Analysis and Metrics of XML Schema

Andrew McDowell
University of Houston-Clear Lake
andrew@rendai.com

Chris Schmidt
University of Houston-Clear Lake
chris@rendai.com

Kwok-Bun Yue
University of Houston-Clear Lake
yue@cl.uh.edu

Abstract

Despite the ubiquity of XML, research in metrics for XML documents is scarce. This paper proposes and discusses eleven metrics to measure the quality and complexity of XML Schema and conforming XML documents. To provide an easy view of these metrics, two composite indices have been defined to measure quality and complexity. An open source metric analyzer tool for XML Schema has been developed. The tool can easily be extended to add new metrics and alter the composition of the indices to best fit the requirements of a given application.

1. Introduction and Related Work

Research of metrics for XML documents is scarce. There has been considerable research done with respect to improving quality in the software engineering process and discovering best practices for knowledge and data capture. Yet, very little has been done to determine how certain artifacts, like XML documents, fit into a mature development process that consistently produces high quality products. The most pertinent research that has been done was by Klettke, Schneider, and Heuer [1]. In it, they developed a set of five metrics for Document Type Definition (DTD) documents to glean some measure of the complexity of XML documents. This was an important first step in XML metric research, but it focused mainly on complexity, and not quality. In this paper, we expand upon this research by analyzing metrics for XML Schema, which is a more descriptive language than DTD for describing the vocabularies of XML documents. We propose eleven metrics for XML Schema and devise two simple formulae that use them to compute complexity and quality indices.

There is also a need to validate the metric research in a practical manner. Thus, a separate activity was done to create an open source metric analyzer of XML Schema. The tool was designed to be flexible in order to easily add new metrics and reformulate the quality and complexity indices. It has been used to generate metrics for a representative collection of XML Schemas.

This paper is divided into seven sections. Sections two and three provide the motivation for research on XML Schema metrics. The fourth section enumerates the eleven metrics identified, followed by an explanation of the formulae created for the indices in the fifth section. The sixth section discusses the XML Schema metric analyzer. Finally, the last section covers future research that should be done to provide additional insight into this field.

2. The Need for Metrics

DeMarco stated that one cannot control something that cannot be measured [2]. Software quality is an elusive goal that both the research community and the industry strive for. Yet, the term 'quality' is vague and can mean different things to different people. In 1983, the Software Engineering Technical committee of the IEEE Computer Society defined quality as "the degree to which software possesses a desired combination of attributes" [3]. Later, the International Standards Organization (ISO) defined quality in ISO standard 9126 as a collection of items in a 'quality model' [4]. This model is broken up into six components, each with multiple subcomponents: functionality, reliability, usability, efficiency, maintainability, and portability. The categorical definition of quality by the ISO committee better describes the different facets of quality and how different stakeholders define it during the software process. Nonetheless, the vagueness of 'quality' has hampered the ability of researchers and corporations alike to build software effectively.

Software metrics can fill the need for measuring the state of software to help control it better. They are defined as the measurement of aspects of software construction and testing that result in an ability to describe the quality, cost and value of the software. Harter and Slaughter posit that there is evidence that ensuring high quality in all phases of software development will result in a higher quality product [5]. By using metrics as a tool to increase quality during the implementation phase, it should be possible to ensure an overall increase in quality for the product.

While Klettke's paper on XML metrics focused on the complexity of elements and their relationship in DTDs, we expanded on their work to also measure quality. This was done by identifying characteristics that helped or hindered the overall quality of XML Schemas.

3. The Need for XML Schema Metrics

In recent years, XML has become a cornerstone of many software applications. XML documents are used for knowledge and data capture. Poor design of XML documents can hinder the ability to effectively use the data. For example, if the XML document is used as an internal format to support the software infrastructure, a poor schema can affect the overall quality of the product. If the XML document is used to store or consume data, then the software handling the document may not be affected, but the net result to users of the product may be that they will get inferior data. In both cases, improving the quality can potentially improve the quality of the software product and/or the quality of the end user's interaction with the product.

XML documents are often used as a communication medium between parties. If the XML data being passed between groups conforms to a common schema, then this schema can be considered a contract. This contract carries a heavy burden since both parties can have products or applications that use the XML data. It is beneficial to have a highly flexible and easily maintained schema to minimize the changes necessary for the end products or applications.

