Report on the Successes and Failures of the U.S. Legal Code Treemapping Project

Joshua Daniel Westmoreland

CS 6985 – Legal Code Visualization

Dr. Lewis Baumstark

Computer Science Department

University of West Georgia

07/30/2009

Introduction And Preliminary Information

This project was undertaken as a directed study in the Summer Semester (Session III) of 2009 at the University of West Georgia under direction from Dr. Lewis Baumstark. Its purpose was multifold:

- Research the topics of Cyclomatic Complexity, Treemapping, and the United States Legal Code (particularly its structure).
- Build a UML data model of the general structure of the United States Legal Code's general structure.
- Build a treemap of one of the Titles of United States Legal Code manually in order to get an idea of what the treemap file (XML) and its interpretation should look like in the chosen treemap visualization program[1].
- Take the Title that was treemapped "by hand", look for places where one section references another, map these references out "by hand", and then attempt to apply simple cyclomatic complexity measures to the result(s).
- Create a program, the language of which was left to choice, to automate the process of interpreting United States Legal Code XML files found on Cornell University Law School's website[2].

Research

There was little problem with the first part of this project which entailed researching the topics of cyclomatic complexity, treemapping, and the U.S Legal Code itself. For the topics of cyclomatic complexity and treemapping I used Wikipedia[3] as a starting point as well as doing several Google searches on these topics and found a great deal of information which I was unfortunately unable to use due to the project not progressing far enough due to time constraints. I was already somewhat familiar with the U.S Legal Code as I had a concentration in Pre-Law as an Undergraduate. However, the Code itself is an incredibly complex set of documents and it would take a great deal of time to understand it completely. Fortunately, my task was reasonably simple: to understand the structure of the Code for the purposes of visualization. This was achieved fairly quickly and led to my next task: creating a UML model of the general structure of the U.S Legal Code.

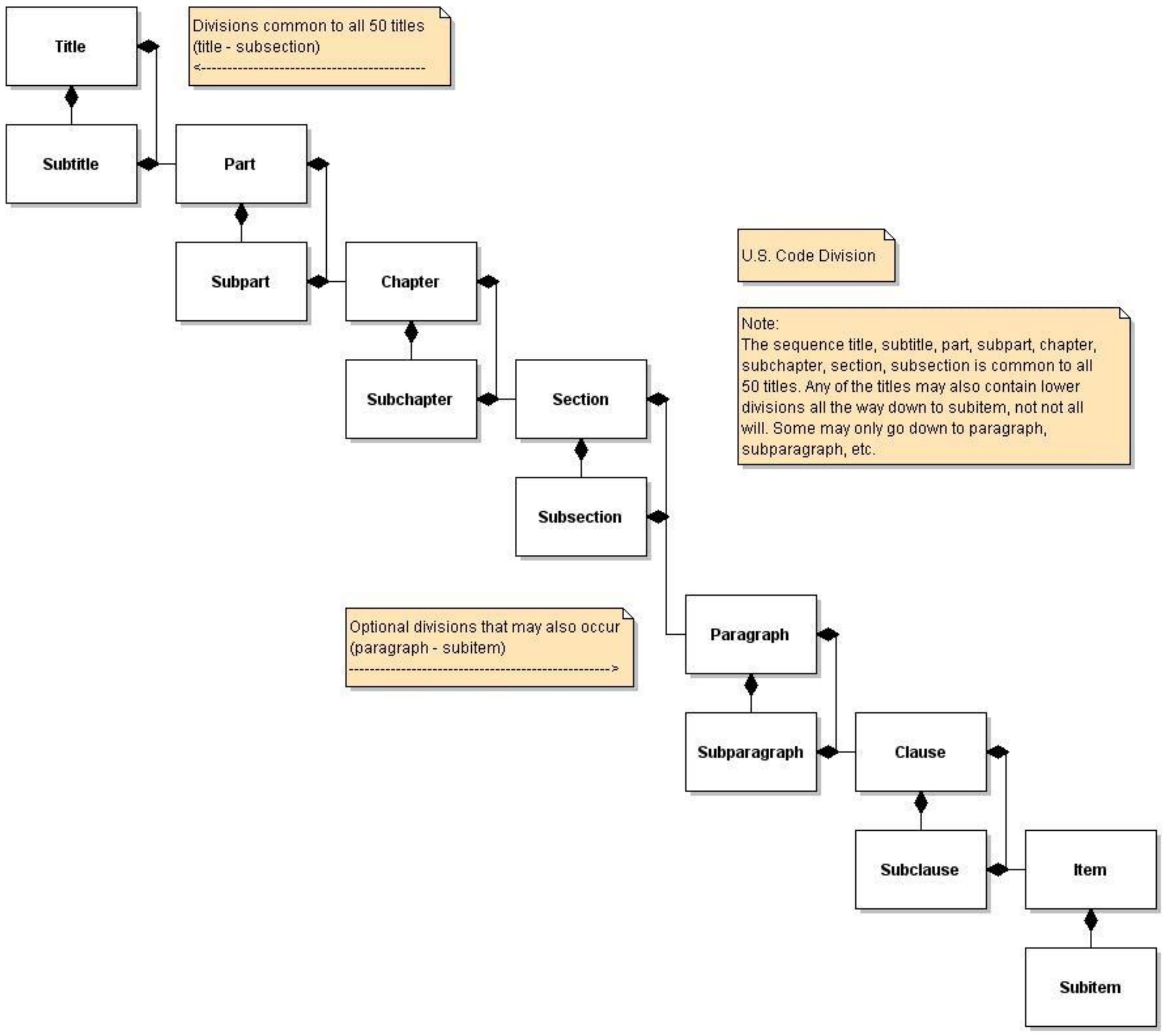
[1] Treemap 4.1, a program for creating visual representations (treemaps) of complex hierarchical structures
Found here: http://www.cs.umd.edu/hcil/treemap-history/

