The Robust Software Metric Data Model Defined in XML

Ng Keng Yap*, Abdul Azim Abdul Ghani, Ali Mamat, Hazura Zulzalil

*Address: Faculty of Computer Science, Universiti Putra Malaysia, 43300 UPM Serdang, Malaysia
*Tel: +6019-3833969, +603-8946 6502, Fax: +603-89466576

Summary
Software metric data model has always been restructured, redefined to fit their respective software metrics, and yet it will never been permanently shaped. It is important to have a generic data model to handle software measurement data in digital format that can actually help making software metrics definition in future much robust, definable and structured. In this paper software metric data model is defined in eXtensible Markup Language (XML) with three main characteristics: portability, extensibility and also reusability. The newly defined software metric data model is SMML. SMML has been tested via proof of concepts through build and evaluate methodology. A testing toolkit and an Application Programming Interface (API) were produced in helping the evaluation of SMML viability. The model has been tested robust with its portability, extensibility and reusability.

Keywords:
Software Metric Database, Software Measurement Data Model, Software Metric, XML Data Model

1. Introduction

Several studies have been carried out to model software measurement data [1, 4, 7, 8, 10]. Some of the studies have been complemented with respective prototypes to prove their concepts [1,10]. Some of these models, however, have not mentioned about the underlying data format used. Nowadays, software development rarely happens in a homogeneous environment, but in various heterogeneous and diverse environments comprising embedded systems, parallel systems, hand-held devices, network and distributed systems, etc. that run on different operating systems. Some literatures have also suggested that data collection should be automated and the metric data thence collected should be stored in a repository [1, 2]. When development happens in a heterogeneous environment, automated data collection and direct storage to the database can be very difficult and troublesome owing to the architectural difference. In such cases, a low-level data format may play its role to map and store the data before storing them into the database. There are several reasons why a low-level data format in software measurement is necessary. The following list shows some fundamental requirements of metric data:

i. the data need to be transferable among measurement tools from different vendors,
ii. the data need to be portable among different operating systems,
iii. the data need to be portable to machines with different architectures, including embedded devices and future systems,
iv. the data format needs to conform to the principles of software measurement, to be more specific and definable via valid software measurement methodologies (scientific),
v. the measurement data need to be reusable among related measurement programs, and
vi. the data structure needs to be visible and definable to man.

In this paper, a data format that can map the metric data using the eXtensible Markup Language (XML) based on the measurement concept asserted by Barbara Kitchenham [5] and ISO/IEC15939 [3, 6] has been proposed. It has been also made possible to handle software quality attributes [11]. The data model has been implemented and tested using the build and evaluates approach by March and Smith. An application programming interface (API) was developed and a testing toolkit was implemented to test for its robustness, portable, extensible and reusable. The evaluation gives positive results.

The paper organization is as follows: In section 2, some related works to metric data modelling have been discussed. In section 3, a much robust software metric data model has been proposed, and section 4 depicts how the model was tested against some predefined criteria. Section 5 concludes and discusses some inconsistency found in regular software metric and data modelling, and some possible future work to this model is discussed in section 6.

2. Related Works

Barbara Kitchenham and Shari Lawrence Pfleeger suggested a validation framework for software measurement [5]. They defined the principles and caveats of software measurement by associating software measurement with other general scientific measures that are widely used. Norman Fenton and Shari Lawrence Pfleeger jointly authored a book on software metric to define some practical approaches towards software metric [2]. They again defined a similar software measurement concept.

Manuscript received February 5, 2008
Manuscript revised February 20, 2008

Barbara Kitchenham et al. specified a model for software data sets in order to capture the definitions and relationships among software measures [1]. The model also suggested the importance of metadata in the definition of software measurement. Metadata in software measurement would define measurement protocols and data specification. The measurement protocol defined by Barbara Kitchenham et al. is closely related to the data collection process. It defines by whom, when and how software metric data should be collected, whereas data specification includes (i) the definition of the entities being measured, (ii) the definition of each attribute measured on each entity, (iii) the definition of each unit for each measurable attribute (if applicable), (iv) the scale type associated with attribute-unit pair, and (v) the definition of scales.

