# Collins Aerospace Artificially Intelligent Requirement Analysis Tool

Design Document  Team 8

04.24.2019

SE 491

Team 8

**Members:**

**Apurva Patel** - *Project Lead, Report Manager, Technical Support, Communicator,  AI Training Lead*

**Ryan Cerveny** - *Scrum Master, Meeting Scribe/Facilitator, Project Lead, Communicator*

**Takao Shibamoto** - *Chief Engineer, Researcher, UI Lead*

**Jonathan Murphy** - *Testing Engineer, Researcher, Requirement Lead*

**Client:**

Collins Aerospace (Representative: Jason Wong)

**Faculty Advisor**

Dr. Simanta Mitra

Design Document Team 8 0

**List of Figures** **4**

**List of Symbols** **4**

**List of Definition & Acronyms** **4**

**1 Introduction** **6**

1.1 Acknowledgement 6

1.2 Problem Statement 6

Figure 1: Use Case Diagram 7

1.3 Operational Environment 7

1.4 Intended Users and Uses 8

1.4.1 Intended Users: 8

1.4.2 Intended Uses: 8

1.5 Assumptions and Limitations 8

1.5.1 Assumptions 8

1.5.2 Limitations 8

1.6 Expected End Product and Deliverables 9

1.6.1 Expected End Product 9

1.6.2 Project Deliverable: 9

**2. Design Specifications** **10**

2.1 Word2Vec with Word Mover's Distance 10

2.2 GloVe and Cosine Similarity 10

2.3 Universal Sentence Encoder 11

2.4 Topic Modeling 11

2.5 Proposed Design for Word/Sentence Embedding 12

Figure 2: Block Diagram 12

2.5.1 Reports Viewer and Reports Finder: 12

2.5.2 Configuration Form/Analysis Initializer and the Configuration Manager: 13

2.5.3 Parser 13

2.5.4 Pretrained Vector Model 14

2.5.5 Word/Sentence Embedding Library 14

2.5.6 Classification Model Training 14

Figure 3: Logistic Regression Example 14

2.5.7 Analysis Algorithm 15

2.5.8 Report Generation 15

2.5.9 Database 15

2.5.10 User Interface 16

2.6 Proposed Design for Standard Enforcing Requirement Hosting Platform 16

Figure 4: Requirement Hosting Tool Design 16
2.6.1 Requirement Creation/Updating 17
2.6.2 Requirement Storage 17
2.6.3 Requirement Viewing 17
2.6.4 Version Control 17
2.6.5 Requirement Analysis 17
2.7 Proposed Design For Topic Modeling 18
Figure 5: Topic Modeling Design 18
2.7.1 Command Line 18
2.7.2 Requirement Directory Parser 18
2.7.3 Topic Modeling Module 18
2.7.4 Gensim Library for Topic Modeling 18
2.7.5 Link Classification Algorithm 19
2.7.6 Reports Module 19
2.7.7 Report Storage Directory 19
2.8 Current Status 19
2.8.1 Conclusion of Work Done so Far 19
Figure 6: Main Page of GUI 20
Figure 7: Training Settings Page of GUI 20
2.8.2 Implementation Issues and Challenges 20

**3 Testing and Implementation** **21**
3.1 Interface Specifications 21
3.2 Hardware and software 21
3.3 Functional Testing 21
3.3.1 Testing Model Accuracy 21
3.3.2 Testing Algorithm Under Configurations 22
3.3.3 Testing Parser with Broken Input 22
3.3.4 Testing Methods 22
3.3.5 Traceability Testing: 22
3.3.5.1 Forward Traceability Test 22
Figure 8: Test Case for Forward Traceability 22
3.3.5.2 Backward Traceability Test 23
Figure 9: Test Case for Backward Traceability 23
3.3.5.3 Bi-Directional Traceability Test 23
Figure 10: Test Case for Bi-Directional Traceability 23
3.3.6 Classification and Prediction Testing: 24
3.4 Non-Functional Testing 24
3.4.1 Security 24

