# Teaching with a Unit Testing Framework

**Dr Mike Lance**
School of Computing
Christchurch Polytechnic Institute of Technology
Christchurch, NZ
lancem@cpit.ac

This paper analyses element usage in a 'real world' XSLT application. A subset of core XSLT elements is identified and the reasons why these particular elements are useful is discussed. Teachers of XSLT may need to modify their introductory examples to cover what is actually needed in larger projects.

## Keywords

XSLT elements

## 1. INTRODUCTION

How much of XML, XSLT and schema languages are actually used in creating real world applications as opposed to student assignments? Is there a minimum subset of the growing WC3 XML standards that can be safely taught that will equip students for the workplace without overburdening them? These questions are addressed by a statistical analysis of the code of an XML and XSLT application written by two level 7 students completing a 45 credit programming project. The application used XSLT to add formatting to XML data and produce output in html and pdf formats. The frequency of use of XSLT elements in this application will be analyzed.

## 2. METHODOLOGY

Students who created an XSLT programme had passed a course in xml. This paper analyzes their use of XSLT in order to better inform subsequent teaching. This is not quiet as dangerous a GIGO situation as it sounds because the students did considerable extra study into the issue of what is good design for XSLT while creating their application and had to justify their design decisions in regular project reviews. In an introductory course covering the whole of XML it is appropriate to demonstrate 'hello world' programmes which are effective proofs of concepts. Good course assignments then require students to apply the concepts to solve practical problems similar to those they will meet in the 'real world'. The weakness of any course assignment is that it is a small scale exercise which does not have the complexity of 'real world' practice. The goal of such an assessment is to give students an effective learning experience and to establish the students' competence. There is also a need to not make assignment tasks too large or repetitive. A sizable applied project, in contrast to an assessment exercise, exposes students a much more significant volume of data and requires them to deal with issues such as scalability and reuse.

The application which was analyzed was a simple and relatively straight forward use of XSLT to transform the structure of an xml document. An XML base document had been used to separate data from presentation and to enable transfer of data between applications. XSLT was being used to add back in the detail of how the xml document was to be displayed. The application assembled the information required to produce course outlines from assorted data sources and then published the course outlines information in xhtml and pdf (via xsltfo) format. The part of the application analyzed in this paper was responsible for producing the output. Before producing output all the required information has been assembled into a single well-formed XML document which has been validated against several schemas to ensure the data is complete. This is a typical use of XSLT to produce output on two major formats .

A Python script was used to count xslt elements in all the xsl files used in the application. The logic

of this script were assembled from the recipes for "Walking directory Trees" (Martelli & Ascher 2002 pp144-145) and "Count Tags in A Document" (Martelli & Ascher 2002 pp 382-383). Output from the Python script was piped into a text file and imported into a spreadsheet for final sorting and display.

It is worth noting that the use of xml namespaces greatly simplified the extraction of the appropriate data. Namespaces were designed to deal with problems of recognition and collision (Bray et al 1999) but have received considerable 'bad press' such as "namespaces in XML is one of the greatest disasters that XML's creators have inflicted upon XML" (St. Laurent 2002). Incorrect use of namespaces were a significant cause of bugs in early versions of the application and took considerable effort to correct. The complications of using namespaces were balanced by the easy with which the xsl data could be selectively extracted and analyzed.

# 3. RESULTS

The XSLT specification is one of the larger W3C recommendations and textbooks introducing XSLT include a formidably large dictionary of all the XSLT elements and their options. In practice an encouragingly small subset of commands was used in the studied applications.

Table 1 shows the occurrence of elements over the 18 files in an xslt application which produced html output.

Table 2 table shows the occurrence of elements over the 19 files in an xslt application which produced an xsltfo file for conversion into pdf output.

## 3.1 Modularity

The high use of xsl:stylesheet, xsl:template and xsl:import elements reflects the architecture of the application. One controlling xslt stylesheet read in the xml data and then used many specialized stylesheets containing templates to produce the desired output. As well as importing all necessary subtemplates the main stylesheet contained one template which matched the top courseOutline element of the xml document. This main template established the overall formatting for the output document. For the html output version this formatting was little more than a declaration of an html element containing head and body elements within each of which was an xsl:apply-templates statement. The xsltfo main template followed the same general pattern as the html output but with a fo:root element which contained elements for a fo:layout-master-set, fo:page-sequence, fo:static-content, and fo:flow and

**Table 1: Elements in xslt to html**

| Element | # of times used | In # of files |
|---|---|---|
| xsl:value-of | 71 | 17 |
| xsl:template | 23 | 18 |
| xsl:for-each | 20 | 15 |
| xsl:if | 20 | 6 |
| xsl:stylesheet | 18 | 18 |
| xsl:variable | 18 | 8 |
| xsl:import | 17 | 1 |
| xsl:apply-templates | 13 | 3 |
| xsl:text | 13 | 4 |
| xsl:number | 12 | 11 |
| xsl:param | 11 | 11 |
| xsl:attribute | 6 | 4 |
| xsl:key | 1 | 1 |
| xsl:strip-space | 1 | 1 |
| xsl:with-param | 1 | 1 |