The above examples are all valid reasons why a high quality schema for XML documents is necessary. When choosing a schema language, you have two main choices: DTD and XML Schema. XML Schema is gaining popularity due to its powerful capabilities. It contains many features missing in DTDs, such as custom data types, object-oriented features and structured documentation [6]. It is generally agreed that XML Schema is the schema language of the future for XML. Thus, we chose to focus this paper on the quality measurement of XML Schema documents and the complexity measurement of conforming XML documents.

Even though XML and XML Schema documents usually only make up a small fraction of a software product, their influence on the overall quality of the software can be large. XML metrics are important components of the overall metrics for predicting quality and complexity of the software development process.

4. XML Schema Metrics

Our proposed XML Schema metrics are based on the five complexity metrics enumerated by Klettke et. al. [1]. When developing the metrics, we also focused on the categories of the ISO 9126 quality model. Since our goal was to build an XML metric analyzer tool that can be easily configured to add new metrics and remove existing metrics, these metrics were not designed to be comprehensive. Instead, additional metrics can be supplemented to the identified metrics on a per business case basis.

Klettke, et. al. focused on measuring *complexity* of DTD. Our metrics expand on their work in two directions:

(1) Metrics that measure the quality of XML Schema. For example, unlike DTD, there is an element called *annotation* to allow formal documentation of the XML Schema. Thus, the abundance of the *annotation* element is usually a good indicator that the XML Schema is well documented and easy to read and maintain.
(2) Metrics that exploit advanced features of XML Schema. For example, unlike DTD, XML Schema allows the definition of user-defined types. Their use may affect both quality and complexity.

