

Metrics for consistency checking in object oriented model transformations

G Ramesh^{1*}, T V Rajini Kanth², A Ananda Rao¹

¹CSE Department, JNT University Anantapur Ananthapuramu, Andhra Pradesh, India

²CSE Department, Sreenidhi Institute of Science and Technology, Ghatkesar Hyderabad, Telangana, India

*Corresponding author's e-mail: ramesh680@gmail.com

Received 29 March 2017, www.cmnt.lv

Abstract

Model transformation is the cornerstone of Model-Driven Engineering (MDE) as it is crucial in Computer Aided Software Engineering (CASE) towards Object Oriented Analysis and Design (OOAD) and Object Oriented Programming (OOP). It also plays vital role in entity relationship model. Therefore it is indispensable to be treated as traditional software artefacts and assess quality of model transformations. Model-to-model transformations are from Platform Independent Model (PIM I) to Platform Independent Model (PIM II) and from PIM to Platform Specific Model (PSM). The goal of our research in this paper is to make these model transformations measurable. However, it is confined to proposing a set of metrics pertaining to consistency checking. The quality of transformations is measured in terms of consistency. The metrics proposed in this paper are general and can be reused. We evaluate the metrics using our framework named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) which supports end-to-end transformations of object oriented models. Our empirical study revealed that the proposed metrics add value to our model consistency checker as they quality in model transformations.

Keywords:

Model Driven Engineering (MDE), XRTSDIC, model transformations, consistency checking, quality measures

1 Introduction

Model Driven Approach (MDA) is an important alternative for developing information systems. The underlying principle of this approach is defining abstract models that can be used for implementations. Unified Modelling Language (UML) is widely used to model information systems that are built in object oriented approach. As part of Model Driven Engineering (MDE) design and exploitation of domain models became important in software development. The conceptual models can help understand development process quickly besides ensuring that the productivity is more with Computer Aided Software Engineering (CASE) tools. Model transformations can be part of CASE tool. This research is our ongoing work on consistency checking in model transformations. This paper focuses on deriving metrics for checking consistency of model transformations.

Our prior works [1-5] provide a series of related research efforts in realizing a framework that supports end-to-end approach for model transformations besides detecting and tracking software design inconsistencies. In [1] we defined a framework named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) which checks model inconsistencies and provide feedback dynamically. The framework is flexible and extensible. It has placeholders for future methods besides having personalized configuration and execution models. In [2] explores the realization of the framework proposed in [1] with consistency rules, provision for tolerance of inconsistencies to support notion of "living with inconsistencies" in the form of a prototype application.

In [3] we improved the framework with rule detector algorithm, consistency checker algorithm, and visualization algorithm. In [4] our framework is further enhanced and evaluated with end-to-end model transformations from Platform Independent Model (PIM) to Platform Specific Model (PSM) often with intermediate PIMs. It focused on class diagram transformation rules, Entity Relationship Diagram (ERD) transformation rules, handling issues with class relationships, and case study to evaluate the work. In [5] the framework is evaluated with UML class diagram to source code of different object oriented languages.

Our contributions in this paper include derivation of metrics for consistency checking of object oriented models and integrating with our framework XRTSDIC to leverage its utility further. A case study is provided to evaluate the framework with the metrics derived. The remainder of the paper is structured as follows. Section II provides review of literature. Section III presents the proposed system in detail. Section IV presents quality attributes. Section V presents the proposed metrics. Section VI shows Evolution methodology and experimental results while section VII concludes the paper.

2 Related works

This sections reviews related works. The reviewed content is categorized into model transformations and metrics used to measure quality of model transformations.

2.1 MODEL TRANSFORMATIONS

Kuzniarz *et al.* (2003) [16] focused on consistency issues in

UML-based software design models. They proposed consistency rules for different transformation models. It has mechanisms for finding inconsistencies in the design models made of UML. Hutchison *et al.* (2009) [8] focused on model-driven software engineering for self-adaptive systems. Paredis *et al.* (2010) [6] model transformations between two languages that are complement to each other. They are known as Modelica and SysML from OMG. SysML is a generalized modelling language while Modelica for analyzing systems with discrete time dynamics. The transformation between them is bi-directional. In [7] Model Driven Interoperability is focused for achieving interoperability transformations in distributed environments.

Biehl *et al.* (2010) [19] made a good review of model transformations. They explored many transformation approaches such as graph-based, template-based, and hybrid approaches besides presenting model transformation languages such as EMF Henshin, ATL, Query/View/Transformation (QVT), SmartQVT, ETL, XSLT and ModelMorf. They opined that synthesis and integration are the two advantages of model transformations broadly. Kessentini *et al.* (2012) [9] focused on search-based model transformation with example. Model Transformation (MT) became very important activity in software engineering as it is supported by Computer Aided Software Engineering (CASE) tools. They proposed an approach that is independent of source and destination formalisms and works for any source model. Model Transformation By Example (MTBE) is the main focus of them. However, they explored different methods for transformation including model transformation based on search.

