Final Report

Version 1.0

WibTeX

Reference Management System

Sponsor:

Dr. James Palmer

Development Team:

Hayden Aupperle

Jarid Bredemeier

Charles Duso

May 8, 2017

Contents

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

The purpose of this document is to provide a complete overview of the WibTeX Reference Management System. This overview will describe the design process, system requirements, system architecture, implementation, testing, development timeline, and future improvements.

1.2 Scope

This report is designed with the intention of it being accessible to any potential user, developer, or contributor such that they can understand the general design of the system and the process with which it was implemented.

2 Process Overview

The WibTeX development team conducted their design and implementation through adherence to the waterfall method [4]. Our process was sequential and mostly non-iterative, following the typical pattern of requirements discovery, system design, system implementation, and then system validation. Teammates did not contribute to this process in any specialized manner; roles were not assigned and workload was distributed based on availability and capability. This process was supported through the use of several tools that will be discussed in the next sub-section.

2.1 Development Toolset

There were several distinct tools and tool types that were used to aid the development process. All programming was completed via various text editors and terminal windows that were dependent on the teammate's preference and their specific operating system. Communication tools included text messaging, email, VoIP and in-person contact. All software version control was conducted via Git.

3 Requirements and Constraints

The purpose of this section is to briefly state the requirements and constraints of the system as determined through the software conception phase. This documentation will serve to provide a clear expectation of what is required of the product.

3.1 Functional Requirements

This section includes the requirements that specify all fundamental actions of the WibTeX Reference Management System.

Specify Style File:	The user should be able to specify the style file they want to use for their document. The user will either specify the style file by way of path in the command line, or through graphical user-interface via drop-down menu.
Extend Style File:	The user should be able to create additional style files from a generic and well-documented template. *This also implies that the user can modify existing style files.*
Complete Style File:	The style file should be complete in that it provides coverage for all resource types supported by a citation style.
Style Support:	The system should support the following reference styles, CCSC, ACM, IEEE, and APA.
Specify BibTeX Database:	The user should be able to specify the BibTeX database, containing a list of references the user plans to cite within their document. The user will either specify the database by way of path in the command line, or through graphical user-interface via drop-down menu.
Complete BibTeX Parser:	The system should be able to support and interpret all resources, fields, and special characters that are valid components of the BibTeX syntax (version 0.99d).
Specify Microsoft Document:	The user shall be able to specify the Microsoft Word document they wish to be read from and the Microsoft Word document they wish to write to. *Microsoft Word documents that are not in the .docx format will not be supported.*
LaTeX Syntax Support:	The system should support valid LaTeX syntax used to generate a reference within a Microsoft Word document from BibTeX databases. *The system must not support all valid LaTeX syntax – only those that interface with BibTeX.*
Error Handling Support:	The system should provide error handling tools that correctly locate the source of the error – whether in the BibTeX file, style file, or Microsoft Word document – and accurately communicate the file and location of error to the user.

TABLE 1: FUNCTIONAL REQUIREMENTS

3.2 Performance Requirements

Below we will list the performance requirements of the WibTeX Reference Management System. We have constructed performance requirements as thresholds of expectation for components of the systems that we have confidence in achieving with high probability. We will consider the response time and scalability for components of the system.

3.2.1 Response Time Requirements

Within this section, we consider the response times of components in the system. Response time requirements envelop all actions by the system that can possibly be conducted by the user. This definition excludes our testing suite and other such functionality that will not be utilized by the primary user. We have constructed a table below that lists the components of the WibTeX system that are of interest for response time.

Component	Metric	Response Time
User Interface	Any event that can be triggered by interface interaction	0.1 to 1.0 seconds until program executes appropriate action related to triggered event
Command Line	Any valid command that can be executed using WibTeX	0.1 to 1.0 seconds until program returns response to user
Document Render	Execution of the render action after sufficient resources have been provided	1.0 to 30.0 seconds until render is complete

TABLE 2: RESPONSE TIME REQUIREMENTS

3.2.2 Scalability Requirements

We will now consider the scalability requirements for the system. We have determined that our system should support the response times listed in the previous section up to a worst-case scenario in which the user has a 100-page Word document and 1,000 references to account for. Although we do not expect any user to reach or exceed the worst-case scenario we have derived, we will still aim to achieve our desired response times should the situation occur.

