Computing and Informatics, Vol. 30, 2011, 531–557

FAULT-BASED TEST OF XML SCHEMAS

Maria Claudia Figueiredo Pereira Emer, Igor Fabiano Nazar
Silvia Regina Vergilio, Mario Jino

Computer Science Department
Federal University of Paraná, Brazil
CP: 19081, CEP: 81531-970
e-mail: {mcemer, jino}@dca.fee.unicamp.br,
{ifnazar, silvia}@inf.ufpr.br

Communicated by Jacek Kitowski

Abstract. XML is largely used by most applications to exchange data among different software components. XML documents, in most cases, follow a grammar or schema that describes which elements and data types are expected by the application. These schemas are translated from specifications written in natural language, and consequently, in this process some mistakes are usually made. Because of this, faults can be introduced in the schemas, and incorrect XML documents can be validated, causing a failure in the application. Hence, to test schemas is a fundamental activity to ensure the integrity of the XML data. With the growing number of Web applications and increased use of XML, there is a demand for specific testing approaches and tools to test schemas. To fulfill this demand, this work introduces a fault-based approach for testing XML schemas. This approach is based on a classification of common faults found in schemas. A supporting tool was implemented and used in evaluation studies. The obtained results show the applicability of the fault-based testing in this context and its efficacy in revealing faults.

Keywords: Fault-based testing techniques, XML Schema, web applications

1 INTRODUCTION

XML (eXtensible Markup Language) [19] is a markup language that has been largely used to manipulate data from different sources. The use of XML to exchange data

among different components is rising, mainly in Web-based applications, such as e-commerce [14] and e-science [6].

XML documents need to be well-formed and valid. A well-formed document obeys XML syntax, and this can be detected by using a checker. A valid document follows rules defined in a schema that specifies the grammar expected for the document. Examples of schemas generally used are: DTD [21] and XML Schema [20].

Since developers are responsible for writing the schemas, some mistakes can be made and incorrect schemas can be generated. A faulty schema can validate an incorrect XML document. If such an incorrect document is manipulated, this can cause a failure in the application that uses the schema. This happens because during the validation process only syntactic aspects are checked. Semantic aspects of the documents are not considered, i.e., the meaning of the stored data in the XML document is not evaluated.

As a consequence, the use of testing techniques and criteria to reveal faults in the schemas is fundamental. They can ensure the data integrity and reliability of the XML documents, and consequently, of the application that manipulates them.

There are three basic testing techniques that can be used: functional, structural and fault-based. The fault-based technique considers the main faults that are generally found in the programs being tested to generate the test data. In some studies of the literature, it has been considered as the most efficacious in terms of the number of revealed faults [24]. Mutation Analysis [7] is the best known and most frequently used fault-based criterion.

We find in the literature some works that explore fault-based testing in the context of Web-based applications. The works described in [12, 15, 25] introduce XML based mutation operators with the goal of testing the interaction between Web components, mainly Web services; but they do not address the testing of schemas. Li and Miller [13] introduce mutation operators for XML schemas and show that schemas modified by these operators are often considered valid by most parsers commonly used to validate schemas. However, the authors do not present a testing process and do not explain how test cases are generated to detect faults in the schemas. Franzotte and Vergilio [10] propose new mutation operators and apply mutation analysis criterion on XML schemas. However, in the proposed testing process, the tester identifies the test data manually, and equivalent mutant schemas can be generated. The manual test data generation and the determination of equivalent schemas increase the costs and efforts of the testing activity.

Considering the promising ideas of the above-mentioned works, in our previous work [9], we introduced a fault-based approach for testing XML schemas, particularly the test of XML Schema documents. The idea of the introduced approach is the automatic generation of test data. Each test data is composed by an XML document that describes a typical fault and queries to this document. The testing result is the result obtained by executing such queries.

In the present paper we grouped the faults in classes and illustrate each class with examples. In addition to that, we implemented a supporting tool, named XTool, to make possible the use and evaluation of the introduced approach. This

tool is also described. By using XTool, we conducted experiments that show the applicability and efficacy of the approach, as well as allowing a comparison with mutation testing.

This paper is organized as follows: Section 2 reviews basic concepts and provides a background on XML, XML Schema and XML query languages. Section 3 describes related works. Section 4 introduces classes of the common faults found in XML schemas. Considering these classes, Section 5 introduces the fault-based approach. Section 6 presents experimental results. Section 7 contains conclusions and future work.

2 BASIC CONCEPTS

2.1 XML

XML (eXtensible Markup Language) is a text format used for data exchange among applications and heterogeneous hardware platforms [19]. XML was defined by W3C (World Wide Web Consortium). Data are represented in XML format as a hierarchy of elements specified by names, optional attributes and optional content. Tags are textual descriptions of data delimited by the symbols "<" and ">", which are used to indicate the logical structure of the document and to identify the data content. Example 1 presents a sample of an XML document. This document contains information about sellers.

Example 1: A fragment of XML document about sellers

```
<?xml version="1.0" ?>
<sellers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<seller sellerID="ac12DD23">
     <sellerName>Eduardo</sellerName>
     <nickName>du</nickName>
     <glancePage>0k</glancePage>
     <about>0k</about>
     <moreAbout>0k</moreAbout>
     <averageFeedbackRating>7</averageFeedbackRating>
     <totalFeedback>2</totalFeedback>
     <totalFeedbackPages>1</totalFeedbackPages>
     <location>
        <city>Blumenau</city>
        <state>Santa Catarina</state>
        <country>Brazil</country>
     </location>
     <sellerFeedback>
           <feedback>
                <rating>7</rating>
                <comment>good</comment>
```

```
                <date>2006-05-26</date>
                <ratedBy>Greg</ratedBy>
            </feedback>
     </sellerFeedback>
 </seller>
...
</sellers>
```

2.2 Schemas for XML

An XML document must meet XML syntax rules to be considered well-formed. Moreover, a well-formed document needs to follow rules defined by a schema to be a valid XML document.

A schema is a grammar that defines the logical structure of an XML document. It defines elements, attributes, and constraints for elements and attributes. Schemas are usually written using DTD (Document Type Definition) [21] and XML Schema (XML Language Schema) [20]. Some advantages of XML Schema against DTD are: XML Schema is written in XML; it has a type system; and it is a much richer language for describing the content of an XML document.

Example 2 presents the XML Schema for the XML document of Example 1. This XML Schema was obtained through hyperModel application [4] and contains information of an XML vocabulary used by Amazon e-commerce service.

