Test Plan

for

CALEX - Temporal Annotation System

Version – 1.0

Prepared by
Ziyan Di, Yantian Ding, Yi Pan, Carl Shen, Hewitt Zhang, Harry Zhao

Stakeholder: Hegler Tissot

Drexel University

April 30th, 2023

Table of Contents

1. Introduction

The primary objective of this written document is to establish a comprehensive system test plan for the CALEXText Python package. The primary purpose of conducting this testing is to guarantee that the package performs as expected and satisfies all of the requirements in the package's accompanying documentation. It is worth noting that all of the information contained within this document was formulated before the initiation of any actual testing and has been compiled here in order to serve as both a primary tool and a reference guide for future testing endeavors.

To achieve the goal of verifying the proper functionality of the CALEXText Python package, the system test plan outlined in this document will cover all aspects of the package's functionality. In addition, such a plan will include testing for various use cases and scenarios and testing for potential errors and exceptions that may arise during package usage.

Any deviations or changes from the testing plan outlined in this document must be thoroughly documented and justified. In addition, any resultant modifications to the testing plan must be communicated to all parties involved in the testing process.

1.1 Definitions and Acronyms

- Time Expression (TIMEX) – TIMEX refers to a temporal annotation schema, which is primarily used to markup explicit temporal expressions, such as times, dates, durations, etc.
- Calendar Expression (CALEX) – CALEX refers to a special temporal annotation schema restricted to time expressions and concepts that can be connected to an absolute timeline.
- Document Creation Time (DCT) – DCT refers to the creation time of a document, such as the creation date.
- Token – Tokens can be individual words, phrases, or even whole sentences.
- Tokenization – Tokenization is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols, and other elements called tokens.

1.2 Scope

In this section, we will detail the specific features of our Python package that will undergo comprehensive testing and those that will not be tested. This information is crucial as it focuses our testing efforts on the package's most critical and relevant features while ensuring that we allocate our resources effectively and efficiently.

- **In Scope**

The main objective of this test is to conduct comprehensive testing of the functions incorporated within the Python package. The test methods we used were unit tests and

acceptance tests. Through this process, we aim to evaluate whether each function and method can effectively process and handle the specified input parameters and generate the expected output as per the predetermined specifications.

Each function will be rigorously assessed for its accuracy, reliability, and efficiency during the testing process. Specifically, we will analyze whether the function can perform the intended operation precisely and error-freely while also checking its speed and resource consumption.

To carry out this testing, we will utilize the code files named "Leveltools.py" and "loadFile.py," which contain all the relevant functions for testing. Upon running the test, if the generated output matches the expected outcome, the program will display "OK"; otherwise, if there are any inconsistencies or failures, the program will display "Failed."

- **Out of Scope**

This test does not include simultaneous testing of multiple functions. Our project aims to develop a Python package, and the potential users are programmers with basic CALEX Annotations knowledge. When using the Python package, functions will be imported individually, and none will be used simultaneously. All tests are based on independent testing of individual functions.

2. Testing Approach

In this section, we will provide a detailed overview of our testing methodology, outlining our approach to testing each component of the system and the procedures and tools we will utilize to perform the tests.

2.1 Unit tests approach

Unit test is a software testing method used to test the minor test units in software - typically a single function or form. Its purpose is to determine whether the behavior of a team meets expectations and to discover and correct errors early in the code. Developers typically write unit tests designed to be quick and repeatable automated tests. This allows these tests to be run quickly when code changes occur to ensure that the code still behaves as expected. Using unit tests enhances code maintainability, reliability, and reusability, providing confidence and support for code refactoring and optimization.

In our CALEX project, unit testing will focus primarily on the various functions and methods in the Python packages developed by our team concerning CALEX annotations. Our testing process will thoroughly evaluate each function or method to verify that its output, when corresponding to a given input, conforms to the expected results based on the specified rules and guidelines.

Our output evaluation will consider content and format to ensure accuracy and consistency. This will allow us to thoroughly validate the functionality of each function and method while ensuring that the output is presented in a manner consistent with the overall structure and format of the system.