3.4.2 Portability 24
3.4.3.Performance 25
3.4.4 Modifiability 25
3.5 Process 25
3.6 Results 26
3.6.1 Word2vec & Word Mover Distance Approach 26
3.6.2 GloVE and Cosine Similarity 26
3.6.3 Universal Sentence Encoder 27
3.6.4 Topic Modeling 27
4 Closing Material 28
4.1 Conclusion 28
4.2 References 29

## List of Figures

1. Figure 1: Use Case Diagram
2. Figure 2: Block Diagram
3. Figure 3: Logistic Regression Example
4. Figure 4: Requirement Hosting Tool Design
5. Figure 5: Topic Modeling Design
6. Figure 6: Main Page of GUI
7. Figure 7: Training Settings Page of GUI
8. Figure 8: Test Case for Forward Traceability
9. Figure 9: Test Case for Backward Traceability
10. Figure 10: Test Case for Bi-Directional Traceability

## List of Symbols

1. Symbols 2: Le is a set of existing links
2. Symbols 3:Lp is a set of predicted links
3. Symbols 4: ∩ intersection

## List of Definition & Acronyms

1. Requirement Tracing

   The process of tracing or recording the links between the higher level requirement system with other individual lower or other higher level requirement system.

2. Requirement Gathering

   The process of gathering the specification as the requirements such as use cases, higher level requirement, functional requirement, technical requirement and many more from the stakeholders that will be used as the formal data for requirement tracing.

3. Supervised learning

   Machine learning task to related every input with a desired learned output based on the example output input pair.

4. Unsupervised learning

Machine learning task to draw inferences from dataset that includes inputs with no labeled response.

5. NLTK

   The Natural Language Toolkit. A python library that was used for stop words.

6. Confusion Matrix

   It is a table with rows and columns that reports the number of false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN).

7. LDA - Latent Dirichlet Allocation, one of the method for Topic Modelling

# 1 Introduction

## 1.1 Acknowledgement

We want to acknowledge all our clients at Collins Aerospace, specially to Jason Wong, Kathleen Knott, and Branden Lange for supporting us and providing us with feedbacks and resources for carrying out the project. We also want to acknowledge our faculty advisor Dr. Simanta Mitra for guiding through the development process, college of engineering and Dr. Thomas Daniels for providing us guidance, expertise and necessary hosting resources.

## 1.2 Problem Statement

Requirement tracing, which is the process of creating logical links between individual requirements, is essential in projects carried out by Collins Aerospace. When working with safety critical systems, it must be ensured that all necessary features are recognized, that no unnecessary features are included, and that we can link the reasoning for including a component to a higher feature. Projects at Collins Aerospace may include thousands of requirements, most of which link to one or more other requirements. As of today, employees at Collins create and review these requirements by hand or with "non-intelligent" tools. Manually reviewing the accuracy of a requirement trace and deciding which links are good and which need to be removed is extremely expensive in terms of the time that must be dedicated to ensure that the requirement trace is sufficient for the project at hand. The purpose of this Capstone project is to develop a tool for Collins in order to automate requirement trace analysis.

Our proposed solution to this problem is to utilize the topic modeling functions of the Gensim Python library. These algorithms analyze the similarities between text by determining the topic of the text through natural language processing techniques that analyze the context, and comparing the determined topic to the topics of relating to other texts in order to calculate similarity. Our tool will take multiple Excel documents as input, which specify all individual requirements and the other requirements that each are linked to. By using the gensim libraries, we will be able to generate similarities between the linked requirements and feed them into a model which will then flag the link as either good, bad, or suspicious. The tool will also recommend possible links for requirements which may have had a bad link or if it finds a very likely match. Finally, a file containing a report generated by the analysis will be stored for the user to view.