Example 2: XML Schema for XML document of Example 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
 <xs:element name="sellers">
  <xs:complexType>
  <xs:sequence>
  <xs:element name="seller" maxOccurs="unbounded">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="sellerName" type="xs:string" minOccurs="0"/>
     <xs:element name="nickName" type="xs:string" minOccurs="0"/>
     <xs:element name="glancePage" type="xs:string" minOccurs="0"/>
     <xs:element name="about" type="xs:string" minOccurs="0"/>
     <xs:element name="moreAbout" type="xs:string" minOccurs="0"/>
     <xs:element name="averageFeedbackRating" type="xs:decimal"
                 minOccurs="0"/>
     <xs:element name="totalFeedback" type="xs:nonNegativeInteger"
                 minOccurs="0"/>
     <xs:element name="totalFeedbackPages" type="xs:nonNegativeInteger"
                 minOccurs="0"/>
     <xs:element name="location" minOccurs="0">
```

```
      <xs:complexType>
       <xs:sequence>
           <xs:element name="city" type="xs:string" minOccurs="0"/>
           <xs:element name="state" type="xs:string" minOccurs="0"/>
           <xs:element name="country" type="xs:string" minOccurs="0"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="sellerFeedback" minOccurs="0">
      <xs:complexType>
       <xs:sequence>
           <xs:element name="feedback" minOccurs="1"
                       maxOccurs="unbounded">
            <xs:complexType>
             <xs:sequence>
                <xs:element name="rating" type="xs:nonNegativeInteger"
                            minOccurs="0"/>
                <xs:element name="comment" minOccurs="0">
                  <xs:simpleType>
                  <xs:restriction base="xs:string">
                      <xs:maxLength value="100"/>
                      <xs:whiteSpace value="preserve"/>
                  </xs:restriction>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="date" type="xs:date" minOccurs="0"/>
                <xs:element name="ratedBy" type="xs:string"
                            minOccurs="0"/>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
   </xs:sequence>
   <xs:attribute name="sellerID" use="required">
     <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:pattern value="[a-zA-Z0-9]{8}"/>
       </xs:restriction>
     </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
  </xs:element>
 </xs:sequence>
 </xs:complexType>
```

```
 </xs:element>
</xs:schema>
```

2.3 XML Query

One way to access data on XML documents is through query languages. XML-QL [8], Quilt [5], XQL [17] and XQuery [11, 18] are query languages created to access data directly from XML documents. In this work, we use XQuery to obtain information from XML documents. It was proposed by W3C to define a standard query language for XML documents [3].

XQuery is a functional language (expressions can be nested), strongly typed (operands, operators and functions must conform to the expected types) and case-sensitive (keywords use lower-case characters) [3]. In XQuery, queries are expressions that are evaluated by generating a value. Example 3 shows a query written in XQuery. This query obtains the element `sellerName` with attribute `sellerID` equal to "ac12DD23" related to the element `seller` of Example 1.

Example 3: Query formulated in XQuery.

