existing information technology architectures. The idea behind the standard KDM is that the community starts to create parsers from different languages to KDM. As a result everything that takes KDM as input can be considered platform and language-independent. For example, a refactoring catalogue for KDM can be used for refactoring systems implemented in different languages [11]. The current version of the KDM is 1.4 and it is being adopted by ISO as ISO/IEC 19506 [7].

SMM is the ADM Metamodel responsible to provide metaclasses to define metrics. A SMM Model could represent a metric definition or a measurement result. Because SMM is a standard, metrics definition and measurements results can be exchange between engineers and can be reused in several projects. The main SMM terms are: measures (metrics) and measurements (models that contains metrics application results). A SMM Measure Model is a model that contains metrics definitions. A SMM Measurement Model is a model that contains the results of a specific measurement instance. To apply SMM Measure Model into KDM Model, we need a tool that receives as input: a XMI File that contains SMM Measures and XMI (one or more) Files that contains KDM instances

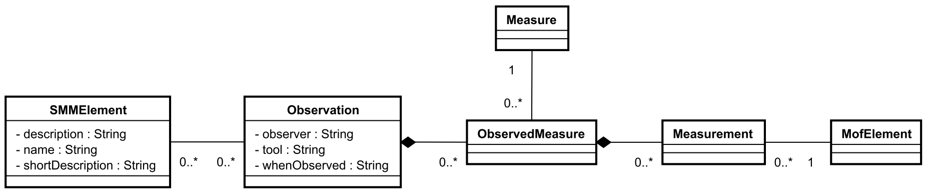


Fig. 1. The Approach of the SMM Metamodel

To work with SMM metamodel you need to instantiate some important SMM metaclasses, as you can see in Figure 1, the minimum requirement to use SMM in a modernization process is to instantiate a Measure, an Observation (Metaclass that represents measurements information, like data, engineer name and tool name), and a measurement, that will be performed in KDM models or any model based on meta object facility OMG metamodel. All metrics instantiation have some attributes: Name, Description, Scope, Characteristic, Operations, etc. Scope is the name of the Metaclass that will be measured. Characteristic is for example: Size, when the goal of the measure is to calculate index related to system size and Operation contains the operation that will be applied in the target model to collect indexes, for example, an OCL Code or XPath code.

3 Modularity-Oriented Modernization (MOM)

KDM-AO is a heavyweight extension for KDM that contains new metaclasses, like AspectUnit, AdviceUnit and PointCutUnit [11]. These and others new metaclasses that were introduced into the KDM Metamodel allow the creation of instances of AOP Software. Thus, it is possible to use this programming paradigm in ADM.

Figure 2 presents the modernization scenario, entitled as Modularity-Oriented Modernization. The main goal is to convert a legacy system (**A**), which contains

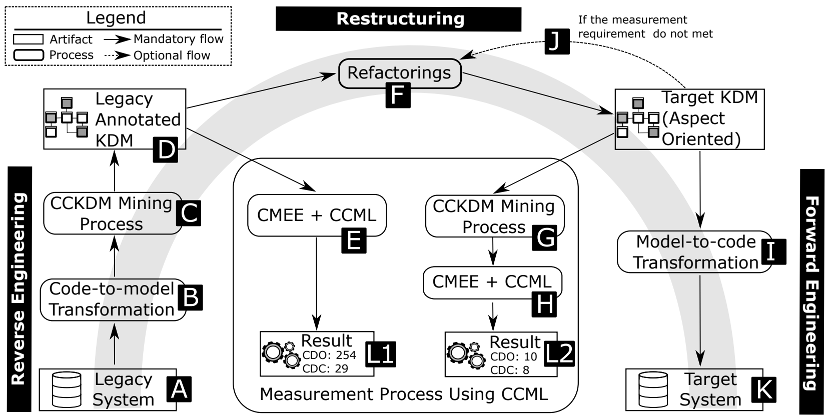


Fig. 2. Modularity-Oriented Modernization Scenario (MOM)

bad modularized concerns, into a new aspect-oriented version (**K**) where the concerns are then well modularized by using proper abstractions.