Rodriguez *et al.* (2010) [10] proposed a method for semi-formal transformation a business process into use case and class diagrams of UML by adapting MDA. They focused on security aspects in the modelling. Towards this end they defined transformation rules to transform business process into class and use case diagrams. Their semi-automated approach could obtain useful artefacts of information systems. Hermann *et al.* (2010) [12] employed triple graph grammars (TGG) for efficient model transformations. Bi-directional model transformations are possible with well known Triple Graph Grammars. Towards this end, they employed Negative Application Conditions (NAC) as well. NACs can improve model transformation specifications. Garcia *et al.* (2012) [18] introduced a semi-automatic process that takes care of model transformation co-evolution. It has two phases namely detection phase and co-evolution phase. The former takes care of detects changes to metamodel while the latter takes care of performing required actions to complete co-evolution process.

2.2 METRICS

Chidamber and Kemerer (1994) [13] focused on a suite of metrics that can be used for improving object oriented design (OOD). Hutchinson *et al.* [14] provided an approach for assessing MDE. Generally MDE promotes software development with advantages such as interoperability, maintainability, portability and productivity. The maturity of MDE is assessed with automation. The degree of code generation is from 65% to 100%. Amstel and Brand (n.d) [22] studied model transformations made using ATL. They assessed quality of transformations using metrics. They classified metrics into different categories. They are rule

metrics, helper metrics, dependency metrics and miscellaneous metrics. They concluded that metrics alone are not adequate to assess quality of model transformations. Moreover those metrics are to be associated with quality attributes so as to relate with quality model of transformations. The quality assessment provided by their metrics and manual assessment is compared to know the error rate in quality assessment of chosen model transformations.

Amstel *et al.* (2008) [25] studied possible measures for quality transformations. They proposed many consistency related metrics such as number of code clones, number of unused variables, number of different types per variable name, and different variable names per type. Kapova *et al.* (n.d) [23] explored code metrics on model-to-model transformations for evaluating maintainability. They used automated metrics such as transformation size metrics, relational metrics, consistency metric and inheritance metrics. Apart from these metrics, they employed manually gathered metrics such as similarity of relations, and number of relations that follow certain design pattern. The computation of metrics is made using QVT transformations and metrics support availability.

Vignaga (2009) [24] applied metrics to measure ATL model transformations. There are many quality metrics such as conciseness, consistency, completeness, modularity, reuse, reusability, modifiability and understandability. The unit metrics available with ATL include Number of Imported Libraries (NIL), Total Number of Imported Libraries (TIL), Number of Helpers (NH), Number of Helpers without Parameters (NHP), Balance of a Unit (BOU) and Number of Helpers per Context (NHC). Other metrics available are categorized into module metrics, library metrics, rule metrics, matched rule metrics, lazy matched metrics, called matched rule metrics, and helper metrics.

Testing model transformations is an important activity in MDA. Baudry *et al.* (2010) [11] identified barriers to systematic testing of model transformations. They considered model transformation example of converting hierarchical state machine to flattened state machine the hierarchical state machine has many incoming and outgoing transmission. The states are of many types namely simple states, initial states and final states. Apart from these, composite states are also available. The barriers identified for model transformations include heterogeneity of transformation languages, lack of tools for model management, and complexity of inputs and outputs.

Kessentini *et al.* (2011) [20] focused on model transformation testing using two steps known as selection of test cases and finding test oracle functions. Their approach also focuses on finding the risk of detected faulty candidates and sorts them in the order of risk. They used immune system metaphor of biological science in order to achieve this. They defined precision and recall measures to evaluate the transformations. Pean (2012) focused on change metrics to measure incrementally built model transformations. They defined language feature metrics and change metrics based on abstract syntax difference model.

Arendt and Taentzer (2013) [21] used Eclipse Modelling Framework and explored it for quality assurance. They employed 6C goals such as correctness, completeness, consistency, comprehensibility, confinement, and changeability. They explored project specific quality

assurance techniques. Therefore it is made possible to specify such techniques based on the need of the project. Their specification supports model smells detection using metrics and anti-patterns. They also employed Domain Specific Modelling Language (DSML) known as SimpleClassModel (SCM) for demonstrating quality assurance of models.

Chitra and Sherly (2016) [15] used graph based models for verification of software design models. The process of model verification is used for observing behaviour preservation. With verification it is possible to have refactoring. Here graph isomorphism is the property utilized for model verification. Rosenberg and Hyatt (n.d) discussed software quality metrics for systems built on object orientation. Then they evaluated metrics using certain criteria such as testability, maintainability, reusability, understandability, complexity and efficiency. The metrics covered by them include cyclomatic complexity, size, comment percentage, weighted methods per class, response for a class, lack of cohesion of methods, coupling between object classes, depth of inheritance tree, and number of children. Amstel *et al.* (n.d) [26] proposed metrics for assessing ASF+SDF model transformations. Their metrics are related to different quality attributes such as understandability, modularity, modifiability, reusability, completeness, and consistency.