```
for $l in document("seller.xml")//seller
where $l/@sellerID="ac12DD23"
return
$l/sellerName
```

There are some tools to execute expressions in XQuery. For example, Qexo [16] is a free software to perform queries in XQuery, on the Java platform.

3 RELATED WORKS

In this section we describe some works from the literature, among them we cite the works that are the basis of our work and others on XML or that address the same subject.

Software test includes test data generation, and XML has been used in many works to support this task. For example, Bulbul and Bakir [2] propose a system based on XML to generate the test data automatically according to the given data definition.

Other works [12, 15, 25] modify XML documents and use these documents to test communication of Web components. Lee and Offutt [12] introduce two mutation operators: `lenghtOf` and `memberOf` for XML documents. The mutants are created by application of the operators and used to test the communication between components of a Web application.

Offutt and Xu [15] explore the use of perturbation operators to test interaction of Web services. In this testing approach, request messages are altered, and response messages are analyzed in relation to the correct behavior. Data perturbation includes data value and interaction perturbation. Data value perturbation modifies

values according to data types in SOAP (Simple Object Access Protocol – message in XML format) messages. Interaction perturbation alters messages in RPC (Remote Procedure Calls – messages with values for the arguments of remote procedure functions) and in data communication (messages to transfer data).

With the same objective, to test communication of Web Services, Xu et al. [25] apply mutation operators in XML schema to generate incorrect XML messages. The schema is represented by a tree T, and operators that change sub-trees and nodes of T are introduced to create incorrect XML messages used in the test of Web services.

The above-mentioned works investigate ideas related to fault-based testing of interaction among Web components, done through XML messages. However, they do not address schema testing. In spite of this different objective, they specify some kinds of faults that are commonly found in XML documents. These faults served as the basis of our classification, since schemas written in XML Schema are XML documents too.

Works with similar objective to ours, addressing testing of schemas, are described in [10, 13]. Li and Miller [13] introduce a set of 18 mutation operators for XML Schema and a mutation analysis model. They can be used to detect faults involving namespaces, complex type definition, simple type definition, type facets and inheritance. The mutation analysis model presents how XML schema mutants are generated, however it does not address about how the Mutation Analysis criterion would be applied in the context of XML Schemas. The authors do not present an experiment to evaluate their mutation operators. They only conduct an experiment to show that commonly used parsers do not reveal most faults described by their operators.

Franzotte and Vergilio [10] apply mutation analysis criterion for testing XML schemas. The authors propose a set of mutation operators that extends the set proposed by Li and Miller. These operators represent possible faults in the schema and are shown in Table 1. Mutation operators are used to alter the XML schema being tested, generating diverse mutants. After this, the tester needs to provide a set of test data, XML documents. Each mutant schema, as well as the original one, is used to validate all the documents in the test set. A mutant is considered dead, if the validation of a test case, by using the mutant, produces a different result from the validation of the same test case against the original schema.

At the end, the mutation score is obtained for the test set provided by the tester. If necessary, additional XML documents can be used to improve the score and to kill the remaining alive mutants. Among these mutants, there can be some ones that are equivalent to the original schema, and they are determined and excluded. A tool, named XTM (Tool for XML Schema Testing Based on Mutation) [10], was implemented to support their operators.

This mutation based approach holds some limitations: it does not offer any automatic support to the generation of the XML documents that are used as test data, and the determination of equivalent mutants is an effort-consuming task, because it can not be completely automated. The authors also do not present a fault model to

Operator	Description
GO – *GroupOrder*	changes the order in which the elements may appear
REQ – *Required*	changes type (optional or obligatory) of the attributes
DT – *DataTypes*	changes data type of elements and attributes
LO – *LengthOf*	changes the name size of the element
CSP – *ChangeSingPlural*	changes the element size by adding or removing characters at the end of the string
CTP – *ChangeTag*	changes the most common node tag used
SO – *SizeOccurs*	changes the maximum and minimum occurrence of the elements
STE – *SubTreeexchange*	inverts the sub-trees below some node
IT – *InsertTree*	adds a node in the structure of the sub-tree
RT – *DeleteTree*	removes the node (or sub-tree) from the structure of the tree

Table 1. Mutation Operators implemented by XTM [10]

allow classification, and the operators need improvement to describe other kinds of faults.

Considering these aspects, we introduce in the next sections a classification for the main faults found in the XML schemas, and a new fault-based approach that automatically generates XML documents and queries to these documents.

4 FAULTS IN XML SCHEMAS

The faults identified in this section were obtained through analyses of XML schemas and investigation of the above-mentioned works [10, 12, 13, 15, 25]. The fault classes represent common mistakes which may be made during the development of the XML schema or in its updating. The focus of this paper is on the semantic faults, that is, we are not interested in syntactical faults easily revealed by the parsers during the validation of the schemas.

The fault classes are described next. They are organized into three fault groups and are illustrated with fragments of XML schemas. Supposing that each fragment is correct according to the data specification, an example of change (fault) that would make the schema incorrect is also presented.

1. **G1 (Group 1): Domain Constraints**: faults related to domain definition of the element or attribute values.

   - IDT – Incorrect Data Type: incorrect definition of data type. The most common data types in XML Schema are: string, decimal, integer, Boolean, date and time. For example:

Correct fragment	Incorrect change
`xs:element name="$totalFeedbackPages" type="xs:nonNegativeInteger"/>`	`... type="xs:integer"/>`

- IV – Incorrect Value: incorrect definition of default or fixed value. The value is defined as default if it should be automatically assigned when no other value is provided. The value is defined as fixed if it is the only acceptable value. In this case the value can be incorrect or the value definition as default or fixed can be incorrect. For example:

Correct fragment	**Incorrect change**
`<xs:attribute name="signal"` `type="xs:string" default="negative"/>`	`...` `default="positive"/>`