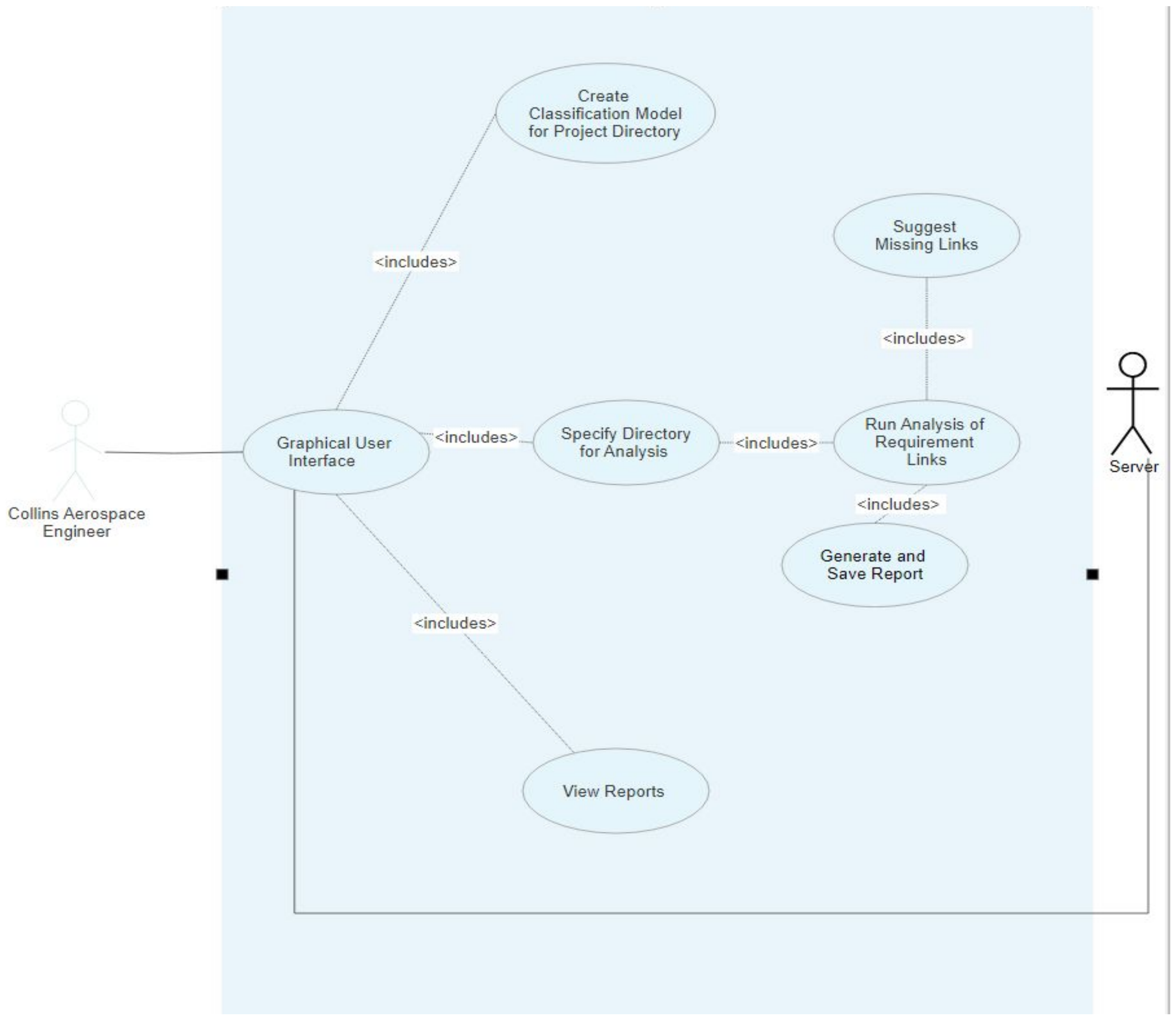


Figure 1: Use Case Diagram

## 1.3 Operational Environment

Collins Aerospace Artificial Intelligence for Requirement Analysis Tool is a software based project. The tool will be operated on the Collins Aerospace server. The user will operate the tool using a command line. Any operating system that has a basic command line would work. The tool will require python3.7, gensim, scikit learn, scipy, spacy, matplotlib, nltk, anaconda 3 and Windows/Linux/iOS operating system to function as per the requirements. The following tool process a sensational data related to security which needs the tool to be highly secure. Thus, secure environment along with a securely implemented tool is a must.