In order to perform this modernization process, two requirements must be met. The first one is to have a way of making the concerns evident in the KDM models representing the legacy system. That means pinpoint which model elements (classes, methods, attributes, etc) are affected or contribute to the implementation of given concerns - this process is known as concern mining [8]. To make them evident, we can use annotations, stereotypes, etc. The second requirement is to have a way of representing aspect-oriented concepts in KDM models. This is necessary for the target model, as it should be able to represent a solution for the crosscutting problem. There are basically two ways of doing that, by employing lightweight extensions or heavyweight ones. In our case, we opted for reusing a heavyweight aspect-oriented KDM extension [11].

Once these two requirements are met, we are able to conduct the whole process. That is, firstly we can apply a measurement process in the legacy annotated KDM to figure out the current modularity problems. Secondly, once we have an aspect-oriented KDM instance representing the target system we can mine it again, annotating and measuring it in order to check whether the modularity problems were solved.

In Figure 2, this process is explained in more details. It starts with the automatic transformation of a legacy source code into a KDM instance (**B**). This process is performed automatically by MODISCO [1]. As the Legacy KDM contains information about the legacy system, it's necessary to discover what components, operations and attributes contribute to the implementation of a concern. To carry out this mining process (**C**), we use CCKDM (Concern Mining KDM) [8], a mining tool that annotates KDM model elements that contribute to the implementation of certain concerns. Listing 1.1 shows an example of an annotated KDM model where the annotation is highlighted.

```
<codeElement  concern="Persistence"  xsi:type="code:MethodUnit" name="connectionDB"...>
.    <attribute %tag="export" value="private"/>
.    <source language="java">
.        <region file="/0/@model.2/@inventoryElement.16" language="java"/>
.    </source>
.    <codeElement xsi:type="code:Signature" name="ConnectionDB">
.    .    <parameterUnit type="/0/@model.o/@codeElement.0/@codeElement.0/@codeEle...">
.    .        <source language="java">
.    .            <region language="java"/>
.    .        </source>
.    .    </parameterUnit.
.    </codeElement>
...
</codeElement>
```

Listing 1.1. Snippet of an annotated KDM model

In this case, CCKDM identified a MethodUnit that is affected by the persistence concern. Notice that the way CCKDM adopts for annotating the models is by using the annotation “concern”. In this example, there is a MethodUnit annotated with <concern=“Persistence”>. Therefore, since the models are annotated we can measure based on these annotations and answer questions as “How many operations implement something about persistence?”.

After the code-to-model transformation (**B**), the CCKDM mining (**C**) produces a Legacy Annotated KDM (**D**) to be measured by Concern Metrics Execution Engine (CMEE) and Crosscutting Concern Metrics Library (CCML) approaches (**E**). CMEE is an eclipse plug-in for applying metrics in a KDM instance. The metrics must be specified by using a SMM model.

Table 1 presents an example of crosscutting concern metrics library with the name of the measure, its scope and unit. CDO and CDC are described in the next section. Concern diffusions over LOC (CDLOC) is the number of transition points for each concern; transition points are points in the code where there is a “concern switch”. Disparity and Concentration metrics quantifies the closeness between program components (such as files or functions) and features.

Table 1. Crosscutting Concern Metrics Library (CCML)

CC_MetricsLibrary.smm (physical file)			
Model			
MeasureLibrary			
Measure	**Scope**	**Unit**	...
CDO	AdviceUnit/MethodUnit	Affected Operations	...
CDC	Aspect/Class/InterfaceUnit	Affected Components	...
CDLOC	SourceRegion	Affected SourceRegion	...
Disparity	Aspect/Class/InterfaceUnit	Affected Components	...
Concentration	Aspect/Class/InterfaceUnit	Affected Components	...
...	...	...	...

The result of this step is a set of metrics calculated (**L1**). Based on these results, the modernization engineer needs to decide what refactorings (**F**) should be applied in order to improve the crosscutting concerns modularization. Then, after the refactorings, the KDM-AO Models (since now they also represent aspect-oriented concepts) needs to be mined (**G**) and measured again (**H**), to collect new information about the modularization (**L2**). If the engineer agrees with the results, then the KDM-AO Models will be transformed into source-code (**I** and **K**). If the engineer is not satisfied with the results (**J**), the process returns to the step (**F**) again.