In this section we present eleven proposed metrics and use the example libraryexample.xsd in Figure 1 to illustrate them. The XML Schema libraryexample.xsd captures a simplified library structure and is used because it contains many of the common XML Schema features pertinent to our discussion.

```
<?xml version="1.0"?>
<xsd:schema
  targetNamespace=
    "http://examples.org/schemaExample"
  xmlns:xsd=
    "http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation>
      This is an example XML Schema for a
       simple library.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleType name="string32">
    <xsd:restriction base="xsd:token">
      <xsd:maxLength value="32"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="bookType">
    <xsd:sequence>
      <xsd:element name="title"
        type="string32"/>
      <xsd:element name="author"
        type="xsd:string"
        maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="isbn"
      type="xsd:string"/>
  </xsd:complexType>
  <xsd:complexType name=”libraryBookType”>
    <xsd:complexContent>
      <xsd:extension base=”bookType”>
        <xsd:attribute name=”deweyDecimalNumber”
          type=”xsd:string”/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="library">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="book"
          type="libraryBookType"
          maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figure 1. A sample XML Schema, libraryexample.xsd

4.1 Number of Complex Type Declarations

XML Schema allows the definitions of complex types and simple types. Complex types allow elements in their contents and can have attributes. They are used to define elements with child elements in their content models. As a result, more complex types usually indicate more complex XML structures. Both globally defined and locally defined complex types are counted. This can be further broken down into three subcategories: Text-only, Element-only and Mixed-Content (may contain both text and child elements.) There are three complex types in libraryexample.xsd: the global types *bookType* and *libraryBookType,* and the anonymous complex type defined locally within the *library* element.

4.2 Number of Simple Type Declarations

Simple types only allow values. They can either be predefined or user-defined. User-defined simple types derive from their parent types by restriction, in order to limit their content. This metric counts the number of user-defined simple type declarations. In libraryexample.xsd, there is one user-defined simple type declaration named *string32*. This simple type is based on the built-in *token* type with the restriction that the value of the element is at most 32 characters.

4.3 Number of Annotations

The *annotation* element in XML Schema allows documentation for the benefit of both human readers and applications. The element may contain *document* elements (for human readers) and *appInfo* elements (for applications). Having more annotation elements present in the XML Schema document usually imply that the XML Schema is better documented. Thus, there is a higher chance that the overall quality is better. In libraryexample.xsd, there is only a single annotation element.

4.4 Number of Derived Complex Types

Complex types can be derived from other complex types or simple types by either expanding or restricting the definitions of their parent types. Derivation of data types is a feature that is not present in DTDs. This can be a very useful feature, which adds flexibility and reusability, but also increases complexity. In libraryexample.xsd, there is one derived complex type, *libraryBookType*, which extends the *bookType* complex type by adding the *deweyDecimalNumber* attribute.

4.5 Average Number of Attributes per Complex Type Declaration

Complex types may have zero or more attributes defined. This metric measures the average number of attributes of the complex types defined within the XML Schema document. This is calculated by dividing the total number of attributes in all complex type declarations, including any attributes that may be inherited from parent type declarations, by the total number of complex type declarations. Of course, this metric is only applicable when there is at least one complex type declaration. In libraryexample.xsd, there are three complex type declarations: *bookType* has a single attribute *isbn; libraryBookType* has two attributes, *deweyDecimalNumber* and *isbn;* the anonymous complex type declared within the library element has no attributes. Since *libraryBookType* derives from *bookType*, *isbn* is included as an attribute for both complex type declarations. This gives an average attribute per complex type declaration of 1.0.

4.6 Number of Global Type Declarations

Type declarations, simple and complex, can be defined at the top level of an XML Schema document. The advantage of using global types is reduced redundancy since elements of the same type can refer to the global type instead of each having a local declaration. In libraryexample.xsd, there is one simple global type, *string32*, and two complex global types, *bookType* and *libraryBookType*.

4.7 Number of Global Type References

Every element declared within an XML Schema document must specify its type. This type may be a built-in simple type, a type defined within the element's body, or a reference to a user-defined global type. This metric measures the number of elements that specify global types as their element types. The ratio of global type references to the number of global type declarations is a good measure of quality, where higher is better. If it is smaller than 1, some global type declarations are not even being used. In libraryexample.xsd, the *title* element refers to the global type *string32* and the *book* element refers to the global type *libraryBookType*.

4.8 Number of Unbounded Elements

Every XML Schema element has attributes called *minOccurs* and *maxOccurs*. The values for these attributes specify how many times the element can occur in a conforming XML document. One possible value for the *maxOccurs* attribute is "unbounded", which means the element may appear any number of times. This metric measures the number of elements that have "unbounded" specified as the value for its *maxOccurs* attribute. Having unbounded elements can greatly increase the complexity of conforming XML documents since having even a single unbounded element allows the XML document to become infinitely large. In libraryexample.xsd, elements *author* and *book* are unbounded elements.

4.9 Average Bounded Element Multiplicity Size

The range of the multiplicity of an element is bounded by the *minOccurs* and *maxOccurs* attributes. The size of this range for a bounded element, called *multiplicity size* here, is equal to (*maxOccurs – minOccurs + 1).* This metric measures the average multiplicity size of all bounded elements. The default value for both *maxOccurs* and *minOccurs* attributes is one. Therefore, if an element does not specify either attribute, its multiplicity size is one. Similar to "unbounded" elements, elements with a high value in multiplicity size can add to the complexity of conforming XML documents. Neither of the two bounded elements in libraryexample.xsd, *library* and *title*, specify a value for *maxOccurs* or *minOccurs* attributes, so the average multiplicity size is 1.0.

4.10 Average Number of Restrictions per Simple Type Declaration