By conducting unit tests in this manner, we can gain a more granular and detailed understanding of each feature and method's functionality and identify any potential issues or errors that may arise during the testing process. This allows us to resolve these issues quickly and efficiently, ultimately providing our end users with a high-quality, reliable and user-friendly system.

2.2 Acceptance tests approach

The acceptance test is a software testing method to test whether the entire software system meets business and user requirements. Its purpose is to verify that the software system meets user expectations and ensure that the software system can run normally in the actual usage environment. These tests typically involve testing the entire software system's functionality, performance, and security aspects.

The use of acceptance testing provides a means to ensure that software systems meet user and business requirements while also allowing potential problems to be identified and resolved before deployment. Through acceptance testing, we can significantly improve the quality and reliability of our software systems, ultimately providing a more valuable and user-friendly experience for our end users.

For our CALEX annotation project, our acceptance testing will focus primarily on evaluating the ability of our developed Python package's ability to handle various input types and extract complete and accurate temporal information using CALEX functionality. As opposed to testing the functionality of multiple functions individually in a UNIT TEST, this is a critical step in verifying that the entire package works as intended, that all the required features and functionality are appropriately integrated, and that they work seamlessly.

By using acceptance tests, we are able to effectively specify boundaries to prevent any potential problems and ensure that the entire Python package stays within reasonable bounds to maximize the user experience. This allows us to deliver a high-quality, reliable and efficient system that meets the needs of our end users while also providing added value to our business.

2.3 Tools to be Used

Test environments are sufficient to run a Python3 environment.

- Modern Operating System:
  Windows 7 or 10
  Mac OS X 10.11 or higher, 64-bit
  Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)
- x86 64-bit CPU (Intel / AMD architecture)
- 4 GB RAM
- 5 GB free disk space

2.4 Testing Schedule and Responsibilities

#	Task	Dependency	Team Member(s)	Timeline
	Test all the functions in Leveltools.py		Ziyan Di, Yi Pan, Yuhao Zhang	3/14/2023-4/30/2023

3. Test Cases

This section contains defined test cases that ascribe to the testing approaches defined above. All test cases were defined before any proper implementation was performed. Each test case is given a number, descriptive ID, and title, in addition to a description of what is being tested along with the steps to perform the test, and pre and post conditions for those steps. Some may optionally include dependencies on other tests. Additionally, the creator of the test is named along with the date they created the test. Finally, space is reserved for whomever performs the test to fill in retroactively the status as to whether the test passes.

3.1 Unit tests cases

- **Leveltools.py**

Number: 1	ID: Test_DatePoint	Priority: High
Title	Test DatePoint Function	
Description	Test if the function can run correctly	
Pre-condition	Function can run properly	
Actions(s)/Steps	1. Assign initial values for each property in DatePoint 2. Compare output with expected output	
Post-conditions	Return “OK” means function can work correctly	

Test Plan for CALEX - Temporal Annotation System

Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 2	ID: Test_DateRange		Priority: High
Title	Test DateRange Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign initial values for each property in DateRange 2. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 3	ID: Test_is_preposition		Priority: High
Title	Test is_preposition Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign different possible input for is_proposition function 2. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 4	ID: Test_is_noun		Priority: High
Title	Test is_noun Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign different possible input for is_noun function 2. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 5	ID: Test_postDP_weight	Priority: High
Title	Test postDP_weight Function	
Description	Test if the function can run correctly	
Pre-condition	Test if the function can run properly	
Actions(s)/Steps	1. Assign different possible input for postDP_weight function	

	2. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 6	ID: Test_preDP_weight	Priority: High	
Title	Test preDP_weight Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign initial values for each property in DatePoint as input 2. Run preDP_weight function with input 3. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 7	ID: Test_postDP	Priority: High	
Title	Test postDP Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign initial values for each property in DatePoint as input 2. Run postDP function with input 3. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 8	ID: Test_is_formerDR	Priority: High	
Title	Test is_formerDR Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign initial values for each property in DateRange as input 2. Run is_formerDR function with input 3. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 9	ID: Test_is_latterDR		Priority: High
Title	Test is_latterDR Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign initial values for each property in DateRange as input 2. Run is_latterDR function with input 3. Compare output with expected output		
Post-conditions	Return "OK" means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 10	ID: Test_is_day		Priority: High
Title	Test is_day Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign initial values for each property in DatePoint as input 2. Run is_day function with input 3. Compare output with expected output		
Post-conditions	Return "OK" means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 11	ID: Test_find_last		Priority: High
Title	Test find_last Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign expected list input for find_last function 2. Compare output with expected output		
Post-conditions	Return "OK" means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 12	ID: Test_find_next	Priority: High
Title	Test find_next Function	
Description	Test if the function can run correctly	
Pre-condition	Test if the function can run properly	

