Finding Hay in the Needle Stack: Doing the Right Thing at the Right Time

Design Document

Team Number: sddec19-09
Client: Collins Aerospace
Adviser: Srikanta Tirthapura
Team Members/Roles
Austin Konsor: Software Architect/Chief Engineer
Mireille Mwiza Iradukunda: Test Engineer/Report Manager
Gooi Yin Hong: Lead Designer/Web Master
Yealim Sung: Project Leader/Meeting Scribe
Team Email: sddec19-09@iastate.edu
Team Website: sddec19-09.sd.ece.iastate.edu

Revised: 3/26/2019/Version1

Table of Contents

List of figures/tables/symbols/definitions 1

1 Introduction 2

1.1 Acknowledgement 2

1.2 Problem and Project Statement 2

1.3 Operational Environment 3

1.4 Intended Users and uses 3

1.5 Assumptions and Limitations 3

1.6 Expected End Product and Deliverables 3

2. Specifications and Analysis 4

2.1 Proposed Design 4

2.2 Design Analysis 6

3. Testing and Implementation 7

3.1 Interface Specifications 7

3.2 Hardware and software 7

3.3 Functional Testing 7

3.4 Non-Functional Testing 8

3.5 Process 8

3.6 Result 9

4 Closing Material 9

4.1 Conclusion 9

4.2 References 9

4.3 Appendices 10

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

Figure 1. Use-case diagram of our project page 4

Figure 2. Design Flow Diagram for the tool page 6

Figure 3. Example input python file 1 page 6

Figure 4. Example input python file 2 page 7

Figure 5. Workflow Diagram page 12

Table 1. Example output of the program page 7

1 Introduction

1.1 Acknowledgement

We are thankful and excited to be assigned this project with Collins Aerospace. Specifically we would like to thank Andy Zobro, Branden Lange, and Kirsten Dawes for their continued help, resources and guidance. We hope to fulfill their needs and deliver a final product to help with their lasting success.

We would also like to thank Srikanta Tirthapura. His support and guidance will be very valuable in the coming months as we prepare to deliver an outstanding product to Collins Aerospace. Finally, we would like to thank all teaching assistants in this process, as much of it is volunteered time, it is greatly appreciated.

1.2 Problem and Project Statement

Problem Statement:

Collins Aerospace has about 8000 common test files that are spread across many directories within a SVN repository. These are python files used for display in airplanes. However, of those 8000 files, there are may be many versions of the same test program. These will have the same file name with the same functions, yet some may work better than others. As of right now, the Collins engineers have no way of knowing which file works the best unless they were to manually search through all of the programs themselves. So the engineers must fix each functions everytime they find a mistake, when that mistake may have been fixed in another version.

In a database called STARWARS, Collins has a collection of test result data regarding each of the programs. This contains data including a time stamp and if a specific function either passed, failed (no errors, however did not run correctly), or crashed (did not compile or run all way through). So, given a user input of a filename + function name, they would like a desktop application to give a suggestion on which version of that specific file to use, relating to the STARWARS test result data.

Project Statement:

Our proposed solution to this problem comes in two parts. The first part being a data aggregation tool to search through the Collins Aerospace SVN directory and identify copies of a program, given a signature by the Collins engineer. The tool will collect data on each of the copies which will include a path, platform, time stamp and status of the function (pass/fail/crash). That way they will know exactly where and how to find the correct program given the feedback. The second part of our solution, the feedback/output, will be visualization of these results which will be shown below. It will include a table table containing all of the different copies of the inputted signature, cross-checking them with the different tests ran on them to display which would be the best program to use. So given one input of a signature, the application will output a table displaying test results on each copy to save the engineer time fixing problems that have already been fixed by another engineer previously.

1.3 Operational Environment

The product will be a desktop application (written in Python 3) that will strictly be run on Collins Aerospace systems. Collins Engineers will only be using this application in the Collins Aerospace office, ISU Research Park computer issued by Collins, or any laptop issued by them in order to work from home or away from the office. This is to ensure that no data is seen outside of Collins employees. Their systems use Windows OS, so there isn't any difficulties with the desktop application. They use an SVN repository and a MySQL repository for STARWARS. A MondoDB will also be upkept throughout the use of this application.

1.4 Intended Users and Uses

– To properly design an end product that will provide the maximum satisfaction and perform in the most efficient manner, it is essential to understand the end user and the associated end uses.