- IEV – Incorrect Enumerated Value: incorrect definition of the list of acceptable values. For example:

Correct fragment	**Incorrect change**

`<xs:element name="semaphore_colors">`
`<simpleType>`
`<xs:restriction base="xs:string">`
`<xs:enumeration value="red"/>`
`<xs:enumeration`
`value="yellow/">`
`<xs:enumeration value="green"/>`
`</xs:restriction>`
`</xs:simpleType>`
`</xs:element>`
	`...`
	`<xs:enumeration`
	`value="orange/">`
	`...`

- IMMV – Incorrect Maximum and Minimum Values: incorrect definition of upper and lower bounds for numeric values. For example:

Correct fragment	**Incorrect change**

`<xs:element name="numbers">`
`<simpleType>`
`<xs:restriction base="xs:integer">`
`<xs:minInclusive value="1"/>`
`<xs:maxInclusive value="6"/>`
`</xs:restriction>`
`</xs:simpleType>`
`</xs:element>`
	`...`
	`<xs:maxInclusive`
	`value="7"/>`
	`...`

- IL – Incorrect Length: incorrect definition of the number of characters allowed for values. For example:

Correct fragment	**Incorrect change**

`<xs:element name="comment" minOccurs="0">`
`<simpleType>`
`<xs:restriction base="xs:string">`
`<xs:maxLength value="100"/>`
`</xs:restriction>`
`</xs:simpleType>`
`</xs:element>`
	`...`
	`<xs:maxLength`
	`value="10"/>`
	`...`

- ID – Incorrect Digits: incorrect definition of the total amount of digits or decimal digits for numeric values. For example:

Correct fragment	Incorrect change
<xs:element name="productid"> <simpleType> <xs:restriction base="xs:integer"> <xs:totalDigits value="4"/> </xs:restriction> </xs:simpleType> </xs:element>	... <xs:totalDigits value="6"/> ...

- IP – Incorrect Pattern: incorrect definition of the sequence of characters or numbers allowed for values. For example:

Correct fragment	Incorrect change
<xs:attributename="sellerID" use="required"> <simpleType> <xs:restrictionbase="xs:string"> <xs:pattern value="[A-Za-z0-9]{8}"/> </xs:restriction> </xs:simpleType> </xs:attribute>	... <xs:pattern value="[a-z0-9]{8}"/> ...

- IWSC – Incorrect White Space Characters: incorrect definition of how white space characters must be treated. It should be preserved, removed or replaced. For example:

Correct fragment	Incorrect change
<xs:element name="comment" minOccurs="0"> <simpleType> <xs:restriction base="xs:string"> <xs:whiteSpace value="preserve"/> </xs:restriction> </xs:simpleType> </xs:element>	... <xs:whiteSpace value="collapse"/> ...

2. **G2 (Group 2): Definition Constraints: faults related to the attribute definition concerning data integrity.**

- IU – Incorrect Use: the attribute is defined incorrectly as optional or obligatory. For example,

Correct fragment	Incorrect change
`<xs:attribute name="sellerID"` `use="required">`	`...` `use="optional">`

3. **G3 (Group 3): Relationship Constraints: faults related to relationship definition among elements.**

- IO – Incorrect Occurrence: incorrect definition of the number of times a same element may occur. This number is defined by minimum and maximum values. The default value for minimum and maximum is 1. For example:

Correct fragment	Incorrect change
`<xs:element name="SellerName"` `type="xs:string" minOccurs="0"/>`	`...` `minOccurs="1"/>`

- IR – Incorrect Order: incorrect definition of the order of the elements. The order is defined through the words *all*, *sequence* and *choice*, that mean respectively: the child elements can appear in any order, the child elements should obey the order of its definition in the schema and only one of the child elements can occur in the XML document. For example:

Correct fragment	Incorrect change
`<xs:element name="location" minOccurs="0">`	
`<xs:complexType>`	`...`
`<xs:sequence>`	`<xs:all>`
`<xs:element name="city"type="xs:string"`	
`minOccurs="0"/>`	`...`
`<xs:element name="state"type="xs:string"`	
`minOccurs="0"/>`	
`<xs:element name="country" type="xs:string"`	
`minOccurs="0"/>`	
`</xs:sequence>`	`</xs:all>`
`</xs:complexType>`	`...`
`</xs:element>`	

- IA – Incorrect Association: incorrect definition of an association, for example, a generalization/specialization. An element may be based on an existing complex element by using complex content. The complex content indicates the intention to restrict or extend the content of a complex type. For example:

Correct fragment	Incorrect change
`<xs:complexType name="person">`	
`<xs:sequence>`	
`  <xs:element name="fullname"`	
`   type="xs:string"/>`	
`</xs:sequence>`	
`</xs:complexType>`	
`<xs:complexType name="client">`	
`  <xs:complexContent>`	`...`
`   <xs:restriction`	`<xs:extension`
`   base="person">`	`base="person">`
`   <xs:sequence>`	`...`
`    <xs:element name="code" type="xs:string"/>`	
`   </xs:sequence>`	
`   </xs:restriction>`	`</xs:extension>`
`  </xs:complexContent>`	`...`
`</xs:complexType>`	

The fault classes defined represent typical faults that can be found in the schemas. The next section describes a testing approach to generate test data capable of detecting these specific faults.

5 THE FAULT-BASED APPROACH

The steps of the approach are presented in Figure 1 and described next. The approach is general and can be applied in the testing of different kinds of schemas; but in this work it is illustrated with XML Schema. This kind of schema was chosen because it is written in XML format and supports data types. In addition to that, XML Schema has become very popular and is used extensively.

The tester provides the XML Schema to be tested and one corresponding valid XML document. This document is named the original XML document.

The schema is represented in a formal way so that its main characteristics (elements, attributes and restrictions) can be automatically identified and associated to possible classes of faults. In spite of this step being completely automatic, the tester can also provide other fault associations or choose specific kinds of faults to be considered.

By using the fault associations, alternatives to the original XML document are generated and validated, as well as some queries that will be executed on the valid documents generated. The results of the queries should be evaluated by the tester, who is in fact the oracle.

In the sequence, each step is better explained and illustrated.

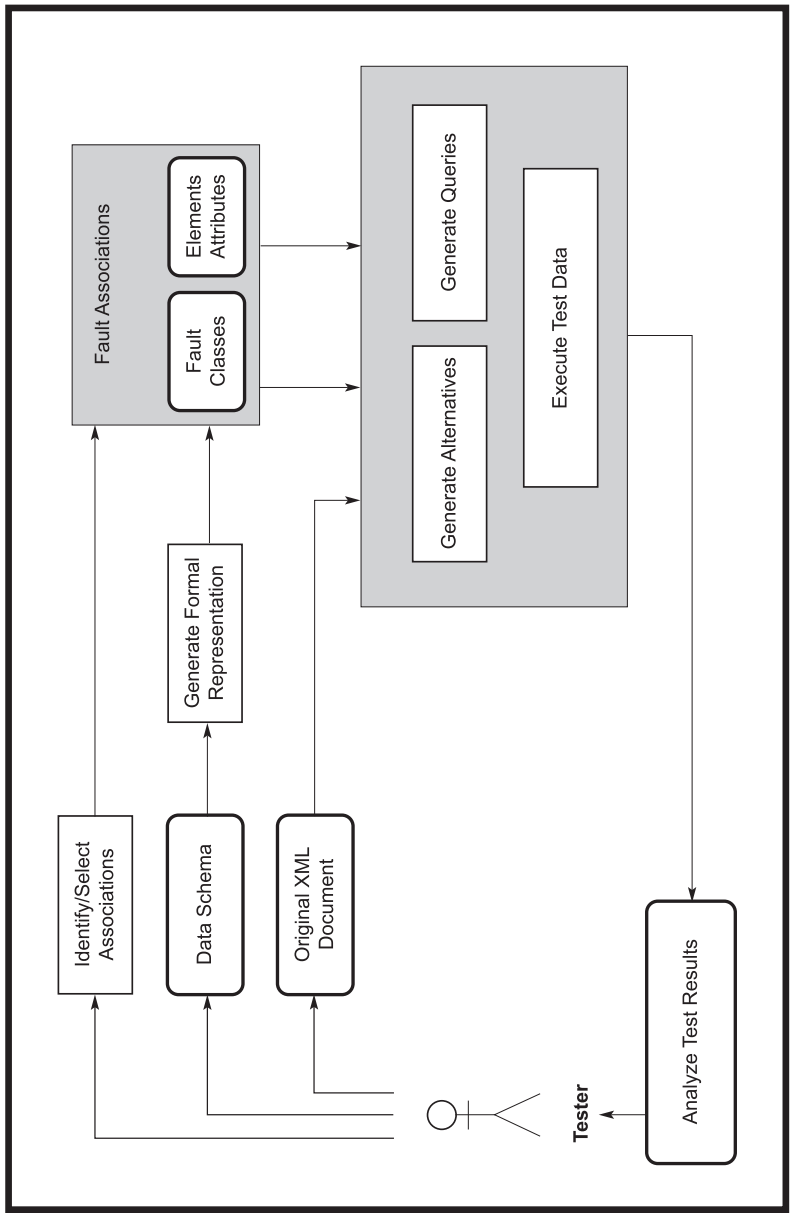


Fig. 1. Testing approach for XML Schemas

5.1 Generate Formal Representation

In this section a formal representation of the schema to be tested is introduced. It allows automatic identification of the elements, attributes, constraints and associations among them. The formal representation is employed to determine fault associations found in the schema under test.

An XML schema S is denoted by $S = (E, A, R, P)$ where:

- E is a finite set of elements;
- A is a finite set of attributes;
- R is a finite set of constraints concerning domain, definition, relationship, and semantics associated to the elements and attributes;
- P is a finite set of association rules among elements, attributes and constraints. Consider $U = E \cup A$, the association rules are represented by:
  - $p(x, y)|x, y, \in U \wedge x \neq y$;
  - $p(x, r)|x \in U \wedge r \in R$;
  - $p(x, r, SU)|x \in U \wedge r \in R \wedge SU = \{u_1, u_2, ..., u_m\} \subset U, \forall u_i \neq x, 1 \leq i \leq m, m \geq 1$, where m is the number of elements and attributes in SU.

Example 4 shows a fragment of the formal representation for the schema of Example 2.

Example 4: Fragment of the formal representation for the schema of Example 2