3 Our framework: XRTSDIC

Our framework defined in [1] is known as Extensible Real Time Software Design Inconsistency Checker (XRTSDIC). It is presented in Figure 1. It gives an overview of the generic approach for model inconsistency checking with provision for personalized configuration and execution model. The framework allows modelling tool selection, consistency rule language selection and visualization approach selection. These are pertaining to personalization which does mean that the models drawn by users are associated with such users and their configurations are retained.

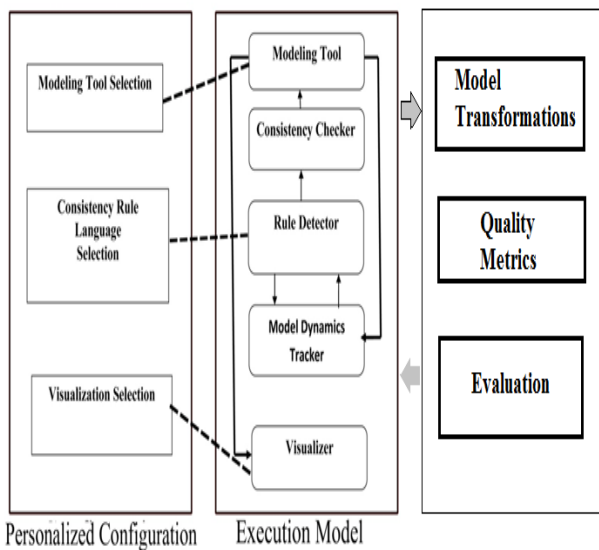


FIGURE 1 Overview of our Framework XRTSDIC

This framework was implemented in [1] and made further enhancements in [2-5]. In this paper we focused on improving

it further to facilitate measures for checking quality of model transformations. Quality attributes and consistency metrics are discussed in section 4 and 5. We considered only model consistency metrics that check the quality of model transformations. The proposed metrics are applied to a case study where model transformations are made from class diagram (PIM) to sequence diagram (PIM). And then the class diagram is transformed into source code (PSM). The metrics are useful to discover any inconsistencies in the way of model transformations from source to target. The source is reused number of times in model transformations as the same source is transformed into multiple targets.

The execution model of the framework helps developers to make use of a modelling tool to build models and then visualize any model inconsistencies. The tool also supports rectification of inconsistencies besides presenting them in chosen phenomenon. The execution model is based on the algorithm 1 presented here.

As shown in algorithm 1, the execution model pseudo code provides useful logic that helped in building the tool. The tool here is enhanced with proposed metrics presented in section 5. However, the consistency rules are taken from our previous work [3] where case study and evaluation of model inconsistencies are demonstrated. In this paper we focused on non only inconsistency checking but also measure quality of model transformations.

```

1 Initialize context vector C
2 Initialize rules vector R
3 Initialize model dynamics vector MD
4 Initialize inconsistencies vector INC
5 Do
6 IF MC is true THEN
7   Notify MDT
8 END IF
9 IF MDT has fresh notification THEN
10  MD = model dynamics
11  R = RuleDetector(MD)
12 END IF
13 IF r!=NULL THEN
14  INC = ConsistencyChecker(R, MD)
15  IF INC !=NULL THEN
16    Update C with INC data and metadata
17    Visualizer(C)
18  END IF
19 While(drawing==true)
    
```

Algorithm 1: Flow of Execution Model [3]

Class diagram to sequence diagram transformation rules

- Class Name → Instance of Class
- Consistency and transformation rule: If(new instance is created) then it should have a corresponding class in class diagram
- Class Method → Interaction in sequence diagram
- Consistency and transformation rule: The operation invoked by source should really exist in destination

Listing 1: Transformation Rules from Class Diagram to Sequence Diagram

Class diagram to source code transformation rules

Class diagram contains class name, attributes and methods. The class diagram is transformed to corresponding source code (classes) according to the object oriented language selected.

<p>Class Name → Class Name Consistency and transformation rule: if(a new class is created then) the class name should be unique and should be available in class diagram</p> <p>Class Attribute → Class Instance Variable Consistency and transformation rule: if(a new attribute is created then) the attribute name should be unique and should be available in the class attributes</p> <p>Class Attribute Type → Class Attribute Type Consistency and transformation rule: if(attribute type is determined then) the attribute type should match or compatible with that of class attribute</p> <p>Class Method → Class Method This method should match or compatible with that of class method.</p> <p>Class Method Arguments → Class Method Arguments The arguments in the generated classes should match arguments of method. However it is subject to the support in UML notation of class diagram.</p> <p>Class Method Return Type → Class Method Return Type The return type of method should have same or compatible type in generated class</p>

Listing 2: Consistency and transformation rules (Class Diagram → Source code)

These rules are applied when the transformation takes place. Again the generated source code is based on the functionality of corresponding dialect chosen. The dialect can provide accurate source code generation.

4 Quality attributes

With respect to model transformations, many quality attributes are identified. These quality attributes can be applied to many software artefacts. Particularly attributes that can be applied to model transformations are described here.