The only user of this product will be Collins Aerospace Engineer. Along with that, as stated above, the single use of the product is a visual displaying the recommended program they should be using given a signature as input.

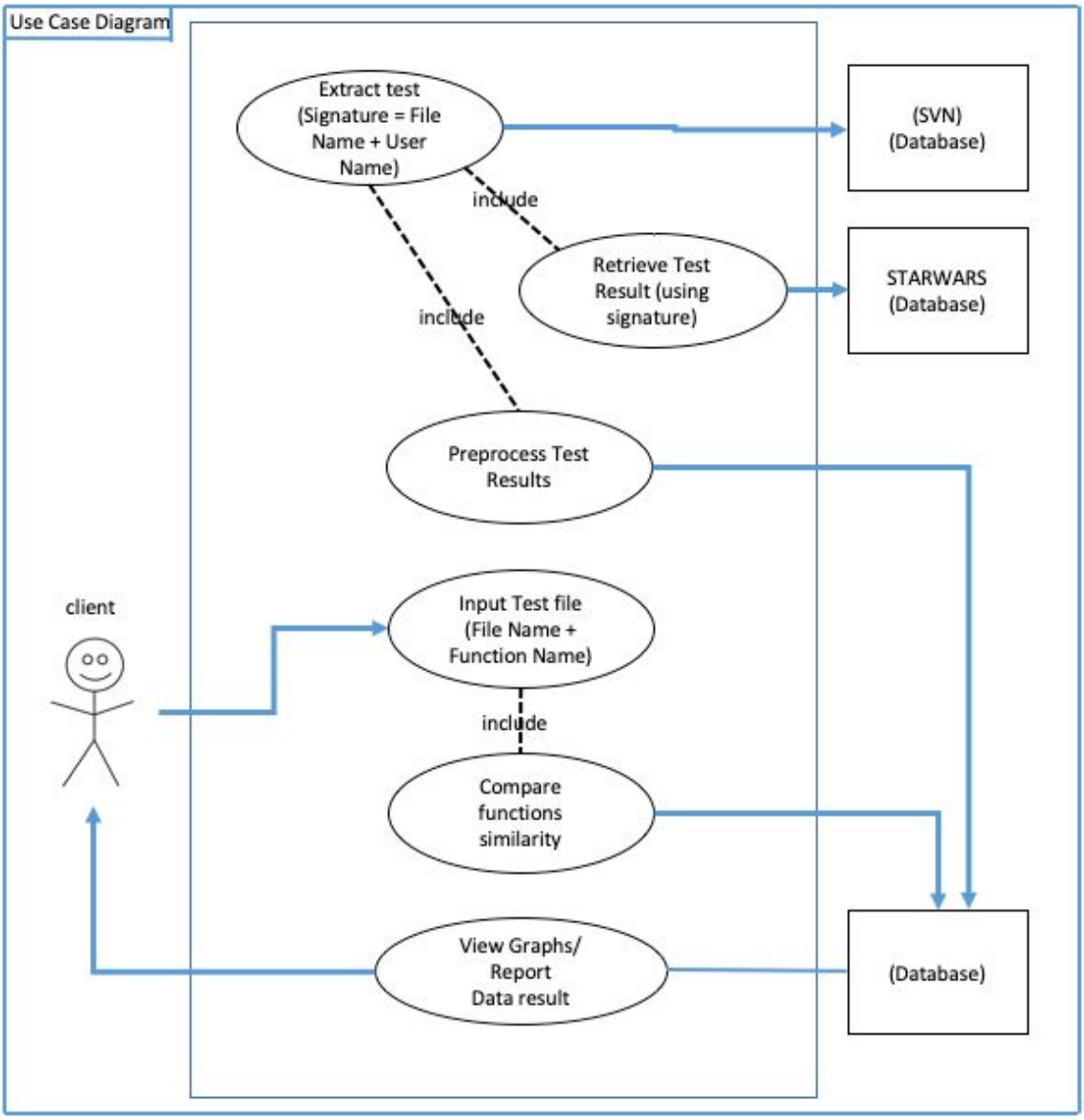

Figure 1: Use-case diagram of our project.

1.5 Assumptions and Limitations

Assumptions

- Mock Test Data will be sufficient to build application - Collins Aerospace is unable to give us the true data from their SQL servers to keep it private, so they will give us a mock data set to work with which will imitate a real set.
- This application will only run on a Collins Aerospace system, even for development and testing.

