Integrated Construction with Plug&Play Mechanism based on Metamodel for Development Process

Woo Sung Jang
Dept. of Software and Communication Engineering
Hongik University
Seoul, Korea
uriel200@hongik.ac.kr

So Young Moon
Dept. of Software and Communication Engineering
Hongik University
Seoul, Korea
whit2@hongik.ac.kr

R. Young Chul Kim
Dept. of Software and Communication Engineering
Hongik University
Seoul, Korea
bob@hongik.ac.kr

***Abstract*— The software development environment of SMEs may not mature in the development process. This can lead to failure in software quality management and asset management. To solve this problem, we study software visualization. Software visualization visualizes the software development process and visualizes the overall workflow within the company. However, in an existing software visualization environment, once a tool is selected, it is difficult to change the tool. Software visualization proposes an easy plug&play environment for process visualization to solve this problem. This method supports easy changes in issue tracking systems, continuous integration tools, and toolchain tools in a visualization environment.**

Keywords—Software Visualization, Process Visualization, Metamodel, Issue Tracking System, Continuous Integration, Metamodel Transformation

I. INTRODUCTION

Recently, the importance of software quality has been increasing. Large companies use excellent software quality management processes. However, SMEs struggle with software quality management due to high costs and a lack of skilled human resources [1,2]. It can be solved in a software visualization [3]. Visualize business and development processes within the enterprise using open source tools purchased from the enterprise. However, it is difficult to change the tool in an existing visualization environment once the tool is selected. As a result, the existing visualization environment is an environment that relies on tools.

We propose an easy plug&play environment for process visualization among software visualization environments. It is a method that supports easy changes in Issue Tracking System (ITS), Continuous Integration (CI) tools, and Toolchain tools, which are tools used in process visualization environments. In addition, when adding a new tool, the user only needs to add a metamodel file and a transformation rule file of a new tool to the transformation engine.

The plan of this paper is as follows. Chapter 2 refers to the related research. Chapter 3 refers to the design of process visualization plug&play environment based on metamodel. Chapter 4 refers to the case study. Chapter 5 refers to the conclusions and future studies.

II. RELATED STUDY

A. Software Visualization

Software visualization [3] focuses on visualization of all tasks related to software, visualization of software structures, and automatic generation of documents. This method provides 1) tracking from requirements to tests. 2) The relationship between requirements, design, and development source codes is identified during development. 3) Software quality scores are quantitatively measured. 4) Development documents are automatically generated. As a result, software visualization reduces the burden of human resources and cost and increases software quality.

The structure of software visualization consists of Process Visualization, Architecture Visualization, and Automatic Documentation, as shown in Figure 1.

Process Visualization visualizes the software development process. First, information from ITS and information from CI is delivered to Toolchain. Toolchain delivers the software structure diagram generated from the received data and the quantitatively measured software score to the Dashboard. The Dashboard outputs the received strategic scores and graphs on the screen. Finally, the Dashboard collects and expresses various information.

Architecture Visualization visualizes the structure of software. Architecture Visualization may be included within Process Visualization. It analyzes the source code, politically measures the analyzed source code to generate scores, and graphically expresses the structure of the source code.

Automatic Documentation automatically documents works that have occurred during software development.

Each visualization method does not depend on a specific tool. The software visualization environment should be established by analyzing and understanding the culture of target SMEs.