```
E = {sellers, seller, sellerName, nickName, ...}
A = {selleID}
R = {type, occurs, pattern, use, ...}
P = { p1(sellers, seller), p2 (sellers, order, seller) ,
      p3 (seller, sellerID), p4 (seller, sellerName), ... ,
      p10 (sellerID, pattern), p11 (sellerID, use),
      p12 (sellerName, type), p13 (sellerName, occurs), ...}
```

5.2 Identify/Select Associations

Fault associations are associations among elements or attributes of the schema under test and the fault classes. These fault associations are identified and selected automatically through elements, attributes and constraints found in the formal representation of the schema. For example, we can observe in the schema of Example 2 two restrictions on the element `sellerName` related to data type and number of occurrences. From this information, the following associations with respective fault classes `G1-IDT` and `G3-IO` are identified: (`sellerName`,G1-IDT) and (`sellerName`,G3-IO).

Based on the fault associations, the alternative XML documents and queries are generated. Moreover, the tester may manually identify and select fault associations,

which are not found in the schema automatically. These associations are related to absent constraint definitions[1]. For example, the minimum occurrence of the element `sellerName` is defined in the specification as zero. Suppose that this constraint is not in the schema, i.e. the minimum occurrence of that element was not defined in the schema. This is an absent constraint. If the tester is not sure about it, that is, whether this restriction should be found in the schema, he or she should add this constraint and analyze the test results. Otherwise, he or she should correct the schema before continuing testing.

5.3 Generate Alternatives

Alternative documents are generated by a single modification in the original XML document and represent possible faults in the schema. The structure of the modifications is defined in a set of modification patterns related to the fault classes. The patterns specify how the original document can be modified by inserting, removing or changing values of elements or attributes. For example, suppose the element `sellerName},`whose number of allowed characters for the content is 20, conforms to the data specification. Considering the fault association `(sellerName,`G1-IL) (Group 1 – Incorrect Length), an alternative document could be generated by inserting characters in the name content to check the content length.

Some generated alternatives may be invalid for the schema under test. The alternatives are separated into valid and invalid ones. Only valid alternatives are queried.

Table 2 shows fragments of valid and invalid alternative XML documents. The alternative is considered invalid because in the schema under test (Example 2) the attribute `sellerId` is obligatory. This information can point out a fault in the schema. So, we can observe that in spite of the queries being executed only in valid alternatives, invalid ones may be used by the tester during the analysis of the test results.

5.4 Generate Queries

Queries are automatically generated based on the fault associations and query patterns defined for each fault class. Example 5 contains a query pattern. It is formulated in XQuery for fault class G3-IO (Group 3 – Incorrect Occurrence). The query returns the number of occurrences of a specified element in the alternative XML document.

[1] This kind of missing constraint is analogous to the missing path error in programs. A functionality that should be implemented by the program is missing, i.e., a path in the program is missing. This constitutes a limitation of structural and fault-based techniques, because they derive test data only based on the program and can not guarantee that this kind of fault is revealed.

Valid alternative	Invalid alternative
... <seller sellerID="ac12DD23"> <sellerName>Eduardo</sellerName> <nickName>du</nickName> <glancePage>0k</glancePage> <about>0k</about> <moreAbout>0k</moreAbout> ... </seller> ...	... <seller> <sellerName>Eduardo</sellerName> <nickName>du</nickName> <glancePage>0k</glancePage> <about>0k</about> <moreAbout>0k</moreAbout> ... </seller> ...

Table 2. Fragments of valid and invalid alternative documents

Example 5: A query pattern in XQuery