There are also other measurement models that are used to model software measurement on specific kinds of software [4, 8, 9]. Some of these models have roughly mentioned about their underlying data structures, but no details have been given on how they designed the data (structure and relationships). Data defined in different ways in different proposed models have made the data difficult to analyze and reuse. It is obvious that software engineering still lacks scientific methods and notations to define software measurement functions unlike what is available to other scientific fields like physics and chemistry. Measurement units, scales, and protocols that are applicable in the measurement of the attributes of an entity must be clearly stated and validated, especially during data analysis, conversion of software metric units and execution of software measurement functions.

Some of the measurement paradigms [1, 2] assert the importance of measurement definition. Measurements are usually defined in a document called the measurement plan. During the data collection or measurement plan execution, this information must be incorporated into or associated with the data collected. In the measurement plan, we must mention clearly the protocol, that is, what, how and when the *entity* is to be measured. Similar to the case with metric data, these factors can have a profound impact upon the data validity during data analysis and measurement function execution. If possible, the data should bear a reference to the description of every entity, or what is specifically known as the metadata.

3. The Robust Data Model

Metadata to the software metric data are obviously important to denote the aforementioned specifications and protocols. Storing data in the Relational Database (RDB) without the associated metadata such as units as well is protocols is prone to analysis failure. The noted problems have prompted us to invent a data format that can fulfil the following requirements:

i. cross-platform, loadable on any open systems and operating systems,
ii. markup of software metric data with the aforementioned specifications and protocols,
iii. human and machine readable so that it can support automated and manual measurements,
iv. formal method of defining measurement plan without additional documents
v. structured and well-formed data structure that can be validated

Although there have been several attempts to create software metric data models [1, 10, 16], none of them have yet fulfilled all the requirements above. The practical way of manipulating collected metric data is to export them into the decision support systems (DSS), project management systems (PMS), benchmarking tools, automated tools, expert systems (ES), executive information systems (EIS) or all other kinds of information system to aid the software process life-cycle [14]. Hence, a formal data format must be created for these integrated systems to communicate and share a common data source. The proposed data model came from the advent Web, or more specifically HyperText Markup Language (HTML) which has created a common markup for different operating systems and web browsers to browse common data on the web servers. Markup is the way of incorporating metadata to data on the web server so that web browsers are able to interpret the data and display them in the correct manners and styles. As the Web's counterpart, markup of metric data may work out in a similar way. With the advent of XML [12, 13], which is an extension to HTML, has given us an opportunity to define software metric data using XML. The XML data must consist of the following elements:

i. the entity identification that clearly states the granularity,
ii. the metrics to be applied to an entity or attributes to be measured,
iii. magnitude or textual value for each metric,
iv. unit associated to each magnitude,
v. annotation or precaution statement for measurement execution,
vi. time-stamp to state the date of data collection,
vii. version number for collected data,
viii. data collector’s information,
ix. project information.

Appendix I attached at the end of the text shows 2 sample XML files that map the software metric data. sample.xml is the implementation of the software metric data model is an instance that marks up the magnitudes or values for entity: 'Software Requirement Specification Document, Chapter 01', Where as, project.xml maps to the project

related constants or final values. The project.xml is adapted from the Project Object Model (POM) proposed by the Apache's Maven Project [15]. Comments are marked and embraced within <!-- and --> in the XML files to give more comprehensive explanations.

The following briefs the elements that are used to model software metric data.

