Software Test Plan

Version 1.0

WibTeX

Reference Management System

Sponsor:

Dr. James Palmer

Development Team:

Hayden Aupperle

Jarid Bredemeier

Charles Duso

March 28, 2017

Contents

Background 3
1 Introduction 3
1.1 Purpose 3
1.2 Scope 4
1.2.1 Unit Tests 4
1.2.2 Integration Testing 4
1.2.3 Usability Testing 4
2 Unit Testing 4
2.1 Unit Testing Section: BibTeX Module 5
2.2 Unit Testing Section: Style Module 6
2.3 Unit Testing Section: Docx Module 7
3 Integration Testing 7
3.1 Integration Point: User Interface, BibTeX, Style, and Docx Modules 7
3.1.1 Testing Approach 8
3.2 Integration Point: BibTeX and Docx Modules 8
3.2.1 Testing Approach 8
3.3 Integration Point: BibTeX and Style Modules 8
3.3.1 Testing Approach 9
3.4 Integration Point: BibTeX, Style, and Docx Modules 9
3.4.1 Testing Approach 9
4 Usability Testing 9
4.1 User Access Testing: Graphical User Interface 10
4.2 User Access Testing: Command Line Interface 10
5 Conclusion 10
Bibliography 12

Background

When publishing research in the field of computer science it is common to be required to use BibTeX and LaTeX to construct documents for publication. LaTeX is a document preparation system designed by Leslie Lamport in 1985 that uses a markup language to structure documents [1]. BibTeX is a reference management system, created in 1985 by Oren Patashnik and Leslie Lamport as a method to construct reference data for documents prepared in LaTeX [2].

When publishing research outside of the computer science field, however, it is commonly required to use Microsoft Word to construct documents for publication. Microsoft Word is a word processor developed by Microsoft for use in constructing documents [3]. Researchers who wish to publish across scientific fields will find themselves a time-consuming endeavor when trying to transition their bibliographic work constructed in BibTeX to a format that is suitable in Microsoft Word. Unfortunately, there does not yet exist a solution for the efficient transfer of bibliographic data constructed in BibTeX to Microsoft Word.

1 Introduction

As a result of the issue described in the previous section, the WibTeX development team has designed the WibTeX Reference Management System to simplify the process of preparing documents for cross-discipline research publications by eliminating the need to reconstruct bibliographic information produced in BibTeX to a format suitable for Microsoft Word. WibTeX will allow the user to construct reference pages and in-text citations sourced from BibTeX bibliographic information from within a Microsoft Word Document, using a LaTeX workflow.

1.1 Purpose

This document describes the plan for testing the WibTeX Reference Management System. The testing plan will contain descriptions and examples of the testing strategies used for the system. The testing plan is designed with the intent to satisfy the requirements of the system as determined in the software design document.

1.2 Scope

The testing plan describes the component, integration, and usability tests that will be conducted on the system prototype, following integration of the subsystems identified in the software design document.

1.2.1 Unit Tests

Component-level testing will be achieved through unit tests for components of the system that are capable of being tested in such a manner. As we cannot test all inputs to a component due to the nature of the components in our system, unit tests will be conducted to satisfy equivalence classes that will target the different subsets of input data. All unit tests will be validated automatically by the system.

1.2.2 Integration Testing

Integration point testing will be achieved through test harnesses tailored with certain input data to elicit a desired response at the various integration points of the system. Integration tests will be validated through developer examination.

1.2.3 Usability Testing

Usability testing will be conducted through direct testing of our prototype with our primary consumer. During usability testing, certain performance metrics as well as the user's subjective experience will be recorded for analysis.

2 Unit Testing

This section serves to detail the unit tests established to ensure that functionality of components satisfies the requirements of the system. We have deemed unit tests as a necessary agent for testing because there are sections of code that are almost entirely logical, and thus integration tests or other methods would not be sufficient. The components that will be tested via unit tests are the BibTeX, Docx, and Style modules. All unit tests will be conducted with *py.test* as *Python* is the sole programming language used within the system and the development team already has prior experience with the package [4].

2.1 Unit Testing Section: BibTeX Module

We will now consider the BibTeX module and the unit tests necessary to effectively assess the functionality of the module as determined by the requirements of the system. As stated in the design document, the BibTeX module is required to support extraction and validation of BibTeX databases.

