Pilot Operating Handbook Semantic Text Analysis

Project Testing and Acceptance Plan

**The Boeing Company**

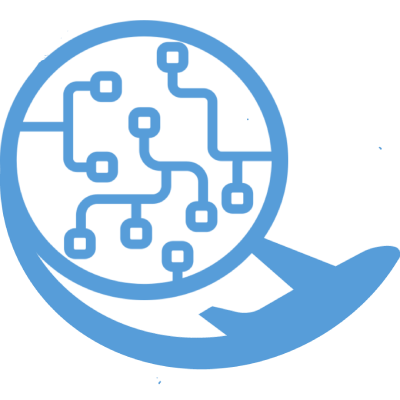


**The Autopilots**

Katie Peterson

Allison Lorphanpaibul

Weiren Lai

Daniel Harker

November 10, 2020

**Note**: Recall that this writing assignment says:

Length = 3 pages text (or more because space goes fast once you start using lists and forms for user feedback) + appendixes as needed.

Not included in 3 page count:

* Cover page
* table of contents
* pictures
* tables
* images
* diagrams

Posted as a single self‐contained file (no links to outside resources.)

Posted as a PDF file.

Typed single‐spaced.

Typed with black text.

Typed with #11 font size.

Typed using Arial font.

Typed with one inch margins on sides, top and bottom.

**Please erase this page in your final document.**

**TABLE OF CONTENTS**

[**I.**](#_heading=h.1fob9te) **Introduction 4**

[I.1.](#_heading=h.3znysh7) Project Overview 4

[I.2.](#_heading=h.2et92p0) Test Objectives and Schedule 4

[I.3.](#_heading=h.tyjcwt) Scope 4

[**II.**](#_heading=h.3dy6vkm) **Testing Strategy 4**

[**III.**](#_heading=h.1t3h5sf) **Test Plans 4**

[III.1.](#_heading=h.4d34og8) Unit Testing 4

[III.2.](#_heading=h.2s8eyo1) Integration Testing 4

[III.3.](#_heading=h.17dp8vu) System Testing 4

[III.3.1.](#_heading=h.3rdcrjn) Functional testing: 4

[III.3.2.](#_heading=h.lnxbz9) Performance testing: 4

[III.3.3.](#_heading=h.35nkun2) User Acceptance Testing 5

[**IV.**](#_heading=h.1ksv4uv) **Environment Requirements 5**

[**V.**](#_heading=h.2jxsxqh) **Glossary 5**

[**VI.**](#_heading=h.z337ya) **References 5**

# Introduction

## Project Overview

This project’s goal is to develop a system with a semantic text parsing algorithm that takes vocabulary from 3-10 Boeing aircraft handbooks as input. In return, the system stores each noun/noun phrase with its designated document name, year, product, location, and 1-50 sentences to demonstrate context for each noun in a MySQL database [1].

## Test Objectives and Schedule

Products to be delivered for this project include the Python code and all system documentation.

The approach for testing has been divided into three major components: unit testing, integration testing, and system testing. System testing is further divided into functional testing, performance testing, and user acceptance testing. Most testing will be performed concurrently with the creation of the Python code, with the exception of user acceptance testing which will be done closer to the completion of the alpha prototype.

Among the required resources for testing are access to a laptop or desktop computer, a commonly used CPU & graphics card, and an operating system capable of running Python. For system specific testing, we will need access to GitLab CI/CD.

Major milestones and a detailed schedule of the project can be found in the team’s Gantt chart at the following link:

<https://docs.google.com/spreadsheets/d/1lJD5gY3dHMCfuv3KDJAOWPmw1VpoPJ7pAH-iBIoO5Ts/edit?usp=sharing>

## Scope

This document will primarily discuss the testing strategy and testing plans that will take place while completing our project. The purpose of testing is to ensure that a fully functioning system is created, and that each requirement given by our customers is met in totality.

# Testing Strategy

We will use GitLab’s CI/CD for our system testing. For other testing, such as performance testing, we may require access to Boeing’s infrastructure to get information such as accuracy metrics.

As our project grows, the pipeline may need to be extended with new required tests. Because we will be adding new features frequently during the initial phases, having a clear workflow for testing will be helpful.

Our testing workflow will be as follows:

1. Identify any new test cases. Tests will be mainly derived from the Software Requirements Specification.
2. Identify which particular tests will be used to test each module.
3. Write necessary tests. Document all test case information.
4. Add new changes. Perform the tests.
5. Submit new changes to unit testing. Document any test data from the testing process.
6. If unit tests pass, the changes will be eligible for integration and system testing.
7. Once all system testing is complete, the change is eligible for deployment.
8. All testing documents and changes will be reviewed by a peer.
9. Changes can be revised and resubmitted for testing.
10. Once passing review, changes can be deployed.
11. In the case of unsuccessful tests, a bug form will be generated describing the the test case, the problem encountered, its possible cause, and the sequence of events that led to the problem. If necessary, this form can be submitted for later analysis.

