BugML: Software Bug Markup Language

Naresh Kumar Nagwani,
Assistant Professor,
Department of Computer Science & Engineering,
National Institute of Technology Raipur

Dr. Shrish Verma,
Associate Professor,
Department of Electronics & Telecommunication,
National Institute of Technology Raipur

ABSTRACT
A software bug contains useful information related to software development. Bug indicates the incorrect behavior of implemented functionalities for the given requirements. Numbers of standards exist to keep the software bug information. Most of the bug tracking tools follows these standards to store the software bug information. XML (eXtensible Markup Language) is another most common and famous standard to represent the data. In this paper the XML is used to represent the software bug information and a new markup language named BugML is proposed. The structure, DTD (Document Type Definition) and XSD (XML Schema Definition) is also explained for the language BugML.

Keywords
BugML, Bug Markup Language, Software Bug Representation, XML Representation.

1. INTRODUCTION

1.1 Software Bug
A software bug is defect or fault in software. A software bug is introduced into the system during the testing phase of software by a quality engineer or tester. A bug indicates the incorrect implementation of given software requirements, or sometimes it may be due to some of the technical limitations also. A software bug can be represented using number of attributes like bug title (or summary), bug description, bug-id (a unique identified for a software bug), date-of-reporting (date on which the bug is reported.), assigned-To (team member to whom the software bug is assigned.), reported-by (team member or tester by whom the bug is reported into the system.) etc.

1.2 Software Bug: Defect or Fault?
Although the defect and fault is treated same and both are considered as software bug, there are some differences between the fault and defects. A fault is a subtype of the super type defect. Every fault is a defect, but not every defect is a fault. A defect is a fault if it is encountered during software execution (thus causing a failure). A defect is not a fault if it is detected by inspection or static analysis and removed prior to executing the software.

1.3 Software Bug Repositories
The bug management of large size projects are typically done with the help of bug tracking tools like Bugzilla, Trac, JIRA, Perforce etc. These bug tracking tools provides various interfaces to log new bugs, access bug information and update it. These bug tracking tools manages the software bugs in online software bug repositories. Example of such bug repository is Mozilla bug repository https://bugzilla.mozilla.org. All these different bug repositories are having their own standard mechanism of storing the software bugs. So there is a need to standard representation technique using which the software bugs can be represented. The work proposed in this paper is to document and represent the standard XML based mark up language using which the software bug can be represented. This mark up language for software bug is named as BugML (Software Bug Markup Language).

1.4 IEEE standards for Software Bug Attributes
There are two standards of IEEE for software defect classifications [4-5]. IEEE Standard Classification for Software Anomalies, IEEE Std 1044-1999 and IEEE Standard Classification for Software Anomalies, IEEE Std 1044-2009 (Revision of IEEE Std 1044-1999). Various important attributes of software bugs are mentioned in those standards. These attributes are considered here to design the BugML. Various IEEE standard defect and failure attributes are shown in table-1 and table-2. The proposed BugML is designed using the mentioned standard attributes in these IEEE standard specifications.

Table 1. Defect Attributes Defined in IEEE Standard 1044-2009

Attribute	Definition
Defect ID	Unique identifier for the defect.
Description	Description of what is missing, wrong, or unnecessary.
Status	Current state within defect report life cycle.
Asset	The software asset (product, component, module, etc.) containing the defect.
Artifact	The specific software work product containing the defect.
Version detected	Identification of the software version in which the defect was detected.
Version corrected	Identification of the software version in which the defect was corrected.
Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.
Severity	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.
Probability	Probability of recurring failure caused by this defect.
Effect	The class of requirement that is impacted by a failure caused by a defect.
Type	A categorization based on the class of code within which the defect is found or the work product within which the defect is found.
Mode	A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.
Insertion activity	The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).
Detection activity	The activity during which the defect was detected (i.e., inspection or testing).
Failure reference(s)	Identifier of the failure(s) caused by the defect.
Change reference	Identifier of the corrective change request initiated to correct the defect.
Disposition	Final disposition of defect report upon closure.