Understandability: This attribute refers to the amount of effort needed for user to understand model transformation. It also promotes reusability and modifiability. As understanding can help in modifications and reusability, it plays important role in model transformations. Model transformation is source→target model and its syntax and semantics are to be easy to understand.

Modifiability: Model transformations can be adapted to different context or altered to have additional functionalities. Changing requirements may force a model transformation to be modified. Another reason for the change is the language. When language needs to be changed, it warrants changes in model transformations. This attribute refers to the amount of effort required for alter model transformation in order to accommodate new requirements.

Reusability: It is the attribute that refers to the extent to which a model transformation or a part of it can be reused in other model transformations without making changes to the model being reused. Thus this attribute differs from modifiability attribute which causes modifications to model transformations. Especially, the reusability attribute comes into picture when a source is transformed to different target

and vice versa.

Reuse: It is somewhat related to reusability. However, it refers to the extent to which a model transformation is actually reused. It is best practice to reuse model transformations as much as possible instead of reinventing the wheel. Moreover MDE advocates reuse. Reuse in model transformations is common as source is common for many transformations. Therefore reuse can be considered as a measure which indicates how best a model can adhere to the principles of MDE.

Modularity: This attribute refers to the extent to which a given model transformation is built systematically. Systematic structure is essential to have modularity and every module in the model transformation should have its own purpose. Again modularity is pertaining to reusability. When functionality is repeated across modules, it is possible to reuse model or part of model transformations. Therefore the number of steps involved in the model transformation also can relate to modularity.

Completeness: This attribute refers to the extent to which model transformation is built fully. A model transformation is said to be complete when it has all parts of source model are completely transformed to target model according to specifications. In other words, the model transformation is made with all functionalities. An incomplete transformation results in target model which is not complete.

Consistency: This is the attribute which refers to extent to which a model transformation is without conflicts surfaced. According to Boehm [28] there are two kinds of consistencies. They are known as internal and external consistencies respectively. When uniform notation is maintained across the model transformation, it is known as internal consistency. It is often related to understandability. Internal inconsistencies can lead to target model in model transformations. External consistency refers to the extent to which model transformation adheres to given specifications.

Conciseness: This attribute refers to the extent to which model transformation has lack of superfluous information such as unused function parameters, code clones and so on.

5 Proposed metrics

This section provides metrics we have defined for measuring consistency and conciseness of model transformations. The main focus of these metrics is to measure consistency in model transformations in general. There are several measures possible. However, we like to define measures that are tool independent. Therefore the consistency measures defined by us are number of signatures with improper arguments (RSIA), number of unused variables (ROUV), number of code clones (NOCC), Population of Clone Class (POP), and Ratio of Non-Repeated Token Sequences (RNRS).

5.1 RATE OF SIGNATURES WITH IMPROPER ARGUMENTS (RSIA)

Model transformations from class to corresponding source code (PIM→PSM) of target language can exhibit inconsistencies. Every function modelled in the class diagram needs to be transformed into a function signature

with appropriate arguments. When it does not happen properly, it results into inconsistency. This kind of inconsistency is measured using NSIA.

$$RSIA = \text{FUN}_{\text{improper}}(A) / \text{FUN}_{\text{whole}}(A)$$

$\text{FUN}_{\text{improper}}(A)$ indicates the number of functions with improper signature and $\text{FUN}_{\text{whole}}(A)$ indicates all the functions that have been transformed. This measure is used to discover inconsistencies in model transformations in terms of number of functions containing improper signature. In other words it finds number of functions that are not consistency in terms of arguments.

5.2 RATE OF UNUSED VARIABLES (ROUV)

This measure is used to know the number of variables which are not used in the model transformations. The unused variables can affect conciseness quality.

$$\text{ROUV} = \text{VAR}_{\text{unused}} / \text{VAR}_{\text{all}}$$

Here $\text{VAR}_{\text{unused}}$ refers to the number of variables that are declared but not used in the application. VAR_{all} refers to all the variables declared in the application.

5.3 NUMBER OF CODE CLONES (NOCC)

Due to signatures with near similar arguments code repetition can occur in model transformations. Therefore this measure is relevant to know model inconsistencies.

$$\text{NOCC} = \text{COUNT}(\text{CC})$$

Where CC refers to code clones and the $\text{COUNT}(\text{CC})$ returns the number of code clones. Number of code clones or duplicate pairs of code is a good measure which may help to discovery model inconsistencies.

5.4 POPULATION OF CLONE CLASS (POP)

The number of clone elements in a clone is measured using POP. A clone class is a class that may contain at least one clone pair. Clone pair is two pieces of code that are identical. The increase in POP reflects increase in clones in system.

$$\text{POP} = \text{Elements}_{\text{clone}} / \text{Elements}_{\text{all}}$$

Here $\text{Elements}_{\text{clone}}$ refers to the count of elements in the code clones while the $\text{Elements}_{\text{all}}$ refers to the number of elements.