```
let $doc := document("[?documento?]")
for $i in $doc[?tempRaiz?]
  for $j in $i[?raiz?]
     let $count :=  count($j[?elemento?])
    return <result> {$count} </result>
```

5.5 Execute Test Data

Test data are obtained automatically, because test data are formed by a valid alternative XML document and a query to this alternative. Hence, the test data are executed by executing the queries in the alternative XML documents. In Table 3 we present an example of test data composed by a valid alternative XML document and a query in XQuery generated by fault association (`sellerName`,G3-IO) of Example 2 according to the pattern of Example 5.

5.6 Analyze Test Results

The result obtained for the test data is the query result. In our test data example (Table 3), the query result is equal to 0. This result needs to be analyzed by the tester. Suppose that in the specification the element `sellerName` is required to appear at least once. Then the schema under test has a fault. The element `sellerName` is defined with the constraint `minOccurs="0"` in the schema (Example 2), which allows a valid document without this element.

In addition to the query results, other testing results include: the number of fault associations, valid and invalid alternative XML documents; the fault associations (elements or attributes associated to fault classes); a description of the query results, and the why alternative XML documents were considered invalid.

Valid alternative	**Query**
... <sellers xmlns:xsi="http:// www.w3.org/2001/ XMLSchema-instance"> <seller sellerID="ac12DD23"> <nickName>du</nickName> <glancePage>Ok</glancePage> <about>Ok</about> <moreAbout>Ok</moreAbout> ... </seller> </sellers>	let $doc := document ("minOccursSellerNameValid.xml") for $i in $doc//sellers for $j in $i/seller let $count := count($j/sellerName) return <result> {$count} </result>

Table 3. Test data example

The test results are compared to the expected results from the specification for the schema. Therefore, if during the process, unexpected results of a query are obtained, the fault described by the corresponding fault class is in the schema and must be corrected.

6 EVALUATION OF THE INTRODUCED APPROACH

To evaluate the approach introduced, we implemented a supporting tool named XTool, and conducted two case studies – the first, to evaluate costs and efficacy; the second, to allow a comparison with the mutation based approach.

6.1 Supporting Tool

XTool (XML Schema Testing Tool) is a tool implemented to support the testing approach based on queries. This tool was developed by using Java. It was implemented to test schemas written in XML Schemas. It uses DOM (Document Object Model) [22] to process XML schema, manipulate and validate alternative XML documents; and XQuery through the framework Qexo [16] to query alternative XML documents. Qexo is a free software executed on the Java platform. XTool implements all steps described in the last section.

6.2 Case Study 1

This first case study was conducted to evaluate the applicability of the approach, and to analyze cost and efficacy, in terms of revealed faults. Because of this, we have used schemas designed during the development of Web-based applications. The faults reported in this study were not seeded.

This case study was conducted on two Web-based systems[2]: library and registration system. The systems were developed by two groups of students enrolled in a graduate course. The students' goal was the development of a Web application using UML models. Specifications of the systems and of the schemas were given to each group.

The library system contains data referring to the registered users and on the available titles in the collection. XML documents are based on two XML Schemas: *user.xsd* and *title.xsd*. The registration system has data referring to the students and the available courses in the program based on two XML Schemas: *student.xsd* and *course.xsd*.

Table 4 presents characteristics such as the number of elements and attributes found in the schemas, and the number of registers of the corresponding XML documents. Such characteristics may influence the application of the testing approaches.

Schema	**Elements**	**Attributes**	**Constraints**	**Schema depth**	**Registers**
User	8	0	4	3	6
Title	11	0	4	4	7
Student	10	0	4	3	5
Course	10	0	4	3	6

Table 4. Characteristics of each schema (Case Study 1)

All the schemas of Table 4 were submitted to XTool. Table 5 presents the number of fault associations identified by XTool and selected manually by the tester. Moreover, this table shows the number of alternative XML documents generated and the number of queries produced by XTool, according to the fault associations selected.

Original schema	**Fault associations**		**Queries**	**Alternative XML documents**	
	Automatic	**Manual**		**Valid**	**Invalid**
User	12	10	22	187	73
Title	16	4	20	252	138
Student	16	14	30	241	42
Course	16	10	26	283	43

Table 5. XTool results (Case Study 1)

In Table 5, we can observe that the number of fault associations is equal to the number of queries generated for the schemas, because for each fault association only one query is generated according to the query patterns. The alternative XML

[2] These systems are the same as those used in the studies reported in [9]. In that our previous work, the results were obtained from the manual application of the approach. We repeated the same study now using XTool; because of this, some results, such as the number of valid and invalid alternatives presented here, are not the same.

documents are generated conform to the modification pattern related to the fault class in the fault association. Because of this, some alternatives generated can not be in conformity with the definitions specified in the schema under test. These alternatives are considered invalid. The test data composed of the valid alternative XML documents and of the corresponding queries were executed. The results obtained from the test data were compared to the specification of the expected results, according to the data specification of each schema and system.

The case study results have shown the presence of faults in the schemas of the XML documents. The faults are related to the incorrect definition of the data type, occurrence, maximum and minimum value, pattern and length of some elements. Table 6 presents the faults revealed by the test, regarding each XML schema and type of fault.

Schema	**Fault Classes**	**Revealed faults**
User	G1-IDT, G1-IP, G1-IL, G1-IMMV	9
Title	G1-IDT, G1-IP, G1-IL	3
Student	G1-IDT, G1-IP, G1-IL	12
Course	G1-IDT, G1-IP, G1-IL, G3-IO	8
Total	5	32

Table 6. Revealed faults

This conducted case study provides the following implications:

1. the cost, associated with the number of queries is not so great. It does not increase proportionally to the number of elements. However, the tester is the only person that knows the expected results (a very common fact in any testing process), and the present version of XTool does not offer any help in this task. This contributed to an increase in the tester's efforts to compare the queries' results to those of the data specification;
2. with respect to efficacy, our testing approach by using XTool was effective in revealing faults covered by the fault classes found in the schemas under test.

6.3 Case Study 2

The goal of the second case study was to evaluate the efficacy of our fault-based testing approach with respect to the faults described by the mutant operators implemented by XTM [10].

Mutant schemas were generated with XTM. Hence, each mutant represents an incorrect version of the schema. XTool was then used to test each mutant and to verify if the fault represented by it is revealed by the test data generated by the introduced approach.

Five XML Schemas were used in the case study. These schemas were obtained from diagrams produced by hyperModel application [4] and were called original

schemas. The schemas are based on vocabularies related to a product catalog (*catalog.xsd*) and to information used by the Amazon e-commerce service (*seller.xsd*, *cart.xsd*, *transaction.xsd*, *customer.xsd*). A valid XML document was available for each original schema. These documents are called original XML documents.