3.3 General Constraints

In this section, we will consider the general constraints for potential users who wish to use the reference management system. These constraints derive out of a need to satisfy legal requirements as well as ensuring a quality product that does not require deprecated software packages and technologies to support previous versions of Microsoft Word, BibTeX, LaTeX, or various operating systems.

Operating System Limitations:	The system shall need one of the following operating systems installed; Windows 10, macOS, or Ubuntu 16.04 LTS. *Later versions of the specified operating systems will also be supported.*
Hardware Limitations:	The system shall meet the hardware requirements of the respective operating system installed on the user's computer. Additional hardware or upgrades are not required of the user.
Licensed Microsoft Office or Word:	The user shall own a valid license of Microsoft Office or Microsoft Word and have sufficient means to create, modify, and store Word documents.
*Python 3.5.x Installation:	The system must have Python version 3.5.x or greater. * *This constraint is potentially irrelevant as the end-goal of our product is to create an executable for each platform – thus not requiring Python installation.*

TABLE 3: GENERAL CONSTRAINTS

4 Implementation Overview

In this section, we will document our general approach to the implementation of the WibTeX reference management system, as well as, the tools and technologies that will be the driver for the construction of the system. As a reminder, we intend for our system to construct a reference page and in-text citations (that conform to a specified citation style), within a Microsoft Word document, using a BibTeX database. This functionality requires that we derive solutions to parse BibTeX databases, parse and manipulate Microsoft Word documents, and parse and interpret a file containing citation style data.

4.1 Design Approach

We have determined that the ideal approach to the construction of our system would be to design a monolithic architecture that employs modularity achieved through object-oriented programming. This means that the final product would be a single, cohesive unit with a distribution of tasks assigned to each module, increasing performance by removing the need of inter-process communication. As we do not intend to expand the system to account for new features, we are not hindered by the tight coupling that can plague software designed with a monolithic architecture.

4.2 Technologies Used

Listed below is the programming languages, and packages that we plan to use to construct the system. We can say with confidence that the following items will remain throughout the implementation of the system, but there may be additional tools used and so this document will be updated accordingly.

4.2.1 Programming Language - Python 3.4.x

We chose Python version 3.4.x as our programming language of choice for several reasons:

- Python contains several pre-existing packages that support BibTeX database parsing, document templating, and XML parsing (necessary for manipulating Word documents)
- Python is a language that all members of our group are comfortable programming with
- Python allows for object-oriented programming - which is necessary for our design architecture
- Programs in Python can be easily prototyped and tested, allowing for a more agile design process

4.2.2 Package - Jinja2

Jinja2 is a package for Python that offers a robust templating engine [5]. A templating engine allows for the generation of human-readable data that can be dynamically substituted - so long as the data being represented maintains a specific form. Jinja2 enables us to generate the reference pages and in-text citations for a Microsoft Word document. We achieve this by constructing the template for the reference pages and in-text citations within our style file, and using that template within the Microsoft Word document we can dynamically fill-in reference data.

4.2.3 Package - BibtexParser

BibtexParser is a package for Python that allows for simple parsing of BibTeX databases [6]. Using the BibtexParser package, we can quickly parse a BibTeX database and store the reference data in a more suitable format. Once we have the reference data, we can then dynamically fill-in the contents of the structures produced by Jinja2 within the Microsoft Word document we are managing.

4.2.4 Package - TkInter

TkInter is a package for Python that allows for the construction of lightweight graphical user interfaces [7]. With TkInter we will construct the user interface that will act as the gateway to execution of the WibTeX Reference Management System.

4.2.5 Package - PythonXml

XML package is a custom Office Open XML implementation written in Python that allows WibTeX to decompress, parse, extract information, save, and compress Microsoft Word files [8].

4.2.6 Package - PyInstaller

PyInstaller is a Python package that is designed to generate executables from Python programs, containing third-party packages [9]. This enables us to freely use external Python packages such as Jinja2 and BibtexParser without having to accommodate for said packages to produce a functional executable.

