ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 3, No. 1, 244-251 (2018)

www.astesj.com

Special issue on Advancement in Engineering Technology

ASTESJ

ISSN: 2415-6698

Measuring modifiability in model driven development using object oriented metrics

Nwe Nwe[*,1], Ei Thu[2]

[1]*Computer University, Monywa, Myanmar*

[2]*University of Computer Studies Mandalay, Myanmar*

ARTICLE INFO

Article history:
Received: 31 October, 2017
Accepted: 02 January, 2018
Online: 30 January, 2018

Keywords :
Model transformation
Drools rule-based
Modifiability

ABSTRACT

Model driven development is an important role in software engineering. It consists of multiple transformation functions. This development is a paradigm for writing and implementing computer program quickly, effectively, at minimum cost and reducing development efforts because it transforms design model to object-oriented code. Our approach is rule-based model driven development in which textual Umple model is used as primary artifact and transformed to mobile applications. In this model driven development, evaluation of quality of transformation is critical. This paper has presented a set of metrics to assess the quality attribute of modifiability and evaluated using these object-oriented metrics. Results represent our approach achieves high efficiency in quality of modifiability.

1. Introduction

Changing with technology, mobile devices and mobile applications are necessary thing for every person and every sector. The burst on the availability of mobile devices is powering a growing mobile application. According to this fact, mobile applications, which are used in these devices, are critical in an industrial development. To fulfill the demand of these things, mobile application development preferably uses model-driven engineering than traditional software development process widely and effectively. It focuses on model for the development of software. There is lower the overall cost of building large internal applications, there is lower the risk of large application, speed time to build large applications and expand the pool of resources that can work on large application are main strategic objectives to use model driven development. Reduction of both direct and indirect development efforts, which enables scripters to contribute to enterprise development and enables task-oriented management of development are benefit of model driven development. It is a superset of model driven development because it goes beyond the traditional development.

In this case, the rule based model driven development of mobile application using Drool Knowledge-based Rule was presented in [1]. They also measured assessment of transformability using object oriented metrics. Drools Knowledge-based is a business rule management system with a forward and backward chaining inference based rules engines [2].

Moreover, this model-driven development is a development paradigm that uses model as the primary artifacts of the development process. It transforms source model to target model/code according to changing requirement and software reused more rapidly than traditional software development. A model transformation consists of multiple transformation functions. These transformation functions transform target language elements from the source language. Most of researchers concentrate on model to model transformation using intermediate meta-model or model to code transformation [3-6]. There is no quality, there is no efficiency in everything. Quality issue also change scale and become more important. The process transforms to new model or code related to quality of final software product and the quality of the model used to generate it. The consistency of source and target model and the assessment of quality of transformation is the critical issue in model transformation domains.

There are many attributes to evaluate quality in software engineering. Among them, most of these quality attributes can be applied to software artifacts in general. However, in [7] authors describe two quality criteria important in model transformation. They are transformability and modifiability. According to their

[*]Corresponding Author: Nwe Nwe, Email: nwenwemdy08@gmail.com

work, it is necessary to extend the assessment to other quality attribute of modifiability, maintainability and reusability using object-oriented metrics. Modifiability is the extent to which a model transformation can be adapted to provide different or additional functionality. The main reason for modifying a model transformation is changing requirements. Another reason is that the (domain specific) language in which the source and/ or target model are described which may be subject to changes. Modifiability captures the amount of effort needed to modify a model transformation. It is the combination of the modularity and reusability, an essential aspect of software engineering that promotes software maintainability. Moreover, it enables transform without affecting other parts of a program that are not directly connected to the changes. This paper is organized as follows: Section II explains the basic concepts of related work. Section III presents the contribution of rule based model transformation and quality attributes of modifiability. Section IV describes our experimental results and comparison results. Finally, section V concludes our approach and evaluation of modifiability using object oriented metrics.

2. Related work

There is increasing attention towards the generation of source code from modeling languages. Several researchers propose model driven approach for the different aspects of mobile applications. The model driven development of mobile applications using Drools knowledge-based Rule was describes in [1]. They developed mobile application by applying Drool rule based. Their work is closely related with JUSE4 android application [8,9]. Moreover, they attempted to address the consistency of source and target model and the assessment of transformability by measuring the accuracy of consistency between source and target model and assessing the transformability using object oriented metrics. According to their work, it is necessary to extend the assessment to other quality attributes of model driven development using object-oriented metrics. Authors performed a comparative study on C++, C# and Java programs using object-oriented metrics in [10]. It consists of class size, complexity, coupling cohersion, inheritance, encapsulation, polymorphism and reusability.