**Table 2: Elements in xsltfo to pdf**

| Element | # of times used | In # of files |
|---|---|---|
| xsl:attribute | 231 | 2 |
| xsl:value-of | 73 | 18 |
| xsl:attribute-set | 39 | 1 |
| xsl:template | 23 | 18 |
| xsl:for-each | 21 | 16 |
| xsl:if | 19 | 5 |
| xsl:stylesheet | 19 | 19 |
| xsl:import | 18 | 1 |
| xsl:variable | 18 | 8 |
| xsl:text | 14 | 3 |
| xsl:apply-templates | 13 | 3 |
| xsl:param | 2 | 2 |
| xsl:key | 1 | 1 |
| xsl:number | 1 | 1 |
| xsl:strip-space | 1 | 1 |

had much more specific document level display detail. The imported sub-templates were each responsible for assembling the information displaying one section of a course outline. This allowed the display characteristic of the different parts of a course outline to be altered independently. As well as producing course outlines, different subsets of the xml source documents can also be extracted for in brochures, posters, course handbooks and marketing documents by different versions of the main stylesheet. Most introductory exercises do not use a modular design but are done in one file. Teaching the use of import statement is the XSLT equivalent of teaching the structured programming technique of functional decomposition.

## 3.2 Procedural programming

The overall processing strategy favoured a 'pull processing' (Kay 2001, p80 Bonneau *et al.* p 187). Each stylesheet template determined the structure of the result document, pulled in required data nodes (with xsl:value-of) and handled the display of them. The XSLT equivalents of conventional control structures (xsl:for-each, xsl:if) were used extensively as were variables and parameters (xsl:param and xsl:variable). This is a very straightforward approach that is recommended if the structure of the source xml document is not going to change significantly over time (Kay 2001, p161). XSLT also supports a declarative programming rules-based 'push' approach (Kay 2001, 161). It is however a mistake to over-emphasize the push approach to using XSLT at the expense of teaching 'normal' programming constructs. It is perfectly sensible to use an XSLT programming style which is readily grasped by procedural trained programmers.

## 3.3 Styling

The head element of the version of the application which output html contained a link to a cascading style sheet which added detailed styling. Using CSS neatly allows for a separation of detailed styling information from the basic layout of a document. Detailed styling was applied to the xsltfo output using the xsl:attribute and xsl:attribute-set elements. This 'trick' allowed a common set of styling attributes to be defined in once and then applied to multiple output elements (Kay 2001 p173-179) to get the same effect as using CSS with html.

## 3.4 Useful tricks

There are some XSLT features which do not get used much, but are very valuable to be aware of as they make life much easier when they are needed. A useful tip is to outputting text with hard spaces by using xsl:text after setting defaults for xsl:strip-space. It is also very useful to have numbered lists and items handled automatically with xsl:number. If you need to replace a numeric value with a word (ie months as words instead of numeric values) then xsl:key can be used to create the equivalent of a index or hastable data structure. Creating a case statement to do the equivalent processing is not a pretty sight in verbose XSLT.

## 3.5 Other tips

There were also some useful development process "discoveries" which are not apparent from an examination of the element count data. It was very useful to develop the application in two steps. In the first iteration all formatting was omitted and the major focus was getting the correct XPATH statements in the select attribute of xsl:value-of elements. Only after all the correct data was appearing in the correct sequence was formatting added. The way an XSLT processor works means that data is always located relative to an ever changing context node. Specifying a full absolute path to data does not allow for differences in the structure of the source data. Trying to debug both errors in locating data and errors in displaying data at the same time is very difficult and not recommended.

It was also valuable to use a command line XSLT parser which produced text output which could be piped to a file and examined in a text editor. XSLT parsers built into browsers render output for display. A bug in the display commands often leaves the programmer looking at a blank screen and unable to get any hint from 'view source' which merely shows the original xml source document with a link to the xsl stylesheet. Fancy xslt-aware integrated development environments also proved to be fragile and unhelpful when debugging.

# 4. CONCLUSIONS

Examination of a hello world program would seem to indicate that printing text and using literal strings are important programming practices. In fact graphical GUI interfaces dominate business applications and it is more usual to use sophisticated model-view-controller constructs to synchronize central data to multiple displays. Concerns about internationalization means some mechanism for "translatable strings" is preferred over using literals. An examination of the frequency of element use within the two versions of a 'real world' XSLT application strongly suggests that beginner focused XSLT examples also can be misleading. A 'real' XSLT application has to deal with pragmatic issues of modularity and control in a sophisticated manner. There are also specialized ways to perform common tasks that are well worth knowing about. The teacher of XSLT does not need to fully cover the whole of the language but can focus on these elements.

# REFERENCES


Bonneau, S. Kohi, T. Tennison, J. Duckett, J. & Williams. K. (2003) XML Design Handbook. Wrox Press Ltd.

Bray, T., Hollander, D. & Layman. A. Editors (1999) Namespaces in XML. Available from http://www.w3.org/TR/1999/REC-xml-names-19990114/

Clarke, J. Editor. (1999) XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999. Available from http://www.w3.org/TR/1999/REC-xslt-19991116

Kay, M. (2001) XSLT Programmer's Reference 2nd Edition. Wrox Press Ltd.

Martelli, A. & Ascher, D. (2002) Python Cookbook. O'Reilly & Associates, Inc

St.Laurent, S. (2002) Namespaces as Opportunity. Available from http://www.monasticxml.org/namespaces.html

Tidwell, D. (2001) XSLT. O'Reilly & Associates, Inc