5 Architectural Overview

Within this section, we will discuss the high-level architecture of the WibTeX Reference Management System. This section will include an architectural diagram to summarize the overall function of the system, and a textual description of the architectural diagram to clearly define the purpose and functionality of each component that constitutes the system.

5.1 Architectural Diagram

Listed below is the architectural diagram for the WibTeX Reference Management System.

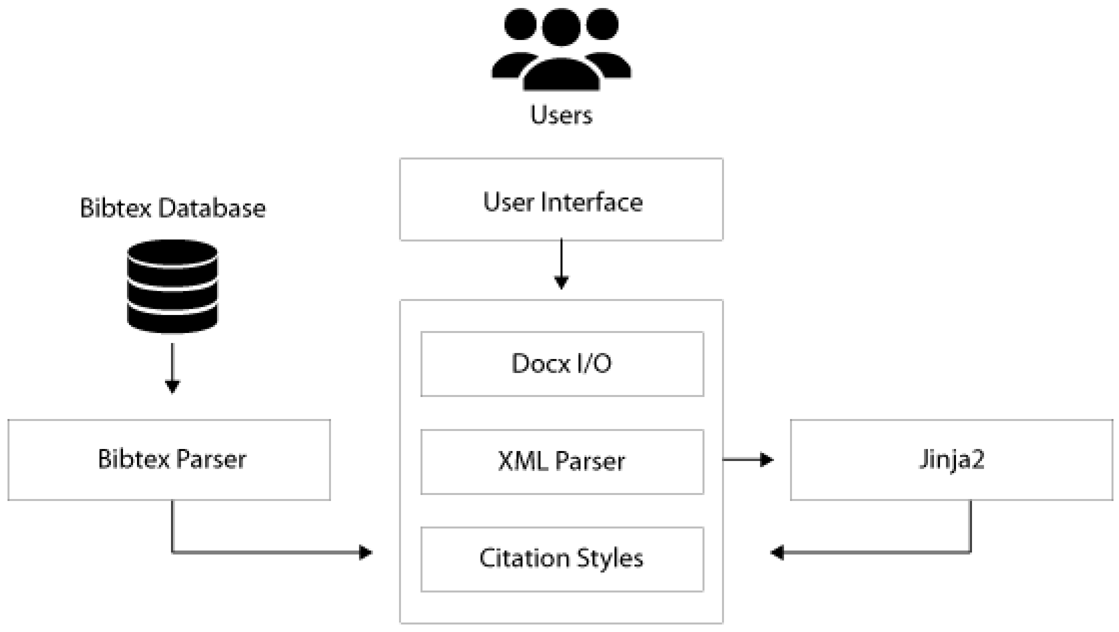


FIGURE 1: SYSTEM ARCHITECTURE

5.2 Component-level Description

In this section, we will describe the components of the system architecture.

5.2.1 User interface

The User Interface module acts as the junction between the user and the WibTeX Reference Management System. The module allows the user to target a BibTeX database, style format, docx document for processing, and docx document for output. The module is responsible for delivering input data to each component that requires it. Listed below is an example of the graphical user interface in the final iteration of the system.

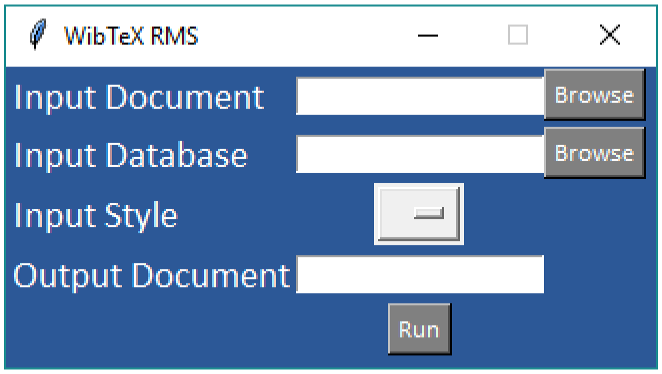


FIGURE 2: WIBTEX RMS GUI

5.2.2 BibTeX Parser

The BibTeX Parser module is responsible for processing BibTeX databases into memory. The module also converts LaTeX symbols to Unicode and logs invalid BibTeX entries.