An evaluation of the quality of model transformation was defined in [11]. They made the quality of model transformation measurable. They presented the quality attributes and a set of metrics to assess these quality attributes. A calculation of metrics values using the same set of standard metrics for three software system of different sizes was described in [12].

In [5], authors presented quality goal in MDE and states that the quality of models is affected by the quality of modeling languages, tools, modeling processes, the knowledge and experience of modelers and the quality assurance techniques applied. In [3, 4], authors defined the meta-model and model transformation rule for model driven android application development. In [13], authors also defined ATLAS transformation rules for UML sequence diagram to generate enterprise java bean code (EJB). In [6], authors presented enhance code generation tool for android source code based on UML class and sequence diagram. In [14], authors specified meta-model with Ecore and transformation rules with Xpand templates for entity relationship diagram to generate android SQLite database model.

In model driven transformation, the approaches are quite different in their respective use of input model. Most of these approaches are based on graphical modeling or textual modeling languages. In contrast to our approach, the previous approaches applied pre-defined meta-modeling while our approach automatically parses and extracts syntax form input model of Umple [15]. Moreover, our approach has specified transformation rules in object pattern matching approach. JUSE4Android is also based on textual modeling languages. Unlike our approach, it is adding annotation into JUSE model and transform into android source according to the predefined meaning of annotation. Therefore, they generated source code contains some more files in their project. The authors [16, 17] have proposed the approach for empirical evaluation of model driven engineering in multiple dimensions. Their case studies include qualitative (expert judgements) and quantitative data (metrics) evaluations. They suppose that the productivity and defect detection rate are the popular metrics for measuring automation degree of MDD processes. Some quality goals such as well-establishment and precision are especially important in MDE [18 -20]. In [21], authors also developed open source tool aims to address quality measurement and prediction process to achieve automatically. In [22], authors presented the most recent challenges faced in the process to make model transformation more sophisticated. According to the literature, it is necessary to extend the assessment to other quality attribute of modifiability using object-oriented metrics. We conducted the comparative study for measuring the modifiability of MDD generated source code using object-oriented metrics. Therefore, we obtain more reliable findings.

3. Rule based model transformation framework

Model transformations become essential with the evolution of model driven development. It is a mechanism of automating the manipulation of models. A transformation is the automatic generation of a target model from source model using transformation definition. These transformations definition is a set of transformation rules that define how a model in the source language can transform into target language. These rules are descriptions of how one or more constructs in the source language can be transformed into one or more constructs in the target language.

In this section, we provide an overview of the framework and their underlying architecture. The proposed architecture is divided into three major parts corresponding to the main capabilities of the proposed framework. These components are parser, transformer and code generator. The parser receives an input model, written in Umple language, tokenizes it and passes it to the next component transformer. The transformer is a knowledge

based rule engines. It has received the tokens previously obtained and transforms them into internal representation consistent with target source code model using predefined set of Drools mapping rules. It is more correctly classified as a production rule system. It is a kind of Rule engine and also an Expert system, the validation and expression evaluation Rule Engines. It is declarative programming and allows to present what to do. The key advantage of rule engine is that using rules can make it easy to express solution to difficult problems and consequently have those solutions verified. The code generator translates the internal representation into target artifacts: source code as Java, XML and android activity class. Each component is tested independently to ensure that the input is processed correctly and the resulted output is validated [1]. Figure 1 describes the overall architecture of the proposed system. We have used Umple as input and transformed to mobile application. In this case of model transformation, we have applied Drool Rule based transformation.