[2] Found at http://voodoo.law.cornell.edu/uscxml/

[3] Cyclomatic Complexity on Wikipedia: http://en.wikipedia.org/wiki/Cyclomatic_complexity
Treemapping on Wikipedia: http://en.wikipedia.org/wiki/Treemapping

<u>UML Model</u>

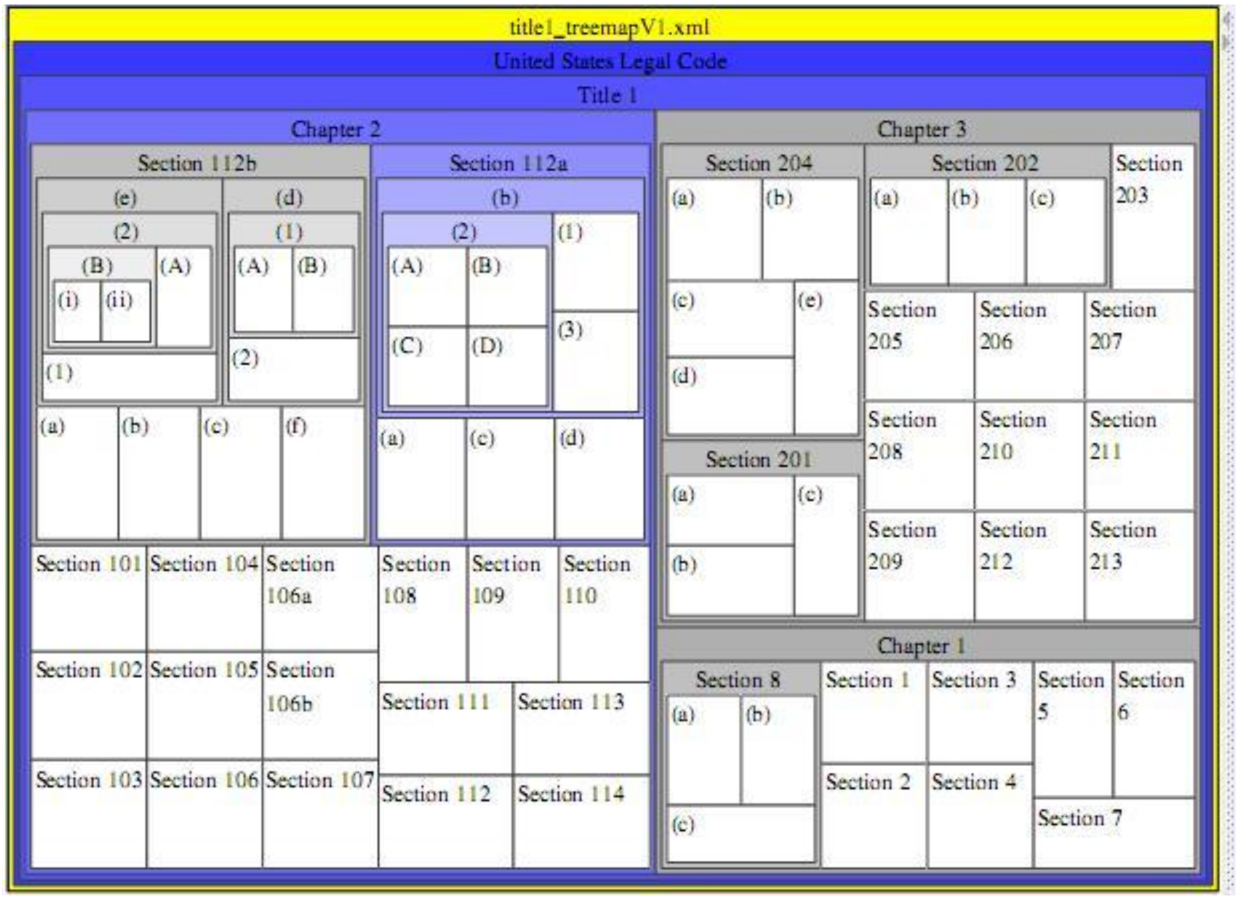
My next task was to create a UML model of the general structure of the U.S. Legal Code. This is the result:


Title
Subtitle
Part
Subpart
Chapter
Subchapter
Section
Subsection
Paragraph
Subparagraph
Clause
Subclause
Item
Subitem
Divisions common to all 50 titles (title - subsection) <---
U.S. Code Division
Note: The sequence title, subtitle, part, subpart, chapter, subchapter, section, subsection is common to all 50 titles. Any of the titles may also contain lower divisions all the way down to subitem, not not all will. Some may only go down to paragraph, subparagraph, etc.
Optional divisions that may also occur (paragraph - subitem) --->