- Entity

As mentioned in Section 2, an entity comprises product, process and resource. In the development planning, we decompose the software product into higher grained granularity, notably submodules, modules, subsystems and systems; and decompose documentation into chapters and sections, development process into phases, and resources into single dependent units. All these decomposed entities must be clearly mapped to the metric data. Line 35 of sample.xml shows the said <*entity*> element. The <*entity*> element consists of a *name* attribute as the unique id to differentiate one from another.

- Metric

Each entity possesses many attributes. The act to measure those attributes contributes to the existence of <*metric*> elements nested within the <*entity*> element. <*metric*> elements will hold the name of the attributes being measured, which is unique, and the units are in each and every magnitude embraced within <*value*> tags. It is worth emphasizing that the *unit* attribute is compulsory for all magnitudes.

- Magnitude and textual values

Magnitudes or textual values are embraced within the <*value*> tags as shown in the lines 38, 41, 45, 48, 53 and so forth. The <*value*> tag may have an attribute known as *type* to denote the datatypes that are allowed for the metric and may have attributes called *min* and *max* to denote the higher and lower bounds of the value.

- Unit

As mentioned above, the *unit* attribute is compulsory for all magnitudes, and it is often used to denote the scale of measures.

- Annotation

The annotation is denoted by the <*annotation*> tag. It may appear as the first sub-element in the <*metric*> or <*entity*> elements. The purpose of annotation is to give remarks to the data collector, especially during manual data collection, about some precautions or caveats that should be paid particular attention to.

- Version number

Version number is used to differentiate among different collections of data in the <*meta*> element. Version numbers that appear in the <*tool*>, <*entity*>, <*function*> elements are used to denote the version of the automated tool used, the entity, and function used for derived measure respectively.

- Time-stamp

A time-stamp is very important in data collection to denote the date of the data collection. The <*timestamp*> element is used for the said purpose and placed within the <*meta*> tags.

- Collector's Information

The Collector's information for a piece of metric data is very handy when we need to contact the collector if we have any doubts about the data. The <*collector*> element will hold the collectors personal information: <*name*>, <*user_id*>, <*email*>, <*designation*>, <*organization*> *and* <*contact*>. If the data have been collected with an automated tool, then the name of the tool and its version number must be enclosed within the <*meta*> element.

- Project Information

The project information is the data regarding the project, which are considered constant, or final, which means these values cannot be reassigned. The Project information is adapted from the Project Object Model (POM) proposed by Apache's Maven Project [15]. The project.xml file is an example adapted from POM. POM can be added to the sample.xml or have it referred to associate the data with a particular project.

4. Build and Evaluate SMML

The proposed data model was tested with Build and Evaluate approach proposed by March and Smith for design sciences was adopted. Build refers to the construction and realization of the software metric data model, i.e. SMML, and the implementation of the model in SMML API. This phase confirms that the model can be constructed and implemented. The SMML API and SMML Toolkit were implemented as per Fig. 1.

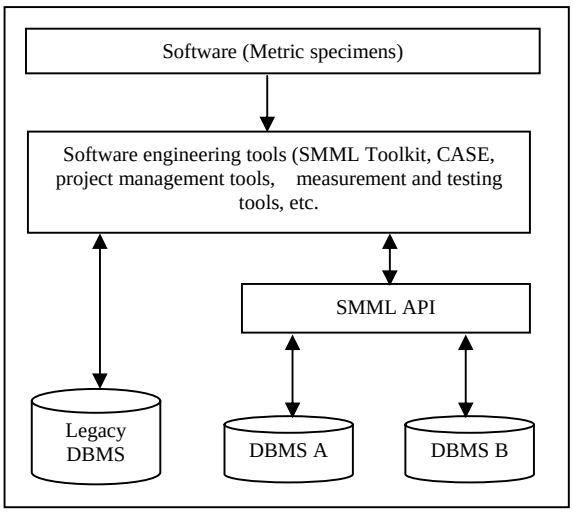

Fig. 1: The architecture of SMML implementation.