4 Specifying Concern Metrics for ADM

In this section the goal is to present how SMM can be used to specify concern metrics. We chosen CDO (Concern Diffusion over Operations) and CDC (Concern Diffusion over Components) metrics because they are specific to measure separation of concerns [9]. CDO purpose is to measure the diffusion of a concern over operations, like methods and advices. CDC measures the diffusion of a concern in the system's components, like Classes, Interfaces and Aspects. These two metrics give to the modernization engineer an overview about concern diffusion over the system. In the following, we will explain how to define these metrics in SMM models.

The fields that you need to fill in the SMM Editor to define the CDO Metric are the following: name, label, description, trait (what characteristic this metric measures?), Scope (what is the target of the metric?), label format, Categories, Operation (the code that will be applied in the target KDM Model), Unit (what the Unit is used in this metric? For example: meters, kilometers, kilo, etc).

The element Operation is one of the most important. This element contains the code that will be applied in the target KDM Model. Thus, it is very important to know how to develop an Operation. All Operation has a name, a description, a language (what language you will use to define the operation?) and body (the body of the operation, where you need to put the code). In this case, we decide to use XPath to write CDO and CDC Operation code. SMM Specification by OMG uses OCL in their examples. In this paper, the choice of XPath for implementing the Operation Code (rather than using OCL) has been made by the author because of their XPath expertise. XPath is a directory language that you can use to query KDM or any MOF-Based Model (A model that is based on the meta-metamodel MOF).

Table 2 depicts the body of CDO and CDC Operations. The goal of these codes is to query the target KDM Model in order to collect data about concern diffusion. CDO retrieves instances of methods and advices of a KDM model that were annotated with a specific tag name and tag value. Similarly, CDC retrieves instances of classes, interfaces and aspects of a KDM model that were annotated with a specific tag name and tag value. Note that tag names and tag values are parameters which can be settled through CMEE in order to capture a specific concern.

Table 2. CDO and CDC Operation Definition

Metric	Language	Body
CDO	XPath	//codeElement='code:MethodUnit' and [@tagName='tagValue'] or //codeElement='code:AdviceUnit and [@tagName='tagValue']
CDC	XPath	//codeElement='code:ClassUnit' and [@tagName='tagValue'] or //codeElement='code:InterfaceUnit and [@tagName='tagValue'] or //codeElement='code:AspectUnit and [@tagName='tagValue']

For instance, it is possible to set "tagName" and "tagValue" with "concern" and "Persistence" respectively, then CMEE will replace these parameters and apply the operation. Listing 1.2 shows the case for the CDO operation and Figure 3 depicts the Object Diagram that is an instance of KDM that represents CDO Metrics [9].

```
//codeElement="code:MethodUnit" AND [@concern="Persistence"] OR
//codeElement="code:AdviceUnit" AND [@concern="Persistence"]
```

Listing 1.2. Replaced parameters CDO Operation

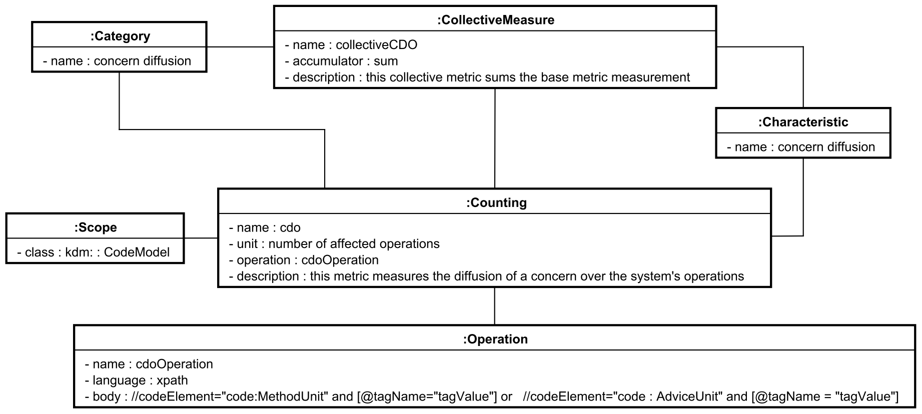

:CollectiveMeasure
- name : collectiveCDO
- accumulator : sum
- description : this collective metric sums the base metric measurement
:Category
- name : concern diffusion
:Characteristic
- name : concern diffusion
:Counting
- name : cdo
- unit : number of affected operations
- operation : cdoOperation
- description : this metric measures the diffusion of a concern over the system's operations
:Scope
- class : kdm: : CodeModel
:Operation
- name : cdoOperation
- language : xpath
- body : //codeElement="code:MethodUnit" and [@tagName="tagValue"] or //codeElement="code : AdviceUnit" and [@tagName = "tagValue"]

Fig. 3. CDO Object Diagram

In the Object Diagram we have instantiated some important metaclasses, like *Counting*. *Counting* is a specialized measure class that serves to define metrics for counting, such as the number of operations affected by some specific crosscutting concern. If we decide to measure more than one target model, we could instantiate *CollectiveMeasure* metaclass, that applies an operation (sum, average, maximum value, minimum value, etc) to a Counting measurement results.