Limitations

- As stated above, we are only able to get a sample data set to work with, however it should be sufficient enough to build the application and work with the true data.
- We must work on the development at the ISU Research Park, there they will provide a Collins Aerospace laptop to work off of. We may code on our own devices, but all data will be stored on that laptop.
- We may need a Collins Aerospace employee present in order to gain access to the laptop, so we will need to work around all of our schedules to do so.
- There are no costs associated with this device, unless Collins Aerospace decides to provide a private repository to work off of.

1.6 Expected End Product and Deliverables

The delivered product will be a desktop application that the developers at Collins Aerospace will run on their systems by Collins engineers. Although developed in two different parts, data aggregation and visualization, they will both be included in the delivered desktop application. The application will mine given data and return a graph to determine the best test file to work with, which will save a lot of time and increase productivity. This product needs to be delivered in December 2019.

We will also need to deliver all related documents such as design, code and architecture documents in order for them to continue to build off of this application in the future if they wish. The delivery date will be December 2019.

2. Specifications and Analysis

2.1 Proposed Design

Design Specification

Below is a flow diagram of our project design. Our test files will be located in Collins SVN repository, that is where we will get them to aggregate the data using our application. We will also be aggregating test result data from “STARWARS”. The application will be storing the test and each one’s performance and in return our tool will be able to display the results back to the user using data visualization tools. The data will be shown in form of graphs showing performance of each test over time and their consistency and there will be reports that could be downloaded.

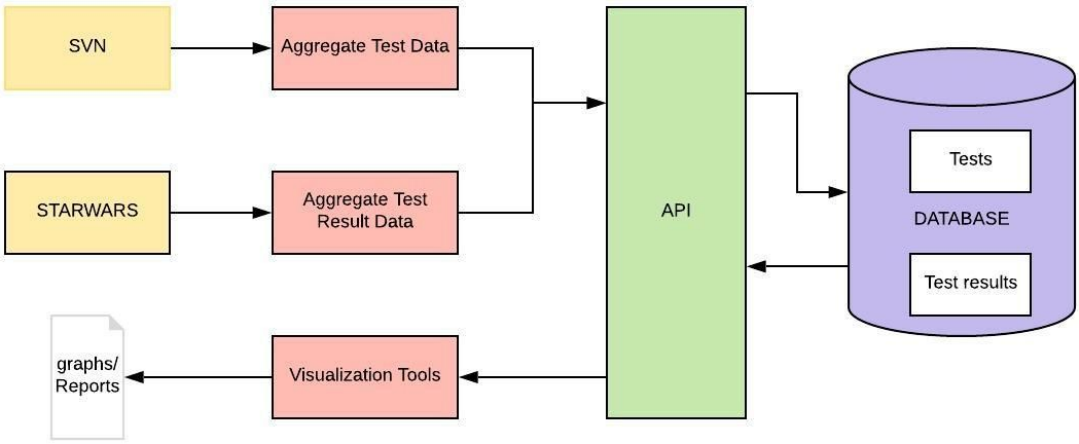

Figure 2. Design Flow Diagram for the tool

Data Aggregation

```
    # how did this happen!?

    assert 5 > 7, (
        "expected to fail" )


def testing_gt_when_lt():
    """
    Make sure the "greater than" operator works when the left is less than
    the right for fixed values
    """
    l, r = ( 18 , 300)

    # ignore this code
    # l += 2

    assert not l > r, """{l} is not bigger than {r},
{r} is like {times}x bigger""".format(
        l=l, r = r,
        times=((1.0 * l)/r))


def testing_gt_when_lt_rand():
    left = random.randint(0, sys.maxint - 1)
    right = random.randint(left + 1, sys.maxint)
    assert not (left > right), \
        'I don\'t think {left} is bigger than {right}'.format(left, right)


def testing_gt_when_equal():
    left = 21
    right = 21
    assert not (left > right), (
        "Equal values can't be greater than eachother")


def just_another_test_but_I_wanted_to_manually_raise_the_assertion_this_time():
    if not 19 > -2:
        raise AssertionError('19 is bigger than -2')


def crashing_test():
    left = 2
    right = 1
    raise Exception('This is an example of a crashing test')
    assert left > right
```