Table 7 presents some characteristics of the original schemas, such as: the number of elements, attributes, and constraints, the schema depth (by considering a tree structure), and the number of registers of the original XML documents used in the case study.

Schema	**Elements**	**Attributes**	**Constraints**	**Schema depth**	**Registers**
Catalog	20	0	5	5	6
Seller	21	1	8	5	1
Cart	36	0	5	5	4
Transaction	51	0	5	6	9
Customer	21	0	7	5	1

Table 7. Characteristics of the original schemas (Case Study 2)

The case study procedure and the results obtained in each step are reported next. First, XTool was executed with the original schemas by using the corresponding original XML document. Table 8 presents some results generated by XTool for the original schemas.

Original schema	**Fault associations**	**Queries**	**Alternative XML documents**	
			Valid	**Invalid**
Catalog	28	28	345	87
Seller	45	45	174	60
Cart	71	71	235	110
Transaction	86	86	723	189
Customer	46	46	139	63

Table 8. XTool results for the original schemas (Case Study 2)

The second step was the creation of schema mutants by using XTM for the original schemas under test. The mutation operators introduced by Franzotte and Vergilio [10] applied in the original schemas by XTM were: DT, LO, CSP, STE, RT. Generated mutants were syntactically validated and invalid mutants were discarded. These mutants were validated by commonly used parsers, such as W3C XML Schema validator [23]. Mutant schemas that are equivalent to the corresponding original one were also discarded. Table 9 shows the number of valid mutants generated by XTM for each original schema.

The mutant schemas generated by XTM were tested by using XTool. To do this, valid XML documents for each valid mutant schema were generated based on corresponding original XML documents. In Table 10, some results generated by XTool

Schema	**Mutation Operator**					**Total**
	DT	LO	CSP	STE	RT	
Catalog	56	20	20	52	39	187
Seller	55	21	20	56	27	179
Cart	27	36	36	160	32	291
Transaction	35	51	51	136	46	319
Customer	16	21	21	65	19	142
Total	189	149	148	469	163	1 118

Table 9. Number of mutants generated by XTM

are presented. The number of fault associations, valid and invalid alternative XML documents are related to the sum of test results of all the mutant schemas generated by each mutant operator. For example, observe that the operator STE generated 52 mutant schemas for the original schema `catalog` (Table 9). In Table 10, 1 456 is the sum of the fault associations identified and selected for these 52 mutants being tested by XTool. Moreover, we can observe that the number of identified fault associations and generated queries for each valid mutant schema were the same as those generated for the corresponding original schema. The only exception occurred with the mutants created by the structural operator RT, because this operator removes elements from the original schema.

To analyze the test results, we considered the results of the original schemas as the expected ones. In this case, the results of the mutant schemas of XTool could be compared automatically. We consider that the fault represented by a determined mutant is revealed if the test results of this mutant differ from the test results of the corresponding original schema. This means that our fault-based testing approach is also capable of revealing the faults described by the mutation operators used in the case study. Table 11 presents the number of revealed faults in the mutant schemas.

We can see in Table 11 that our fault-based testing approach is able to reveal all faults represented by the mutants. It is important to remark that most faults represented by the valid mutant schemas created by XTM would not be revealed by common parsers used to validate schemas. As a result of this case study we can observe that the fault classes of our approach covered all faults described by the mutant operators.

Beside this, we have observed that the fault classes can represent other types of faults not represented by the mutant operator approach. For example, a mutant operator to describe a fault related to definitions of pattern constraints for elements or attributes was not proposed by the operator approach. Consider the schema of Example 2, and suppose that in the data specification, the attribute `sellerID` can contain only letters of “A” to “Z” and “a” to “z”. Hence, the schema of Example 2 is incorrect because it also allows numbers for the content of the attribute `sellerID`. However, XTM does not have a mutation operator to describe that fault. Then, the fault would not be revealed by the mutant operator approach but would be revealed by our fault-based testing approach.

Mutant schema	Mutant operator	Fault associations	Alternative XML documents	
			Valid	Invalid
Catalog	STE	1 456	17 948	4 516
	CSP	560	6 564	1 740
	DT	1 400	16 832	4 768
	RT	904	7 466	2 138
	LO	560	6 564	1 740
Seller	STE	2 464	9 688	3 360
	CSP	880	3 462	1 359
	DT	2 422	9 256	3 561
	RT	1 082	4 143	1 423
	LO	924	3 633	1 260
Cart	STE	11 360	13 440	2 080
	CSP	2 556	2 989	433
	DT	1 915	2 285	370
	RT	2 144	2 425	419
	LO	2 556	2 984	428
Transaction	STE	11 696	15 112	3 416
	CSP	4 386	5 567	1 181
	DT	3 008	3 889	881
	RT	3 772	4 765	996
	LO	4 386	5 561	1 175
Customer	STE	2 990	3 835	845
	CSP	966	1 295	329
	DT	731	944	211
	RT	782	1 006	224
	LO	966	1 295	329

Table 10. XTool results for each generated mutant schema by mutant operators

7 CONCLUSIONS

We have presented a fault-based testing approach for XML schemas. This approach is based on introduced fault classes, which represent common faults found in schemas. Three groups of faults were presented. They are used to establish fault associations for a schema under testing. The associations are related to pat-

Mutant schema	Revealed faults
Catalog	187
Seller	179
Cart	291
Transaction	319
Customer	142

Table 11. Number of revealed faults described by the operators

terns previously defined that are used to generate alternative XML documents and queries. The results of these queries are the testing results to be analyzed by the tester.

We have implemented a tool, named XTool, to support the approach. The tool is fundamental to making this approach practical. This tool provides the test of schemas written in XML Schema. However, the testing approach may be applied in other types of schemas for XML.

We show results of two case studies conducted by using XTool. The first case study used four schemas written by students during the development process of Web applications. The case study results indicate that the cost and efforts related to the use of XTool are associated with the analysis of the queries' results by the tester. Another result from this study is that our testing approach is applicable and effective in revealing faults covered by the fault classes introduced.

In the second case study, our testing approach was evaluated by considering faults described by some mutation operators found in the literature and implemented by XTM. Differently from the first case study, the XML schemas (original schemas) tested in this case study are used in real applications. The results of this case study show that our testing approach is effective in revealing faults described by the mutation operators. The faults described by all the mutants generated by XTM were revealed by the test data generated by XTool.

An advantage of our testing approach is the automatic test data generation, because in applying mutation testing by using XTM, the tester generates test data manually, and needs to determine equivalent mutants.

Another advantage is that the testing approach allows the tester to identify other kinds of faults related to missing restrictions. These missing restrictions are related to functionalities not found in the schema.

Moreover, the fault-based approach can test schemas even if the application is not available and the different generated data instances may also be used, in the context of XML, to test applications that manipulate a document in XML format, interaction among Web components and Web services.

