TOWARDS A 'UNIVERSAL' SOFTWARE METRICS TOOL
Motivation, Process and a Prototype

Gordana Rakić, Zoran Budimac
Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Novi Sad, Serbia

Klaus Bothe
Institute of Informatics, Humboldt University, Berlin, Germany

Keywords: Software engineering, Software metrics, Software metrics tool, Compiler construction, Parser generator.

Abstract: In this paper we investigate main limitations of actual software metrics techniques/tools, propose a unified intermediate representation for calculation of software metrics, and describe a promising prototype of a new metrics tool. The motivation was the evident lack of wider utilization of software metrics in raising the quality of software products.

1 INTRODUCTION

Software metric (SM) can be defined as a numerical value that reflects some property of: a whole software product, its one piece or its specification. There are numerous categorizations of SM. Considering the measurement target, metrics could be divided in three main categories: product metrics, process metrics and project metrics (Kahn, 2003). In this paper we shall deal with the product metrics and primarily code metrics as its sub-category.

SM tools are being used for calculation, processing and analysis of the SM values. Improvements in the field of SM tools, such is creating of a new SM tool with advanced features may lead to better results of software projects.

This paper will introduce the reader to the development of one such tool. However, in this paper we concentrate on just some aspects of such a tool – independency on input programming language (IPL) and on SM algorithms to be applied.

Motivations behind designing a new tool lay in reports on existing tools' flaws and in tools review (section 2). We list some of recognized flaws:

- SM tools are generally not independent on IPL. The different tools are often used for different projects, for different software components, or even within a single component.
- SM tools usually compute only a subset of possible SM and rarely combine them to gain higher measure quality.
- SM tools rarely interpret the meaning of computed numerical results and their correlations in order to suggest what typical actions should be taken in order to improve the quality.
- SM tools are usually insensitive to the existence of additional, useless and duplicate code, as well as to attempts to 'cheat' the metrics algorithm.

Developing an SM tool that will solve the enumerated flaws would increase the level of application of SM in practice and improve the development process and final product quality. This is underlined as our implicit objective.

The paper is organized as follows. In section 2 the state of the art and open problems in the field will be presented. This has been used as a guideline for a development of a new tool. Section 3 explains the process of designing a tool. Description of the developed prototype of the new tool follows in section 4, and conclusions and further work are given in section 5.

2 RELATED WORK

One of the main problems in wider application of SM techniques and tools lays in limitations and inadequacy of available tools.

With the intention to discover main of mentioned weaknesses, review of available SM tools has been

Rakić G., Budimac Z. and Bothe K. (2010).
TOWARDS A 'UNIVERSAL' SOFTWARE METRICS TOOL - Motivation, Process and a Prototype.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 263-266
DOI: 10.5220/0002916502630266

Table 1: The overview of the results of software metric tools review.

Tool	See ref	Platform independ.	IPL independ.	Supported SM					Code hist.	Metrics storing
				CC	H	LOC	OO	others		
SLOC	(Wheeler, 2009)	-	+	-	-	+	-	-	-	+
Code Counter Pro	(Geronesoft, 2009)	-	+	-	-	+	-	-	-	+
Source Monitor	(Campwood Software, 2009)	-	-	+	-	+	+	+	-	+
Understand	(ScientificToolworks, 2009)	+	-	+	+	+	+	+	-	+
RSM	(MSquaredTechnologies, 2009)	+	-	+	+	+	+	+	+	+
Krakatau	Power Software, 2009	-	+*	+	+	+	+	+	-	+

done. Criteria for evaluation of each of analyzed tool are related to the possibility of wide usage of the tool. Those are: platform dependency, IPL dependency, and supported SM. Additionally, following two criteria are related to storing of produced results and intermediate results: history of code and metrics storing facility.

The analysis included 20 tools, but actual situation can be represented by restricted set of six representative tools (Table 1). Symbol "+" in a cell of the table indicates that listed tool possess corresponding characteristic, while "-" indicates that this criterion is not satisfied. Mark "*" next to the symbol "+" means that tool only partially satisfies specified criterion.

The table contains analysis of support for the following SM: Cyclomatic Complexity (CC), Halstead Metrics (H), Lines of Code (LOC) SM family including Comment LOC (CLOC), Source LOC (SLOC), etc., Object Oriented Metrics (OO) and any other SM which is not in list. For details see (Kan S., 2003).

The most important conclusions of the review follow.