## 1.4 Intended Users and Uses

### 1.4.1 Intended Users:

The engineers at different departments at Collins Aerospace will be using the tool. Each department will have their own tool configured separately. Thus, every engineer would access the tool as a common user.

### 1.4.2 Intended Uses:

Artificial Intelligence for requirement analysis tools is a software based tool that will be used as requirement tracing. The main uses of this tool is to classify the higher level requirement link and lower level requirement links as good, bad and suspicious. Moreover, it will also provide a prediction of lower level requirement links that relates to the higher level requirement. At the end of the analysis and prediction process, a report will be generated mentioning the good, bad, and suspicious link along with the prediction of the links.

## 1.5 Assumptions and Limitations

### 1.5.1 Assumptions

- A web form will be sufficient for configuring settings in the analysis process.
- HTML, CSS, Bootstrap and JavaScript will be sufficient to develop the front-end UI.
- A configuration file mentioning the higher level platform and lower level platform and how they relate will be provided in the form of settings.py.
- The user will operate the command line or GUI when new data is been throw into the AI.
- A group of users will use the tool as a single type of user, any amount of groups can access the tools.
- Clean and readable output will be provided, representing the data with an appropriate color coding along with detailed labeling
- System will be sufficient enough to handle multiple numerous user request

### 1.5.2 Limitations

- The size of the data varies from department to department.
- The format of the input file will vary.
- The training model needs to be configured continuously in order to make the Artificial Intelligence tool continue to learn newly updated terms and definitions.

## 1.6 Expected End Product and Deliverables

### 1.6.1 Expected End Product

Our team plans to deliver a tool to read requirement documents provided to it by a user, analyze the documents, and classify links between requirements as either good, bad, or suspicious. The tool will be hosted on a linux server and will be capable of being accessed by either command line or web GUI. Secondary deliverables include expanding the tool to include the following features:

- Identify/suggest possible missing links between requirements
- Analyze links between code and requirements
- Analyze links between tests and code

### 1.6.2 Project Deliverable:

Collins Aerospace will be provided with the following deliverables:

- Well commented source code
- Executable for the Tool (batch file)
- GUI
- Instruction Manual such as written documentation
- Step-by-Step video instruction manual on how to rebuild the executable and how to setup the tool
    - Updated READ_ME.txt file with a list of required tools and libraries such as follows:
        - Python 3.7
        - Anaconda 3
        - Gensim
        - Matplotlib
        - Pandas
        - Numpy
        - NLTK
        - Scikit learn
        - Spacy
        - Scipy
- Step-by-Step video on how to use the tool
    - Short Tutorials on different segments of the tools in order to help the engineer at Collins Aerospace to quickly grasp the tool
- Updated UML Diagram, Use Case Diagram, Block Diagram
- Documentation related to the algorithms like Word2Vec, Word Mover Distance, GloVE, Sentence Encoding, Topic Modeling. The following document will include a brief explanation to all the above algorithms along with their pros and cons related to the project. Moreover, it will also include a status related to the use of the algorithm based on the project's final result.
- Deliver a live presentation of the final product

# 2. Design Specifications

There have been several design approaches to this project which has evolved throughout the semester. Approaches we have experimented with or ideated on include:

- Word2Vec with Word Mover's Distance
- GloVe and Cosine Similarity
- Universal Sentence Encoder
- Topic Modeling
- Standard Enforcing Requirement Hosting Tool

More explanation on the ideas themselves can be seen in our final project plan. The following sections describe the designs for each approach or group of approach:

## 2.1 Word2Vec with Word Mover's Distance

Word2Vec is a word embedding software which we used via the Gensim python library. In short, textual data is first provided to a Word2Vec model. From there, the model creates unique vectors which store the "meaning" or context of each provided word. Words of similar meaning or context will be located in the same general vector space. More in depth explanation and visualization can be found in reference [6].