5.5 RATIO OF NON-REPEATED TOKEN SEQUENCES (RNRS)

Ratio of non-repeated token sequences refers to the ratio of non-repeated token sequences in a given clone set. Higher rate of RNRS indicates the presence of more non-repeated token sequences in code clone. This metric is computed as follows.

$$\text{RNR}(S) = \frac{\sum_{i=1}^n \text{LOS}_{\text{non-repeated}}(c_i)}{\sum_{i=1}^n \text{LOS}_{\text{while}}(c_i)}$$

$\text{LOS}_{\text{non-repeated}}(c_i)$ refers to length of the non-repeated

token sequence of code clone c_i . In the same fasion, $\text{LOS}_{\text{while}}(c_i)$ refers to while token sequence of code clone c_i . LOS stands for Length of token Sequence.

6 Evaluation methodology

The tool implemented by us is Extensible Real Time Software Design Inconsistency Checker (XRTSDIC). It is used to perform model transformations and consistency checking. For evaluating metrics discussed in this paper along with the performance of the tool, we invited five industry experts who are aware of software engineering and model transformations well. They spent their valuable time on our request to provide ground truth for the case study described in this paper. The ground truth is evaluated with the system generated values with respect to metrics that are used to evaluate the quality of model transformations done by the tool. An average is computed for the independent values given by the experts. The average values are considered to be the ground truth and used in comparison.

6.1 CASE STUDY AND RESULTS

Our framework XRTSDIC with prototype application is used to have a case study which helps in model transformations with consistency checking. Besides it helps in using the metrics presented in this paper to know quality of model transformations. UML class diagram is transformed into corresponding ERD. This process is done by using transformation and consistency rules. There is intermediate result in the form of XML file that encapsulates classes in the PIM. Then the class diagram is transformed into source code. The case study class diagram considered is related to Hospital Management System (HMS). The class diagram is as shown in Figure 2.

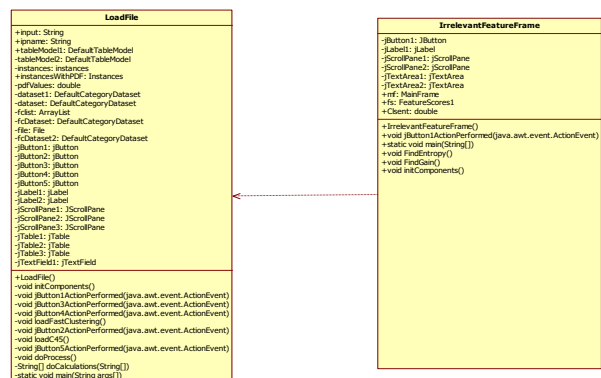


FIGURE 2 UML class diagram for FSC project case study (PIM I)

The class diagram is drawn using our framework. The model transformation is done with two experiments. In the first experiment, the class diagram is transformed into ERD. Then the class diagram is also transformed to source code using Java syntax and semantics. In either case, the model transformation rules and consistency rules are employed. The generated ERD is presented in Figure 3.

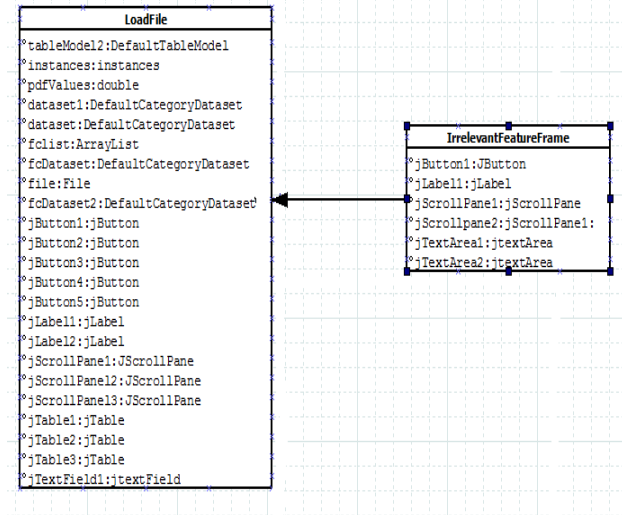


FIGURE 3 Transformed ERD (PIM II)

We made another empirical study on model transformations. The UML class diagram (PIM) is first of all transformed into another PIM model known as sequence diagram. Afterwards the class diagram is transformed into PSM known as source code of object oriented programming languages like C++, Java and C#. We proposed a Dialect hierarchy in Java language to handle transformation semantics for Java, C# and C++ [5]. The transformation dialect is a class that takes care of syntactical and semantic differences based on the target language chosen. The model transformation procedure and its flow are in our prior work [5]. In this paper our focus is more on checking quality of model transformations. Our enhanced framework XRTSDIC is used to apply metrics to measure quality of model transformations. Section 4 and 5 provided more details on quality attributes and proposed metrics for quality of consistency in model transformations. Following are the details of metrics applied to know the quality of transformations.