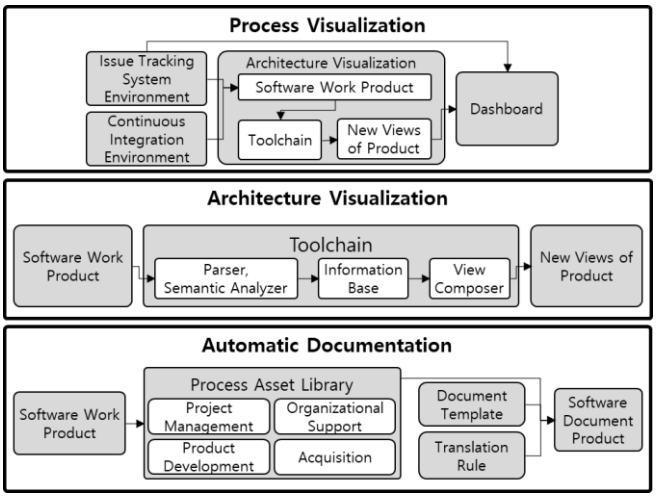

Fig. 1. Software Visualization

III. Design of Process Visualization Plug&Play Environment based on Metamodel

A. *Design of Process Visualization Plug&Play Environment*

We propose a plug&play environment for Process Visualization in software visualization. Figure 2 below is the design of the Process Visualization plug&play Environment. Furthermore, it explains plug&play process in detail of Toolchain.

(a) Process Visualization

Figure 2-(a) shows Process Visualization in Software Visualization. ITS tools used in practice include Redmine[4] and Jira[5]. CI tools include Jenkins [6], Buddy[7], etc. Companies select one of the various tools and apply it to a visualization environment. Toolchain tool is one of the components of Architecture Visualization. The source code structure is analyzed and quantitatively measured to calculate the software quality score. The dashboard should collect, calculate, and output information desired by the administrator from ITS, CI, Toolchain, etc.

(b) Plug&Play Environment of Tools for Process Visualization

Figure 2-(b) describes in detail the automatic plug&play process of ITS, CI, and Toolchain tools in Figure 2-(a). Data source is information stored in ITS, CI, and Toolchain. Data Target is information output on the dashboard. The Independent CI Model, the Independent ITS Model, and the Independent Toolchain Model store information on tools. For example, the Independent ITS Model consists of the Redmine Model and the Jira Model. The Redmine Model stores project information of the Redmine tool, and the Jira Model stores project information of the Jira tool. The Specialized CI Model, Specialized ITS Model, and Specialized Toolchain Model store the integrated information of each tool. For example, the Specialized ITS Model can store all information from Independent ITS Models (Redmine Model, Jira Model, etc.). Adapter is a rule that automatically transforms the Independent Model into a Specialized Model. The adapter performs model transformation by referring to the Data Catalog. Data Catalog stores Metamodels of Independent Models and Specialized Models.

(c) Plug&Play Environment for Toolchain

Figure 2-(c) describes the plug&play process of Toolchain in detail in Figure 2-(b). Bad Smell[8] measures a bad smell in the source code. If the user uses Bad Smell Toolchain, Bad Smell Toolchain measures the bad smell score from the source code stored in the Data Source and stores it in the Bad Smell Toolchain Model. Moreover, the stored model is automatically transformed into a Specialized Toolchain Model by the Bad Small Transformation Rule. The Data Target loads the transformed model.

(d) Bad Smell Metamodel Transformation Rule

Figure 2-(d) is the detailed flow of the Bad Smell Metamodel Transformation Rule in Figure 2-(c). The Transformation Engine reads files stored in the Bad Smell Toolchain Model. And run Bad Smell Transformation Language. Bad Smell Transformation Language automatically transforms the Bad Smell Toolchain Model into a Specialized Toolchain Model by referring to two metamodel information. Finally, the Transformation Engine stores the automatically transformed model as an XML Metadata Interchange (XMI) file.

B. *Bad Smell Toolchain Metamodel*

Figure 3 is the Bad Smell Toolchain Metamodel. It stores the meta-structure of the Bad Small Toolchain Model. Badsmell includes several BSclass. BSClass includes several BSmethods. In the source code, class information is stored in the name of the BSclass. The bad smell of the class is stored in the type of BSclass. In the source code, method information is stored in the name of the BSmethod. The bad smell of the method is stored in the type of BSmethod.

C. *Specialized Toolchain Metamodel*

Figure 4 is the Specialized Toolchain Metamodel. It stores the meta-structure of the Specialized Toolchain Model. Classes stores class information and method information in the source code. Connections stores the relationship between the class and the method in the source code. Scores store quality scores of classes, methods, and relationships.