It is not always possible to define concern metrics faithfully using the ADM metamodels. This is the case of the CDLOC [9]. This metric measures the implementation alternation of concerns between lines of code, but in KDM, there is no element that represents a line of code, but there is an element that represent a region of the source code.

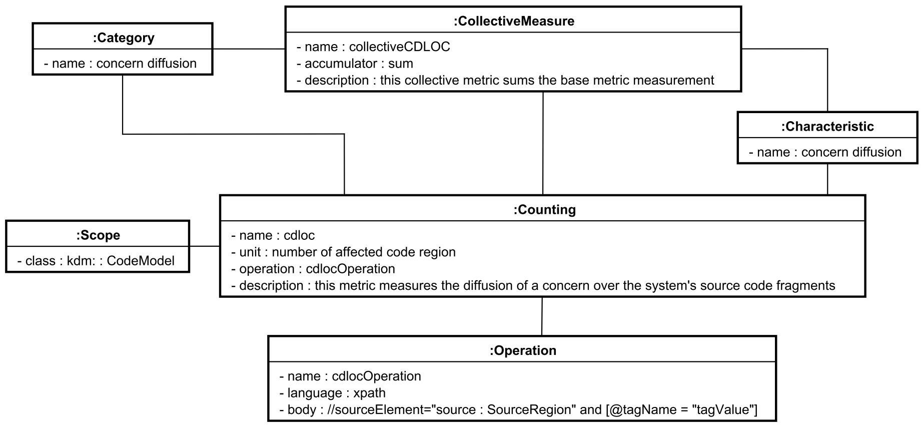

:Category
- name : concern diffusion
:CollectiveMeasure
- name : collectiveCDLOC
- accumulator : sum
- description : this collective metric sums the base metric measurement
:Characteristic
- name : concern diffusion
:Scope
- class : kdm: : CodeModel
:Counting
- name : cdloc
- unit : number of affected code region
- operation : cdlocOperation
- description : this metric measures the diffusion of a concern over the system's source code fragments
:Operation
- name : cdlocOperation
- language : xpath
- body : //sourceElement="source : SourceRegion" and [@tagName = "tagValue"]

Fig. 4. CDLOC Object Diagram

Thus, it was necessary to adapt this metric to measure SourceRegion elements instead of Lines of Code. This adaptation can be seen in Figure 4, where we can see that the code makes mention for an Operation element that searches for SourceRegion elements. To define our Concern Metrics Library we have used MoDisco SMM Editor. This editor is part of MoDisco Framework [2] and allows creating SMM metrics models using graphic elements. The library structure is shown in the Table 1.

In Table 1 it is possible to see the structure of our metrics library. This library was developed with the objective to group a set of concern metrics. The library in contained inside the CC_MetricsLibrary.smm physical file, and contains the following elements: Model (the main element, that represents a SMM Model), Library (our concern metrics library), Measures (organized by Category, Characteristic or Scope), Operations (contains the OCL or XPath or XQuery code that will be applied in the target KDM Model). All these elements are useful when the engineer wants to search metrics or measurement results in a repository.

5 Tool Support

In this section we show how CMEE (Concern Metrics Execution Engine) works. To help in the understanding process, we have created an UML Activity Diagram with the flow to be executed that can be seen in Figure 5. The flow starts when the Modernization Engineer (User) opens the CMEE Plugin on Eclipse IDE. In the first action, User needs to choose the .SMM Model file and one or more