Table 2. Failure Attributes Defined in IEEE Standard 1044-2009

Attribute	Definition
Failure ID	Unique identifier for the failure.
Status	Current state within failure report life cycle.
Title	Brief description of the failure for summary reporting purposes.
Description	Full description of the anomalous behavior and the conditions under which it occurred, including the sequence of events and/or user actions that preceded the failure.
Environment	Identification of the operating environment in which the failure was observed.
Configuration	Configuration details including relevant product and version identifiers.
Severity	As determined by (from the perspective of) the organization responsible for software engineering.
Analysis	Final results of causal analysis on conclusion of failure investigation.
Disposition	Final disposition of the failure report.
Observed by	Person who observed the failure (and from whom additional detail can be obtained).
Opened by	Person who opened (submitted) the failure report.
Assigned to	Person or organization assigned to investigate the cause of the failure.
Closed by	Person who closed the failure report.
Date observed	Date/time the failure was observed.
Date opened	Date/time the failure report is opened (submitted).
Date closed	Date/time the failure report is closed and the final disposition is assigned.
Test reference	Identification of the specific test being conducted (if any) when the failure occurred.
Incident reference	Identification of the associated incident if the failure report was precipitated by a service desk or help desk call/contact.
Defect reference	Identification of the defect asserted to be the cause of the failure.
Failure reference	Identification of a related failure report.

This paper is divided into the seven sections. Section two discusses about the related work done so far in the data representation techniques using XML. In section three overview of the BugML is given. Section four is consisting of DTD (Document Type Definition) of the BugML. In section five the BugML syntax is explained using example, section six is about the XSD (XML Schema Definition) for the BugML and the seventh section is consisting of the conclusion of the proposed work.

2. RELATED WORK

Markup languages are designed for various purposes. In this section some of the designed mark up languages is mentioned.

The aim of the Predictive Model Markup Language (PMML) is to support the exchange of data mining models between different applications and visualization tools. It is the result of a standardization effort by a group of vendors. PMML is an XML-based language (grammar) for describing data mining models. Despite its name, it is not limited to predictive models. The contribution of this paper is two-fold: an encouragement for researchers to base their work on PMML and a slightly enhanced PMML DTD for multi-relational rule models [1].

A XML application which provides the representation of Java source code is called JavaML, is more natural for tools and permits easy specification of numerous software-engineering analyses by leveraging the abundance of XML tools and techniques. A robust converter built with the Jikes Java compiler framework translates from the classical Java source code representation to JavaML, and an XSLT stylesheet converts from JavaML back into the classical textual form [2]. The design of RuleML, a rule markup language for the Semantic Web is given by Boley et. al RuleML implementations via XSLT is also explained [3].

Two popular XML based applications are XGMML and LOGML. XGMML is graph description language and LOGML is a web-log-report description language. The usefulness of both of the applications is also explained in web mining [6]. A framework is proposed by Suri and Singh which stores the design elements in the form of a text document using DGML (DesiGn Markup Language). A new syntax is created for DGML based documents. This representation of pictorials design elements in the form of text helps in design optimization, reusing the existing design and early prediction of error prone modules. A fresh new design can be obtained from existing design after parsing it for well defined project requirements [8].

3. OVERVIEW OF BUGML

BugML is proposed to provide a standard of storing the software bug information. Presently most of the minor functionalities implementation and enhancement tasks are also managed using the bug tracking systems, so BugML can also be used for maintaining the enhancement and minor functionalities implementation tasks. BugML is a structural document consisting of BugML tags. These tags are designed with parameters to store all the useful information regarding the software bugs. A well-defined DTD is also there for the grammatical verification of any document to be used for further processing. And XSD is also defined which can be used optionally to follow the schema of the bug documents.

```
<bug id=''>
<summary>          </summary>
<description user='' time=''></description>
<created-by id='' email=''>  <created-by>
<assigned-to id='' email=''> <assigned-to>
< environment>  </environment>
< status >  </ status >
< version-detected > </version-detected >
< version-corrected > </version-corrected>
<priority>          <priority>
< severity > </ severity>
<product> </product>
<component> </component>
<version> </version>
<platform> </platform>
<importance>  </importance>
<target-milestone> ---  </target-milestone>
<qa-contact> </qa-contact>
<reported>
        <reported-time> </reported-time>
        <reported-by id='1' user=''> </reported-by>
</reported>
<modified>  </modified>
< cc-list>
        <cc  id=''  email=''>                  </cc>
        <cc  id=''  email=''>                  </cc>
        ……
        <cc  id=''  email=''>                  </cc>
</ cc-list >
< bcc-list>
        <bcc  id=''  email=''>                 </bcc>
        <bcc  id=''  email=''>                 </bcc>
        ……
        <bcc  id=''  email=''>                 </bcc>
</ bcc-list >
<comments>
        <comment  id=''  user=''  time=''>
        </comment>
        <comment  id=''  user=''  time=''>
        </comment>
        ……
        <comment  id=''  user=''  time=''>
        </comment>
</comments>
</bug>
```

Fig. 1: Structure of BugML with Attributes.

Figure-1 depicts the structure of BugML with various attributes and Figure-2 shows a sample of Mozilla Bug.