# Test Plans

Our team will use several testing methods for our system, and we will follow the test pipeline outlined above. Most system and unit testing will be written before or in tandem as their corresponding features. Performance, chaos, and user acceptance testing will most likely be done after the development of our working prototype.

Members will be in charge of developing test cases for the features that they are in charge of.

## Unit Testing - Each member writes their own unit tests for subsystem

The primary goal of unit testing is to take the smallest unit of testable software in the application, isolate it from the remainder of the code, and test it for bugs and unexpected behavior.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | System | Test Data | Expected Result |
| Test sentence tokenization base case | Sentence Parser | A paragraph | Sentences should be parsed properly |
| Test sentence tokenization no stop punctuation | Sentence Parser | A paragraph with no punctuation | Paragraph should be unchanged |
| Test incorrect command | User Interface | Misspelled/incorrect command | Error message. Help menu displayed. |
| Test table implementation | Database | System run with blank PDF | Empty tables are created |
| Test table accuracy (1 PDF) | Database | System run with PDF containing a single paragraph | Tables are populated properly and with correct data |
| Test table accuracy (multiple PDF’s) | Database | System run with 2 PDF’s each containing a single paragraph | Tables are populated properly and with correct data |
| Test text parsing algorithm | Sentence parser | A PDF written in simplified English that has at least 100 pages | Parsed words with information tagged to each word such as year, location and etc.. |
| Test system response time | File import, sentence parser, user interface | A PDF written in simplified English that has at least 100 pages | output each function’s response time after running the entire system through |

## Integration Testing

Integration testing detects faults that have not been detected during unit testing by focusing on small groups of components. Two or more components are integrated and tested, and when no new faults are revealed, additional components are added to the group.

**III.2.1 Test Incorrect Command with Sentence Tokenization**

After knowing that the user interface works for obtaining specific documents to open for reading, we need to be sure it can actually be used for tokenizing the document by sentences. This will be tested using a small sample size, in which the sentences will simply be output to the terminal for confirmation. Once it’s determined that the two components work together, we will be able to use actual Boeing documents.

**III.2.2 Test Sentence Tokenization with Noun Parser**

Now that it’s confirmed that we can obtain sentences from an input file, we need to confirm that we can parse nouns from the sentences. This will likely focus mainly on the variable type of the sentences and how that type is compatible with the noun parser program we decide to use. Similar to testing incorrect commands with sentence tokenization, the nouns from each sentence will be displayed to the terminal for confirmation. This integration test will start firstly with a single sentence, then progress to a paragraph, to ensure that multiple sentences perform identical to a singular sentence as input.

**III.2.3 Test Sentence Tokenization and Noun Parser with Data Storage**

This integration test has two connected parts to it: storing sentences and storing nouns. Since sentences and nouns are linked together across their respective data tables, they’ll need to be stored together. This will be done by storing a sentence in its respective data table and linking each noun detected by the noun parser as described in WA3. Testing will start with a paragraph in order to easily check for accuracy, then increase in size to multiple paragraphs. After it’s determined that these parts work together flawlessly, we will be able to run the program on actual Boeing documents.

## System Testing

System testing is a type of black box testing that tests all the components together, seen as a single system to identify faults with respect to the scenarios from the overall requirements specifications. Entire system is tested as per the requirements.

During system testing, several activities are performed:

## Functional testing:

Test of functional requirements (from requirements specification). The goal is to select those tests that are relevant to the user and have a high probability of uncovering a failure.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Functional Requirement | Steps | Test Data | Expected Result |
| Test invalid system input | The system must read as input 3-10 Boeing pilot operating handbooks in  PDF format. | 1. Input a file in the incorrect file format  2. Wait for results  3. Check the database | A file with an invalid extension (doc, csv, etc) | The system should report an error-- it should not crash. No data should be produced. |
| Test non-simplified English system input | The system must read as input 3-10 Boeing pilot operating handbooks in  PDF format. | 1. Input a file in the incorrect file format  2. Wait for results  3. Check the database | Any pdf with text not written in simplified English. | The system should output to the database. |
| Test output if there is an existing database previously created by system | The system must store data collected in a MySQL database. | 1. Run system with file(s) so that a database is created  2. Run system again with same file(s)  3. Check the database | Any PDF(s) written in simplified English | The system should create a new database and not overwrite the existing database |
| Test text parsing algorithm | The system is able to parse all nouns from text written in Boeing simplified English | 1. Run system with an imported PDF file.  2. Parse the text  3. Check the result in the database. | At least a PDF written in simplified English from Boeing. | The algorithm parsed the text in the PDF and outputted the results into the database. |
| Test nouns with specific requirements. | The system must parse and identify 1 to 50 sentences from the various documents that contain the noun, the name of the document(s) the noun appears in, the product associated with the document(s), the publication year of the document(s), and the location of the document(s). | 1. Run the system with an imported file.  2. Parse the files.  3. Identify a list of information about each noun. | PDFs written in simplified English | A dataset that contains a list of nouns and a list of information that is related to each noun such as location, year etc.. |