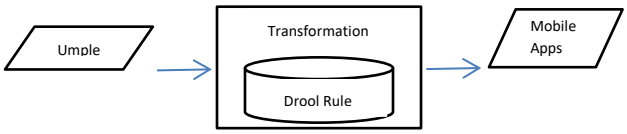

Figure 1 Overall Architecture of model transformation

3.1. Rule-based Inference System

In this system, we present a rule-based model driven approach to generate android application from text-based modeling language. Rule languages and inference engines incorporate reasoning capabilities are used in mobile application development system. A rule is made up of a collection of conditions associated with a sequence of actions to be applied to each collection of facts matching the rule condition. The proposed model transformation rules are based on Drools rule inference engine [2]. It is improved to reach the generation of mobile applications source code and introducing new concern in model driven mobile engineering. The core of the Drool suites and advanced Inference Engine using are improved Rete algorithm for object pattern matching. Rules are stored in the production memory, while facts are maintained in the working memory. The production memory remains the same during an analysis session, i. e, rules cannot be added or removed or changed. The contents of the working memory on the other hand can change. Facts may be modified, removed or added by executing rules or from external sources. After changing in the working memory, the inference engine is triggered and it works out which rules become "true" for the given facts. If there are multiple selected rules, their execution order will be managed via the Agenda, using a conflict resolution strategy.

3.2. Drools Transformation Rule

Drools rules are defined using Java-like language. It is a Business Logic integration Platform (BLiP). With the runtime, we create a working memory. The syntax of rule is shown as follows:

Rule
Rule <Rule Name>
When <Condition>
then <Action>

Rule: A rule is nothing but the logic that will be applied to incoming data. It has two main parts; when and then.

When: works out the condition on which Rule will be fired.

Then: the action; if the rules met the condition, they define what work this rule performs.

Step 1: Create a.drl (droolRule.drl)file where we will define the rules.

Step 2: Create Person POJO class.

The proposed rule engine consists three parts: umple2model, umple2view and umple2controller according to android model, view and controller perspective. Table 1 shows the sample form of Drools transformation rule for simple variable declaration for Account Title. Umple2Model.drl transforms incoming abstract syntax model (ASM) into plain java object (POJO). Umple2View.drl transforms ASM into android user interface XML file and Umple2Controller.drl transforms ASM into android activity class. The code generator receives the POJO model for model layer, XML model for view layer and android model for controller layer. The generator use the java development tool (JDT-core) to generate POJO class and android class source code. It is also used the JDOM to generate XML user interface file.

Table 1: Transformation Rule Sample

```
Umple          String AccountTitle;

Rule "VariableDeclaration"
Dialect "java"
when
$st : SyntaxTree(status==SyntaxTree.VAR_DECLARE)
then
TypeDeclaration type=AST2Android.Variable_Decl($st.getType());
CompilationUnit cu=$st.getCu();
$st.setStatus(SyntaxTree.ACTIVITY_CREATE);
$st.setType(type);
$st.setCu(cu);
update($st);
end
```

```
POJO (Model Layer) String AccountTitle;
PublicvoidsetAccountTitle(String at){
accountTitle=at;}
public String getAccountTitle(){
return accountTitle;}
}

XML (View Layer)
<EditText android:id=
"@+id/ txtaccountTitle"
android:layout_height=
"wrap_content"
android:layout_width=
"wrap_content"/>

Android (Controller Layer)
private EditText txtaccountTitle;
private String accountTitle;
txtaccount-
Title=(EditText)findViewById(R.id.accountTitle);
```

4. Experimental results and comparison

In model driven development, there are two important criteria to evaluate the quality of model transformation. They are transformability and modifiability. The consistency of source and target model and the assessment of transformability are evaluated in [1]. According to these results, we extend to evaluate the assessment of quality of modifiability in this model driven transformation.

4.1. Modifiability

Changes made for the requirements are rendered quality of the code in the models the code. This fact becomes challenges in quality of model driven development [quality]. To address these issues, this paper has proposed the evaluation of modifiability of model driven transformation using object oriented metrics. This modifiability is decomposed into traceability of model elements and well-designated or not being too complex. Moreover, the extent to which a model transformation can be adapted to provide different or additional functionality. The main reason for modifying a model transformation is changing requirements. Another reason is that the (domain specific) language in which the source and/ or target model is described which may be subject to changes. Modifiability captures the amount of effort need to modify a model transformation. It controls the visibility of system development. Such controls contribute to modularity, an essential aspect of software engineering that promotes software maintainability. In object-oriented programming, we note that these classes form the modules of programs. From the modularity perspective, modules should be as independent as possible with minimal coupling.

We have also performed a comparative study using both our proposed system and JUSE4 Android [8, 9] based on object oriented metrics. These metrics indicate quality of source code directly. We evaluate the quality of model driven for model transformation of generated source code quality and prior approach's generated source code quality. The results are used to evaluate a model is complete or suitable for automation or a modeling technique is appropriate for a target transformation. Therefore we have identified metrics to examine in this process.