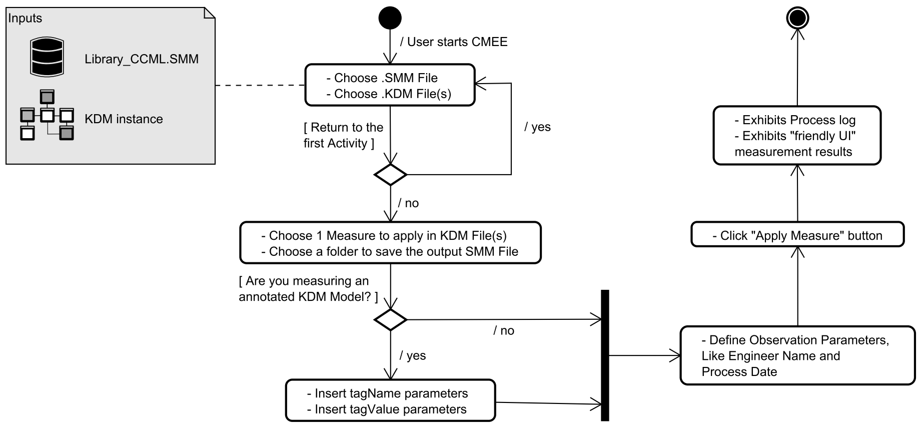

Inputs
Library_CCML.SMM
KDM instance
/ User starts CMEE
- Choose .SMM File
- Choose .KDM File(s)
[Return to the first Activity]
/ yes
/ no
- Choose 1 Measure to apply in KDM File(s)
- Choose a folder to save the output SMM File
[Are you measuring an annotated KDM Model?]
/ no
/ yes
- Insert tagName parameters
- Insert tagValue parameters
- Define Observation Parameters, Like Engineer Name and Process Date
- Click "Apply Measure" button
- Exhibits Process log
- Exhibits "friendly UI" measurement results

Fig. 5. Concern Metrics Execution Engine – CMEE Activity Diagram

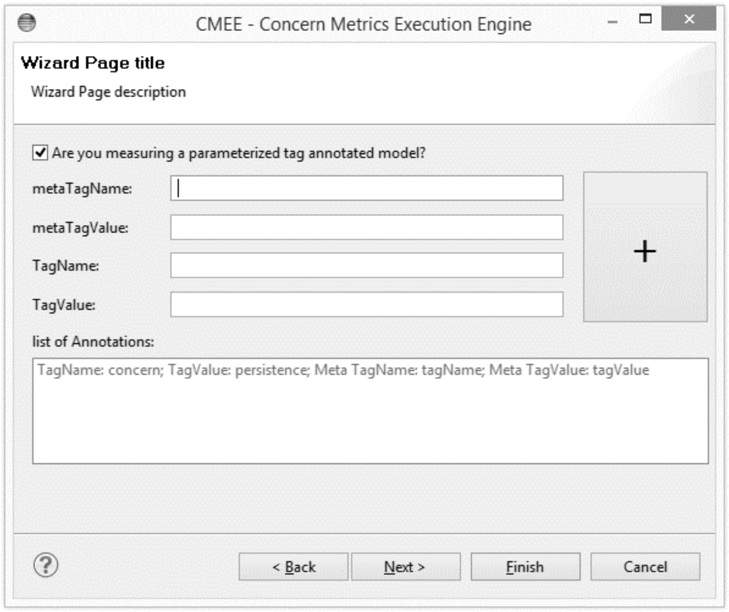
.KDM Model. If any of these models is not well-formed CMEE will detect and return to the first phase. Since all models are ready, the next step is choosing one measure to apply on KDM Model and choosing a directory to save the measurement results file.

Then, User need to inform if the target KDM Model is an Annotated model (previously mining) or just a KDM Model, without annotations (See Figure 6). If the KDM Model contains Annotations, it will be necessary to inform the parameters and CMEE will replace this information on the Operation Code, for instance: if the user had used a concern mining tool that annotated the KDM models with the following pattern "concern=concern_name" and the user is also using our parameterized metrics, it will be necessary to inform CMEE that the meta tag name is "tagName", the meta tag value is "tagValue", then, it will necessary to inform the tag name value, like "concern" and the concern name for tag value, like "persistence". The CMEE allow the inclusion of multiple patterns of annotations simply by clicking on the symbol with the add button.

Finally, the user needs to provide some information about the measuring process, like engineer name and date, then click on the button "Apply Measure" and see the results.

