|  |  |  |  |
| --- | --- | --- | --- |
|  | 3/2/2015 |  | |
| |  |  | | --- | --- | |  |  | |  |  | |  |  | |  |  | | |  | |
| SSE 554 Project 2 Report  *Test Driven Dev., Distributed Version Control, and Extensible Markup Language* | | | |
|  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |
|  |  | |  |

|  |  |  |
| --- | --- | --- |
|  | Daryl Ebanks, David May, Josh Deremer, Kei’Shawn Tention |  |

Table of Contents

[Requirements and Testing 2](#_Toc413089757)

[Summary 2](#_Toc413089758)

[Requirements (for I/O Class) 2](#_Toc413089759)

[Testing 2](#_Toc413089760)

[Description of Code 2](#_Toc413089761)

[XMLIO 2](#_Toc413089762)

[Database 2](#_Toc413089763)

[GUI 2](#_Toc413089764)

[Application at Work 2](#_Toc413089765)

[Distributed Version Control System 2](#_Toc413089766)

[An Picture is worth 1000 words 2](#_Toc413089767)

[Appendix I Non-Direct Activity Reports 2](#_Toc413089768)

[Appendix II Full Unit Testing Code 2](#_Toc413089769)

[Appendix III Source Code 2](#_Toc413089770)

# Requirements and Testing

## Summary

Once again the TDD process was used to develop portions of this code. A set of tests were developed from the requirements of the classes required to write this application.

## Requirements (for I/O Class)

The requirements were as follows:

* Use XML as a means of storing the information in a file
  + Reading from an XML file and ~~parsing the document for its information~~
    - Building the document tree so it can be used within the program
  + Writing to an XML file with the same
    - ~~Building the document tree so the xml document will be properly formatted~~.
    - Accepting the document tree

These requirements created the XML I/O Test classes and XML I/O classes.

The lines that have a strike were later removed to keep the File I/O about files exclusively.

<image of XMLIO class diagram>

## Testing

# Description of Code

## XMLIO

## Database

## GUI

### Using a DTD for XML File Validation

In most circumstances, it becomes essential that XML code has been written in the proper format. In many cases, XML processors are expecting an XML file that has a certain format, including specific tags and attributes associated with those tags. Because of this, different testing methods have been developed for XML files to ensure they are of the proper format. However, as stated earlier, one of the key and powerful advantages of XML is that it is extensible – users can create their own tags to be included in a file. With many different XML files that require a certain set of tags, one XML file may very well require a different set of rules than another. In this situation, a data validation file is practically essential.

There are two different types of validation files that are used often with XML files: Document Type Definitions (DTDs) and XML Schemas. The first, DTD, was utilized in this project to ensure the GUI XML files followed the proper XML standard for tags and attributes that the program was expecting. The difference between these two file types is basic: complexity. DTDs are simple to write, but not very powerful, whereas conversely, XML schema files can be extremely powerful, but become much more complicated.

DTDs contain markup declarations that define document content. A variety of document content rules can be defined by a DTD involving elements, attributes, entities, notations, processing instructions, comments and parameter entity references. The DTD used in this project only focuses on those first two: elements and attributes.

In order to associate a DTD with an XML file, the DTD must be declared directly within the XML file directly after the document declaration using a line such as bellow:  
 <<!DOCTYPE gridbag SYSTEM "gridbag.dtd">  
The gridbag.dtd is the text file that contains the DTD. One thing to note is that a file does not need to be referenced. The rules for the DTD can simply be included within the “!DOCTYPE” tag before closing the tag. DTDs have their own coding format that must be followed so that the XML file and the XML file processor can read the DTD.

Using a DTD with a file can be essential in saving the heartache associated with hunting through the file to find a tag that should not be there or some other form of error. DTDs can help with writing an XML file to check the file along the development path for any discrepancies. In addition, a program that uses the XML file can first compare it to the DTD before attempting to parse the file. If the file is not validated by the DTD, than the program can throw an error or choose not to open the file, both of which are preferable to crashing due to a poorly-formatted XML file.

# Application at Work

# Distributed Version Control System

## An Picture is worth 1000 words

# Appendix I Non-Direct Activity Reports

# Appendix II Full Unit Testing Code

# Appendix III Source Code