A problem with our approach is the cost, because a great number of test data can be generated and the testing results need to be analyzed. This is also a limitation of the testing activity, the oracle can be only partially automated, and because of this the tester plays a fundamental role. Fault-based techniques, such as mutation analysis, are generally most expensive because of the great number of mutants generated and test cases required. Some works have pointed to some strategies, such as determination of essential operators to reduce this cost [1]. We think this kind of strategy can also be used to refine and select the most essential fault classes related to XML schemas.

We should conduct other evaluation studies with other kinds of schemas. These studies can be used to refine the proposed fault classification. Other refinements should be implemented in XTool. We are now implementing an "oracle" module to help the tester in the analysis of the results.

As mentioned above, the approach can be applied in other contexts. We intend to investigate its application in the context of relational database schemas and Web services.

Acknowledgements

We would like to thank André L. S. Solino, Everton F. R. Seára, Luciana Umburanas, Rafael Caiuta and Wendel G. Pedrozo for their assistance during the case studies.

REFERENCES

[1] Barbosa, E. F.—Maldonado, J. C.—Vicenzi, A. M. R.: Towards the Determination of Sufficient Mutant Operators for C. Software Testing, Verification and Reliability 2011, No. 11, pp. 113–136.

[2] Bulbul, H. I.—Bakir, T.: XML-Based Automatic Test Data Generation. Computing and Informatics. Vol. 27, 2008, No. 4, pp. 681–698.

[3] Boag, S.—Chamberlin, D.—Fernandez, M. F. et al.: XQuery 1.0: An XML Query Language. Working Draft, W3C World Wide Web Consortium, November 2003. Available on: `http://www.w3.org/TR/2003/WD-xquery-20031112` (accessed May 2004).

[4] Carlson, D.: HyperModel Application. 2006. Available on: `http://www.xmlmodeling.com/models/index.html` (Accessed May 2006).

[5] Chamberlin, D.—Robie, J.—Florescu, D.: Quilt: An XML Language for Heterogeneous Data Sources. Journal Lecture Notes in Computer Science, Vol. 1997, 2001, available on: `http://citeseer.ist.psu.edu/chamberlin00quilt.html` (accessed September 2004).

[6] Chen, H.—Ma, J.—Wang, Y.—Wu, Z.: A Survey on Semantic E-Science Applications. Computing and Informatics, Vol. 27, 2008, No. 1, pp. 5–20.

[7] DeMillo, R. A.—Lipton, R. J.—Sayward, F. G.: Hints on Test Data Selection: Help for the Practicing Programmer. IEEE Computer, Vol. 11, 1998, No. 4, pp. 34–41.

[8] Deutsch, A.—Fernandez, M.—Florescu, D.—Levy, A.—Suciu, D.: XML-QL: The Query Language Goes XML. 2004, available on: `http://www.w3c.org/TR/NOTE-xml-ql` (Accessed September 2004).

[9] Emer, M. C. F. P.—Vergilio, S. R.—Jino, M.: A Testing Approach for XML Schemas. In: The 29th Annual International Computer Software and Applications Conference COMPSAC – QATWBA, July 2005.

[10] Franzotte, L.—Vergilio, S. R.: Applying Mutation Testing to XML Schemas. In: The 18th International Conference on Software Engineering and Knowledge Engineering (SEKE '06), July 2006.

[11] Katz, H. (Ed.): XQuery from the Experts-Guide to the W3C XML Query Language. Addison-Wesley 2003.

[12] Lee, S. C.—Offutt, J.: Generating Test Cases for XML-based Web Component Interaction Using Mutation Analysis. In: The 12th International Symposium on Software Reliability Engineering, Hong Kong, China November 2001, pp. 200–209.

[13] Li, J. B.—Miller, J.: Testing the Semantics of W3C XML Schema. In: The 29th Annual International Computer Software and Applications Conference COMPSAC, July 2005.

[14] Offutt, J.: Quality attributes of Web software applications. IEEE Software, Vol. 19, 2002, No. 2, pp. 25–32.

[15] Offutt, J.—Xu, W.: Generating Test Cases for Web Services Using Data Perturbation. In: TAV-WEB Proceedings, September 2004.

[16] Qexo: The GNU Kawa implementation of XQuery. Available on: `http://www.gnu.org/software/qexo` (a ccessed October 2004).

[17] Robie, J.—Lapp, J.—Schach, J.: XML Query Language (XQL). XQL Proposal, W3C World Wide Web Consortium 2004, available on: `http://www.w3.or/TandS/QL/QL98/pp/xql.html` (accessed September 2004).

[18] Sousa, A. A.—Pereira, J. L.—Carvalho, J. A.: Querying XML Databases. In Proc. of the XXII International Conference of the Chilean Computer Science Society, November 2002, pp. 142–150.

[19] W3C: Extensible Markup Language (XML) 1.0 (second edition) W3C recommendation, October 2000. Available on: `http://www.w3.org/XML` (accessed January 2005).

[20] W3C: XML Schema recommendation. May 2001, available on: `http://www.w3.org/tr` (accessed January 2005).

[21] W3C: Document Type Definition. Available on: `http://www.w3schools.com/dtd/default.asp` (accessed February 2005).

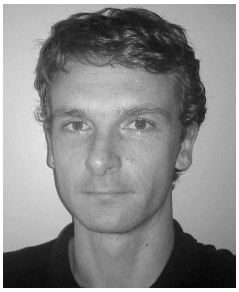
[22] W3C: DOM – Document ObjectModel. 2005. Available on: `http://www.w3.org/DOM` (accessed November 2005).

[23] W3C: Validator for XML Schema. Available on: `http://www.w3.org/2001/03/webdata/xsv` (accessed November 2005).

[24] Wong, W. E.—Mathur, A. P.—Maldonado, J. C.: Mutation Versus All-uses: An Empirical Evaluation of Cost, Strength and Effectiveness. In: Software Quality and Productivity Theory, Practice, Education and Training. Hong Kong, December 1994.

[25] Xu, W.—Offutt, J.—Luo, J.: Testing Web Services by XML Perturbation. In: The 16th IEEE International Symposium on Software Reliability Engineering November 2005.

Maria Claudia Figueiredo Pereira Emer is currently an Assistant Professor at the Informatics Department of the Federal University of Technology of Paraná (UTFPR), Brazil. She received her Dr. Sc. degree in Electrical Engineering from State University of Campinas (UNICAMP), São Paulo, Brazil, in 2007, her M. Sc. degree in Informatics from Federal University of Paraná, Brazil, in 2002, and her B. Sc. degree in Informatics from State University of Western Paraná (UNIOESTE), Brazil, in 1997. Her research interests include software testing.



Igor Fabiano Nazar received his M. Sc. degree in Informatics from Federal University of Paraná (UFPR), Brazil, in 2007, and his B. Sc. degree in Informatics from State University of Ponta Grossa (UEPG), Brazil, in 2004. His current research interests include software testing and development methodologies.

Silvia Regina Vergilio received the M. Sc. (1991) and Dr. Sc. (1997) degrees from University of Campinas (UNICAMP), Brazil. She is currently at the Computer Science Department of the Federal University of Paraná (UFPR), Brazil, where she has been a faculty member since 1993. She has been involved in several projects and her research interests are in the area of software engineering, such as search based software engineering, software testing, software quality and software metrics. She is a member of ACM and SBC – the Brazilian Computer Society.

Mario Jino received the B. Sc. Degree In Electronic Engineering from Instituto Tecnológico de Aeronáutica (ITA), Brazil in 1967, the M. Sc. Degree in Electrical Engineering from the State University of Campinas (UNICAMP), Brazil, in 1974, and the Ph. D. degree in Computer Sciences from the University of Illinois, Urbana-Champaign, Illinois, USA, in 1978. Since 1971 he is a teacher at UNICAMP where he is presently a Full Professor in the Department of Computer Engineering and Industrial Automation of the School of Electrical and Computer Engineering. His current research interests include software quality; software testing, debugging and maintenance; object orientation; and software metrics. He has been involved in several technological development projects with government and private entities, and has served as technical advisor/referee for Brazilian research funding agencies as well as in several scientific conferences and symposiums. He is a member of the IEEE Computer Society, the IEEE, the ACM, SBA – the Brazilian Society of Automatic Control, and SBC – the Brazilian Computer Society.