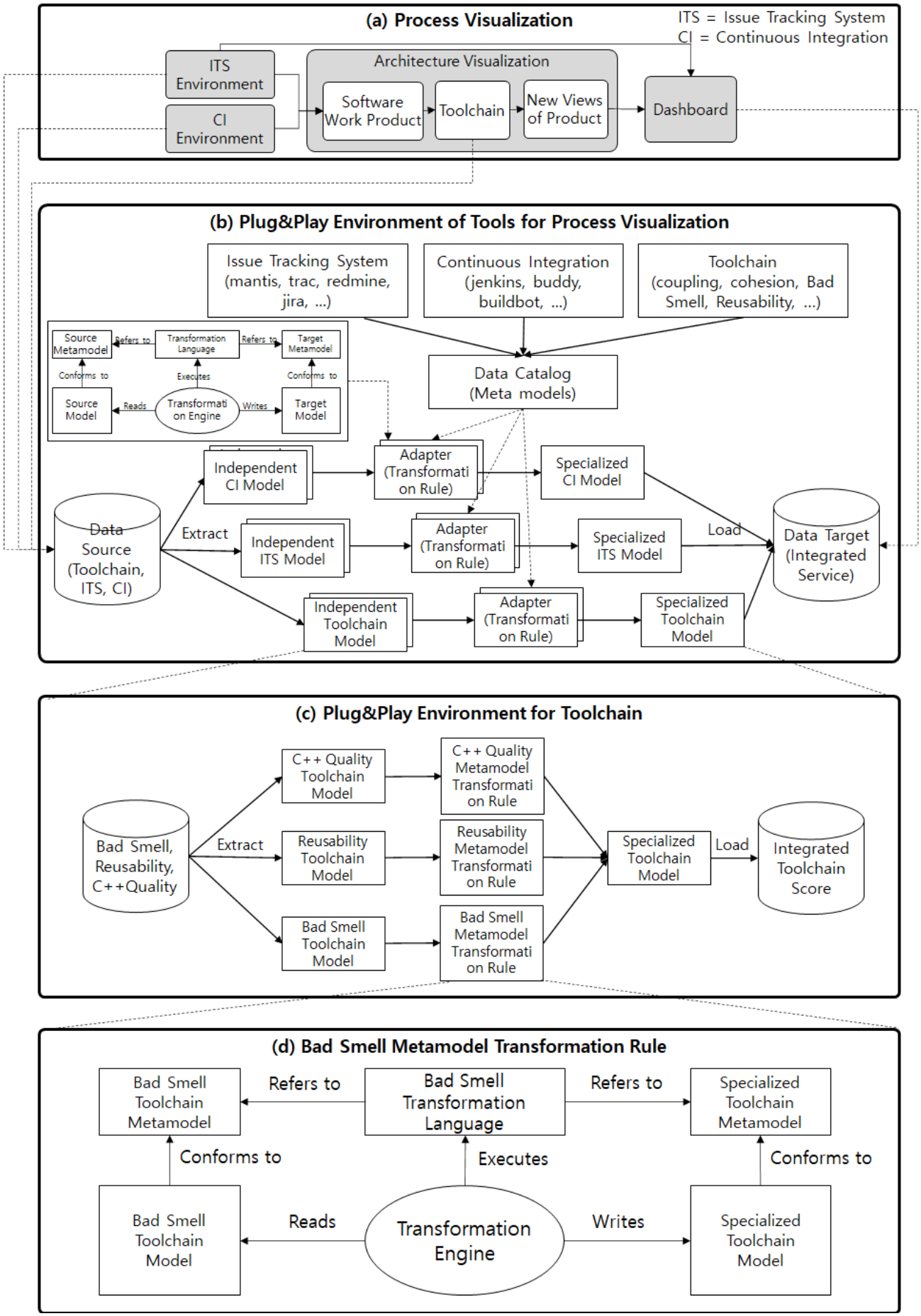

Fig. 2. Design of Process Visualization Plug&Play Environment

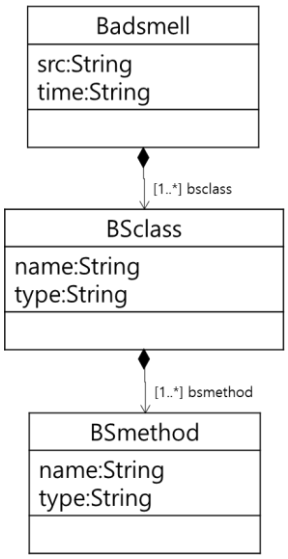

Fig. 3. Bad Smell Toolchain Metamodel

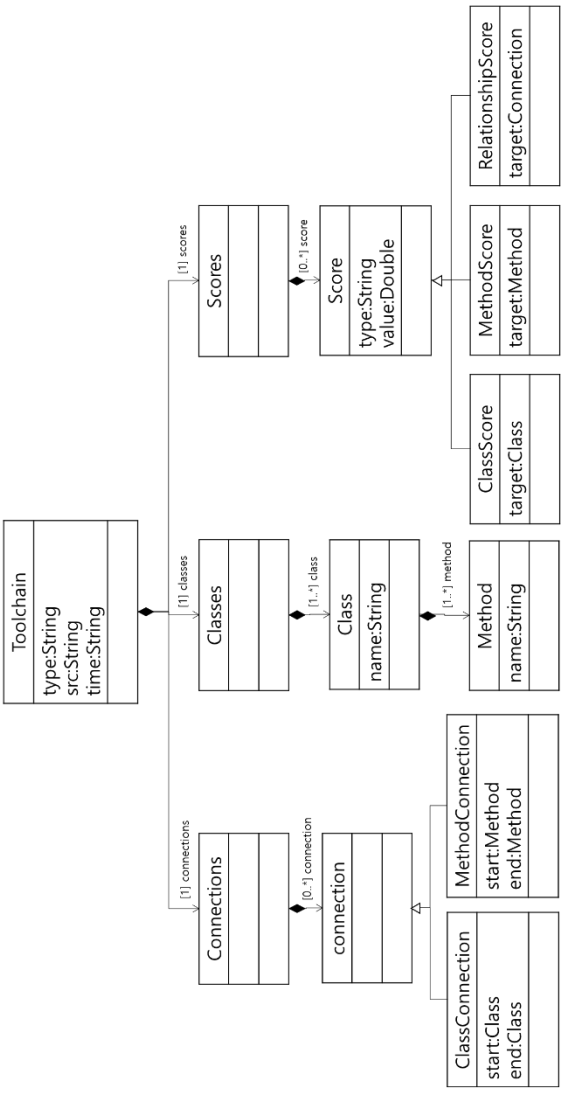

Fig. 4. Specialized Toolchain Metamodel

D. Metamodel Transformation Engine

Figure 5 shows the Bad Smell Transformation Language algorithm executed by Metamodel Transformation Engine. The Bad Smell Tolchain Metamodel components are automatically transformed into the Specialized Toolchain Metamodel components. For example, the src of Badsmell is automatically transformed to the src of Toolchain. This algorithm is implemented through Atlas Transformation Language (ATL)[9].