6 Execution Engine and Metrics Library Case Study

To validate and exemplify the use of the CMEE, we decide to measure KDM models of a system named CD Store. This system was written in Java and serves to manage a store that sells CD/DVD. This case study has two parts. In the first part, we have applied the CDO and CDC metrics [9] on the legacy CD Store KDM Models and, in the second part, we have applied CDO and CDC Metrics in KDM-AO modernized models.


CMEE - Concern Metrics Execution Engine

Wizard Page title

Wizard Page description

[✓] Are you measuring a parameterized tag annotated model?

metaTagName:

metaTagValue:

TagName:

TagValue:

list of Annotations:

TagName: concern; TagValue: persistence; Meta TagName: tagName; Meta TagValue: tagValue

Fig. 6. CMEE Screen – Tag parametrization

Legacy CD Store uses an Object-Oriented Framework to persist data on database. The problem is that a big number of classes in the application implements actions related to persistence. Therefore, the modernization aims to change this OO Persistence Framework for an AO Persistence Framework, in order to remove persistence implementation from the base application.

To discover where the crosscutting concern "persistence" is, we have used the CCKDM Tool [8] to mining concerns in our CD Store KDM and KDM-AO Models. Then we use our tool CMEE to measure the models using CDO and CDC. The CDO and CDC Indexes can be seen in Table 3. Note that before the modularity-oriented modernization process, the Base Application CD Store had a lot of persistence code in its classes and after these numbers decrease a lot, leaving only the Persistence AO Framework implementing the persistence of the application.

Table 3. Case Study Measurement Results - CDO and CDC Indexes Before and After Modernization

Application	Metric Application	CDO	CDC
Base App	Before	13	13
	After	1	1
Persistence Framework	Before	87	14
	After	87	19

This kind of measurement helps the engineer to discover if the ADM modernization process was effective or not. In this case, the base app CD Store has been modernized successfully, because the diffusion of the concern “persistence” decreases from 13 to 1. But, if the engineer is not satisfied with the indexes, he can reapply the refactorings and then measure again. Notice that the Persistence Framework did not have a significant change since its purpose was to perform the persistence of the application.

7 Related Works

The oldest approach found was METRINO [6], a tool that works in four phases. In the first, named “Rule Management”, the user could define a rule or use an existing rule. In the second phase, named “Measure Generation”, metric models are generated based on the previously defined rules. In the third phase, named “Measure Management”, it is possible to change the automatically generated metric models. Then, in the last phase, named “Measure Evaluation”, METRINO applies the selected metrics (selected by the user) on the MOF-based target models.

Another related approach is Gra2Mol Motor [3]. The mechanism Gra2Mol receives as input metrics defined in a DSL (Domain Specific Language) named Medea and KDM files to apply the metrics. The tool has a translator that transforms Medea code in SMM Measures and then apply the selected measures in the KDM Model that represents the system.

And the last related approach found was MAMBA Framework (Measurement Architecture for Model-Based Analysis) [5]. MAMBA could receive both SMM and MDL Files. MDL (metrics definition language) is a DSL developed by MAMBA Team to facilitate the process of creating SMM Metrics files. As with other approaches, it is necessary to put a KDM File to measure and then the MAMBA Execution Engine starts the measurement process and generates an output SMM File.

It’s important to emphasize that none of the related works can interpret aspect-oriented metrics and parameterized concern metrics.

8 Conclusion

By means of this research it is fairly evident that our approach fills a gap in the ADM context. Now, it is possible to measure Aspect-Oriented Software represented in KDM Models using Concern Metrics, like CDO and CDC.

We argue this approach is useful in modernization scenarios to improve the modularization of software systems, for instance, a modernization process that applies refactorings in order to modularize the persistence code which is scattered and tangled over operations, components and/or lines of code.

With our approach the Modernization Engineer could use SMM models with XPath Operations Code to collect data about crosscutting concerns and other metrics. Note that KDM model measurements is different to measurements in the source code. KDM models are versatile and can have multiple levels of abstraction, but cannot represent lines of code, for example.

We can also mention that measure crosscutting concerns is not something native on KDM, so it had to be extended (KDM-AO) and annotated by a crosscutting concern mining tool and then it could be measured.

This type of measurement (concern measurements) requires that the Operations of SMM metrics to be dynamic enough so that the user can specify which concern is required to be measured and how the mining tool annotate KDM models, that is, which annotation pattern the mining tool uses.