5.2.3 Docx I/O

The Docx I/O module is responsible for decompressing and opening a docx document for modification. The module extracts XML content while parsing LaTeX markup into a data structure. The module writes valid XML content to docx documents.

5.2.4 XML Parser

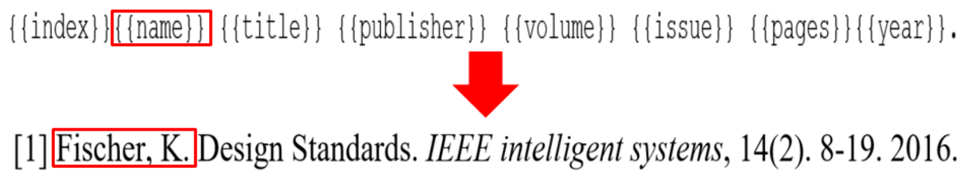
The XML Parser module is responsible for applying styles to BibTeX data and updates the information inside the data structure. The module is capable of converting HTML styling markup to Word-suitable XML markup. XML Parser is dependent on Jinja2 to substitute reference data into a Microsoft Word document.

5.2.5 Citation Styles

The Citation Styles module is essential to validating style templates and constructing reference pages based on a user’s selected style format. Citation Styles utilizes Jinja2 to achieve its prescribed functionality.

5.2.6 Jinja2

Jinja2 is a templating engine that is used for two discrete purposes: replacing LaTeX markup inside a docx XML with the markup processed by the XML parser and constructing reference material from a style template and BibTeX database. This functionality requires information be passed from the user interface to both the Citation Styles module and XML Parser to properly execute. Listed below is a simplified example of how the Jinja2 engine interacts with the Citation Styles module to produce reference material.



```
{{index}}{{name}} {{title}} {{publisher}} {{volume}} {{issue}} {{pages}}{{year}}.
```

[1] Fischer, K. Design Standards. *IEEE intelligent systems*, 14(2). 8-19. 2016.

FIGURE 3: JINJA2 USAGE EXAMPLE

5.3 Reflection

As with all software development, the inception of a software design may differ from its reality and our system is not an exception. In terms of components, the previously described structure is almost entirely consistent with the final implementation. The difference is apparent at the functionality level; several features we had intended to develop for each component are either conditionally functional or unimplemented due to time constraints. These features include robust Unicode conversion, BibTeX database entry inheritance, and consistent GUI appearance across platforms. In the advent that we had more time we would have dedicated our remaining resources to these features.

6 Testing

In this section, we consider the testing plan inception, describing the component, integration, and usability tests that were constructed to be conducted on the system prototype, following integration of the subsystems identified in the software design document. We will consider several facets of testing and how these facets were utilized to assess our system.

6.1 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].

6.1.1 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 4: 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 5: 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 6: BIBTEX UNIT TEST 3

6.1.2 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 7: 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 8: STYLE FILE UNIT TEST 2

6.1.3 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 $\in$ *{ BibTeX Standard }*
Input Data:	Microsoft Word document with BibTeX Markup $\in$ *{ BibTeX Standard }*
Erroneous Input Data:	N/A

TABLE 9: DOCX UNIT TEST 1

6.2 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.

6.2.1 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.

6.2.1.1 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.

6.2.2 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.

6.2.2.1 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.

6.2.3 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.

6.2.3.1 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.

6.2.4 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.

6.2.4.1 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.

6.3 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.

6.3.1 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.

6.3.2 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.

6.4 Testing Results

Unit tests and integration tests were built and assessed through the project's development. Unit tests were primarily successful with most aspects of the system, however, LaTeX to Unicode conversion proved particularly difficult and unreliable. Integration tests were painstaking, but mostly successful in that we have a functional system that utilizes each component. Usability tests were not conducted due to time constraints and so we had to validate its use amongst the developers – which is not indicative of a successful system, but it was the most appropriate option we had. Had we more time, we would have derived more unit tests and attempted to ensure success on all areas of testing.