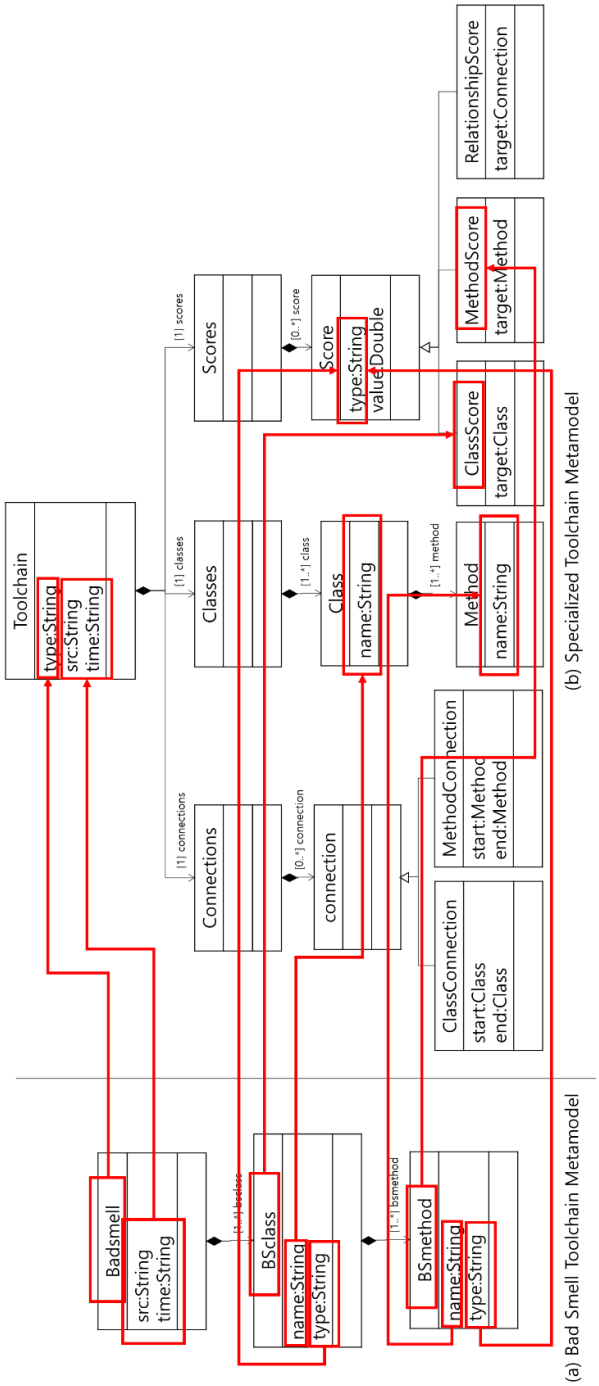

Fig. 5. Transformation Rule between Bad Smell Toolchain Metamodel and Specialized Toolchain Metamodel

E. *Bad Smell Toolchain Model*

The Bad Smell Toolchain Model stores the bad odor score of the source code. The model is stored as an XMI file. Table 1 is an example of the XMI code of the Bad Smell Toolchain Model.

Table 1. XMI Code of Bad Smell Toolchain Model

XMI Code

```
<?xml version="1.0" encoding="UTF-8"?>
<bs:badsmell xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:bs="http://tc/1.0" src="1_badsmell.jpg" time="2021/11/17-13:35:33">
 <bsclass name="A1" type="">
  <bsmethod name="A1" type="Large Class" />
  <bsmethod name="A1_run" type="" />
  <bsmethod name="A1_runA2" type="" />
 </bsclass>
 <bsclass name="A2" type="Data Class">
  <bsmethod name="A2" type="" />
  <bsmethod name="A2_run" type="" />
 </bsclass>
 <bsclass name="Main" type="">
  <bsmethod name="Main" type="" />
  <bsmethod name="Main_main" type="" />
 </bsclass>
</bs:badsmell>
```

F. *Specialized Toolchain Model*

The Specialized Toolchain Model can store information on all Toolchain models. Table 2 is an example of the XMI code of the Specialized Toolchain Model automatically transformed from the XMI code of Table 1.

Table 2. XMI Code of Specialized Toolchain Model

XMI Code

```
<tc:Toolchain xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tc="http://tc/1.0" type="badsmell" src="1_badsmell.jpg"
time="2021/11/17-13:35:33">
 <connections>
  <connection xmi:startclass="" endclass=""/>
 </connections>
 <classes>
  <class name="A1">
   <method name="A1"/>
   <method name="A1_run"/>
   <method name="A1_runA2"/>
  </class>
  <class name="A2">
   <method name="A2"/>
   <method name="A2_run"/>
  </class>
  <class name="Main">
   <method name="Main"/>
   <method name="Main_main"/>
  </class>
 </classes>
 <scores>
  <score target="//@class.0" type="Large Class" value="" />
  <score target="//@class.1" type="Data Class" value="" />
 </scores>
</tc:Toolchain>
```

IV. CASE STUDY

Figure 6 shows the implementation results of plug&play Environment for Toolchain in Process Visualization plug&play Environment. Users can select one of Bad Smell, Reusability, and Quality (C++ Quality) and upload the source code to measure the source code's score.