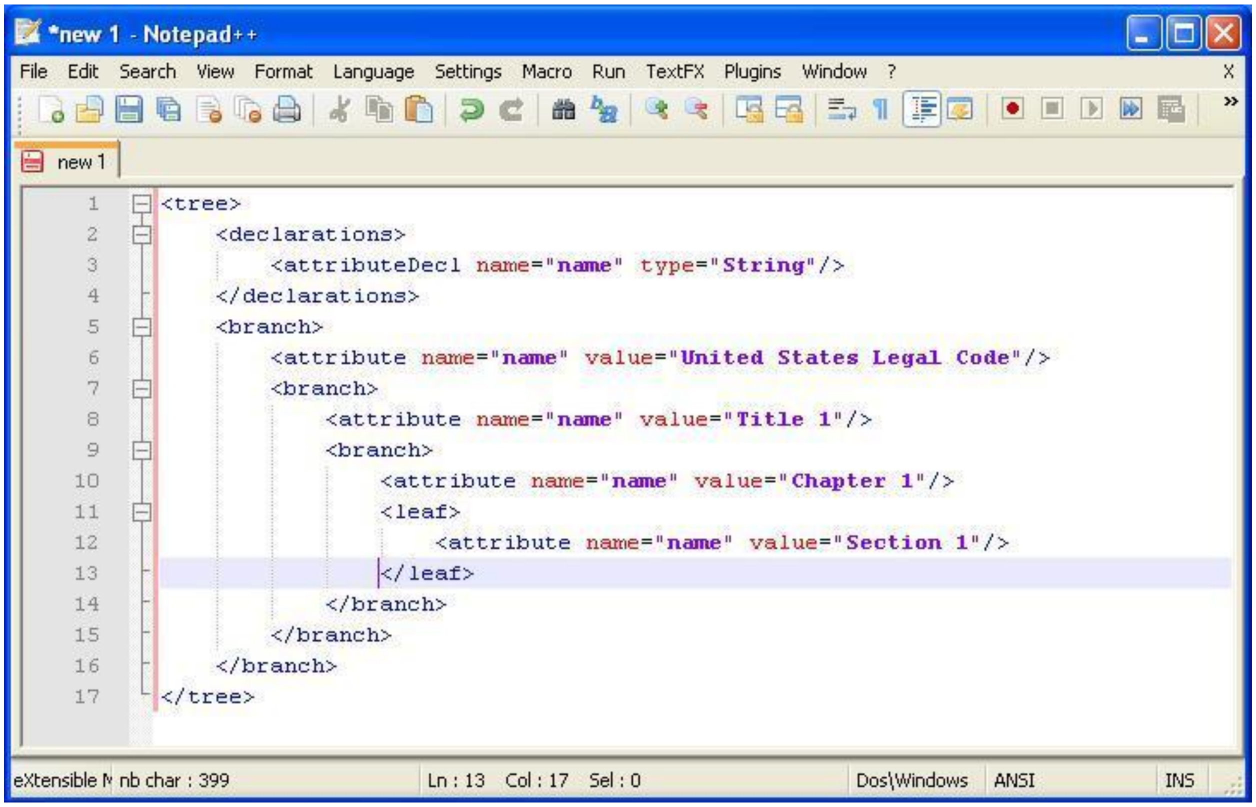
The notes in this images help to explain the structure of this code in some degree of detail, but the following can generally be said of the U.S. Legal Code in general regards to its structure: The U.S. Legal Code has many Titles, a Title may have many Subtitles, Titles and Subtitles may have many Chapters, a Chapter may have many Subchapters, Chapters and Subchapters may have many Sections, a Section may have many Subsections. The following entails subdivisions that exist below Subsection that are not present in all 50 titles: Sections and subsections may have a number of Paragraphs, Paragraphs may have a number of Subparagraphs, Paragraphs and Subparagraphs may have a number of Clauses, Clauses may have a number of Subclauses, Clauses and Subclauses may have a number of Items, Items may have a number of Subitems. A very complex structure, of this there is no doubt.

Manually Built Treemap

The following is a manually built, that is to say built without the aid of any program or script, treemap of Title 1 of the U.S. Legal Code:



This entailed building an XML file with a structure amenable to being interpreted by Treemap 4.1. An example of this:


*new 1 - Notepad++
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window ?
new 1
eXtensible M nb char : 399
Ln : 13 Col : 17 Sel : 0
Dos\Windows
ANSI
INS

```xml
<tree>
    <declarations>
        <attributeDecl name="name" type="String"/>
    </declarations>
    <branch>
        <attribute name="name" value="United States Legal Code"/>
        <branch>
            <attribute name="name" value="Title 1"/>
            <branch>
                <attribute name="name" value="Chapter 1"/>
                <leaf>
                    <attribute name="name" value="Section 1"/>
                </leaf>
            </branch>
        </branch>
    </branch>
</tree>
```

The XML structure seen above is but a small snapshot of how a complete XML representation of a Title of the U.S. Legal Code would look, but it provides a general idea of the structure of such a file. The tree must have a root, literally "tree" in this case, followed by branches off of the root of the tree as well as branches off of other branches as well. After that there are leaves and leaves can have other leaves, but that outermost node in a tree such as this must be a leaf.