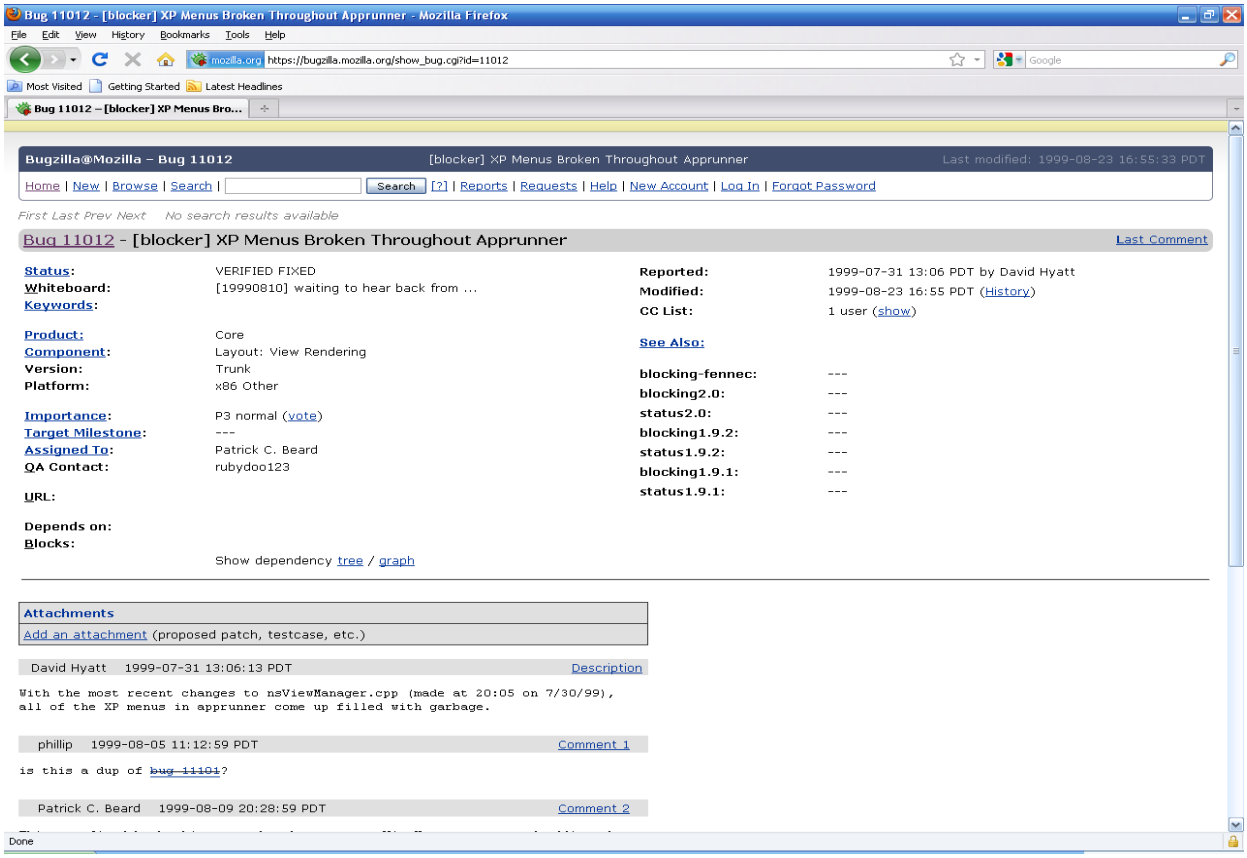

Fig.2: Snapshot of a Mozilla bug (bug id = 11012)