6.2 RESULTS AND DISCUSSION

We considered the case study pertaining to a data mining application named FSC (Feature Selection and Classification). Out of this project two important classes are considered for empirical study. LoadFile and IrrelevantFeatureFrame are the two classes presented in the class diagram shown in Figure 2. These two classes in the diagram are transformed into corresponding ERD as shown in Figure 3. This is achieved by generating some intermediate file in XML format. With the XML file, the model transformation is verified for correctness. Then the class diagram is transformed to source code using Java. This too was verified for consistency. Then the source code is implemented and subjected to metrics proposed in section 5.

6.3 RATE OF SIGNATURES WITH IMPROPER ARGUMENTS (NSIA)

This metric applied to the source code in Java that is LoadFile class. $FUN_{improper}(A)$ value obtained is 0 and the value for $FUN_{whole}(A)$ is 12. RSIA is computed as follows.

$$RSIA = 0/12 = 0$$

6.4 RATE OF UNUSED VARIABLES (ROUV)

This metric when applied to LoadFile class of PSM, the unused variables (VAR_{unused}) obtained is 0 and all variables in the class (VAR_{all}) is 27. The ROUV is finally computed as follows.

$$ROUV = 0/27 = 0$$

The result of ROUV metric is 0.24 which indicates rate of unused variables.

6.5 NUMBER OF CODE CLONES (NOCC)

This metric is applied to LoadFile class in the source code. The result obtained by the tool is 13. It is the count of code clones which is the functionality of our tool which detects clones and visualizes the same.

$$NOCC = 13$$

6.6 POPULATION OF CLONE CLASS (POP)

This metric when applied to LoadFile, the tool has returned values for two variables involved in the metric. Number of elements in clone $Elements_{clone}$ has got 13 while the total number of elements $Elements_{all}$ has got 313. The result of the metric is as given below.

$$POP = 13/313 = 0.041533$$

Here $Elements_{clone}$ refers to the count of elements in the code clones while the $Elements_{all}$ refers to the number of elements.

6.7 RATIO OF NON-REPEATED TOKEN SEQUENCES (RNRS)

This metric is applied to LoadFile class using our tool. The tool obtained the sum of length of the non-repeated token sequence of code clones and the sum of token sequence of code clones. The results are as shown below.

$$LOS_{non-repeated}(C_i) = 11$$

$$LOS_{while}(C_i) = 13$$

When these values are substituted into the metric, the result is as shown below.

$$RNR = 11/13 = 0.846153$$

High RNR value indicates ratio of non-repeated token sequences is more while lesser value indicates the repeated token sequences is more. According to the methodology described in section 6, the group truth is obtained from human experts and the results are presented in Table 1.

TABLE 1 Results of metrics compared with ground truth

Measures	Ground Truth	Tool Result
RSIA	1	0
NOCC	1	0
ROUV	1	0
POP	0.9	0.041533
RNR	1.2	0.846153

From Table 1, it is evident that the results of metrics computed by our tool and the results of metrics computed by human experts are presented.

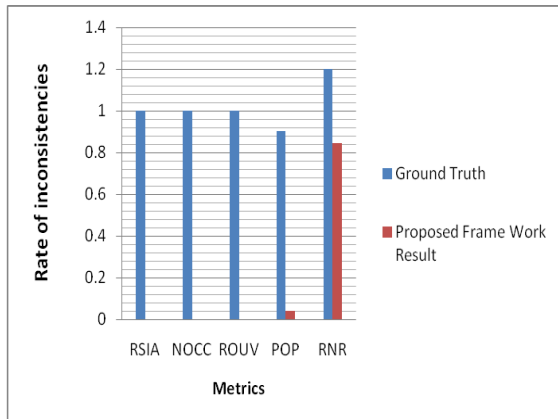


FIGURE 4 Results of quality metrics

The metrics are applied to the model transformations and the results are compared with the ground truth. The results are evaluated using metrics such as RSIA, NOCC, ROUV, POP, and RNR. A new metric is derived from the aforementioned metric. The details are as follows.

$$NewMetric = \frac{(P_1 + A_1) + (P_2 + A_2) + (P_3 + A_3) + \dots + (P_n + A_n)}{n}$$

$$A_n = P_n \times W_n \%$$

$$P_n = final\ value$$

$$W_n \% = weight\ percentage$$

$$A_n = weight\ value$$

RSIA

$$A_1 = P_1 \times W_1 \% = 1 \times (30/100) = 0.3$$

$$P_1 + A_1 = 1 + 0.3 = 1.3$$

ROUV

$$A_2 = P_2 \times W_2 \% = 1 \times (25/100) = 0.25$$

$$P_2 + A_2 = 1 + 0.25 = 1.25$$

POP

$$A_3 = P_3 \times W_3 \% = 0.9 \times (25/100) = 0.225$$

$$P_3 + A_3 = 0.9 + 0.225 = 1.125$$

RNR

$$A_4 = P_4 \times W_4 \% = 1.2 \times (20/100) = 0.24$$

$$P_4 + A_4 = 1.2 + 0.24 = 1.44$$

Substitute all these values in new metric equation
Then we get = (1.3+1.25+1.125+1.44)/4=1.27

TABLE 2 Result Comparison

Tool	Result of new metric
Solid SDD	0.82
ConQAT	0.91
XRTSDIC	1.27

As shown in Table 2, it is evident that the derived metric which provides overall performance of the framework in terms of finding quality of transformations is compared with other tools such as SolidSDD and ConQAT. The XRTSDIC shows better performance.