Every simple type is a restriction of another simple type. This metric measures the average number of restrictions placed on all of the simple type declarations within the XML Schema document. Having more restrictions on a type reduces the range of valid values for the type, thus reducing the complexity of the XML document. However, some restrictions, such as enumeration, are more restrictive than others. Having more restrictions does not always produce a smaller set of values. On the other hand, having more restrictions may mean the authors have spent more effort to apply domain constraints, a sign of good quality. In libraryexample.xsd, there is only one simple type defined, *string32*, which has a single restriction.

4.11 Element Fanning

Element Fanning is a composite of two metrics defined for DTDs in [1]: Fan-In and Fan-Out. In XML Schema terms, Fan-Out is the number of child elements that an element has. The other metric, Fan-In, measures how many times an element is referenced within the XML Schema document.

It is easier to understand this metric by modeling the XML Schema document as a graph. A node is an element and an edge is a parent-child relationship in element declarations. Fan-In and Fan-Out of an element are the number of incoming and outgoing edges of the corresponding node respectively.

In libraryexample.xsd, the Fan-In of the elements *library*, *book*, *title* and *author* are 0, 1, 1 and 1 respectively. Their Fan-Out values are 1, 2, 0 and 0.

Element Fanning is the average Fan-In and Fan-Out for all elements in the XML Schema document. It does not matter which is used for the average since the average Fan-In will always equal the average Fan-Out. The Element Fanning can be computed by dividing the number of edges in the graph, by the number of nodes in the graph. The graph for libraryexample.xsd consists of three edges and four nodes, giving an Element Fanning of 0.75.

5. XML Schema Complexity and Quality Index

Although we have presented eleven metrics, it is easy to develop many more. It is difficult to grasp the meaning of these individual metric values. In an attempt to provide an easy to understand interpretation of these metrics, we have formulated two indices for measuring quality and complexity.

However, it is important to emphasize that there are both advantages and disadvantages of using composite indices. A single index is easily understood and quantified. However, both quality and complexity are subjective to a certain degree and they are difficult to be quantified in general. This is especially true for quality, as discussed in Section 2. Many facets must be considered to present an overall view of the underlying quality and complexity. Thus, a single index has the potential to be overly simplistic and inaccurate [7]. As a result, the indices we defined below should be considered as a first attempt at quantifying an elusive ideal. Readers should be cautious when interpreting the results. Taken into context, the indices provide a general indication of the quality and complexity of XML Schema documents and their resulting XML documents.

Other indicators could also be used to devise the quality or complexity indices. Based on our own experience with XML Schema, we identified a subset of the metrics as having more impact on overall quality and complexity, both positively and negatively. Some of these metrics have more weight than others in the formulae. For instance, in our opinion, the number of annotations describes the maintainability factor of a document better than the total number of global types, and thus has a higher weight in the quality index. Each metric in the subset is weighted with a multiplier of 1 to 5, based on the importance of the metric compared to others. Some metrics are subtracted from the indices because we feel that they detract from quality or complexity. The formulae follow:

Quality Index = (Ratio of simple to complex type declarations) * 5 + (Percentage of annotations over total number of elements) * 4 + (Average restrictions per simple type declarations) * 4 + (Percentage of derived complex type declarations over total number of complex type declarations) * 3 – (Average bounded element multiplicity size) * 2 – (Average attributes per complex type declaration) * 2

Complexity Index = (Number of unbounded elements) * 5 + (Element fanning) * 3 + (Number of complex type declarations) + (Number of simple type declarations) + (Number of attributes per complex type declaration)

Note that the actual values of the weights are set based only on our experience in analyzing many XML Schemas. They should be considered preliminary and are subject to changes when more experiments are performed. Our metrics analysis tool is developed in a way that these weights can easily be changed in the future.

The end result of either formula will be a number that by itself does not have much meaning. The metric analyzer tool also stores the average of the indices of XML Schemas it has analyzed. Once a number of documents have been analyzed, the comparison between the average and the document's indices will give a relative indication of quality and complexity.

It is also important to note that the quality index is intended to provide an indication of the quality of the XML Schema document, that is, how well the XML Schemas are constructed. On the other hand, the complexity index is used for XML documents that are validated by the XML Schema document. This is an important distinction, since the indices are tuned to two different groups of documents.

6. XML Schema Metric Analyzer Tool

The tool created by the authors was designed from the beginning to be an open source application to allow others to contribute and strengthen it. A secondary benefit of opening the code to the public is in the hopes that software development groups will incorporate the metric analyzer as a part of their processes. Leon Osterweil makes a strong argument for increasing the communication between the research community and software practitioners in [8]. Ideas to increase quality in the software process that have been generated by researchers are lagging by as much as 20 years before entering the commercial realm [8]. By doing research and developing a concrete tool in parallel, it is the authors' hope that any benefits that come from this research can be applied in the software development process quickly.

Besides being an open source tool, it also makes use of openly available tools to help in extracting the metrics from XML Schema documents. One such underlying tool is the Castor schema object model parser [9]. The Castor code was invaluable in providing a base to parse XML Schema documents and manipulate the objects generated. The application was developed in Java 1.4 to allow for greater portability, and two interfaces were created to allow the analyzer to be used in either windowed environments

(Windows, OS X, X Windows, etc) or command line environments (DOS, UNIX, etc).

The analyzer was developed in an extensible manner. A property file specifies what objects are instantiated to control certain aspects of the system, like the user interface, storage method of previously analyzed schemas, and the parser that extracts the metrics from XML Schema documents. Each of these items can be updated and swapped out easily. For instance, the metrics are stored in an XML document, but a new storage Java class can be created if the metrics need to be stored in a database or remote system. The base code of the analyzer does not need to be changed to incorporate the new functionality.

Even with the ability to manipulate the individual sections of an XML Schema document, the metric analyzer tool is unable to capture all aspects of a XML Schema document for quality analysis. For example, unintelligible element names, poor layout and improper spacing will reduce understandability of an XML Schema document. This is difficult for a program to determine, but is easy for humans.

Figure 2 shows the output of using the command line mode of the analyzer to analyze libraryexample.xsl.

Sample Metric Output