The above architecture depicts where SMML API plays its role in filling up the gap between structured software metric data in RDBMS and software engineering tools. The SMML API is implemented in Java programming language. It helps in transforming structured data from the RDB databases into XML format like appendix 1. The extraction, transformation and loading of data from the RDB to the XML data can be partial and selective based on criteria such as by project, date or module. SMML API adopted their respective JDBC technology to fill up the communicate gap between itself and different RDB provided by different vendors. Software engineering such CASE tools, project management tools, etc can be developed and extended it capability to adopt software metric data model with SMML API to gain access to RDB. The comprehensive data model was implemented in order to enable evaluation of the model in latter stage.

Evaluate refers to the development of criteria and the assessment of the output performance against those criteria of the model. There were three main characteristics were tested in during the evaluation phases. In order to evaluate the SMML API, the realization of the model, we developed the SMML Toolkit using SMML API to manipulate software metric data. The SMML Toolkit was then tested for the follow criteria to see if the implemented model complies with the criteria: portability, extensibility and reusability. Portability was tested at three levels namely operating system, database as well as application levels. SMML Toolkit was tested based on the test cases defined to evaluate if SMML Toolkit conforms to all the aforementioned criteria. The test cases used were:

a. Test case 1: The SMML Toolkit must be able to run on several common environments, Microsoft Windows, Linux distro Fedora Core 4 were selected in the experiment. VMware Workstation™ system virtualization was used to enable multiple operating systems to run on a single Pentium 4 machine. These operating systems were used through out the evaluation process which includes Test Case 1, 2, 3, 4, 5.
b. Test case 2: SMML Toolkit must be able to be used by other. It was used in the Web Usability Testing Tool developed by Kamsiah Mohamed [15]
c. The application made use of the SMML API, keeps metric data in SMML API objects for analysis purpose.
d. Test case 3: SMML Toolkit must be able to extract, transform and load data even if different databases are used. SMML Toolkit was tested on MySQL and Microsoft SQL Server via their respective JDBCs.
e. Test case 4: SMML Toolkit must be able to load and merge two or more data chunks, and save them into the database.
f. Test case 5: SMML Toolkit must allow new metric definition to be extended to the data model.

The test cases were running on a personal computer with Pentium 4 processor, 512 megabytes of memory, with VMware Workstation™ virtualization installed with Linux Fedora Core 4 and Microsoft Windows XP Professional. The experiment shown positive results as Table 1 below:

Table 1: Evaluation Result

	Linux			Windows		
	Ex	Tr	Ld	Ex	Tr	ld
Test case 1	✓	✓	✓	✓	✓	✓
Test case 2	✓	✓	✓	✓	✓	✓
Test case 3	✓	✓	✓	✓	✓	✓
Test case 4	✓	✓	✓	✓	✓	✓
Test case 5	✓	✓	✓	✓	✓	✓

Legends:
✓ : The test case ran with positive result.
Ex : Extracting data from the RDMBS
Tr : Transforming to XML format and vice versa
Ld : Loading back to RDB

5. Discussion and Conclusion

The use of XML in software metric data will definitely make software metric data more definable and unified for all software engineering processes. It also reduces the incompatibility of the software metric data during analysis. Current stage of research will test on the feasibility of XML in metric data mapping. There are still many factors to be taken into the account to make it more flexible across software engineering processes, such as measurement instrument to units and scales standard, scientific methodologies used in software engineering, etc. These factors give a great impact to the software metric data analysis. Research to make this technique more reliable and definable is necessary so that the software industry can meet a consensus on the standardized data format for software metric in future.

6. Future Works

This model is tested using Build and Evaluate approach. SMML Toolkit were used to evaluate the SMML API. More tools can be developed using SMML API to ensure that data are exported to relational database easily regardless the variant of RDB. With the advents object oriented database and OR mapping tools, SMML API can be extended to enable software metric data to be stored on object oriented database.