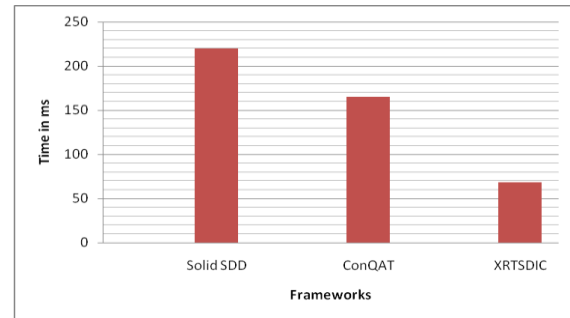


FIGURE 5 Performance comparison


From Figure 5, it is evident that the performance of XRTSDIC is better when compared with SolidSDD and ConQAT. SolidSDD and ConQAT are tools have inconsistency metrics including code clones. However, they do not have model transformation capabilities. XRTSDIC thus shows superior performance.

7 Conclusions and future work

This paper presented our research made on Model Driven Engineering (MDE) in terms of proposing metrics for measuring quality of model transformations. It is our ongoing research and the framework we built earlier [1] named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) which supports end-to-end transformations of object oriented models. In this paper we focused on defining consistency metrics meant for measuring quality of model transformations. The metrics are pertaining to model consistency as our research was focusing on this area. Model-to-model transformations are from Platform Independent Model (PIM I) to Platform Independent Model (PIM II) and from PIM to Platform Specific Model (PSM). The goal of our research in this paper is to make these model transformations measurable. Towards this end we proposed different metrics namely number of signatures with improper arguments (RSIA), number of unused variables (ROUV), number of code clones (NOCC), Population of Clone Class (POP), and Ratio of Non-Repeated Token Sequences (RNR). We enhanced our tool [1] to demonstrate the proof of concept of the application these metrics to know quality of model transformations. Our empirical study revealed that the proposed metrics add value to our model consistency checker as they quality in model transformations.

References

- [1] Ramesh G, Rajini Kanth T V, Ananda Rao A 2016 XRTSDIC: Towards a flexible and scalable framework for detecting and tracking software design inconsistencies *Proceedings of the first A.P. Science Congress*
- [2] Ramesh G, Rajini Kanth T V, Ananda Rao A 2016 Extensible real time software design inconsistency checker: a model driven approach *Proceedings of the International multi conference of engineers and computer scientists 2016 Vol I, IMECS Hong Kong*
- [3] Ananda Rao A, Rajini Kanth T V, Ramesh G 2016 A model driven framework for automatic detection and tracking inconsistencies *Journal of software* 11(6) 538-53
- [4] Ramesh G, Rajini Kanth T V, Ananda Rao A 2016 An extended model driven framework for end-to-end consistent model transformation *Indian journal of computer science and engineering (IJCSSE)* 7(4) 118-32 Aug-Sep 2016
- [5] Ramesh G, Rajini Kanth T V, Ananda Rao A 2016 XRTSDIC: Model Transformation from PIM to PSM *Communicated to IET Software*
- [6] Paredis C J J, Bernard Y, Burkhart R M, de Koning H-P, Friedenthal S, Fritzson P, Rouquette N F, Schamai W 2010 An overview of the sysml-modelica transformation specification *IEEE* 1-14
- [7] Bézin J, Soley R M, Vallecillo A 2010 Editorial to the proceedings of the first international workshop on model-driven interoperability *ACM* 1-112
- [8] Cheng B H C, Lemos R de, Giese H, Inverardi P, Magee J 2009 *Software engineering for self-adaptive systems* Springer 1-270
- [9] Kessentini M, Sahraoui H, Boukadoum M, Omar O B 2012 Search-based model transformation by example *Softw Syst Model* 209-26
- [10] Rodríguez A, García-Rodríguez de Guzmán I, Fernández-Medina E, Piattini M 2010 Semi-formal transformation of secure business processes into analysis class and use case models: An MDA approach *Information and Software Technology* 52 945-71
- [11] Baudry B, Ghosh S, Fleurey F, France R, Traon Y L, Mottu J-M 2010 Barriers to systematic model transformation testing *IEEE* 1-12
- [12] Hermann F, Ehrig H, Golas U, Orejas F 2010 Efficient analysis and execution of correct and complete model transformations based on triple graph grammars *ACM* 1-10
- [13] Chidamber S R, Kemerer C F 1994 A metrics suite for object oriented design *IEEE transactions on software engineering* 20(6) 1-18
- [14] Hutchinson J, Whittle J, Rouncefield M, Kristoffersen S 2011 Empirical assessment of MDE in industry *ACM* 1-10
- [15] Chitra M T, Sherly E 2016 Verification of behavior preservation in uml sequence diagrams using graph models *Indian journal of computer science and engineering* 7(4) 1-6
- [16] Kuzniarz L, Huzar Z, Reggio G, Sourrouille J L, Staron M 2003 Workshop on Consistency Problems in UML-based Software Development II *IEEE* 1-89
- [17] Rosenberg L H, Hyatt L E 2010 Software quality metrics for object-oriented environments *IEEE* 1-6
- [18] García J, Diaz O, Azanza M 2013 *Model transformation co-evolution: a semi-automatic approach* Springer 144-53
- [19] Biehl M 2010 Literature study on model transformations *Royal institute of technology* 1-28
- [20] Kessentini M, Sahraoui H, Boukadoum M 2011 Example-based model-transformation testing *Autom Softw Eng*, 199-224
- [21] Arendt T, Taentzer G 2013 A tool environment for quality assurance based on the eclipse modeling framework *Autom Softw Eng* 141-84
- [22] van Amstel M F, van den Brand M G J 2010 Quality assessment of ATL model transformations using metrics *IEEE* 1-15
- [23] Kapova L, Goldschmidt T, Becker S, Henss J 2011 Evaluating maintainability with code metrics for model-to-model transformations *ACM* 1-16
- [24] Vignaga A 2009 Metrics for measuring ATL model transformations *IEEE* 1-15
- [25] van Amstel M F, Lange C F J, van den Brand M G J 2008 Metrics for analyzing the quality of model transformations *ACM* 1-11
- [26] van Amstel M F, Lange C F J, van den Brand M G J 2009 *Using metrics for assessing the quality of ASF+SDF model transformations* Springer 239-48
- [27] Paen E 2012 Measuring incrementally developed model transformations using change metrics *Queen's University* 1-125
- [28] Boehm B W, Brown J R, Kaspar H, Lipow M, Macleod G J, Merrit M J 1978 Characteristics of software quality North-Holland