4. THE DTD FOR THE BUGML

All the BugML tags for software bug representation are having the DTD (Document Type Definition) entry for syntax check. A strong characterization of an XML document is validity. An XML document is valid if and only if it both is well-formed and adheres to its specified document type definition, or DTD. A document type definition is a formal description of the grammar of the specific language to be used by a class of XML documents. It defines all the permitted element names and describes the attributes that each kind of element may possess. It also restricts the structure of the nesting within a valid XML document. Figure 3 shows the DTD for the BugML. Here all the attributes and their corresponding data types and requirements are explained. Figure 5 depicts the DTD for the BugML.

```
<!ELEMENT assigned-to ( #PCDATA ) >

<!ELEMENT bug ( title, status, product, component, version,
platform, importance, target-milestone, assigned-to, qa-contact,
reported, modified, cc-list, description, comments ) >
<!ATTLIST bug id NMTOKEN #REQUIRED >
<!ELEMENT cc ( #PCDATA ) >
<!ATTLIST cc id NMTOKEN #REQUIRED >
<!ATTLIST cc user NMTOKEN #REQUIRED >
<!ELEMENT cc-list ( cc+ ) >
<!ELEMENT comment ( #PCDATA ) >
<!ATTLIST comment id NMTOKEN #REQUIRED >
<!ATTLIST comment time CDATA #REQUIRED >
<!ATTLIST comment user CDATA #REQUIRED >
<!ELEMENT comments ( comment+ ) >
<!ELEMENT component ( #PCDATA ) >
<!ELEMENT description ( #PCDATA ) >
<!ATTLIST description time CDATA #REQUIRED >
<!ATTLIST description user CDATA #REQUIRED >
<!ELEMENT importance ( #PCDATA ) >
<!ELEMENT modified ( #PCDATA ) >
<!ELEMENT platform ( #PCDATA ) >
<!ELEMENT product ( #PCDATA ) >
<!ELEMENT qa-contact ( #PCDATA ) >
<!ELEMENT reported ( reported-time, reported-by ) >
<!ELEMENT reported-by ( #PCDATA ) >
<!ATTLIST reported-by id NMTOKEN #REQUIRED >
<!ATTLIST reported-by user NMTOKEN #REQUIRED >
<!ELEMENT reported-time ( #PCDATA ) >
<!ELEMENT status ( #PCDATA ) >
<!ELEMENT target-milestone ( #PCDATA ) >
<!ELEMENT title ( #PCDATA ) >
<!ELEMENT version ( #PCDATA ) >
```

Fig. 3: DTD for BugML used to represent the software bug information.

According to this DTD, there are several element types. The bug element must contain exactly one id (unique bug identifier) followed by other bug attribute body element. The CC (Carbon Copy) mailing list can have zero or more number of elements. All the attributes like summary, description etc. are of text data types. The attribute reported is the combination of two elements reported-time and reported-by, which is time at which the software bug was reported and by whom the software bug was

reported. Just like CC mailing list, the comment list is there, where there can be zero of more number of comment elements can be associated with a software bug. For each comment element, comment-id, comment-user and comment-time are associated.

5. SYNTAX OF BUGML

To discuss the syntax of BugML, let us take the example of the software bug taken from an open source software's bug repository, named Mozilla bug repository [7] (https://bugzilla.mozilla.org). Here the software bug id is selected to demonstrate the BugML syntax. Most of the important BugML tags are used here to represent the bug document using BugML. The example of BugML syntax is given in figure 4, a Mozilla bug with bug id 11012 is represented using BugML syntax in this figure.

```
<bug id='11012'>

<title>[blocker] XP Menus Broken Throughout Apprunner
</title>

<status>VERIFIED FIXED</status>
<product>Core</product>
<component>Layout: View Rendering</component>
<version>Trunk</version>
<platform>x86 Other</platform>
<importance>P3 normal</importance>
<target-milestone> ---  </target-milestone>
<assigned-to>Patrick C. Beard</assigned-to>
<qa-contact>rubydoo123</qa-contact>
<reported>
        <reported-time>1999-07-31 13:06 PDT</reported-
time>
        <reported-by id='1' user='chofmann'> David Hyatt
</reported-by>
</reported>
<modified>  1999-08-23 16:55 PDT (History) </modified>

<cc-list>
        <cc id='1' user='chofmann'>chofmann</cc>
</cc-list>

<description user='David Hyatt' time='1999-07-31 13:06:13
PDT'>With the most recent changes to nsViewManager.cpp
(made at 20:05 on 7/30/99), \n all of the XP menus in apprunner
come up filled with garbage.\n  </description>

<comments>
        <comment id='1' user='phillip' time='1999-08-05
11:12:59 PDT'>is this a dup of bug 11101?</comment>
        <comment id='2' user='Patrick C. Beard' time='1999-
08-09 20:28:59 PDT'>This was fixed by backing out the
changes to nsViewManager.cpp, and adding them \n again more
carefully. Hyatt can verify this. </comment>
        <comment id='3' user='rubydoo123' time='1999-08-10
07:12:59 PDT'>Hyatt -- can you please verify this one?
thanks</comment>
        <comment id='4' user='David Hyatt' time='1999-08-11
10:19:59 PDT'>Yes.</comment>
        <comment id='5' user='rubydoo123' time='1999-08-23
16:55:59 PDT'>marking verified based on hyatt's
input.</comment>
</comments>

</bug>
```

Fig. 4: Syntax of BugML for Example Mozilla bug (bug id = 11012)

6. XSD FOR THE BUGML

XSD (Xml Schema Definition) for the BugML is represented in figure 5 for the corresponding BugML DTD given in figure-3.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
name="bug" type="xs:string">