4.2. Metrics

We describe the metrics for assessing the quality attributes for model transformation. Those metrics are applicable to language definition and characteristics of languages. For modifiability, we determine encapsulation, polymorphism and reusability as the quality criteria. These are described in table 2.

Table 2: Object oriented quality criteria

Quality of Criteria for Comparative Study				
No	Quality Criteria	Metrics	Acronyms	Desired Results
1	Encapsulation	Methods of hiding factor	MHF	High
		Attribute hiding factor	AHF	High
2	Polymorphism	Number of method overridden by a subclass	NMO	High
		Polymorphism factor	PF	High
3	Reusability	Reuse ratio	RP	High
		Specialization ratio	SR	High

A. Data Collection

To evaluate the modifiability of transformation, we have collected the metric values by using Eclipse metrics Plug-in [23]. It is an open source metrics calculation tools which measures various metrics and detects cycle in package and type dependencies. At first, we have generated the android application from different approaches of proposed and prior approach respectively. In the next step, we enable Eclipse Metrics Plug-in on each generate source code that give common solution. Finally, we extracted the mean, standard deviation and maximum metric values for each generate source code. In our comparative study, we have collected the average metric values from the proposed

and prior generated source code with respect to the quality criteria. Table 3 shows extracted metric values.

4.3. Encapsulation

It is the bringing together of a set of fields and methods into an object definition and hiding their internal working from the users of the object. By encapsulation, the way an object or its fields and methods are structured which is not visible to the users of the object. It is also facilitated by bundling and information hiding. It enhances the software maintainability. Encapsulation increases the cohesiveness of data and methods through bundling and reduces the strength of coupling between software components through information hiding. For encapsulation, we have measured method hiding factor (MHF) and attribute hiding factors (AHF) using the following equation 1 and 2. MHF and AHF are indicators to show how well methods and attributes are hidden inside classes. The results are presented in table 3. The comparisons of MHF values and AHF values of proposed system and prior systems are shown in the following figure 2 and 3 respectively. These metrics are measured at system level and high metric values are expected. The results are compared with prior approach and our proposed approach. This result means that we achieve the higher method hiding factor and attribute hiding factors.

Let V (M)= number of classes where the method M is visible, then

$$MHF= 1-\frac{\sum V(M)/(Total\ numbers\ of\ classes-1)}{Number\ of\ methods\ in\ all\ classes} \quad (1)$$

Let V (A) = number of classes where the attribute A is visible, then

$$AHF= 1-\frac{\sum V(A)/(Total\ numbers\ of\ classes-1)}{Number\ of\ attributrd\ in\ all\ classes} \quad (2)$$

By using the equation 1, we have calculated the MHF value. The result described that the ratio of number of classes where the visible method M is higher, the MHF value is lower.

By using the equation 2, we have calculated the AHF value. The result described that the ratio of number of classes where the visible attribute A is higher, the AHF value is lower.