- Available tools could be divided in two categories. The first category includes tools that calculate only simple metrics as are metrics from LOC family, but for wide set of IPL. The second category is characterized with wide range of metrics, but limited to small set of IPL. There are attempts to bridge the gap between these categories, but without final success. This is a big limitation if we take into account that currently most software projects are being written in more than one PL, usually different by nature and type. There is also a significant number of legacy software written in 'ancient' languages such are FORTRAN and COBOL. To all these subsystems, one and uniform SM tool should be applied to get reliable and uniform results and interpretation.
- Even if tools support some object-oriented metrics, this is still weak point of available tools, in opposite to the wide application of the object-oriented approach in software development.

General conclusion is that a new tool is needed.

3 TOWARD THE NEW TOOL

The basic idea is to split complete tool development in three steps with the following explicit goals for each step (Figure 1):

- Step 1 - to generate an appropriate intermediate structure for the representation of a source code to which SM algorithms can be applied.
- Step 2 - to apply SM algorithms to the given structure and to produce appropriate numerical values as a result.
- Step 3 - to apply advanced algorithms to the values of SM calculated in step 2, in order to produce more usable information to the end user.

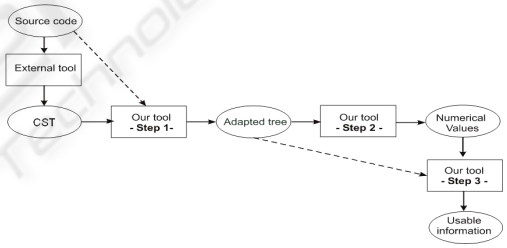

Figure 1: Development roadmap.

3.1 Step 1 - Intermediate Structure

Achieving the IPL independency requires creating a special intermediate structure for particular program representation. Such structure has to be suitable for representing source code written in "any" IPL.

Many other tools aiming for language independency show (e.g., Christodoulakis et al., 1989; CodeSquale, 2009) that usual intermediate structure for this purpose is some sort of syntax tree.

The basic idea is to start from parser generator (e.g., ANTLR (Parr, 2007)) which as input receives a IPL grammar and as output provides the IPL scanner and parser.

Parser generators usually generate Abstract (AST) and Concrete (CST) Syntax Tree, as intermediate structures. Structure and content of these trees is determined by IPL grammar and used

parser generator. The CST contains all information about IPL constructions and elements of the source code, so it would be possible to apply SM algorithms to such structure directly.

ANTLR generates AST and CST which is easy to be extended with additional (imaginary) nodes and enriched with additional information. This enrichment is possible by simple changes inserted in the IPL grammar – see (Parr, 2007) for details.

Generally, structure of the CST is always the same and independent on the IPL. This is not the case for the content of the nodes which differs for different languages even if it represents analogue IPL constructions. Application of SM algorithms to this structure requires modification of the CST to avoid implementation of SM algorithms for each IPL.

For this purpose a separate tree structure that is suitable for representation of the CST generated by the parser generator was developed and called 'enriched CST' (eCST). It is based on XML structure that provides independency with respect to the IPL and SM. Tree representation of the source code prepared in this way is the starting point in the second step.

3.2 Step 2 - Calculating Metrics Values

The eCST representation of the source code is the starting point for implementation of as many SM algorithms as possible and to produce rich enough set of numerical characteristics of the source. The set of SM which is to be calculated consists of code metrics and other SM which could be calculated on a source code represented by the given structure.

Calculated values should be stored and well organized for further manipulation in the third step.

3.3 Step 3 - Usable Information

After application of all SM algorithms and collecting required values, calculated data should be input parameters to advanced algorithms for delivering useful information to the end user in the form of advice for improving the software product or its elements.

4 THE PROTOTYPE

Determination of the eCST structure was based on:

- comparative analysis of application of a single SM to different IPLs;
- comparative analysis of application of different SM to a single IPL.

eCST is designed to be suitable for unique representation of a source code written in different IPLs and for application of different SM algorithms (see section 4.1)

The prototype of the new SM tool has been implemented in Java. It dynamically recognizes IPL, after which the source code is being parsed and eCTS is generated and stored to an XML file (section 4.1). Production of eCST is a result of a simple modification of the language grammar rules by adding generation of imaginary nodes in the tree.

The calculated SM values are also stored into XML document together with brief information about corresponding elements of the source code.

4.1 Storing the Generated eCST

Generated eCST consists of nodes and branches. Some of the nodes are imaginary and provide useful additional information about structure of the source code and IPL elements. These imaginary nodes have been added by modification of IPL grammar to enrich the tree for IPL independency purpose by enabling application of unique implementation of the SM algorithms for different languages.

For example one of the CC calculation algorithms is based on counting certain IPL constructs indicating loops, branches, logical operations, etc. These constructs are usually different in different IPLs. This is the reason for adding unique imaginary node before each branch, each loop, etc. which will initiate recognition and counting of the factor independently of IPL.