  <xs:complexType>
        <xs:sequence>
                <xs:element ref="title"/>
                <xs:element ref="status"/>
                <xs:element ref="product"/>
                <xs:element ref="component"/>
                <xs:element ref="version"/>
                <xs:element ref="platform"/>
                <xs:element ref="importance"/>
                <xs:element ref="target-milestone"/>
                <xs:element ref="assigned-to"/>
                <xs:element ref="qa-contact"/>
                <xs:element ref="reported"/>
                <xs:element ref="modified"/>
                <xs:element ref="cc-list"/>
                <xs:element ref="description"/>
                <xs:element ref="comments"/>
        </xs:sequence>
    <xs:attribute name="id" type="xs:NMTOKEN"
         use="required"/>
  </xs:complexType>

 <xs:complexType>
     <xs:simpleContent>
       <xs:extension base="xs:string">
          <xs:attribute name="id" type="xs:NMTOKEN"
           use="required"/>
          <xs:attribute name="user" type="xs:NMTOKEN"
           use="required"/>
         </xs:extension>
      </xs:simpleContent>
 </xs:complexType>

 <xs:complexType>
    <xs:sequence>
          <xs:element maxOccurs="unbounded" ref="cc"/>
    </xs:sequence>
 </xs:complexType>

 <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
          <xs:attribute name="id" type="xs:NMTOKEN"
           use="required"/>
          <xs:attribute name="time" type="xs:string"
```

```
            use="required"/>
            <xs:attribute name="user" type="xs:string"
            use="required"/>
          </xs:extension>
      </xs:simpleContent>
  </xs:complexType>
  <xs:complexType>
     <xs:sequence>
            <xs:element maxOccurs="unbounded"
ref="comment"/>
      </xs:sequence>
   </xs:complexType>

  <xs:complexType>
      <xs:simpleContent>
            <xs:extension base="xs:string">
               <xs:attribute name="time" type="xs:string"
                 use="required"/>
              <xs:attribute name="user" type="xs:string"
                use="required"/>
            </xs:extension>
       </xs:simpleContent>
   </xs:complexType>


  <xs:complexType>
     <xs:sequence>
            <xs:element ref="reported-time"/>
            <xs:element ref="reported-by"/>
      </xs:sequence>
   </xs:complexType>


  <xs:complexType>
      <xs:simpleContent>
            <xs:extension base="xs:string">
               <xs:attribute name="id" type="xs:NMTOKEN"
                   use="required"/>
               <xs:attribute name="user" type="xs:NMTOKEN"
                  use="required"/>
            </xs:extension>
      </xs:simpleContent>
            </xs:complexType>
</xs:schema>
```

Fig. 5: XSD of BugML.

7. CONCLUSION

In this paper a new standard named BugML is proposed for storing the software bug information. The standard is based on XML, which is a very famous way for organizing the data. The DTD and XSD for the BugML are also discussed, and syntax of BugML is explained using example. The proposed standard can be used for any software bug repositories to represent the software bug information. This can also be used effectively for software bug data transfer using web services and other web standards of data transfer.

8. REFERENCES

[1] Dietrich Wettschereck and Stefan M¨uller, "Exchanging Data Mining Models with the Predictive Modelling Markup Language", In Proc. of Int. Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning, Freiburg, 2001.

[2] Greg J. Badros, "JavaML: A Markup Language for Java Source Code", In Proceedings of the 9th International World Wide Web Conference, 2000.

[3] Harold Boley, Said Tabet and Gerd Wagner, "Design Rationale of RuleML: A Markup Language for Semantic Web Rules", In International Semantic Web Work- ing Symposium (SWWS), 2001.

[4] IEEE Standard Classification for Software Anomalies, IEEE Std 1044-1999.

[5] IEEE Standard Classification for Software Anomalies, IEEE Std 1044-2009 (Revision of IEEE Std 1044-1999).

[6] John R. Punin, Mukkai S. Krishnamoorthy, Mohammed J. Zaki, "LOGML: Log Markup Language for Web Usage Mining", WEBKDD 2001 - Mining Web Log Data Across All Customers Touch Points, Third International Workshop, San Francisco, CA, USA, August 26, 2001. Revised Papers, volume 2356 of Lecture Notes in Computer Science, pages 88–112. Springer, 2002.

[7] Mozilla bug repository, https://bugzilla.mozilla.org.

[8] P. K. Suri and Gurdev Singh, "Framework to represent the software design elements in markup text – Design Markup Language (DGML)", IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.1, pp. 164- 170,January 2010.