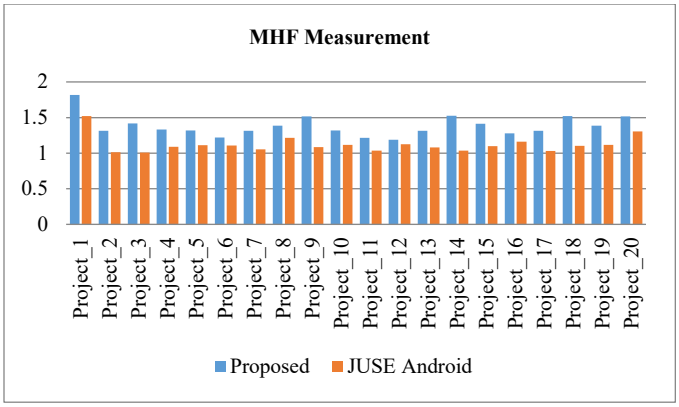

Figure 2 Comparison results of method of hiding factors

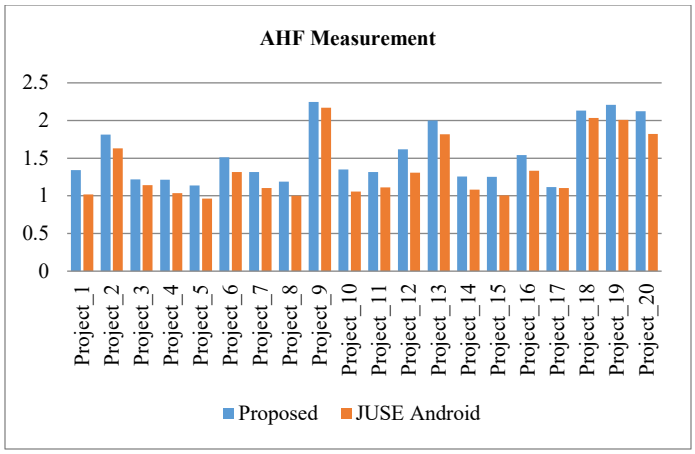

Figure 3 Comparison results of attribute hiding factors

4.4. Polymorphism

It is the ability of objects to respond to the same message but with the appropriate method based on their class definitions. For polymorphism, we have measured the number of method overridden by a sub class (NMO) and polymorphism factors (PF). Results are described in figure 4 and 5 respectively.

For NMO, we have determined the number of methods in a subclass overridden from its base class by using equation 3. Moreover, we determine the PF by using following equation 4. By using this equation 4, we present the PF value in table 3. To be specific, NMO is a class-level metric, which refers to the number of methods overridden by a single subclass, while PF is a system level metric, which measures the degree of method overriding in the whole type tree. These values are desired to be high. The results are compared with prior approach and our proposed approach. This result means that we achieve the higher number of methods overridden by a subclass and polymorphism factor.

NMO= the number of methods in a subclass overridden from its base class (3)

$$PF= \frac{\sum_{i=0}^{TC} M0(Ci)}{\sum_{i=0}^{TC}[(Mn\ Ci)x\ DC\ (Ci)]} \quad (4)$$

Where TC = the total number of classes

M_n (C_i)= Number of new methods of the class C_i

M_o (C_i)= Number of overriding methods of the class C_i

DC(C_i)= Number of Descendant of the class C_i

We have calculated the NMO value by using the number of methods in a subclass overridden from its base class and applying equation 3. They are presented in table 3.

We have calculated the PF value by using equation 4 and the results are described in table 3. The results present that number of overriding methods of the class is higher, the PF value is higher.

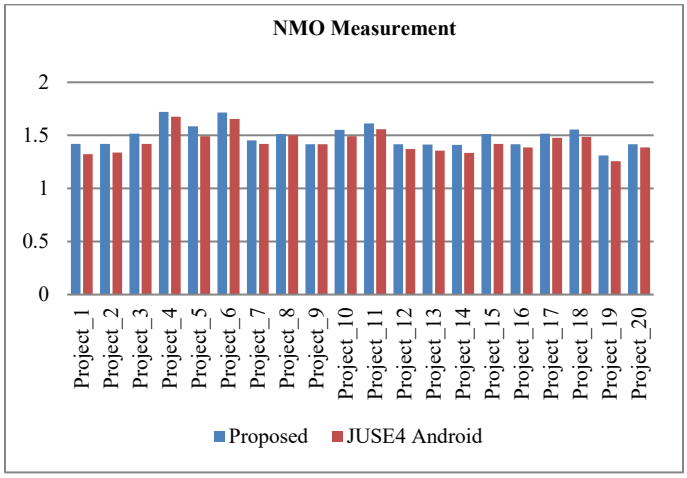

Figure 4 Comparison results of number of methods overridden by a sub class

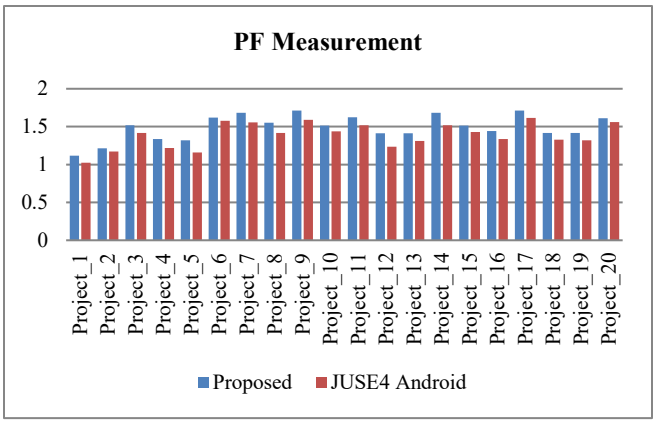

Figure 5 Comparison results of polymorphism factors

4.5. Reusability

Reusability is the extent to which a model transformation can be reused by other model transformations. It refers to as-is reuse. It is especially relevant for model transformations when a source model has to be transformed into different target models or vice versa. For reusability, we have measured reuse ratio (RR) and specialization ratio (SR) and results are presented in figure 6 and 7.

