Development of Plug-in Platform for Metrics Measurement

Akira Saito, Goro Yamada, Tatsuya Miyake, Yoshiki Higo, Shinji Kusumoto, Katsuro Inoue

† Graduate School of Information Science and Technology, Osaka University

Metrics Assessment plugin-platform for Software Unit of multiple programming languages

Problems

Many metrics tools have been proposed

However, these are three problems

Solutions

- Metrics measurement is completely separated from source code analysis
 - Users easily implement their original logic
 - MASU stores the result of analysis to internal database
 - Users only have to implement measurement logic as a plug-in

Ambiguous definitions

A metric has several definitions

Each tool has its own measurement logic

 High cost of implementation
 In case of new metrics, source code analysis has to be implemented in addition to logic of measurement

Lack of compatibility

Existing tools can hardly handle multiple programming languages MASU handles multiple programming languages
Java and C#

For instance, a RFC plug-in is presented as below:

import jp.ac.osaka_u.ist.sel.metricstool.main.plugin.AbstractClassMetricPlugin; import ...;

```
public class RfcPlugin extends AbstractClassMetricPlugin {
    protected Number measureClassMetric(TargetClassInfo targetClass) {
        final Set<CallableUnitInfo> rfcMethods = new HashSet<CallableUnitInfo>();
```

```
// gets defined methods
final Set<TargetMethodInfo> localMethods = targetClass.getDefinedMethods();
rfcMethods.addAll(localMethods);
```

```
// gets called methods from localmethods
for (final TargetMethodInfo m : localMethods) {
    rfcMethods.addAll(MethodCallInfo.getCallees(m.getCalls()));
```

```
return new Integer(rfcMethods.size());
```

```
protected String getDescription() { return "Measuring the RFC metric."; }
protected String getMetricName() { return "RFC"; }
protected boolean useFieldInfo() { return true; }
protected boolean useMethodInfo() { return true; }
```





Development of Eclipse Front-end

File met

Class m

Method

Field me

GUI front-end of MASU

Easily measure the metrics from Eclipse

			/	MASU Metrics Measurement	X
Ƴ Select output	MASU Metrics Measurement	s M	Select metrics to measure At least one plugin must be selected		I MASU
			Available Metrics FLOC	add > Metrics t	o measure
Output to:		1	MLOC	< remove	
View in Eclipse					
○ Text(CSV) files					
Output file paths		_		<< remove all	

Features

- Selects files or directories that you want to measure the metrics
- Two types of output format
 - Outputs the result to view in Eclipse
 - Also shows difference of the result measured before and after
 - Exports the result to a text file

In the future work, we will implement the function of generating a template of MASU plug-ins

rics:	Bro Plugin Description This plugin measures the LCOM1 metric The LCOM1 is one of the class cohes Bro Bro Bro	ric(CK-metrics's LCOM). ion metrics measured by following steps: ch do not share any field.If all methods do ich share some fields. as P - Q , otherwise 0.
< <u>B</u> ack <u>N</u> ext >	 ✓ masu_launcher.view ≅ metric name date	> <u>F</u> inish Cancel
of	target name gppa.data.FileInformation#getDirectory() gppa.data.FileInformation#getFileName() gppa.data.FileList#createPatternInstanceList(gppa.data.PatternList) gppa.data.Group#addPattern(gppa.data.Pattern) gppa.data.Group#setRoot(gppa.data.Pattern) gppa.data.Group#getRoot() gppa.data.Group#getGID() gppa.data.Group#getGID() gppa.data.GroupList#putNewGroup(gppa.data.Pattern gppa.data.Patter gppa.data.GroupList#mergeAll(gppa.data.Pattern java.util.TreeSet <gp gppa.data.GroupList#mergeGroup(gppa.data.Pattern gppa.data.Group gppa.data.GroupList#mergeGroup(gppa.data.Pattern gppa.data.GroupList#mergeGroup(gppa.data.Pattern gppa.data.GroupList#mergeGroup(gppa.data.PatternList) gppa.data.GroupList#mergeGroup(gppa.data.PatternList)</gp 	value ^ 3 = 3 = 12 4 3 = 3 = 3 = 3 = 3 = 3 = 3 = 3 = 3 = 4 = 9 = 11 = 18 = 81 = 3 =

http://sourceforge.net/projects/masu/