Acknowledgement
This project is funded by the Ministry of Science, Technology and Innovation (formerly known as the Ministry of Science, Technology and Environment), Malaysia under its Intensification of Research in Priority Areas (IRPA) research grant. We sincerely thank all parties who have been involved and rendered their supports directly or indirectly to the development of this project. Should some credits go to them.

REFERENCES

[1] B.A. Kichenham and R.T. Hughes (2001). "Modeling Software Measurement Data". IEEE pp788-804.
[2] N.E. Fenton and S.L. Pfleeger (1998). "Software Metrics: A Rigorous and Practical Approach", PWS.
[3] ISO/IEC15939:2001 (2001). International Standard, Software Engineering-"Software Measurement Process".
[4] D. A. Lamb and J. R. Abounader (1997). "Data Model for Object-Oriented Design Metrics," Queen's University, Kingston, ON.
[5] B.A. Kitchenham, S.L. Pfleeger and N.E. Fenton (1997). "Towards a Framework for Software Measurement Validation", IEEE Transactions on Software Engineering, 21(12), pp929-944.
[6] J. McGary, D. Card, C. Jones, B. Layman, E. Clark, J. Dean and F. Hall (2001). "Practical Software Measurement", Addison-Wesley.
[7] L. Olsina, G. Lafuente, O. Pastor (2002). "Towards a Reusable Repository for Web Metrics", Journal of Web Engineering 1(1): 61-73.
[8] L. Olsina, G. Rossi (2002). "Measuring Web Application Quality with WebQEM". IEEE MultiMedia 9(4), 20-29.
[9] S. H. Kan (1995). "Metrics and Models in Software Quality Engineering", Addison-Wesley.
[10] B.A. Kitchenham, S.G. Linkman, A. Pasquini, and V. Nanni (1997). "The SQUID Approach to Defining a Quality Model". Software Quality Journal, 6, pp 211-233.
[11] ISO/IEC 9126-1:2001 (2001). International Standard, Software Engineering - "Product Quality - Part 1: Quality Model".
[12] D. C. Fallside (2001) "XML Schema Part 0: Primer", http://www.w3.org/TR/xmlschema-0.
Last access: July 2007.
[13] Apache Software Foundation, "Maven Project", http://maven.apache.org. Last access: July 2007.
[14] A.W.B. Peter Hitchcock, R. Weedon, A.N. Earl, R.P. Whittington and D.S. Robinson (1986). "The Use of Databases for Software Engineering," *British National Conference on Databases* pp. 55-70.
[15] Kamsiah Mohamed, and Abdul Azim Abd. Ghani, and Hazura Zulzalil , and Azrina Kamaruddin , (2004) *Usability Metrics for Web Site: A Case Study*. In: Proceedings of the Joint Conference on Informatics and Research on Women in ICT (RWICT) 2004 , 28 - 30 July 2004 , Putra World Trade Center, Kuala Lumpur, Malaysia .

Ng Keng Yap is a tutor at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. He is a Master Science holder, graduated from the same university with his research dissertation entitled 'Development and Evaluation of a Software Metric Markup Language'. He has 7 years of working experience as a system analyst. He is currently working on research related to data model to be used spanning across software engineering process, in making the software measurement data much more structured and usable.

Abdul Azim Abdul Ghani is an associate professor cum dean at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. His research interests are software metrics, software ethics and cyberlaw besides software requirement engineering.

Ali Mamat is an associate professor in computer science at Universiti Putra Malaysia, Serdang, Malaysia. He obtained his PhD in computer science from University of Bradford, U.K. His interests include databases, XML and semantic web.

Hazura Zulzalil is a lecturer at the Department of Information Systems, Faculty of Computer Science and Information Technology, University Putra Malaysia (UPM).
She received her First Degree in Computer Science and Master of Science (Software Engineering) from UPM.
Currently, she is pursuing a PhD Degree at Faculty of Computer Science and Information Technology, UPM.
Her research interests include software metrics, software quality, software product evaluation and multi-criteria aggregation procedure.

