MAIJ - Metric Analyser and Identification for Java

A Guide to adding your own metrics.

Implementing the API---------------------------------------------------------------------

To make a metric you need to create a class that extends the metricAnalysis.Metrics.Metric class.

The framework supports 4 different levels of metrics:

Project Level - metrics that are evaluated on the entire project

Package Level - metrics that are evaluated on a single package

Class Level - metrics that are evaluated on a single class/Interface/Enum

Method Level - metrics that are evaluated on a single method/constructor

When making your custom metric you need to indicate which level the metric may be applied on

by setting the appropriate MetricLevel enum values in the levels field. eg.

public class LOCMetric extends Metric {

/\*\*

\* Sets the identifier and the levels that the metric may be applied on

\*/

public LOCMetric(){

super();

this.levels = Arrays.asList(MetricLevel.PROJECT, MetricLevel.PACKAGE, MetricLevel.CLASS, MetricLevel.METHOD);

}

Along with that you need to implement the appropriate run methods for the levels selected.

Project Level - run(ProjectAST):void

Package Level - run(PackageAST):void

Class Level - run(ClassAST):void

Method Level - run(MethodAST):void

where the parameters are the wrapper class for the Abstract Syntax Tree of the source code.

They represent a particular project version, package, class or method and may be used

to access the parsed abstract syntax tree or the source code file directly. See the JavaDoc

for more details.

Storing the Result---------------------------------------------------------------------

Once the result is calculated it needs to be stored in a MetricResult object which has two

constructors

public MetricResult(String identifier, Double i)

public MetricResult(String identifier, Double i, List<Range> poi)

where identifier is the metric identifier and i is the calculated value of the metric

poi is an optional list or JavaParser.Range objects to indicate points of interest in the file

where the value of the metric was updated. Refer to the JavaParser documentation for more details.

Once created, the MetricResult object needs to be added to the wrapper AST class via

x.getResults().put(this.identifier, MetricResult)

which will store it so that it may be read by the program.

Registering the Metric---------------------------------------------------------------------

Once the metric is done you need to register it in the metricAnalysis.AnalysisHandler class by adding

it to the registeredMetrics field, simply add it to the static section where indicated using the given

format:

public class AnalysisHandler {

private static final Map<String,Class<?>> registeredMetrics;

static {

Map<String, Class<?>> aMap = new HashMap<String, Class<?>>();

aMap.put(new ATFDMetric().getIdentifier(), ATFDMetric.class);

aMap.put(new CycloMetric().getIdentifier(), CycloMetric.class);

aMap.put(new LOCMetric().getIdentifier(), LOCMetric.class);

aMap.put(new MaxNestingMetric().getIdentifier(), MaxNestingMetric.class);

aMap.put(new NOPAMetric().getIdentifier(), NOPAMetric.class);

aMap.put(new NOSMetric().getIdentifier(), NOSMetric.class);

aMap.put(new NOAMMetric().getIdentifier(), NOAMMetric.class);

aMap.put(new FanOutMetric().getIdentifier(), FanOutMetric.class);

---->>>>> aMap.put(new CustomMetric().getIdentifier(), CustomMetric.class);

//Register new metrics here in the same format as Above

registeredMetrics = Collections.unmodifiableMap(aMap);

}

After that you'll be able to use your metric with the program.

Calculating the Value---------------------------------------------------------------------

The existing metrics use the parsed Abstract Syntax Tree to calculate their values

Use the existing metrics source code as examples and refer to the JavaParser and

JavaParserSymbolSolver documentation for more details.

Alternatively you may wish to work with the source code directly, such as when using third party

libraries or a custom parser, in which case you may get the files or folder from the AST.

For a method level metric you may get the file path using

methAst.getNode().findCompilationUnit().get().getStorage().get().getPath();

For a class level metric you may get the file path using

clasAst.getNode().findCompilationUnit().get().getStorage().get().getPath();

for a package level metric you may get the file paths using.

packAst.getCUS(); to get the set of Compilation Units and for each of those

cu.get.get().getStorage().get().getPath(); to get the file path.

or use cu.getStorage().get().getSourceRoot(); to get the path of the package directory

but the last will only work when the package name and file structure are the same.

For a project level metric use

projAst.getDirectory(); to the the path of the directory used to import the project.

In this manner you should be able to wrap third party tools so that the program may use

them.