Test 1 - Valid BibTeX Entry

Description:	Test that valid BibTeX entry types are stored and that invalid entry types produce an error. (Assume that valid fields provided)
Equivalence Partition:	*Entry Types ∈{ BibTeX Standard }*
Input Data:	*Entry Types ∈{ BibTeX Standard }*
Erroneous Input Data:	*Entry Types ∉{ BibTeX Standard }*

TABLE 1: BIBTEX UNIT TEST 1

Test 2 - Valid BibTeX Fields

Description:	Test that required BibTeX fields for an entry results in storage of the entry, and failure to include required fields produces an error.
Equivalence Partition:	All fields in the set of required fields for a given BibTeX entry that is part of the BibTex Entry Standard
Input Data:	• *Entry Types ∈{ BibTeX Standard }* • *Fields ∈{ BibTeX Standard }*
Erroneous Input Data:	All entries lacking the required fields designated by the BibTeX Entry Standard for that specific entry type

TABLE 2: BIBTEX UNIT TEST 2

Test 3 – Extract Special Characters

Description:	Test that special characters that have been escaped, using BibTeX syntax can be mapped to their Unicode equivalents
Equivalence Partition:	Escape Characters ⊆ { BibTeX Escape Characters }
Input Data:	BibTeX entry containing string of escaped characters to be converted to Unicode
Erroneous Input Data:	N/A

TABLE 3: BIBTEX UNIT TEST 3

2.2 Unit Testing Section: Style Module

In this section, we will list and describe the unit tests necessary to effectively assess the functionality of the Style module as determined by the requirements of the system. The Style module is required to interpret, validate, and utilize system-inherent and user-designed style files.

Test 1 – Validate Required Fields Provided

Description:	Test that required fields can be validated and that missing fields results in error
Equivalence Partition:	*Fields ∈{ Style File Standard }*
Input Data:	Style file containing *Fields ∈{ Style File Standard }*
Erroneous Input Data:	Style file lacking one or more *Fields ∈{ Style File Standard }*

TABLE 4: STYLE FILE UNIT TEST 1

Test 2 – Validate Fields

Description:	Test that applicable fields included in a style file can have their input validated and thus result in error if invalid
Equivalence Partition:	*Applicable Fields ∈{ Style File Standard }*

Input Data:	Style file containing *Fields ∈{ Style File Standard }*
Erroneous Input Data:	N/A

TABLE 5: STYLE FILE UNIT TEST 2

2.3 Unit Testing Section: Docx Module

Within this section, we will consider the Docx module and the unit tests that have been created to assess the required functionality of the module. The Docx module must be capable of manipulating and parsing Microsoft Word documents to extract necessary citation information.

Test 1 – Extract BibTeX Tags

Description:	Test that the Docx module can extract all BibTeX-supported markup from a Microsoft Word document containing said tags
Equivalence Partition:	BibTeX Markup *∈{ BibTeX Standard }*
Input Data:	Microsoft Word document with BibTeX Markup *∈{ BibTeX Standard }*
Erroneous Input Data:	N/A

TABLE 6: DOCX UNIT TEST 1

3 Integration Testing

In this section, we will list and detail the integration tests for the system. With these tests, we aim to assess the interfaces between major modules of the system, focusing on the interactions and data exchanges. We abstract implementation details from each module so that each module is a single unit that offers one or more functions. Four primary interactions between modules of the system will be tested as integration points.

3.1 Integration Point: User Interface, BibTeX, Style, and Docx Modules

We now consider the interaction between user interface (both graphical and command line), BibTeX, Style, and Docx modules. The primary interaction is the transfer of file paths from user interface to BibTeX, Style and Docx modules with which the modules can extract relevant data from the supplied files. The test harness cannot be automated completely for this scenario, but

we can control the degree to which the program executes: our primary concern is that the appropriate modules receive the appropriate file paths from the user interface.

3.1.1 Testing Approach

The integration tests will be conducted as a top-down approach with which the data gathered by user interface will be funneled down to relevant modules within the system. The team will test both valid and invalid datasets to evaluate the interaction between user interface and subsequent modules.

3.2 Integration Point: BibTeX and Docx Modules

The BibTeX and Docx modules have a specific interaction that must be tested via integration tests. During execution of the system, the Docx module is intended to supply the BibTeX module with a list of unique citations that were collected from the user's submitted input document. Upon receiving the citation data, the BibTeX module is tasked with referencing the user's input BibTeX database and extracting citation data matching that transferred by the Docx module. Should a citation not be listed in the database, the BibTeX module is tasked with notifying the user of the error.

3.2.1 Testing Approach

The integration tests will be conducted in isolation so that the BibTeX and Docx modules are the only active components of the system. The team will monitor the transfer of data from Docx module to BibTeX module. Data transferred will be derived from a discrete set of inputs that will be determined at the time of testing. Data transferred will be evaluated for its expected structure and content.