Cyclomatic Complexity Measure

This was not done due to both constraints on time and a shift in focus from the theoretical measures presented by cyclomatic complexity to concrete visualization in the form of treemapping. From this point work was begun on a program to automate conversion of the uscodexml found online to a format that was readable by Treemap 4.1.

<u>Programmatic XML conversion</u>

See next section for thorough details on how this part of the project worked out, including both successes and failures.

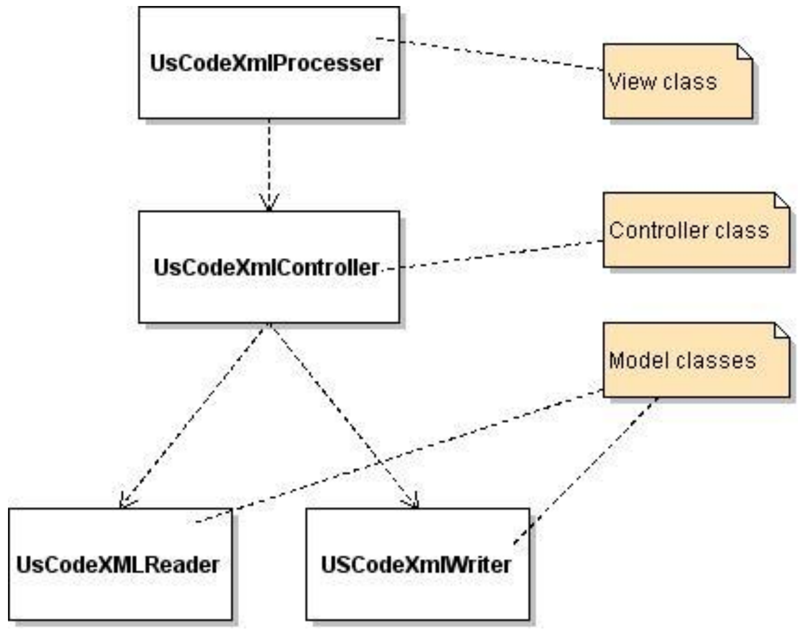
Programmatic XML Conversion: The Project As It Stands

At the beginning of July 2009 work was begun on a program to automate the process of converting the uscodexml found at the Cornell University Law School website into a format that could be read and visualized by Treemap 4.1. In this section the following topics will be covered:

- Initial steps taken in writing the program
- Details about the program as it currently stands
- Difficulties that arose during the process

Initial Steps

The first thing that had to be done when the decision was made to begin the process of writing a program to achieve the above mentioned goals was to select a language in which to write said program. C# was chosen at this point because of my familiarity with the language and the fact that C# has a reasonable well-constructed XML parser (System.Xml) included in its libraries. With the language selected I then began planning out what functionalities, and there for classes, that would be needed and it was decided that a simple model-view-controller structure would work best for this program as it works for most any type of program. Below is a UML model of the classes and methods as they came to be:



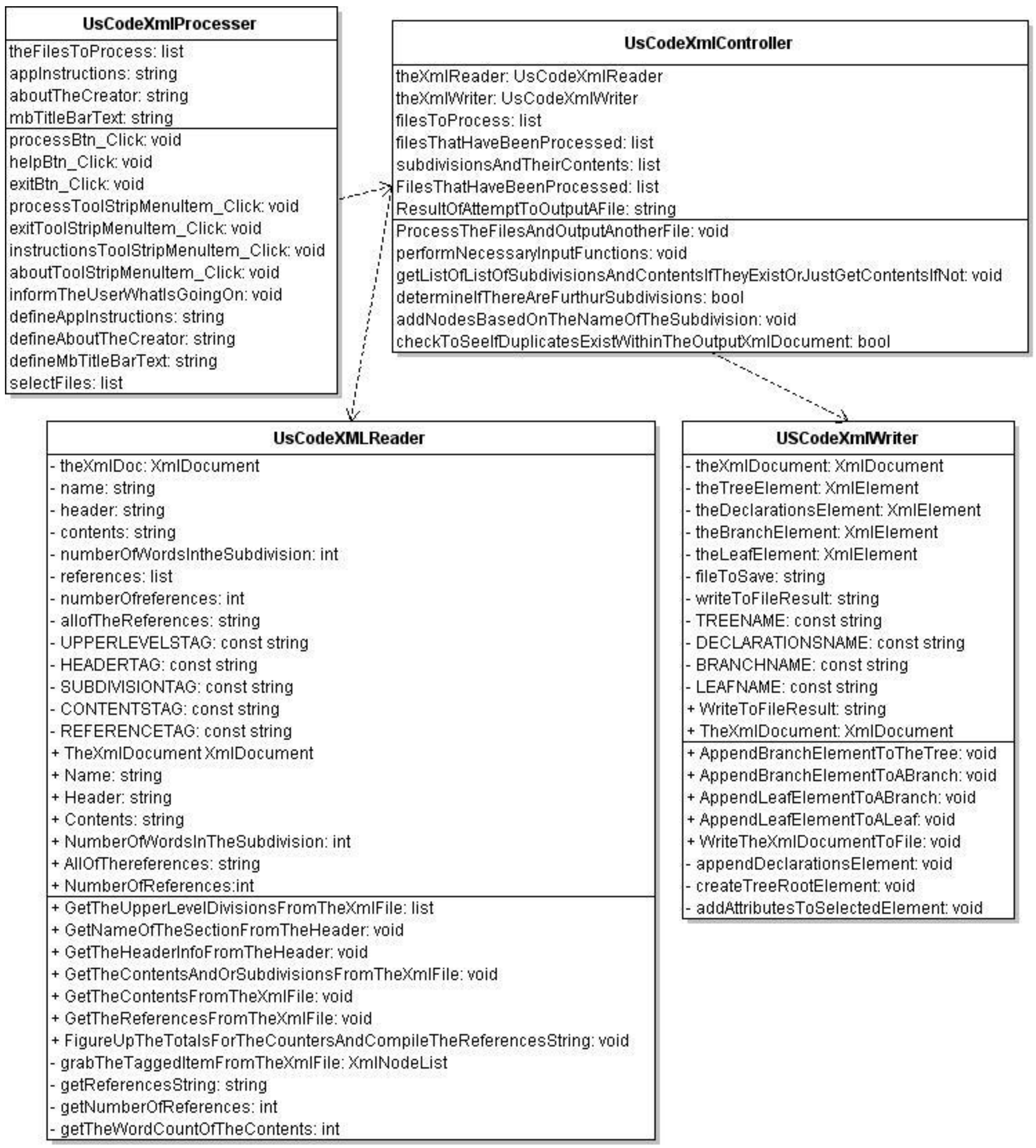
The intended functions of these classes were as follows:

- The UsCodeXmlReader model class was to handle all of the input processes necessary, such as grabbing the necessary things from the uscodexml files being processed
- The UsCodeXmlWriter model class was to handle all of the output processes necessary, such as writing XmlElements with specified attributes to an XmlDocument to later be saved out to a file.
- The UsCodeController controller class was to actually specify how the input and output processes were to be handled.
- The UsCodeProcessor view class was to be a GUI, something much friendlier than a command line interface, to handle the user's interactions with the program.

It was not until the implementation of these classes that problems began to occur.

The Program as It Currently Stands

The program as it currently stands looks something like this (the image can be seen on the next page due to its large size and degree of detail:


UsCodeXmlProcesser
theFilesToProcess: list
appInstructions: string
aboutTheCreator: string
mbTitleBarText: string
processBtn_Click: void
helpBtn_Click: void
exitBtn_Click: void
processToolStripMenuItem_Click: void
exitToolStripMenuItem_Click: void
instructionsToolStripMenuItem_Click: void
aboutToolStripMenuItem_Click: void
informTheUserWhatIsGoingOn: void
defineAppInstructions: string
defineAboutTheCreator: string
defineMbTitleBarText: string
selectFiles: list
UsCodeXmlController
theXmlReader: UsCodeXmlReader
theXmlWriter: UsCodeXmlWriter
filesToProcess: list
filesThatHaveBeenProcessed: list
subdivisionsAndTheirContents: list
FilesThatHaveBeenProcessed: list
ResultOfAttemptToOutputAFile: string
ProcessTheFilesAndOutputAnotherFile: void
performNecessaryInputFunctions: void
getListOfListOfSubdivisionsAndContentsIfTheyExistOrJustGetContentsIfNot: void
determineIfThereAreFurthurSubdivisions: bool
addNodesBasedOnTheNameOfTheSubdivision: void
checkToSeeIfDuplicatesExistWithinTheOutputXmlDocument: bool
UsCodeXMLReader
- theXmlDoc: XmlDocument
- name: string
- header: string
- contents: string
- numberOfWordsIntheSubdivision: int
- references: list
- numberOfreferences: int
- allofTheReferences: string
- UPPERLEVELSTAG: const string
- HEADERTAG: const string
- SUBDIVISIONTAG: const string
- CONTENTSTAG: const string
- REFERENCETAG: const string
+ TheXmlDocument XmlDocument
+ Name: string
+ Header: string
+ Contents: string
+ NumberOfWordsInTheSubdivision: int
+ AllOfThereferences: string
+ NumberOfReferences:int
+ GetTheUpperLevelDivisionsFromTheXmlFile: list
+ GetNameOfTheSectionFromTheHeader: void
+ GetTheHeaderInfoFromTheHeader: void
+ GetTheContentsAndOrSubdivisionsFromTheXmlFile: void
+ GetTheContentsFromTheXmlFile: void
+ GetTheReferencesFromTheXmlFile: void
+ FigureUpTheTotalsForTheCountersAndCompileTheReferencesString: void
- grabTheTaggedItemFromTheXmlFile: XmlNodeList
- getReferencesString: string
- getNumberOfReferences: int
- getTheWordCountOfTheContents: int
USCodeXmlWriter
- theXmlDocument: XmlDocument
- theTreeElement: XmlElement
- theDeclarationsElement: XmlElement
- theBranchElement: XmlElement
- theLeafElement: XmlElement
- fileToSave: string
- writeToFileResult: string
- TREENAME: const string
- DECLARATIONSNAME: const string
- BRANCHNAME: const string
- LEAFNAME: const string
+ WriteToFileResult: string
+ TheXmlDocument: XmlDocument
+ AppendBranchElementToTheTree: void
+ AppendBranchElementToABranch: void
+ AppendLeafElementToABranch: void
+ AppendLeafElementToALeaf: void
+ WriteTheXmlDocumentToFile: void
- appendDeclarationsElement: void
- createTreeRootElement: void
- addAttributesToSelectedElement: void

