Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 7 (2013) 336 – 343

Procedia Technology

The 2013 Iberoamerican Conference on Electronics Engineering and Computer Science

A Methodology for Obtaining Universal Software Code Metrics

Alberto Núñez-Varela[a], Hector G. Perez-Gonzalez[a], Juan Carlos Cuevas-Tello[a], Carlos Soubervielle-Montalvo[a]

[a]*Facultad de Ingeneieria, Universidad Autonoma de San Luis Potosi, Manuel Nava No.8, San Luis Potosí, 78216, Mexico*

Abstract

The development of quality software is a basic requirement that must be observed. Measuring software is a tool that allows the development of quality software for its entire life cycle. For software measurement, software metrics are used, among other techniques, which allow us to obtain a numerical value from a software product. There are two problems with these measurements: a value obtained can have different meanings depending on the project and what is desired as a result from the measurement, and the other problem is that the number and type of measurements is limited by the capabilities of the used tool. This paper presents a promising solution to the problem above by presenting a technique with which users can obtain any desired metrics and apply them to code in any programming language.

Keywords: Software measurement. Software Metrics, Software Code Metrics

1. Introduction

Software measurement is the process by which we can map a software product or process attribute to a numeric value. Comparing these values with a set of standards (defined by each individual, software group, organization, etc.), it is possible to obtain conclusions about the software quality. Software measurement is accomplished through metrics [1].

Software metrics deal with the measurement of the software product and the process by which it is developed. They are commonly classified in process metrics and product metrics. In this paper we focus

* Corresponding author. Tel. +52 (444) 826 2330 al 36 ext. 2111.
E-mail addresses: alberto_snv@hotmail.com (Alberto Núñez-Varela), hectorgerardo@acm.org (Hector G. Perez-Gonzalez), cuevas@uaslp.mx (Juan Carlos Cuevas-Tello), csober22@gmail.com (Carlos Soubervielle-Montalvo)

doi:10.1016/j.protcy.2013.04.042

exclusively in product metrics, and more precisely in those metrics related to the software source code and how to collect them.

Product metrics are measures of the software product at any stage of its development; from requirements to installed system and are widely used by the software industry for their measurement processes, including industries widely recognized [1].

A large number of product metrics have been defined, including a number of variants for each metric. Due to the lack of well-established standards and methodologies for software metrics measurement, a large number of tools have been developed in order to help the users to collect them.

The process of choosing the tool that best suits the user needs is not simple, since we cannot assume that all tools may handle and support the same metrics and programming languages the same way [2], mainly because products metrics are sometimes so ambiguous that the interpretation and implementation can be very different from tool to tool, and programming languages are always increasing in size and in number. What is more important is that tools used to measure product metrics only allow to analyze a reduced set of programming languages, mostly those of major use such as: C/C++, C# and Java. But even if the tool natively incorporates a language, there is always the problem of extensibility; if the language modifies or incorporates new syntax the tools will not be able to analyze some code until a new version is released. Even though the last represents a big problem, the major difficulty is that there are hundreds of programming languages which code cannot be measured because existing tools do not support them.

A number of issues related to metrics tools are: Accept a relatively small number of languages, they accept a predefined set of metrics, the user does not define the metrics, some tools are specific to a particular programming paradigm, especially for structured programming or object-oriented programming and the user cannot apply conditions to the output.

We do not want to adjust the set of metrics and programming languages we use in our software processes according to the tool or tools we have in hand, the tools must be adapted to our software processes; otherwise, we can end up with results that were not initially desired. This last statement is supported by the existence of well-established software measurement ontologies such as Goal Question Metric (GQM) [3, 4, 5] and Practical Software and Systems Measurement (PSM) [7]. In these methodologies is intended that the user has complete control in the measurements that he wants to perform according to his needs, so there are no limitations regarding the measurement processes.

To accomplish that goal, it is necessary a generic, extensible and scalable method of source code measurement for collecting metrics that can be applied to any language. We propose a method that allows us to obtain metrics from any source code, written in any programming language.

The remainder of this paper is organized as follows: Section 2 presents the related work to this research. Section 3 introduces our methodology for generic source code measurement. Section 4 presents some experiments and results and section 5 finally presents conclusions and further research.

2. Software Metrics Technology

Here we present some measurement methodologies, software metric models and architectures, commercial and non-commercial software metric tools.