```
----------------------------------------------------------------
Metric Analysis of XML Schema File libraryexample.xsd
----------------------------------------------------------------
Number of Complex Type Declarations:                    3
-Number of Text Complex Type Declarations:              0
-Number of Element Complex Type Declarations:           3
-Number of Mixed Complex Type Declarations:             0
Number of Simple Type Declarations:                     1
Number of Annotations:                                  1
Number of Derived Complex types:                        1
Average Number of Attributes per Complex Type:          1
Number of Global Type Declarations:                     2
Number of Global Type References:                       2
Number of Unbounded Elements:                           2
Average Bounded Element Multiplicity:                1.00
Average Number of Restrictions per Simple Type:      1.00
Element Fanning:                                     0.75


Quality Index for schema:                            8.51
Complexity Index for schema:                        17.92
```

Figure 2. Metrics and indices output of analyzing libraryexample.xsd by the tool

7. Conclusions and Future Work

In this paper, we have presented eleven metrics for XML Schema and two indices for interpreting them. An open source metric analysis tool has been developed to implement these metrics and indices. Due to the fast paced nature of the XML technologies, the metrics, indices and the tool will need to be updated continuously.

Additional metrics can obviously be developed. Furthermore, a given metric will have different importance to different application. For example, a new metric can be proposed to measure the number of *appInfo* elements, which provides documentation about the XML Schemas to applications. This metric may be more important if the targeted XML documents are used as internal data format for an application. The authors are working on a more systematic approach and categorization of metrics for XML Schemas.

A very important problem with any software metric is how it should be interpreted. One person may not have the same opinion as to what makes a good XML Schema compared to someone else. The relative importance of a metric may also be application and domain dependent. Although the tool is written in a way that the formulae of the indices can be changed easily, we plan to make it even more flexible by storing the formulae in property files. The most appropriate property file can then be selected for a given scenario.

Another extension to the configurable formulae can be an implementation of a Bayesian learning algorithm so that users can denote those XML Schema documents that have a higher quality factor for their business needs, and others that do not. Given enough XML Schema documents, the analyzer will then be trained to better identify the quality of an XML Schema documents in the same domain.

An important future task is the validation or refutation of the current formulae to determine the relative complexity and quality of XML Schema documents. An email based survey and a Web survey have been created to gather feedback from key XML Schema developers to correlate their view of complexity and quality to those of the tools. Unfortunately, there were not enough responses to get a consensus. We will continue to work in this direction.

Acknowledgement. This work is partially supported by the Faculty Leave Fund of the University of Houston-Clear Lake.

8. References

[1] Klettke, M., Schneider, L., Heuer, A., "Metrics for XML document collections", *XMLDM Workshop*, Czech Republic, 2002, pp. 162-176

[2] DeMarco, T., *Controlling Software Projects*, Yourdon, New York, 1982

[3] Software Engineering Technical Committee of the IEEE Computer society. *IEEE Standard Glossary of Software Engineering Terminology*, IEEE-STD-729-1983, New York, 1983, p. 32

[4] Fenton, N. E. and Pfleeger, S. L., *Software Metrics, a rigorous and practical approach*. 2nd Ed, PWS Publishing Co., Boston, 1997

[5] Harter, D. E. and Slaughter, S. A., "Process Maturity and Software Quality: a Field Study", *International Conference on Information Systems, Proceedings of the twenty first international conference on Information systems, Association for Information Systems*, Brisbane, Australia, 2000, pp. 407-411

[6] W3C, "W3C XML Schema", XML Schema Specification, http://www.w3.org/XML/Schema

[7] Boehm, B.W., Brown, J. R., Lipow, M., "Quantitative Evaluation of Software Quality", *2nd ICSE*, Vol .2, 1976, pgs. 592-605

[8] Osterweil, L., "Strategic Directions in Software Quality", *ACM Computing Surveys (CSUR)*, Volume 28, Issue 4, New York, 1996, pp. 738-750

[9] Castor, "Castor Schema Support", Castor Schema Object Model, http://www.castor.org/xmlschema.html