AUTHORS	
	<p>G. Ramesh</p> <p>University studies: received B. Tech Degree in Information Technology from RGM CET, Nandyal, Kurnool Dist. Andhra Pradesh, M. Tech Degree in Software Engineering from JNTUA college of Engineering, Ananthapuramu, Andhra Pradesh, India, Perusing Ph. D at JNTUA, Anantapuramu, Andhra Pradesh, India</p> <p>Scientific interests: Software Engineering and Big Data</p> <p>Publications: several papers in various International Journals/ Conference</p>
	<p>Dr. T.V. Rajinikanth</p> <p>University studies: received M.Tech degree in Computer Science & Engineering from Osmania University Hyderabad, Andhra Pradesh, India and he received PhD degree from Osmania University Hyderabad, Andhra Pradesh, India. He is Professor of Computer Science & Engineering Department, SNIST, Hyderabad, Andhra Pradesh, India.</p> <p>Publications: more than 50 publications in various National and International Journals/Conferences. Organised and Program Chair 2 International Conferences, 2 grants received from UGC, AICTE. Editorial Board Member for several International Journals.</p> <p>Best Paper Award: "Design and Analysis of Novel Similarity Measure for Clustering and Classification Of High Dimensional Text Documents" in the Proceedings of 15th ACM-International Conference on Computer Systems and Technologies (CompSysTech-2014), pg: 1-8, 2014, Ruse, Bulgaria, Europe. His main research interest includes Image Processing, Data Mining, Machine Learning.</p>
	<p>Dr. Ananda Rao Akepogu</p> <p>University studies: received B.Tech degree in Computer Science & Engineering from University of Hyderabad, Andhra Pradesh, India and M.Tech degree in A.I & Robotics from University of Hyderabad, Andhra Pradesh, India. He received PhD degree from Indian Institute of Technology Madras, Chennai, India. He is Professor of Computer Science & Engineering Department and currently working as Director Academic and Planning, of JNTUA College of Engineering, Anantapur, Jawaharlal Nehru Technological University, Andhra Pradesh, India.</p> <p>Publications: more than 100 publications in various National and International Journals/Conferences.</p> <p>Best Research Paper award for the paper titled "An Approach to Test Case Design for Cost Effective Software Testing" in an International Conference on Software Engineering held at Hong Kong, 18-20 March 2009. Received Best Paper Award: "Design and Analysis of Novel Similarity Measure for Clustering and Classification Of High Dimensional Text Documents" in the Proceedings of 15th ACM-International Conference on Computer Systems and Technologies (CompSysTech-2014), pg:1-8,2014, Ruse, Bulgaria, Europe. Also received Best Educationist Award, Bharat Vidya Shiromani Award, Rashtriya Vidya Gaurav Gold Medal Award, Best Computer Teacher Award and Best Teacher Award from the Andhra Pradesh chief minister for the year 2014. His main research interest includes software engineering and data mining.</p>