2.1. Measurement Methodologies

There are several research works about Software Measurement Ontologies, which pretend to generate software measurement standards. Two of the most representative methodologies are Goal Question Metric (GQM) and Practical Software and Systems Measurement (PSM). These methodologies define processes that

must be done to make measurements "successful". Some other methodologies and common practices have been proposed as in [8]. On top of these ontologies some well-defined meta-languages are constructed that allow the creation of software measurement generic models. These kind of languages are commonly know as Domain Specific Languages (DSL) and provide a degree of high-level abstraction with no distinction between programming languages, they describe a common model of what to measure, but they do not define how to measure. Software Measurement Language (SMML) is a proposed meta-language that allows the user to define this kind of generic models [9]; other more powerful languages such as ATL (ATLAS Transformation Language) [10] and QVT [11] also allow describing transformations that can be made between the models.

In order to get the "real" product measurements it is necessary to use different techniques or tools to collect the metrics from the source code. There is no much research on generic solution on this topic, and this might be due to the ever-increasing variety of programming languages and the differences between their syntax definitions. B.I. Cogan proposed a measurement approach based on language rather than on the model [12], so measurements are defined directly in the software itself. A marked grammar defines the syntax of the language along with the metrics (if possible), the grammar then is matched against the input source code and compiles into the value of the metric. Other common implementations collect the metrics through control structures, such as trees or graphs that represent the structure of the code [13].

We point out that from the above methodologies, there is not methodology that can obtain any metric for any language, because each metric is defined directly according to the syntax of the language under analysis. In some cases, the metric is defined according to its intermediate representations (parse tree or control graph). This paper presents a methodology in which the metrics are based on the language syntax definition but are defined independently. There is no evidence in the literature and previous use of this approach.

2.2. Software Metric Models, Architectures and tools

Some researchers have defined and developed Software Measurement Framework Models with which they are intended to define theoretically a set of metrics that are specific to a paradigm, application or use in particular. These frameworks also define how to obtain these metrics from source code, but not all provide the tools to accomplish this. Example of a Framework that defines a set of metrics for object-oriented paradigm is found in literature [9].

Many commercial and non-commercial tools for collecting source code metrics have been developed over the years. Some of these tools use measurement techniques like the ones described previously. In the Introduction of this paper we listed a set of issues regarding the use of these tools, but the most important problem is that each tool supports just a finite set of languages and metrics. Table 1 shows a short list of commercial and open source code tools indicating the supported programming languages and metrics (actual information when conducting the investigation, could have changed to this date).

The tool that supports the greater number of languages is McCabeIQ, but the set of metrics it supports is somehow reduced. RSM is the tool that supports the greater number of metrics, but it only supports four languages. With a superficial review of the table, it can be determined that there is not a middle point between the characteristics of each tool, forcing the user to use two or more of these tools for his measurement processes, with the disadvantage of having to fit to the characteristics of the tools limiting measurement needs.

One of the most recent tools we could find is the one developed from the DXCore Framework, which is a command line tool that allows the user to collect three different predefined metrics, but also allows creating metrics using the components in the Framework. The creation of user defined metrics is a good characteristic of the tool, but unfortunately it has two main drawbacks: the user has to hard code the metrics and the new

metrics must be created using limited information about the source code given by the framework components, reducing the set of metrics that can be created.

Table 1. Software Metrics Tools

Tool	Metrics	Languages
McCabeIQ	43 metrics including: cyclomatic complexity, Halstead metrics, lines of code, etc.	Ada, ASM86, C, C#, C++.NET, C++, COBOL, FORTRAN, JAVA, JSP, Perl, PL1, VB. .NET
CMT++	Cyclomatic complexity, Halstead metrics, lines of code.	C, C++
CMTJava	Cyclomatic complexity, Halstead metrics, lines of code.	Java
RSM	Around 100 metrics: lines of code, number of functions and classes, etc. All metrics by class, namespace, package, etc.	C, C++, Java, C#
JMetric	Cyclomatic complexity, LDC, number of classes, packages,etc	Java
Essential metrics	Counting lines metrics, cyclomatic complexity, Halstead metrics, object oriented metrics, etc.	C, C++, Java
DXCore	Cyclomatic complexity, line count, maintenance complexity	C#, Visual basic, C++, XML, HTML, XAML, ASP.NET

3. Generic Source Code Measurement

To achieve source code generic measurement, we propose a measurement technique that will allow obtaining a set of substrings arranged with a common meaning of the source code of a program regardless of the programming language in which it is written. These set of substrings will have a specific meaning; a common meaning in software measurement, that is, well known and defined product metrics like the ones presented in [16], or a meaning given by the user.