3.3 Integration Point: BibTeX and Style Modules

In this section, we discuss integration testing of the interaction between the BibTeX and Style modules. During execution of the system, the BibTeX module is tasked with transferring a data structure containing all relevant information for the citations listed within the user's input document. Upon receiving the data structure, it is the task of the Style module to use the reference data and the style file that the user supplied prior to execution, to generate an appropriate template for producing a formatted reference page.

3.3.1 Testing Approach

Integration tests for this interaction will be conducted with the complete system intact, save for the user interface so that interactions can be tested automatically. Testing will be conducted as a top-down approach so that we can monitor the transfer of data throughout the system – as it would during non-simulated execution. System execution will be halted after the interaction between BibTeX and Style modules completes. Output data will be compared against predefined output expectations derived from predefined input data that will be used during testing.

3.4 Integration Point: BibTeX, Style, and Docx Modules

We now consider the final and most crucial interaction point for the system that will be tested via integration testing: the interface between BibTeX, Style, and Docx modules. During execution of the system, the output document will need to be rendered and a specific set of operations must occur in order to successfully produce the appropriate document. First, the BibTeX module must supply the Docx module with the reference data structure. Second, the Style module must supply the Docx module with the reference template string. Third, the Docx module must use both the reference data and template string to render the output document with an appropriately structured reference page and corresponding in-text citations.

3.4.1 Testing Approach

As discussed in the previous integration testing section, testing will be conducted using a top-down approach with the complete system intact, save for the user interface. This interaction almost entirely encompasses the system's flow of execution and thus its success is crucial. All input data and expected output data will be predefined. All output data, except for the final output document, will be compared automatically for efficient testing. The final output document will be evaluated by the development team as the content of the document must have its physical appearance compared.

4 Usability Testing

In this section, we consider the usability tests for the WibTeX Reference Management System. Usability tests are essential to improving the organization of interface information, access to functional elements of the system, and the time to execution of a task using the system. We will

measure the user's degree of success with accessing the provided functionality. Usability tests will be conducted via direct contact with our primary consumer, from whom we will gather their subjective opinion on the usability of the system and the objective data we gather from observing their usage.

4.1 User Access Testing: Graphical User Interface

We will now consider the method of testing the usability of the graphical user interface. Our user will be given the task to execute the WibTeX Reference Management System with a given set of inputs and an expected output. The user will not be given explicit instructions as to how to operate the interface. The user's interaction will be timed. Following the completion of the task, we will document their opinion of their overall interaction with the interface.

4.2 User Access Testing: Command Line Interface

We will now consider the method of testing the usability of the command line interface. Our user will be given the task to execute the WibTeX Reference Management System with a given set of inputs and an expected output. The user will be given explicit instructions that detail how to operate the command line interface - or at least a means to receive discrete instructions via the command line interface. The user's interaction will be timed. Following the completion of the task, we will document their opinion of their overall interaction with the interface as well as the time to completion of task and error rate.

5 Conclusion

The WibTeX Reference Management System simplifies the process of publishing documents across different scientific fields. Researchers - especially those in the field of computer science - often create written publications through LaTeX and BibTeX. If a researcher who works in the LaTeX and BibTeX environment wishes to publish across different scientific fields, they may be required to create documents in Microsoft Word. The transition from LaTeX and BibTeX can be a time-consuming endeavor as the reference material created in BibTeX does not easily transition to Microsoft Word. Fortunately, WibTeX aims to remediate this problem by allowing the user to retain their BibTeX bibliographic information in its original format and use that information as they would in a LaTeX environment; but, for Microsoft Word documents.

This system will be validated through unit, integration, and usability testing methods that aim to assess required functionalities. The WibTeX Reference Management Team is confident that the testing methods listed within this document will be sufficient to providing a complete view of the system's functionality. Should new features be added, or certain tests be ineffective the team will update this document to reflect such events.

Bibliography

[1] LaTeX – A document preparation system. (2017). Retrieved March 5, 2017 from https://www.latex-project.org/

[2] Alexander Feder. 2006. BibTeX. (2006). Retrieved March 5, 2017 from http://www.bibtex.org/

[3] Word. (2017). Retrieved March 6, 2017 from https://products.office.com/en-us/word

[4] Holger Krekel. 2015. Full pytest documentation. (2015). Retrieved March 26, 2017 from http://doc.pytest.org/en/latest/contents.html