Simply from looking at the size of these classes one can begin to see the problems that must have occurred in the course of writing the program. The classes are far too large and complicated and should have been split up into smaller, more specific helper classes in order to make this program function properly. Now, let us go through each class, one method at a time, and try to find out what went wrong and where.

The `UsCodeXmlProcessor` Class

This class was to serve as a view, handling all interaction with the user. I will not go into a great amount of detail here as this is simply a view for the program and because everything in this class actually functioned as intended. The methods and attributes can be viewed in the class diagram on page 9.

The `UsCodeController` Class

This class was to serve as a controller, managing the 2 model classes UsCodeXmlWriter and UsCodeXmlReader and thus managing all of the input and output processes, for the program and contained the following methods:

- `public void ProcessTheFilesAndOutputAnotherFile()`
    - Loops through the list of files to be processed, performs all necessary input functions, performs all necessary output functions, and finally adds the file that was just processed to the list of files that have been processed
    - **Did not work due to problems with helper methods**
- `private void performNecessaryInputFunctions()`
    - Called all of the necessary input functions
    - **Did not work due to problems with helper methods**
- `private void getListOfListOfSubdivisionsAndContentsIfTheyExistOrJustGetContentsIfNot()`
    - Did just what the method name said it was supposed to do
- `private bool determineIfThereAreFurthurSubdivisions()`
    - Determines if there are any further subdivisions in the section
- `private void addNodesBasedOnTheNameOfTheSubdivision()`
    - Appends nodes to the tree based on the name being read in from the file
    - **Did not work due to a faulty algorithm**
- `private bool checkToSeeIfDuplicatesExistWithinTheOutputXmlDocument()`
    - Checks to see if any elements already exist with the name being read in from the uscodexml file

The `UsCodeXmlWriter` Class

This model class defined all output processes necessary for writing an XML file that would have been readable by the Treemap 4.1 program and contained the following methods:

- `public void AppendBranchElementToTheTree(string nameValue, string headerValue, string contentsValue, int wordCountValue, string referencesValue, int numberOfReferencesValue)`
  - Appends a branch element to the tree with the passed in attribute values
  - Parameters:
    - `nameValue` - the value of the name (of the subdivision) attribute to be written to the element
    - `headerValue` - the value of the header (of the subdivision) attribute to be written to the element
    - `contentsValue` - the value of the contents (of the subdivision) attribute to be written to the element
    - `wordCountValue` - the value of the wordCount (of the subdivision) attribute to be written to the element
    - `referencesValue –` the value of the references (of the subdivision) attribute to be written to the element
    - `numberOfReferencesValue –` the value of the numberofReferences (of the subdivision) attribute to be written to the element
- `public void AppendBranchElementToABranch(string nameValue, string headerValue, string contentsValue, int wordCountValue, string referencesValue, int numberOfReferencesValue)`
  - Appends a branch element to a branch the passed in attribute values
  - Parameters:
    - `nameValue` - the value of the name (of the subdivision) attribute to be written to the element
    - `headerValue` - the value of the header (of the subdivision) attribute to be written to the element
    - `contentsValue` - the value of the contents (of the subdivision) attribute to be written to the element
    - `wordCountValue` - the value of the wordCount (of the subdivision) attribute to be written to the element
    - `referencesValue –` the value of the references (of the subdivision) attribute to be written to the element
    - `numberOfReferencesValue –` the value of the numberofReferences (of the subdivision) attribute to be written to the element

- `public void AppendLeafElementToABranch(string nameValue, string headerValue, string contentsValue, int wordCountValue, string referencesValue, int numberOfReferencesValue)`
    - Appends a leaf element to a branch with the passed in attribute values
    - Parameters:
        - `nameValue` - the value of the name (of the subdivision) attribute to be written to the element
        - `headerValue` - the value of the header (of the subdivision) attribute to be written to the element
        - `contentsValue` - the value of the contents (of the subdivision) attribute to be written to the element
        - `wordCountValue` - the value of the wordCount (of the subdivision) attribute to be written to the element
        - `referencesValue –` the value of the references (of the subdivision) attribute to be written to the element
        - `numberOfReferencesValue –` the value of the numberofReferences (of the subdivision) attribute to be written to the element
- `public void AppendLeafElementToALeaf(string nameValue, string headerValue, string contentsValue, int wordCountValue, string referencesValue, int numberOfReferencesValue)`
    - Appends a leaf element to a leaf with the passed in attribute values
    - Parameters:
        - `nameValue` - the value of the name (of the subdivision) attribute to be written to the element
        - `headerValue` - the value of the header (of the subdivision) attribute to be written to the element
        - `contentsValue` - the value of the contents (of the subdivision) attribute to be written to the element
        - `wordCountValue` - the value of the wordCount (of the subdivision) attribute to be written to the element
        - `referencesValue –` the value of the references (of the subdivision) attribute to be written to the element
        - `numberOfReferencesValue –` the value of the numberofReferences (of the subdivision) attribute to be written to the element
- `public void WriteTheXmlDocumentToFile()`