We have determined the RR and SR by using the following equation 5 and 6. RR and SR are both system level reusability metrics. They are calculated as the ratios of subclass to all classes and to super classes, respectively. The results are presented in table 3. We have expected to be highly reused, large reusability metrics values are desirable.

$$RR = \frac{(Total\ number\ of\ Super\ classes)}{(Total\ number\ of\ classes)} \quad (5)$$

$$SR = \frac{(Total\ number\ of\ Sub\ classes)}{(Total\ number\ of\ super\ classes)} \quad (6)$$

We have calculated the RR and SR values by applying equations 5 and 6 respectively. There are more subclasses, the higher the RR and SR values.

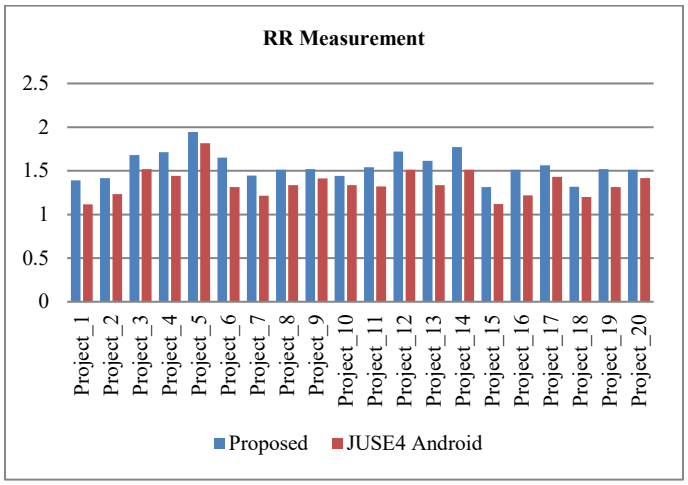

Figure 6 Comparison results of reuse ratios

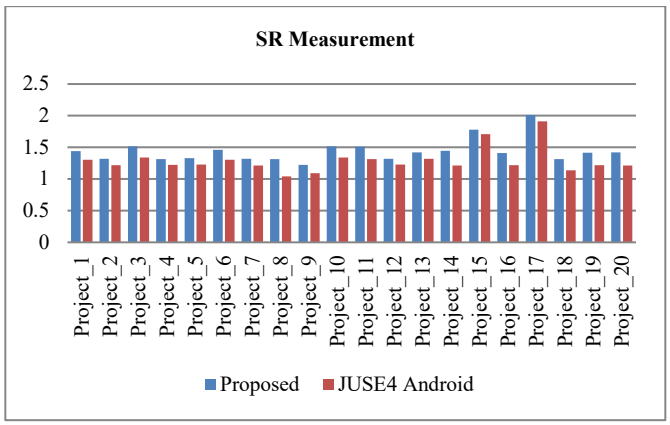

Figure 7 Comparison results of specialization ratios

5. Results and discussion

We have presented comparison results of our measurement. Expected results for our evaluation are shown in table 2. The larger a model transformation, the harder it is to understand and modify. Moreover, the number of signature and equations per function has a negative effect on consistency. If more similar signatures or equations have to be written, it is more likely that a different style is used.

The comparison results of our measurements are described in table 3. We have employed the result from MHF and AHF metrics for encapsulation, NMO and PF metrics for polymorphism and RR and SR for reusability to compare our approach and prior approach.

We have used private, protected and public keywords to control the accessibility to the method and attributes inside a class. According to these facts, we have planned quality attributes of our system to achieve higher modifiability.