Actions(s)/Steps	1. Assign expected list input for find_next function 2. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 13	ID: Test_multi		Priority: High
Title	Test multi Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign initial values for each property in DatePoint as input 2. Run multi function with input 3. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 14	ID: Test_add		Priority: High
Title	Test add Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign initial values for each property in DatePoint as input 2. Run add function with input 3. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 15	ID: Test_find_year		Priority: High
Title	Test find_year Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign expected list and integer input for find_year function 2. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 16	ID: Test_find_decade	Priority: High

Title	Test find_decade Function		
Description	Test if the function can run correctly		
Pre-condition	Test if the function can run properly		
Actions(s)/Steps	1. Assign expected list and integer input for find_decade function 2. Compare output with expected output		
Post-conditions	Return “OK” means function can run correctly		
Created by:	Ziyan Di	Created on:	4/25/2023
Tested by:	Ziyan Di	Tested on:	3/15/2023
Test Status	Successful		

Number: 17	ID: Test_find_centry	Priority: High
Title	Test find_centry Function	
Description	Test if the function can run correctly	
Pre-condition	Test if the function can run properly	
Actions(s)/Steps	1. Assign expected list and integer input for find_centry function 2. Compare output with expected output	
Post-conditions	Return “OK” means function can run correctly	
Created by:	Ziyan Di	Created on: 4/25/2023
Tested by:	Ziyan Di	Tested on: 3/15/2023
Test Status	Successful	

3.2 Acceptance tests cases

4. Risks and Assumptions

4.1 Time

The occurrence of unforeseen events such as natural disasters, extreme weather, and plagues like Covid-19 can significantly impact the progress of the overall project process. It may

disrupt or even terminate certain portions of the project. While we cannot predict or prevent these events, we can be prepared to respond to them promptly and effectively.

If such an event occurs, all project team members will work together to discuss potential worst-case scenarios and develop appropriate responses. In addition, we will strive to update the project schedule and Gantt chart in real time, taking into account specific circumstances and adjusting the project plan as needed to ensure that progress continues despite any obstacles.

If time constraints prevent us from completing all project requirements as initially planned, we will prioritize the most critical needs and consider removing any relatively unnecessary ones. This approach will allow us to deliver essential requirements within the specified timeframe, ensuring the successful completion of the project despite any unforeseen challenges.

4.2 Scope Creep

The project scope contains all the work needed to complete a project. Before a project begins, we work closely with stakeholders to develop an initial project plan that identifies all of the work that will be required. However, it is essential to note that the project scope may change over time, especially when the project client or other stakeholders request changes.

Scope creep is a common problem that occurs when project requirements are added after project execution has already begun. In many cases, these changes must be adequately reviewed or evaluated, resulting in the project team being expected to complete more tasks, deliverables, and milestones with the same resources and timeframe as the original project scope.

In our CALEX annotation projects, scope changes may occur as we refine our understanding of project requirements and identify opportunities to enhance our packages. For example, we may need to adjust our rules or modify output functionality to meet our end users' needs better. As with any project, it is essential to effectively manage scope changes to ensure the project is successful, on time, and within budget.

4.3 Lack of clarity

Lack of clarity may result in miscommunication from stakeholders, vague project scopes, or unclear deadlines. The result can be a lack of visibility due to siloed work, going over budget, falling behind project deadlines, changing project requirements, having to pivot project direction or disappointing project outcomes. To better ensure clarity, we should create a proper plan for the project and check the requirements to ensure that everything and every team member is in place. It's also essential to make sure everyone has access to project information. By keeping the information in one central tool, you can ensure that everyone stays up to date as the project progresses.