The CMEE tool supports multiple sets of parameters and can accept even one, two or more patterns of annotations in KDM target model. Thus, as a limitation for this research we can mention the need for prior concern mining of target models. To measure crosscutting concerns it is necessary to know where they are implemented. Therefore, mining techniques and tools should be used mandatory.

The dependence of the extension KDM-AO for representing aspect oriented software systems is another limitation for the reuse of metrics since our predefined parameterized metrics mention some terms like “AspectUnit" and “AdviceUnit" that is a particularity of our extension. Another limitation is the dependence of MoDisco tool to represent java source code, which so far does not have support for all KDM packages.

As a future work, we will perform a more in-depth evaluation of the approach and tool, involving real systems developed in different programming languages and software engineers from industry to evaluate whether the SMM instances are generic enough to measure different instances. Another future work is to enrich the SMM metrics library with other quality metrics and also to improve the output results in order to provide useful reports that could help software engineers in the decision making process.

9 Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and by FAPESP process number 2016/03104-0. We also would like to thank CONICYT PFCHA/-DOCTORADO BECAS CHILE/2016 - 72170024.

References

1. Bruneliere, H., Cabot, J., Jouault, F., Madiot, F.: Modisco: A generic and extensible framework for model driven reverse engineering. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. pp. 173–174. ASE ’10, ACM, New York, NY, USA (2010). https://doi.org/10.1145/1858996.1859032

2. Brunelière, H., Cabot, J., Dupé, G., Madiot, F.: Modisco: A model driven reverse engineering framework. Information and Software Technology **56**(8), 1012 – 1032 (2014). https://doi.org/10.1016/j.infsof.2014.04.007
3. Canovas Izquierdo, J., Zapata, B., Molina, J.: Definición y ejecución de métricas en el contexto de adm. In: Taller sobre Desarrollo de Software Dirigido por Modelos (DSDM). pp. 1–10 (2009)
4. Durelli, R.S., Santibáñez, D.S.M., Marinho, B., Honda, R., Delamaro, M.E., Anquetil, N., Camargo, V.V.: A mapping study on architecture-driven modernization. In: Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014). pp. 577–584 (Aug 2014). https://doi.org/10.1109/IRI.2014.7051941
5. Frey, S., Hoorn, A., Jung, R., Kiel, B., Hasselbring, W.: Mamba: Model-based software analysis utilizing omg's smm. Softwaretechnik-Trends **32**(2), 49–50 (2012)
6. Hein, C., Engelhardt, M., Ritter, T., Wagner, M.: Generation of formal model metrics for MOF based domain specific languages. ECEASST **24** (2009). https://doi.org/10.14279/tuj.eceasst.24.339
7. Pérez-Castillo, R., de Guzmán, I.G.R., Piattini, M.: Knowledge discovery metamodel-iso/iec 19506: A standard to modernize legacy systems. Computer Standards & Interfaces **33**(6), 519 – 532 (2011). https://doi.org/10.1016/j.csi.2011.02.007
8. San Martín Santibáñez, D., Durelli, R.S., Camargo, V.V.: A combined approach for concern identification in kdm models. Journal of the Brazilian Computer Society **21**(1), 10 (Aug 2015). https://doi.org/10.1186/s13173-015-0030-3
9. Sant'anna, C., Garcia, A., Chavez, C., Lucena, C., v. von Staa, A.: On the reuse and maintenance of aspect-oriented software: An assessment framework. In: Proceedings XVII Brazilian Symposium on Software Engineering (2003), http://twiki.im.ufba.br/pub/Aside/NossasPublicacoes/sbes2003-135.PDF
10. Santos, B.M., de Guzmán, I.G., Camargo, V.V., Piattini, M., Ebert, C.: Software refactoring for system modernization. IEEE Software **35**(6), 62–67 (November 2018). https://doi.org/10.1109/MS.2018.4321236
11. Santos, B.M., de S. Landi, A., Santibáñez, D.S., Durelli, R.S., Camargo, V.V.: Evaluating the extension mechanisms of the knowledge discovery metamodel for aspect-oriented modernizations. Journal of Systems and Software **149**, 285 – 304 (2019). https://doi.org/10.1016/j.jss.2018.12.011
12. Visaggio, G.: Ageing of a data-intensive legacy system: symptoms and remedies. Journal of Software Maintenance and Evolution: Research and Practice **13**(5), 281–308 (2001). https://doi.org/10.1002/smr.234