Table 3: Metric values collected by Eclipse Metrics Plug in

Project	MHF		AHF		NMO		PF		RR		SR	
	Proposed	Prior	Proposed	Prior	Proposed	Prior	Proposed	Prior	Proposed	Prior	Proposed	Prior
Project_1	1.817	1.52	1.341	1.017	1.418	1.324	1.118	1.024	1.391	1.117	1.441	1.305
Project_2	1.312	1.012	1.814	1.631	1.418	1.339	1.214	1.172	1.415	1.234	1.318	1.216
Project_3	1.418	1.01	1.218	1.141	1.516	1.418	1.516	1.418	1.681	1.519	1.515	1.341
Project_4	1.332	1.091	1.216	1.037	1.721	1.675	1.338	1.219	1.712	1.441	1.312	1.225
Project_5	1.318	1.113	1.137	0.964	1.584	1.492	1.321	1.158	1.944	1.814	1.331	1.226
Project_6	1.218	1.108	1.512	1.314	1.714	1.654	1.62	1.578	1.651	1.315	1.461	1.306
Project_7	1.314	1.052	1.315	1.103	1.454	1.418	1.681	1.554	1.445	1.215	1.317	1.211
Project_8	1.386	1.216	1.189	1.003	1.512	1.5	1.551	1.418	1.512	1.335	1.312	1.041
Project_9	1.514	1.084	2.247	2.171	1.416	1.415	1.712	1.589	1.518	1.412	1.224	1.091
Project_10	1.317	1.117	1.351	1.056	1.551	1.491	1.512	1.438	1.441	1.337	1.516	1.338
Project_11	1.215	1.034	1.314	1.114	1.612	1.558	1.623	1.519	1.541	1.319	1.511	1.314
Project_12	1.188	1.127	1.618	1.306	1.416	1.371	1.412	1.237	1.721	1.512	1.317	1.227
Project_13	1.315	1.081	2.001	1.818	1.412	1.356	1.412	1.311	1.615	1.336	1.418	1.318
Project_14	1.523	1.034	1.258	1.084	1.411	1.336	1.681	1.518	1.771	1.512	1.446	1.215
Project_15	1.412	1.098	1.252	1.004	1.513	1.418	1.512	1.429	1.314	1.118	1.781	1.711
Project_16	1.278	1.161	1.541	1.331	1.417	1.387	1.441	1.337	1.512	1.217	1.412	1.219
Project_17	1.314	1.033	1.118	1.105	1.516	1.477	1.711	1.616	1.561	1.431	2.012	1.911
Project_18	1.521	1.102	2.132	2.034	1.554	1.486	1.417	1.327	1.318	1.201	1.316	1.138
Project_19	1.386	1.117	2.21	2.007	1.312	1.258	1.415	1.318	1.518	1.312	1.416	1.216
Project_20	1.517	1.305	2.124	1.823	1.416	1.387	1.612	1.559	1.512	1.416	1.418	1.213

By applying the experimental results, our approach has higher value than prior approach in encapsulation, polymorphism and reusability. Moreover, our expected result is that the high metric values are preferable.

These experimental results show how well methods and attributes are hidden inside classes. Therefore, our approach can help system develop methods and attributes which are hidden inside classes more efficiently. This means that our approach is more efficient than others. However, these results describes that they are a little bit higher than prior approach. By using these results, we will enhance our approach to achieve more efficiently and effectively model driven development process.

6. Conclusion

Model transformations become essential with the evolution of model driven development. It is an automatic generation of a target model from source model by using transformation definition. Modifiability is key issues in quality of this transformation. To address this issue, we have evaluated the modifiability of quality of model transformation using object oriented metric. This modifiability is decomposed into traceability of model elements and well-designated or not being too complex. Moreover, the extent to which a model transformation can be adapted to provide different or additional functionality. The main reason for modifiability of a model transformation is changing requirements. In this paper, we have performed the comparative study on our approach and prior approach to determine modifiability to develop more efficient mobile application system. We have determined the encapsulation, polymorphism and reusability as quality metrics. These metrics are measured at system level. We have used private, protected and public keywords to control the accessibility to the method and attributes inside a class. According to these facts, we have planned quality attributes of our system to achieve higher modifiability. The determination of experimental results represent that we achieve high score from comparison of our approach and prior approach. This means that our system is more traceability and

well-designated. Using these findings, we will enhance our approach to achieve higher efficiency and quality. In the future work, we will investigate more quality attribute for high accuracy of system development. Moreover, we will also evaluate the impact of transformation rules.

References

[1] Thu, E. E, Nwe N, "Model Driven Development of Mobile Applications Using Drools Knowledge-based rule" proceeding of SERA 2017, June 7-9, 2017, London, UK.