## Performance testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Non-Functional Requirement | Steps | Test Data | Expected Result |
| Test the readability of the database the system creates. | With the input to the program being Boeing-provided documents and manuals that are hundreds of pages long, the output must be clear and concise. It will take the form of a database that can be read by humans and parsed by computers. | 1. Run system with specified documents  2. Wait for results  3. Check to ensure the database was created, is accessible, and contains data from each of the 10 documents. | 10 PDF documents written in Boeing simplified English, each with at least 100 pages. | The system creates a database with the necessary amount of memory to store all data from the 10 documents.. The database is able to be opened on the computer and be viewed by the user. |
| Test the accuracy of noun parsing | When parsing a sentence by each word, the system should be able to determine if each word is a noun or not. | 1. Run the system  2. Print nouns and their counts  3. Check the output compared to the known, desired results | Start small with a paragraph. Progress into using multiple paragraphs | The system should generate 100% accuracy in finding nouns. |
| Test the response time of the program. | After inputting Boeing documents, the system is able to run and output a response time to each process accordingly. | 1.Run the system.  2. Check the duration time of each process. | 10 PDF documents with Boeing simplified-English. | The system is able to run and output the time taken for each process. |

## User Acceptance Testing:

The team plans to meet with our customers, Don and Rocky, on December 3rd to perform user acceptance testing. We will be demonstrating the alpha prototype we have created, and will receive feedback from Don and Rocky. They will communicate with the team on whether the system is satisfactory and meets the initial requirements given.

* + 1. **Chaos Testing:**

Once the prototype is in a working state, we can perform chaos testing on it. The goal of chaos testing is to cause system failures. To start, we can perform naive chaos testing by simply inputting bad or volatile data into the system. If the system becomes robust enough, we can implement a chaos monkey to automatically and randomly perform tests on the system that will cause it to fail.

# Environment Requirements - Will

## Physical Characteristics

We need a computer to run and test the program, so a laptop or a PC is necessary.

## Hardware

We will likely need an Intel series CPU & Graphics card / AMD series CPU & Graphics card. There is not a specific hardware requirement in terms of testing. As long as they are commonly used CPUs and Graphics cards, just in case if problems happen, we are familiar with what we have so we could approach the problems easier.

## System Software

Our computers will require MacOS, Windows, Linux, or any operating system that python runs on.

## Testing Tools

For our testing tools, we came up with a few ideas. We have decided to use pytest and unittest or either one. We are also aware of how different they are in comparison.

Unittest provides test cases, test suites, test devices, and related classes of test running programs, making the test clearer, more convenient, and controllable.

To write use cases using unittest, the following rules must be followed:

1. The test file must be imported into unittest first.

2. The test class must inherit unittest.TestCase.

3. The test method must start with “test\_”.

4. The test class must have the unittest.main() method.

However, pytest is a third-party testing framework for python and an extended framework based on unittest, which is more concise and more efficient than unittest.

To write use cases using pytest, the following rules must be followed:

1. The test file name must begin with “test\_” (eg: test\_ab.py).

2. The test method must start with “test\_”. (It is obvious here that pytest is much simpler than unittest for writing test cases. pytest can execute unittest-style test cases without modifying any code of unittest case, which has better compatibility).

Also, there are lots of pytest plug-ins, such as flask plug-ins, which can be used for error reruns of use cases; and xdist plug-ins, which can be used for parallel execution of devices.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Complexity | Compatibility | Plug-ins |
| Unittest | Complex | None | Barely |
| Pytest | Simple | Supports unittest | Many |

# Glossary

**Generative Adversarial Networks (GANs) -**  A popular machine learning model used to create synthetic data on-demand.

**Natural Language Processing (NLP)** - The field pertaining to how to program machines to

understand human language.

**Simplified English -** The standard for written aerospace documents. Utilizes a restricted vocabulary and simple grammar rules.

# References

[1] Rakshit Bhatt & Don Brancato, “Pilot Operating Handbook Semantic Text Analysis”, The Boeing Company, Washington, USA, 2020.