Figure 3. Example input python file 1

```
    # How did this happen!?

    assert 5 > 7, (
        "expected to fail" )


def testing_gt_when_lt():
    """
    Make sure the "greater than" operator works when the left is less than
    the right for fixed values
    """
    l, r = ( 418 , 300)

    # ignore this code
    # l += 2

    assert not l > r, """{l} is not bigger than {r},
{r} is like {times}x bigger""".format(
        l=l,
        r=r,
        times=((1.0 * l)/r))


def testing_gt_when_lt_random():
    left = random.randint(0, sys.maxint - 1)
    right = random.randint(left + 1, sys.maxint)
    assert not (left > right), \
        'I don\'t think {left} is bigger than {right}'.format(left, right)


def just_another_test_but_I_wanted_to_manually_raise_the_assertion_this_time():
    if not 19 > -2:
        raise AssertionError('19 is bigger than -2')


def crashing_test():
    left = 2
    right = 1
    raise Exception('This is an example of a crashing test')
    assert left > right
```

Figure 4. Example input python file 2

From the Collins SVN directory we get the python files that looks like the images above. They have the same function name and functionality but may have different newlines and indentations. By using the key, our program extracts the function name and gets the information regarding the testing of the file from the STARWARS MySQL database. By doing so, the program should be able to figure out which file has the better version of the function and recommend that file and its' path to the user.

Data Visualization

The table below shows our main visualization for the report that will be the end product of our program according to the functional requirement. Depending on the signature provided by our user, the program will show the comparison against the other versions from different programs and show the result of the test and time of the test. P represents pass, C represents crash, and F represents fail. Depending on the status, the percentage is displayed.

			Week														
Sig	%	Program	1	2	3	4	1	2	7	8	9	10	11	12	13	14	15	16
		A							P	P			P					

A1XHD	100	B		P	P			P		P			P		P		P		
		C	P	F	P				P										
		D														P			
BD4AO	95	A	C	C			F		C										
		F	C	P															
		G			C								P						
C489H	80	E				P													
		D	P		C			C					C						

Table 1. Example output of the program

Data Structure

As per our functional requirement, the data structure is a key-value pair represented in spatial-temporal way. It will represent the hierarchy of the test values and link each key signature to the attribute values. For example:

linkage: { nodeId, inLinks: (1,2,3), outLinks: (7) }

In order to store the key-value pair, we are using MongoDB which is a NoSQL and is widely used and known for key-value pair data storage.

Test Result Attributes

1. Test instance: the test instance will have the path to the file, file name, and function name. It will be achieved by recursively iterating through the repository path and parsing the python files.
2. Platform: it will show whether or not it is in the host or target.
3. Status: as mentioned above, it will show if it passed, failed, or crashed.
4. Degree of failure: will be determined from the STARWARS("Software Test Archive and Reporting With user Authentication and Registration Support") database result.
5. Timestamp: will show when did the test run.
6. Number of runs: will show how many times the test was ran.

2.2 Design Analysis

What we have done: We have no real design implemented at the moment. Instead, we have been discussing with our clients about the layers and flows of the design of how everything should work with each other.

For this project, we separated it into 2 parts, back-end, and front-end. The team on the back-end will be working on data aggregation, while the front-end will be working on data visualization.

Back-end: This part of the project is greatly involved with databases and algorithm, which will be extremely challenging for us while we are still trying to figure a good data model to adapt with. The

back-end will be developing a tool that will mine data from multiple data sources and make data relationships from the aggregated data.

One of their data sources is “Subversion” A.K.A “SVN” an open source control system, which currently has more than 80 in branches that contain .py test files, and “STARWARS” which contain the “Test Run Data” for each .py test files. In order to extract those “test data”, we have to implement a recursive iteration through the repository path in the SVN and identify all programs in the path and all branches in the program. With the given “path URL + File Name + Function Name” we will be able to get the test instance of every test file from STARWARS. Each test instance stores historical results of the test file, and we will make use of those aggregated test results and function names to make relationships for data visualization.

Each path in the SVN leads to a test file. Each test file is a python file which has a file name, and functions implemented in it. The whole function code will be hashed to create a “signature” and stored in a new database system along with other attributes that will be cover in this section. The function that has the same name could have multiple versions throughout the SVN. Before the function code is being hashed, we have to design a code that will remove comments in the code, line indentation, spacing, and etc without affecting the functionality of the original code. Each of this function code could occur multiple times in different programs and branches in the SVN, our clients called it “flavors” for the different versions of the function code. After obtaining the flavor of the current test file it will be stored in a database with its path URL, the file name, function name, function signature, program, and branch. This process will keep repeating until all test files are covered in the SVN and stored in a new database system.