[2] De Lay, E, Jacobs D, "Rules-based Analysis with JBoss Drools: Adding Intelligence to Automation", ICALEPCS 2011, Proceeding of ICALEPCS Genoble, France

[3] Son, H.S, Kim, W.Y and Chul, R.Y, "MOF based Code Generation Method for Android Platform", International Journal of Software Engineering and Its application, Vol 7, No 3, Hongik University, Sehong Campus, Korea, 2013.

[4] Son, H. S, Kim, J.S, Chul, R. Y, SMTL Oriented Model Transformation Mechanism for Heterogeneous Smart Mobile Models, International Journal of Software Engineering and its Applications, Volume 7, No3, Sejong Campus, Korea, 2013.

[5] Parada, A.G, Lisane, B, A Model Driven Approach for Android Application Development, Brazilian, Symposium on Computing System Engineering (SBESC), Pelotas, Brazil, 2012.

[6] Parada, A.G.; Milena, R.S.; Automating mobile application development: UML-based code generation for Android and Window phone, Journal of Theoretical and Applied Information (RITA), volume 22, Pelotas, Brazil, 2015.

[7] Solheim, L and Neple, T, Model Quality in the Context of Model-Driven Development, Proceeding of 2nd International Workshop on Model- Driven Enterprise Information Systems (MDEIS'06), pp. 27-35, 2006.

[8] Silva, L.P, Abreu,F.B.: A Model-Driven Approach for Mobile Business Information Systems Applications, ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, MODELS 2014, QUASAR/ISTAR/ISCTE-IUL, Lisboa, Portugal.

[9] Silva, L.P, Abreu,F.B.: Model-Driven GUI Generation and Navigation for Android BIS Apps, MODELSWARD 2014, Portugal.

[10] Baowen, X, Di.W.: A metrics-based Comparative Study on Object-Oriented Programming Languages, 27th International Conference on Software Engineering and Knowledge Engineering, SEKE' 2015, USA.

[11] van Amstel, M.F; Lange, C.F.J.; van den Bran, M.G. J.: Metrics for analyzing the quality of model transformations, Proceeding s 12th ECOOP Workshop on Quantitative Approaches on Object Oriented Software Engineering (QAOOSE 08, Paphos, Cyprus, July 8, 2008.

[12] Rudiger, L, Jonas, L.: Comparing Software Metrics Tools, International Symposium on Software Testing and Analysis, ISSTA'2008, USA.

[13] Omar, E.B, Bragun, B, Automatic Code Generation by Model Transformation from Sequence Diagram of System's Internal Behavior International Journal of Information Technology, Hassan 1st University, Morocco, 2012.

[14] Steeg, C.C, Gotz, F, Model Driven, Data Management in Android with the Android Content Provider, https:// code.goodle.com/archieve/p/mdsd-android-content-provider/Bingen, Germany, 2011.

[15] https://github.com/umple/umple

[16] Mohagheghi, P, Aagedal, J, Evaluating Quality in Model-Driven Engineering, Proceeding of 29th International Conference on Software Engineering Workshops (ICSEW'07), 2007

[17] Mohagheghi,p.: An Approach for Empirical Evaluation of Model Driven Engineering in Multiple Dimensions, MODELPLEX, Oslo, Norway, 2010.

[18] Chidamber, S.R, Kamerer, C.F.: A metrics suite for object-oriented design, IEEE Transaction of Software Engineering, pp-(20)6, 1994: 476-493.

[19] Henderson-Sellers, B.: Object-oriented metrics: measures of complexity, Prentice Hall, 1995.

[20] Badreddin, O, Lethbridge, T.C, Forward, A, A Novel Approach to Versioning and Merging Model and Code Uniformaly, in MODELSWARD 2014, Proceeding of the 2nd International Conference on Model-Driven Engineering and Software Development, 2014.

[21] Kocaguneli, Ekrem, et al.: Prest: An Intelligent Software Metrics Extraction, Analysis and Defect Prediction Tool, 21th International Conference on Software Engineering and Knowledge Engineering, SEKE'2009, USA.

[22] Pallavi, K, Suita, P.U: Model Transformation, Concept, Current Trends and Challenges, International Journal of Computer Applications, Volume 119, No 14, Nasik, Manarashtra, India, 2015.

[23] http://metrics.sourceforge.net