The generic measurement will be achieved by allowing the user to query the source code directly. The result of these queries will be the set of substrings obtained from the source code.

3.1. Queries

The queries are generated from the programming language syntax definition whose source code is going to be measured. The syntax definition can be written in definition languages such as the BNF notation and Context-Free grammars (CFG). In this paper we use the CFG as our syntax definition language. Each non-terminal symbol in the grammar, individually or linked with other as described below, will form a path. A set of two or more related symbols is called a path chain, this chain will be considered as a query. The result of the query will be a set of substrings extracted from the source code that correspond to the real lexemes which are read from the source code that match the patterns defined by the terminal symbols (tokens). For syntax purposes, each path in the chain is linked by the symbol "!" as shown in the example in section 3.3.

3.2. Path Tree

A Path Tree is a simplified representation of the grammar defining the language. The tree contains all the valid, non-ambiguous and non-recursive paths that can be obtained from the grammar. The tree is built from the non-terminals symbols of the grammar and always respects the parent/children relationship rule in the

path construction. Each tree node is labeled with a non-terminal symbol and may have from zero to n children. To construct the path tree the following rules apply:

1. The root node of the tree is labeled with the symbol found on the left side of the initial production.
2. For each node N labeled with the symbol S is due to meet to:
    2.1. The labels of child nodes of N may not be repeated and are different from S.
    2.2. In the branch that is built from node N, node N can only be labeled with the symbol S.

Valid paths are those that are formed from a top - down tour from any node to any other node. If the paths begin to form from the root of the tree (which is the start symbol by rule 1) does not get any ambiguous path. Rule 2 prevents recursion.

3.3. A practical example

In this example we use the grammar shown in Fig 1.a, which allows defining a set of classes whose content is a list of variable declarations. This grammar generates the tree of paths shown in Fig 1.b. We will use the source code shown Fig 1.c as input:

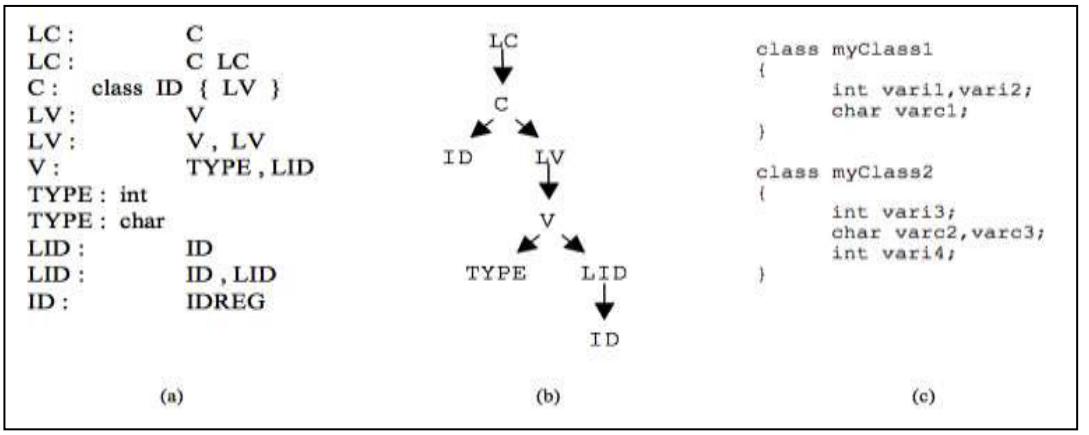

Figure 1. (a) Language grammar; (b) Tree of paths; (c) Example code

To view the operation and how it generates the response we will take the first declared class. Figure 2 shows how the input will divide according to the grammar.