- Writes the XML document out to file

- `private void appendDeclarationsElement()`
  - Adds the attribute declarations element to the tree
  - **Does not work properly due to the nature of the attribute "nodes" required by the Treemap 4.1 program. These were added as InnerText properties, but the < and > did not display and were instead replaced by *&lt;* and *&gt;***
- `private void createTreeRootElement()`
  - Creates the XML declaration and root of the tree
- `private void addAttributesToSelectedElement(XmlElement selectedElement, string nameValue, string headerValue, string contentsValue, int wordCountValue, string referencesValue, int numberOfReferencesValue)`
  - Adds attributes with the passed in values to the selected XML element
  - Parameters:
    - `selectedElement` - the element to which these things are to be added
    - `nameValue` - the value of the name (of the subdivision) attribute to be written to the element
    - `headerValue` - the value of the header (of the subdivision) attribute to be written to the element
    - `contentsValue` - the value of the contents (of the subdivision) attribute to be written to the element
    - `wordCountValue` - the value of the wordCount (of the subdivision) attribute to be written to the element
    - `referencesValue –` the value of the references (of the subdivision) attribute to be written to the element
    - `numberOfReferencesValue –` the value of the numberofReferences (of the subdivision) attribute to be written to the element

The `UsCodeXmlReader` Class

This model class defined all input processes necessary for reading an XML file that was part of the uscodexml set of files and extracting data necessary to output a file that would have been readable by the Treemap 4.1 program and contained the following methods:

- `public List<string> GetTheUpperLevelDivisionsFromTheXmlFile()`
  - Grabs the nodes with the upper level subdivisions and adds the inner text, and thus the names of those upper level subdivisions, of those nodes to a list

    - Returns: A list of string items representing the subdivisions above section
- `public void GetNameOfTheSectionFromTheHeader()`
    - Grabs the header text from the XML file, grabs the section number out of it, and assigns the value to the data member "name"
    - **Does not function as intended; a better algorithm for extracting the name should have been formulated**
- `public void GetTheHeaderInfoFromTheHeader()`
    - Grabs the header text from the XML file, grabs everything after the section number out of it, and assigns the value to the data member "header"
    - **Does not function as intended; a better algorithm for extracting the name should have been formulated**
- public List<List<string>> GetTheContentsAndOrSubdivisionsFromTheXmlFile()
    - Returns: A list of lists (2) of strings, the first list [0] containing the names of the subdivisions below section and the second list [1] containing the contents of those subdivisions at a corresponding index
- `public void GetTheContentsFromTheXmlFile()`
    - Gets the text from the contents nodes and appends it to the “contents” string
    - **Does not function as intended; a better algorithm for extracting the name should have been formulated**
- `public void GetTheReferencesFromTheXmlFile()`
    - Gets the text from the reference nodes and adds it to the list of references
- `public void FigureUpTheTotalsForTheCountersAndCompileTheReferencesString()`
    - Figures up the totals of the number of words in the subdivision, the number of references, and compiles a string of all of the references in a subdivision
- `private XmlNodeList grabTheTaggedItemFromTheXmlFile(string tagName)`
    - Returns a XmlNodeList of the elements with tags matching the passed in string tagName and returns an XmlNodeList of those items
- `private string getReferencesString()`
    - Iterates through the list of references and returns a string containing all of the references, each on a new line
- `private int getNumberOfReferences()`
    - Returns the number of references in the references list
- `private int getTheWordCountOfTheContents()`
    - Returns the number of words in the contents string

Difficulties

Several difficulties arose throughout the course of this project. How these could have been avoided and potential resolutions will be covered in the next section.

- The primary difficulty that arose during the course of this project was dealing with the format of the XML files being read in. These XML files were clearly meant to be the back end of a set of web pages and did not translate very well for the intended purpose of this project.
- Another problem may have been that I decided to construct the translation program in C#, a programming language I have not actively used in over a year. The XML libraries of the language are significant and would have served me well had I been using a language I was more familiar with.
- Yet another issue that arose was my general unfamiliarity with XML in general and the fact that I ended up doing a lot of reading towards the end of the project when time began to become an issue.
- The relatively short length of the summer semester presented some major time management challenges, but I do not believe that this severely impacted the project in any significantly negative way.

Programmatic XML Conversion: The Project As It Should Have Been

First off, I believe there is absolutely no need to redesign this program from the ground up as I do not believe the design flaws were quite that massive. The existing program could have been made to function if more time had been allowed. However, there were a number of improvements that could have been made.

The `UsCodeXmlProcessor` Class

There are no major design flaws in this class as it was designed as a simple view to interact with the program and therefore redesigning it is unnecessary.