Front-End: The front-end will be mainly developing a tool that will help our client to visualize relationships from the aggregated data. There will be no complicated computations happening on the front-end, therefore we have come up with a solution by using a client-server model with Socket API to communicate between both ends for data transmission. The reason we chose Socket API is that we could benefit from the TCP protocol, this should work well because we should expect more than multiple users using the same application and no data should be lost with an in-ordered delivery to the front-end. Besides, the front-end will be receiving fetched results from the database and will be able to generate a visualization e.g. (graphs/report) as mention in 2.1 proposed design.

Strengths: The Strength of our design is that we preprocess the SVN database and STARWARS database and place them into a new MongoDB database. The reason why we use MongoDB is because it uses key/value pairs. By preprocessing it, when the engineers are actually using the application, it would by faster by just sending query to MongoDB which improves the search performance and also saves much time on waiting to get the data representation.

Weakness: The weakness of the design is, during the preprocessing stage, we might have to leave the application running overnight, as we have mentioned earlier there are more than 8000 python test files, but it shouldn’t pose a problem because we have discussed about this issue before.

2.3 Functional Requirements

Analyze input files Given the file and key, the program should find what the key represents. It should also identify how similar one program is to others and how many of them are using the same version.

Visualize end result After comparing the input file with other files, it should be able to output the result in a tabular form to help the user to clearly see the result.

Generate final report The summary of analyzation and visualization should be included in this final report that will be generated when the program ends.

Quick, accurate, and reliable output The result should help the engineers to quickly and easily determine what to do and provide deeper understanding.

Offer suggestions Given a file the program should see the trends and offer suggestions that will improve the result.

2.4 Non-Functional Requirements

Maintainability The Collins Engineers should be able to maintain it for their own future use possibly without the team's help.

Usability The Collins Engineers should be able to modify it and implement it to fit their environment.

Compatibility It should be compatible with the Collins system.

3 Testing and Implementation

3.1 Interface Specifications

Hardware Interface: We will not have any hardware interface as the project is fully software-based.

Software Interface: The main interface would be Python. Interface for connecting front-end and back-end development would be Socket API. To test the connection, we will use Postman. As the product would be a desktop application, we would possibly use WinAppDriver which is a free automation tool for Windows desktop apps developed by Microsoft.

3.2 Hardware and software

There will be no hardware used for this project. The only physical hardware that will be using are computers installed/running with the software application.

Software: The software of this application should be able to run in the latest Windows operating system. The software application should be able to communicate with multiple data sources and gather the required information. The software application will be design to aggregate data and display a data visualization. The data visualization includes showing historical data results of a test, give a better recommendation of a test, and relationships between multiple tests in a graph/table format.

3.3 Functional Testing

For unit testing, we will be using the Python library 'unittest'. This is a choice made due to the fact that it has good documentation and it comes standard with Python. This will not require our team to make any additional integrations. 'unittest' supports automation and other object-oriented concepts which is going to be useful for our team. We will create specific test to check the coverage of our code against the given .py files. The more coverage we get will ultimately determine the best test which will be used in our phase two of determining the best test and later in giving them out as suggestions. We will also use the coverage.py package to better test our code coverage

Integration testing should this should work hand in hand with our already mentioned unit tests that will be conducted. Both unit and integration testing will be helping us as developers identify breaking code changes. This will be carried out later in the process after our unit testing has proven to be successful. The team will then proceed to test our tool's interaction with the existing Tests at Collins Aerospace. The tool's installation and overall integration in the Collins system will not be a problem since the tool will be an application that gets installed on the computer.

System testing and Acceptance testing will be carried out by our client, and some of Collins engineers since they will be our only users. This process will come after the first two phases of testing.

3.4 Non-Functional Testing

Our criteria for performance testing will be making sure that the tool is able to analyze all tests and how fast this tool will be. We will be using the Locust framework for testing the tool's performance, with Locust we will be writing performance scripts. Locust is one of the top libraries for load and performance testing. It will require us to install it since it is not standard with Python but this will cause no issues.

Regarding security, we do not have to worry about our tool being corrupted since it will only be used in house at Collins aerospace. There is no risk of data corruption because Collins Aerospace is very particular of their data. In development, we will only be able to test the data on a Collins computer and our repository will be private and only accessible to us and our client.