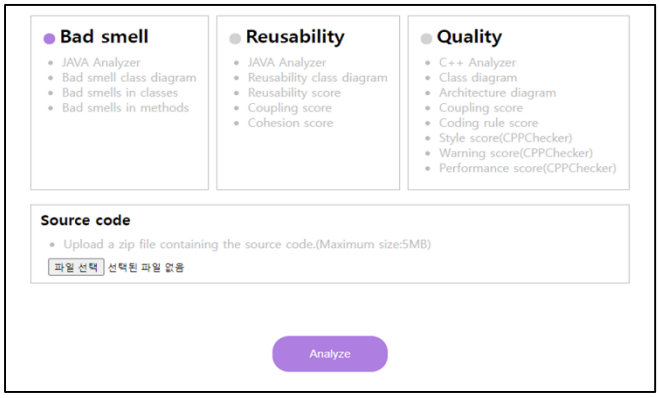

Fig. 6. Implementation of Plug&Play Environment for Toolchain

Figure 7 is the result of measuring the Bad Smell score of the source code. Bad Smell scores of classes and methods are displayed on the dashboard.

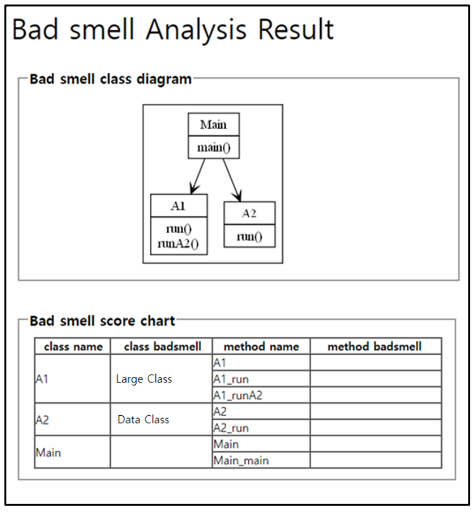

Fig. 7. Printed Bad Smell Score in the Dashboard

V. CONCLUSIONS

We propose an environment for free plug&play of tools in Process Visualization. Software Visualization uses a variety of tools to visualize organizational structures and processes. However, it is difficult to change once you select a tool. Our study supports easy plug&play of tools in a software visualization environment. As a case study, three toolchain models were transformed into specialized toolchain models and output to the dashboard. If the user wants to add a new Toolchain Model, the user can only perform the model transformation by adding a new model's metamodel (XMI file) and model transformation rules (ATL file). At this time, the source code of the engine is not changed.

In the future, we will study the plug&play environment for Architecture Visualization and Automatic Documentation.

ACKNOWLEDGMENT

The research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1I1A305040711) and the Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2021R1I1A1A01044060) and the BK21 FOUR (Fostering Outstanding Universities for Research) funded by the Ministry of Education (MOE, Korea) (F21YY8102068).

REFERENCES

[1] B.K. Park, H.E. Kwon, H.S. Son, Y.S. Kim, S.E. Lee, R.Y.C. Kim, "A Case Study on Improving SW Quality through Software Visualization", Korean Institute of Information Scientists and Engineers, Vol.41, No.11, pp.935-942, Nov 2014.

[2] NIPA Software Engineering Center, “SW Development Quality Management Manual”, Dec 2013.

[3] W.S. Jang, J.H. Kim, R.Y.C. Kim, "Best Practice on Software Traceability Environment based on PaaS Cloud Service", The International Journal of Advanced Smart Convergence(IJASC), Vol.9, No.4, pp.149-155, 2020.

[4] Redmine, https://www.redmine.org/, December 2021.

[5] Jira, https://www.atlassian.com/software/jira, December 2021.

[6] Jenkins, https://www.jenkins.io/, December 2021.

[7] Buddy, https://buddy.works/, December 2021.

[8] J.H. Park, H.S. Son, R..Y.C. Kim, "Developing an Automatic Tool for Visualizing Source Code against Bad Smell Patterns", Global Conference on Engineering and Applied Science, 2017.

[9] ATL, https://www.eclipse.org/atl/, December 2021.