This tree modification does not affect the structure of the tree. Each node consists of general data about character and position of the source code element and possible sub-nodes. This is basic structure of syntax trees, and parsers generated by different parser generators are usually producing trees in that or in slightly modified form.XML schema for keeping generated eCST is presented in figure 2.

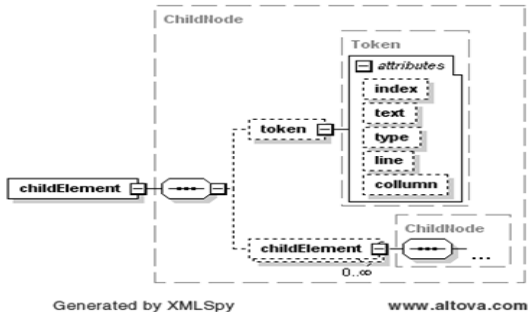

Figure 2: XML structure of an eCST.

The following example shows how the simple "if" statement is stored to the given structure. Let the statement that we want to store be the following one.

```
if (a >= b) //SomeStatement;
```

Let part "//SomeStatement(s)" represents list of statements. The graphical representation of the matching part of the eCST is presented in figure 3.

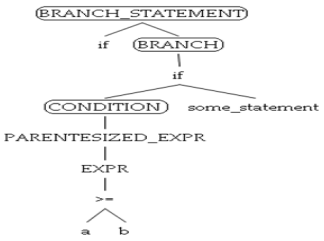

Figure 3: Simple "if" statement.

Figure 4 illustrates equivalent part of the XML tree.

Figure 4: XML tree representing "if" statement.

"BRANCH_STATEMENT", "BRANCH" and "CONDITION" are imaginary nodes added to achieve IPL independency. "BRANCH_STATEMENT" represents the beginning of the block that contains "if" branching. It may contain one or more sub-trees whose root is node named "BRANCH". It represents start of the each branch in the branching block. Moreover, each sub-tree that contains single branch may contain sub-tree representing condition. Root of this sub-tree is "CONDITION" node.

5 CONCLUSIONS

SM tools are at this moment a weak point in SM field and their wider application because of the numerous limitations of available implementations.

In this paper, the most important weaknesses in this area have been examined and presented together with possible solutions. In that direction the basic idea for development of a new SM tool and its prototype were proposed.

Current prototype works for IPLs Modula-2 and Java, calculating two characteristic SM (LOC and CC). It is based on usage of parser generator producing eCST which is stored in XML structure.

The immediate following task is to add more IPLs by generating appropriate scanners and parsers. Similarly much more SM algorithms will be supported, primarily by adding imaginary nodes.

ACKNOWLEDGEMENTS

We acknowledge support of DAAD (German Academic Exchange Service), project "Software Engineering: Computer Science Education and Research Cooperation" for partial support of the reported work. We are also grateful to a bilateral project between Serbia and Slovenia (project no. 27) that enabled the exchange of visits and ideas with colleagues of Faculty of Electronics, Computing and Informatics (Maribor, Slovenia).

REFERENCES

CampwoodSoftware, 2009, *Source Monitor*, http://www.campwoodsw.com/sourcemonitor.html

Christodoulakis D.N, Tsalidis C, C.J.M. van Gogh,. Stinesen V.W, 1989, Towards an automated tool for Software Certification, , *International Workshop on Tools for Artificial Intelligence,. Architectures, Languages and Algorithms*, IEEE, ISBN: 0-8186-1984-8, pp. 670-676

CodeSquale, 2009, http://codesquale.googlepages.com/

Geronesoft, 2009, *Code Counter Pro* http://www.geronesoft.com/

Kan S., 2003, *Metrics and Models in Software Quality Engineering - Second Edition*, Addison-Wesley, Boston, ISBN 0-201-72915-6

MSquaredTechnologies, 2009, *Resource Standard Metrics – RSM*, http://msquaredtechnologies.com/

Parr T., 2007, *The Definitive ANTLR Reference - Building Domain-Specific Languages*, The Pragmatic Bookshelf, USA, ISBN: 0-9787392-5-6

PowerSoftware, 2009, *Krakatau Essential PM (KEPM)-User guide 1.11.0.0*, http://www.powersoftware.com/

PowerSoftware, 2009, *Krakatau Suite Management Overview*, http://www.powersoftware.com/

ScientificToolworks, 2009, *Understand 2.0 User Guide and Reference Manual* March 2008, http://www.scitools.com

Wheeler D. A., 2009, *SLOCCount User's Guide*, http://www.dwheeler.com/sloccount/