For usability, this tool will be simple with simple features that will make is user friendly, as shown before, it will show performances of each test in a neat table. Later on we will look into the possibility of making suggestions about tests. For our usability testing we will be testing again our requirements and making sure that it is consistent. We will be running our tool on several of the Collins computers as well as getting feedback from the client.

For the compatibility testing, we will be making sure that the tool we will have developed, is compatible with Collins systems, we will be using Python 3 and since it is what the Collins engineers use currently, we will have to just make sure that our windows application is compatible with their systems.

3.5 Process

Workflow

1. The input will be a problematic test.
2. The program will analyze and compare the input test against the test with same name to report on all variations of the test, based on the code signatures.
3. Report will be generated as an output to show the comparison against other versions of the test and other programs' test results history for the same test name.
4. Rapid and intelligent data-driven changes will be applied to the artifacts.

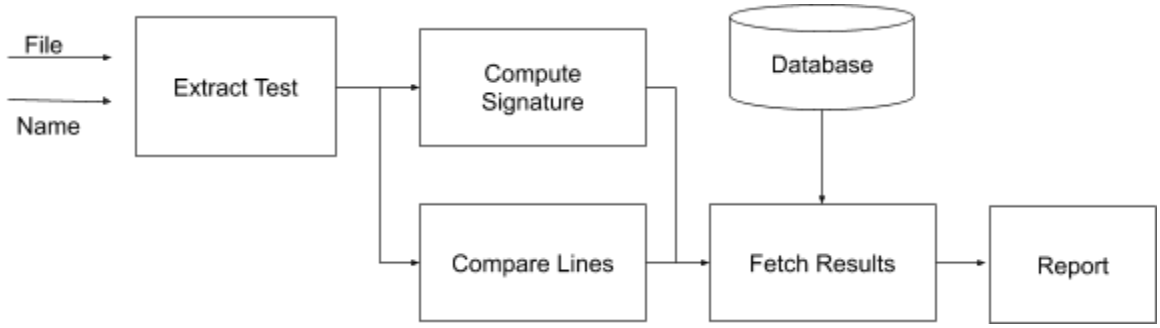

Figure 5. Workflow Diagram

3.6 Results

No testing results has been obtain at the moment, testing results will be able to obtain during the implementation of the system.

Front-end: None

Back-end: None

4 Closing Material

4.1 Conclusion

To this date, our team has just received access into the Collins Aerospace systems at the ISU Research Park. We have a project plan in place including the specific software libraries we will be using, along with our design diagrams to build off of.

Our goal of this project is to deliver a desktop application to Collins Aerospace that, given a signature input from an engineer, will output a suggested program for them to use. This will save them a lot of time fixing mistakes in their files that have already been fixed by another engineer. To achieve this product, we will have a two-part system, data aggregation (back-end) and the visualization of the data (front-end). The aggregation will mine through multiple data sources in order to receive a specific and unique signature, which is given by a "file name + function name".

Along with each signature will be a sample of "test data", which shows how well that function is ran for a given test. The visualization side will then take all copies of a given signature, and create a table in order to see how each different copy compares with the tests ran on them. There weren't too many possibilities of solutions, as it is a single use case, however it is a large one at that. They had experimented with the idea of completely reconstructing their database, however with 50 engineers and over 8000 files, that would just make more of a mess than necessary. This is also a product that they are able to build off of, possibly automate. While working on a program, it would make a note that their are better options to be using, saving themselves even more time. This product is to be delivered to Collins Aerospace by December 2019.

4.2 References

Docs.python.org. (2019). socket — Low-level networking interface — Python 3.7.3 documentation. [online] Available at: https://docs.python.org/3/library/socket.html [Accessed 26 Mar. 2019].

"Unittest - Unit Testing Framework¶." *Unittest - Unit Testing Framework - Python 3.7.3 Documentation*, docs.python.org/3/library/unittest.html.

EDUCBA. (2019). MySQL vs NoSQL - Which One Is More Useful (With Infographics). [online] Available at: https://www.educba.com/mysql-vs-nosql/ [Accessed 26 Mar. 2019].

Bushnev, Yuri. "JMeter vs. Locust - Which One Should You Choose?" *BlazeMeter*, BlazeMeter, 17 Oct. 2017, www.blazemeter.com/blog/jmeter-vs-locust-which-one-should-you-choose.

4.3 Appendices

Socket API manual: https://docs.python.org/3/library/socket.html

Locust Documentation: https://docs.locust.io/en/stable/

Coverage.py Documentation: https://coverage.readthedocs.io/en/v4.5.x/