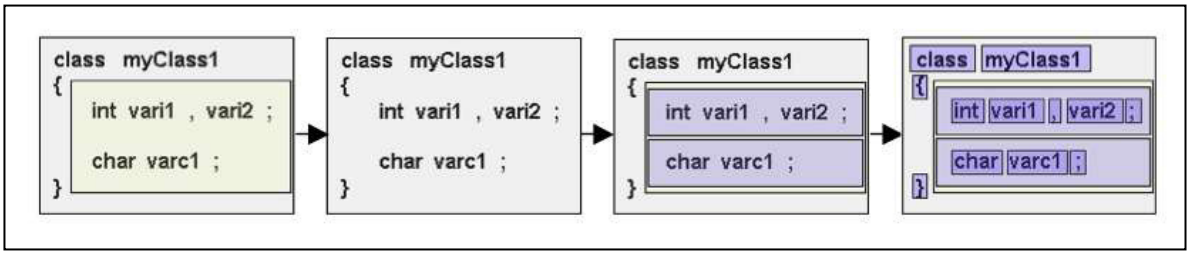

Figure 2: Graphical representation of the divisions that are generated from the context-free grammar during parsing.

Focusing on the text portion derived from LV we have the representation shown in Fig. 3

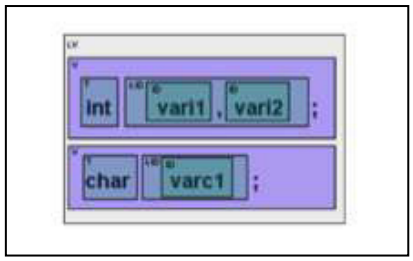

Figure 3: Graphic representation of a portion of the input and its relationship with the path tree.

With the knowledge obtained, many queries can be performed. Fig 4 shows queries generated to obtain the metrics: Names of all classes and Names of all declared variables.

Meaning: Names of all classes
Query: *LC!C!ID*
Result:

myClass1
myClass2

Meaning: Names of all declared variables
Query: *LC!C!LV!V!LID!ID*
Result:

vari1
vari2
varc1
vari3
varc2
varc3
vari4

Figure 4: Queries generated to obtain the metrics: Names of all classes and Names of all declared variables

Queries shown in figure 4 contain only one route, adding more routes to the chain gives depth to the results. Figure 5 shows queries generated to obtain the metrics: Declarations of variables per class and Variables declared by type and class

Meaning: Declarations of variables per class
Query: *LC!C!ID?LC!C!LV!V*
Result:

myClass1	int vari1,vari2
	char varc1
myClass2	int vari3
	char varc2,varc3
	int vari4

Meaning: Variables declared by type and class
Query: *LC!C!ID?LC!C!LV!V!TYPE?LC!C!LV!V!LID!ID*
Result:

myClass1	int	vari1
		vari2
	char	varc1
myClass2	int	vari3
	char	varc2
		varc3
	int	vari4

Figure 5: Queries generated to obtain the metrics: Declarations of variables per class and Variables declared by type and class.

4. Experiments and Future Work

To test the methodology presented in this work, we measured a C# project with around 20 classes and 7000 lines of code. We generated most of the well known set metrics such as: Namespaces used in each class, Code from each class, Class's names, Class's y inheritance, Methods per class, Methods per class and its parameters, Parameter per method and class average, All data types used in variables, Member variables per class, Fan-In, Cyclomatic complexity, etc.

We are working on a table comparison between the results from our methodology and results from existing software metric tools so we can take conclusions on whether our methodology can supersede those tools.

We are also working on proposing a Measurement Query Language (MQL) as a script-based language that extends the query notation. In general, each query is a script that can be extended applying operators to each path that conforms the query, conditions, built-in functions, etc. The actions and modifications that can be generated by these extensions are applied to the query output; so, no additional modification is required in the algorithms presented here making this language independent and scalable.

This methodology can measure any text in general, so any text written in native languages such as Spanish or English can be measured. Some queries of interest are: all the words, all the numbers or all the symbols that the text contains, how many sentences or paragraphs compose the text, how many words a paragraph contains, how many times a given word is repeated, etc.

As mentioned before, the methodology presented is not only useful for source code measurement, but for any regular language accepted by the parser. The input for any of these languages is always text, so we can affirm that the methodology can be used for generic text measurement.

With support of this methodology several applications of general use can be created such as:

- Metric tools with user interaction and user defined languages and metrics.
- Applications for source code automatic generation of documentation.
- Any type of application involving XML documents.
- Any type of application involving XHTML documents for web pages measurement.
- Any application involving text can be created or supported by this methodology.