Appendix I

sample.xml file

```
<?xml version="1.0"?>
<data>
    <!-- this portion hold the metadata of this data collection -->
    <meta>
        <!-- Information on data collection personnel -->
        <collector>
            <name> </name>
            <user_id> </user_id>
            <email> </email>
            <designation> </designation>
            <organization> </organization>
            <contact> </contact>
        </collector>
        <!-- if automated tool is used, this element will hold its information -->
        <tool>
            <name> </name>
            <version> </version>
            <vendor> </vendor>
            <type> </type>
            <description> </description>
        </tool>
        <!-- time-stamp of the data collection -->
        <timestamp> </timestamp>
        <!-- version of this data collection -->
        <version> </version>
    </meta>
    <!--  The POM element goes here, please refer to POM below.
    Alternative way of defining POM is using include tag like this:
    <include url="http://202.184.29.77/~wbc/project.xml" /> -->
    <project>
    ...
    ...
    ...
    </project>
    <entity name="SRS_Chap01" version="1.0">
        <annotation>Software Requirement Specification Document, Chapter 01</annotation>
        <metric name="size" unit="page">
            <value type="int"/>
     </metric>
     <metric name="manpower" unit="man">
            <value type="int"/>
     </metric>
     <metric name="dayUsed" unit="day">
            <annotation>{dayUsed: 1, 2, 3 ...}</annotation>
            <value type="int"/>
     </metric>
     <metric unit="manday" name="effort">
            <value type="double" format="##0.00">
            <?smml version="1.0" function="manpower*dayUsed"?>
            </value>
     </metric>
```

```
52.      <metric type="indirect" unit="page/manday" name="productivity">
53.           <value>
54.                <function language="java">
55.                     size/effort
56.                </function>
57.                <!--<?smml version="1.0" function="size/effort"?> -->
58.           </value>
59.      </metric>
60.   </entity>
61. </data>
```

project.xml file

```
1.  <!-- Project Object Model (POM) adapted from Apache's Maven project. -->
2.  <project>
3.      <version></version>
4.      <name></name>
5.      <id></id>
6.      <currentVersion></currentVersion>
7.      <organization>
8.          <name></name>
9.          <url></url>
10.     </organization>
11.     <inceptionYear></inceptionYear>
12.     <package></package>
13.     <shortDescription></shortDescription>
14.     <description></description>
15.     <url></url>
16.     <cvsWebUrl></cvsWebUrl>
17.     <issueTrackingUrl></issueTrackingUrl>
18.     <siteAddress></siteAddress>
19.     <siteDirectory></siteDirectory>
20.     <distributionDirectory></distributionDirectory>
21.     <cvsRoot></cvsRoot>
22.     <cvsModule></cvsModule>
23.
24.     <distributions>
25.       <distribution>
26.       <id></id>
27.       <version></version>
28.       <tag></tag>
29.       </distribution>
30.     </distributions>
31.     <branches/>
32.     <mailingLists>
33.       <mailingList>
34.           <name></name>
35.           <subscribe></subscribe>
36.           <unsubscribe></unsubscribe>
37.           <archive></archive>
38.       </mailingList>
39.     </mailingLists>
40.     <developers>
41.       <developer>
42.           <name></name>
```

```
43.             <id></id>
44.             <email></email>
45.             <organization>
46.                 <name></name>
47.                 <url></url>
48.               </organization>
49.         </developer>
50.         <developer>
51.             <name></name>
52.             <id></id>
53.             <email></email>
54.             <organization>
55.                 <name></name>
56.                 <url></url>
57.             </organization>
58.       </developer>
59.     </developers>
60.     <dependencies>
61.         <dependency>
62.             <name></name>
63.           <type></type>
64.           <version></version>
65.           <jar></jar>
66.       </dependency>
67.     </dependencies>
70.       ...
71. </project>
```