7 Project Timeline

In this section, we will consider the timeline for the project from its inception to our current progress at the time of writing this document. We will briefly discuss important dates, phases, and milestones. Pictured below is a Gantt chart depicting our schedule. *At the time of writing this, the items listed on the schedule are not all yet completed; we are currently working on completion of the user manual and the product delivery tasks.*

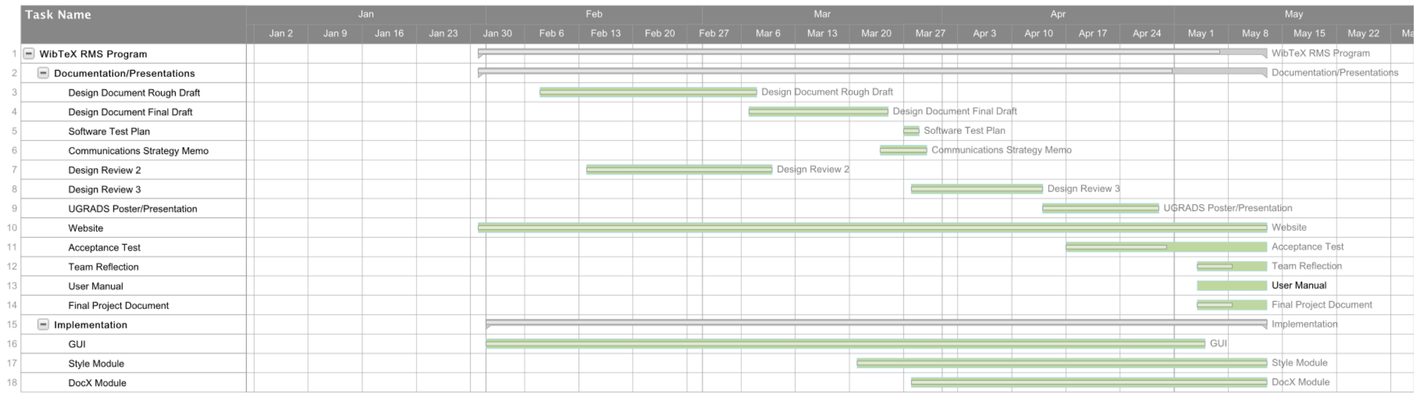


FIGURE 4: GANTT CHART SCHEDULE

7.1 Important Documents

There are two documents that could be considered essential to the system's success: the design document, and testing plan. These documents comprise the entirety of the system, including requirements, architecture, implementation, and testing methods. These documents served as guidelines throughout the construction of the WibTeX Reference Management System. Fortunately, we completed the documents within the specified time frame and received positive reviews from our mentor.

7.2 Schedule Milestones

There are several important events that could be considered milestones for the system and its development. These events include the design review presentations, capstone presentation, acceptance test demo and project delivery. We consider these events milestones because they are indicative of significant progress towards our goal of the system's deployment. With each event being iterative, and presented in front of several peers each time we had to ensure that progress was made from one event to the other.

8 Future Work

In this final section, we would like to list several improvements and features we believe would be beneficial to the WibTeX Reference Management System. These features would extend the functionality of the system seemingly beyond its current capabilities while retaining the identity of the original system.

- System support for mobile devices to allow reference construction when a desktop environment is not available
- Tool to help users easily generate style templates for WibTeX Reference Management System
- Functionality with web documents to allow reference construction for blogs, articles, and other online material

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.

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] SDLC Waterfall Model. (2017). Retrieved May 6, 2017 from https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

[5] Armin Ronacher. 2008. Welcome to Jinja2. (2008). Retrieved March 6, 2017 from http://jinja.pocoo.org/docs/2.9/

[6] François Boulogne. 2014. Welcome to BibtexParser's documentation!. (2014). Retrieved March 6, 2017 from http://bibtexparser.readthedocs.io/en/v0.6.2/

[7] Tkinter Wiki. (March 2014). Retrieved March 6, 2017 from http://tkinter.unpythonic.net/wiki/

[8] PythonXml. (January 2012). Retrieved March 7, 2017 from https://wiki.python.org/moin/PythonXml

[9] Welcome to PyInstaller official website. (February 2013). Retrieved March 7, 2017 from http://www.pyinstaller.org/