5. Conclusions

In this work we present a generic method for text measurement that can be applied in a wide range of applications. The use given to this method here is for collecting software product metrics. According to the best of our knowledge, there is not a generic method similar to the one introduced in this paper.

The use of this methodology as a base for the creation of other applications is useful for both developers and the end users. The applications built on top of it are scalable and provide a high level of interaction with the user and functionality. Even though the final objective of this method is to really be able to obtain any characteristic of a text, it is not possible to determine if that objective can be accomplish because hundreds of different queries can be made along with many special cases of consideration. The method presented here was designed for source code measurement in general, in order to obtain the largest number of metrics. Fortunately the method is extensible to query any text, extending its range of application, not only in the computer science area. Another very important point of this method is that queries can be conditioned. This represents something new in the field of measurement because it allows increasing significantly the number of queries that can be made. Possibly the greatest achievement when implementing this method is that any application can be powered with hundreds of languages definitions and queries, making unnecessary the use of metrics extraction tools such as those mentioned in this article. This methodology can achieve the same

results generated by measurement tools but with a greater degree of accuracy, because the user has complete control of what he wants to query. As a future work we propose the development of a more intuitive generic software metrics generator tool using the concepts proposed here.

6. References

[1] Ordonez, MJ., Haddad, HM. The State of Metrics in Software Industry. Fifth International Conference on Information Technology: New Generations, April 2008 Page(s):453 - 458

[2] Rüdiger L., Lundberg J. and Löwe W. Comparing Software Metrics Tools. ISSTA'08, July 20–24, 2008, Seattle, Washington, USA.

[3] Mettam GR, Adams LB. How to prepare an electronic version of your article. In: Jones BS, Smith RZ, editors.Introduction to the electronic age, New York: E-Publishing Inc; 1999, p. 281-304

[4] Basili V. Applying the Goal/Question/Metric Paradigm in the Experience Factory. Proceedings of the Tenth Annual CSR (Centre for Software Reliability) Workshop, Application of Software Metrics and Quality Assurance in Industry, Amsterdam, Holland, 29th September - 1st October, 1993.

[5] Basili, V. Caldiera, G. and Rombach HD. Goal Question Metric Approach. Encyclopedia of Software Engineering, , John Wiley & Sons, Inc., 1994. pp. 528-532.

[6] Basili, V. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. University of Maryland, CS-TR-2956, UMIACS-TR-92-96, September 1992.

[7] Clark, E., Dean J., and Hall, F., Practical Software Measurement: Objective Information for Decision Makers, Addison-Wesley, 2001

[8] Garcia, F., Ruiz, F. Calero, C. Bertoa, MF. Vallecillo, A., Mora1, B and Piattini M. Effective use of ontologies in software measurement. The Knowledge Engineering Review. Volume 24 , Issue 1 (March 2009). pp. 23-40.

[9] Mora, B., García, F. Ruiz, F., Piattini M. SMML: Software Measurement Modeling Language. CIbSE 2009, Medellín, Colombia, Abril -17, 2009. pp.: 181-194

[10] Jouault, F., Allilaire, F. , Bézivin, J. Kurtev, I and Valduriez P.. ATL: a QVT-like Transformation Language, 2006.

[11] OMG, "QVT Standard Specification," 2005.

[12] Cogan, BI, ,Hunter RB. Language-based approaches to software measurement. Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results, 1996. pp: 3-9

[13] Cogan, B. Shalfeeva E. A Generalized Structural Model of Structured Programs for Software Metrics Definition. Software Quality Journal, 10, 2002, pp.149–167.

[14] Mao, M. Jiang Y. A Coherent Object-Oriented (OO) Software Metric Framework Model: Software Engineering. International Conference on Computer Science and Software Engineering, Volume 2, 12-14 Dec. 2008. pp: 68 - 72

[15] Lincke, R., Lundberg, J. and Löwe, J, Comparing software metrics tools, Proceedings of the 2008 international symposium on Software testing and analysis, July 20-24, 2008, Seattle, WA, USA

[16] Mills E. Software Metrics. SEI Curriculum Module SEI-CM-12-1.1, December 1988. Carnegie Mellon University, Software Engineering Institute.