The `UsCodeController` Class

No major redesign of this class is necessary except for fixing the problems that currently exist in the class. These are:

- `public void ProcessTheFilesAndOutputAnotherFile()`
  - **Did not work due to problems with helper methods**
  - Resolution: Fix the helper methods
- `private void performNecessaryInputFunctions()`
  - **Did not work due to problems with helper methods**
  - Resolution: Fix the helper methods
- `private void addNodesBasedOnTheNameOfTheSubdivision()`
  - **Did not work due to a faulty algorithm**
  - Resolution: Take the time to formulate a better algorithm

The `UsCodeXmlWriter` Class

Some of the methods in this class function as intended, but several still do not. This could be resolved with a few simple modifications.

- None of the `Append` Element methods work as intended as they do not add the proper element in the proper place
  - Resolution: this could be solved by adding in a few conditionals to make the decisions of what needs to be added where based on the name of the previous subdivisions.

        - Example: if the previous XmlElement added was a section (leaf) that was appended to a Part (branch) and the next XmlElement to be appended is another section then that XmlElement should be appended to the Part below the previous leaf (a section) that was added to the branch (also a section).

```
<branch>
        <attribute name=”name” value=” Part X”/>
        <leaf>
                <attribute name=”name” value=” Section X”/>
        </leaf>
        <leaf>
                <attribute name=”name” value=” Section Y”/>
        </leaf>
</branch>
```

- The `appendDeclarationsElement()` and `addAttributesToSelectedElement()` methods does not work correctly in its current form. It could be fixed by finding a way to append literals of “<” and “>” instead of “&lt;” and “&gt;” in the inner text of an XmlElement. Otherwise, there is no way to add the attributes in the necessary format, which is a sort of unusual half-node, to the respective XmlElements.

<u>The `UsCodeXmlReader` Class</u>

As with the UsCodeXmlWriter class, some of the methods in this class function as intended, but several still do not. These problems are more severe than those of the UsCodeXmlWriter class and would require more complex problem solving.

- The `GetNameOfTheSectionFromTheHeader()`, `GetTheHeaderInfoFromTheHeader()`, and `GetTheContentsFromTheXmlFile()` methods all suffered from what was essentially the same problem: getting the incorrect items from the XML file being read in. This could be resolved by:
    - First, doing further research into how the U.S. Code is broken up in the uscodexml files.
    - Secondly, grab the proper items from the uscodexml file.

The primary problem with this class was that it was far too large and clunky. This could have been solved by refactoring this class and extracting a helper class or two and having objects of those classes present in this class instead of having everything jammed into this class as it was. A good way to do this perhaps would have been to:

- Extract the methods, as well as the appropriate data members and properties, that grabbed the necessary items from the XML file an extract these into a helper class that I would have called `UsCodeXmlElementReader`. The methods I would have extracted include:
    - `GetTheUpperLevelDivisionsFromTheXmlFile()`
    - `GetNameOfTheSectionFromTheHeader()`
    - `GetTheHeaderInfoFromTheHeader()`
    - `GetTheContentsAndOrSubdivisionsFromTheXmlFile()`
    - `GetTheContentsFromTheXmlFile()`
    - `GetTheReferencesFromTheXmlFile()`
    - `grabTheTaggedItemFromTheXmlFile()`

What was left could have been left in the class as it was and the `UsCodeXmlElementReader` object could have performed it necessary function as a helper class.

General Comments about Problem Resolution in Version 2.0

- *Problem:* Clunky format of the XML files being read in by the program
  Resolution: I would read in the data from each XML file as text, that is to say grab the string information from the nodes that contain such data, and create a new text document for each XML file and would then read each of these files and take what I needed from the text as I am much more comfortable dealing with text files than with XML.
- *Problem:* C# unfamiliarity
  Resolution: If I actually were to begin again from the beginning I would certainly not use C# as my programming language of choice. I would use a language with which I am much more familiar with: Ruby. Ruby has an XML processing module called REXML that is apparently very efficient and might have been more useful than C#'s System.Xml.
- Problem: I was unfamiliar with XML at the beginning
  Resolution: I would more extensively research XML in the early stages of planning in order to prepare myself for the tasks necessary later in the course of the project. I borrowed several books and read them towards the end of the project, but it was too late by that point to correct the problems that had cropped up in my program.
- *Problem:* The relatively short length of the summer semester
  Resolution: I would budget my time more efficiently, especially towards the beginning when there was an abundance of time to do research and prepare myself for what lay ahead of me: the program itself.

Conclusions

In conclusion, I would like to say, although it might sound incredibly cliché, that I have learned quite a lot from doing this project. I learned quite a good deal about the following:

- C#'s XML processing abilities via `System.Xml`
- *Cyclomatic Complexity* as a theoretical complexity measure for complex structures, although I never actually got to apply it to the project
- *Treemapping* as a visual complexity measure for complex structures
- That the "*best laid plans of mice and men often go astray*" when developing software of any size or complexity. These problems just become larger and more complex in direct correlation to the project where they occur.

I am grateful for the opportunity I was given to learn about these topics and that I was given the opportunity to write this report to explain the project and its results. I hope it has been illuminating.