Once the Word2Vec model is trained, the Word Mover's Distance algorithm comes into play to calculate similarity between texts, or in our case, requirements. Word Mover's Distance uses the Word2Vec vectors from the first text, and then finds the minimum distance to translate to the Word2Vec vectors in the second text. The smaller the distance it takes Word Mover's Distance to translate between the sentences, the more similar they are. More in depth explanation and visualization can be found in reference [7].

The idea behind using Word2Vec and Word Mover's Distance is that requirements that should be linked, should have text that are similar.

## 2.2 GloVe and Cosine Similarity

GloVe, short for Global Vectors, is a common approach for determining the semantic similarity of words. This comparison is done through embedding the words to be compared,then calculating the cosine similarity between them (e.g. the cosine of the angle between two vectors of a product space that can be used to measure the similarity between the vectors) based on their positions in the multi-dimensional global vectors. The cosine similarity is returned as a decimal number representing the similarity of the words, where two of the same words are given the maximum similarity value of 1. For example, when comparing the words "man" and "woman", the resulting cosine similarity would be very large, but very small if you were comparing two completely unrelated words.

## 2.3 Universal Sentence Encoder

The following algorithm takes word sentences into account and converts them into word vectors of length 512 irrespective of the input size. Based on the word embedding vector, it calculates the semantic textual similarity. Thus, this approach is similar to what we have discussed earlier in Word2Vec and GloVE and Cosine Similarity algorithm.

## 2.4 Topic Modeling

The idea of topic modeling came up in our discussion after the experiments that showed that word2vec, WMD, and Glove which are the approaches that directly compare the similarity of texts turned out to make bad predictions. Topic modeling approach, on the other hand, doesn’t directly compare the similarity of two texts. Instead, it compares the topic probability distribution of two texts. So far it is showing some interesting results and we are focusing on this approach.

We have used the most popular topic model called Latent Dirichlet allocation (LDA). It puts that each document (in our case, each REQ/UC) is a mixture of a small number of topics with weight and each topic constitutes of small number of related words with weight. [4]

There are multiple factors to consider in topic modeling. There are mainly two things we need to consider: 1) how to make topics and 2) how to compare topics. We performed experiments with slightly different variations and the following is what we have tried.

Topic modeling experiment 1 - Separate LDA models for REQs and UCs

1. Find top N most used words and prompt engineers to choose stop words (this technique would apply to any kind of algorithms we would use)
    - eg. we probably don’t need “shall” in REQ and “user” in UC
    - maybe completely remove any verbs
2. Somehow unify similar words
    - eg. unify “misspell”, “misspells”, “misspelling”, “misspelled” into “misspell”
3. Make separate LDA models for REQs and UCS
4. Compare the probability distribution of words for each topic for each req for each uc
5. Each REQ chooses top X similar UCs

Topic Modeling Experiment 2 - One LDA model for both REQ/UC

1. Filter out all words except nouns
2. Combine REQs and UCs into one table
3. Make a LDA model with N topics
4. Choose the topic which has the highest score for each REQ/UC
5. For each REQ, find UCs that are in the same topic

Topic Modeling Experiment 3 - Same as Experiment 2 except that for step 4 and 5, we compare the probability distribution of topics.

For example, assume we find three topics for the requirement data. REQ1 has a topic distribution of { (0, 0.5), (1, 0.25), (2, 0.25) } and UC1 has a topic distribution of { (0, 0.25), (1, 0.25), (2, 0.5) } and UC2 has a topic distribution of { (0, 0.6), (1, 0.2), (2, 0.2) }, where topic distribution is a set of tuples and each tuple represents pair of the topic index and its weight. We now use the method of least squares to calculate the distance of these REQ/UC.

For REQ1 and UC1, $(0.5-0.25)^2 + (0.25-0.25)^2 + (0.25-0.5)^2 = 0.125$

For REQ1 and UC2, $(0.5-0.6)^2 + (0.25-0.2)^2 + (0.25-0.2)^2 = 0.015$

Thus UC2 is the better link for REQ1 because it is closer in similarity.

## 2.5 Proposed Design for Word/Sentence Embedding

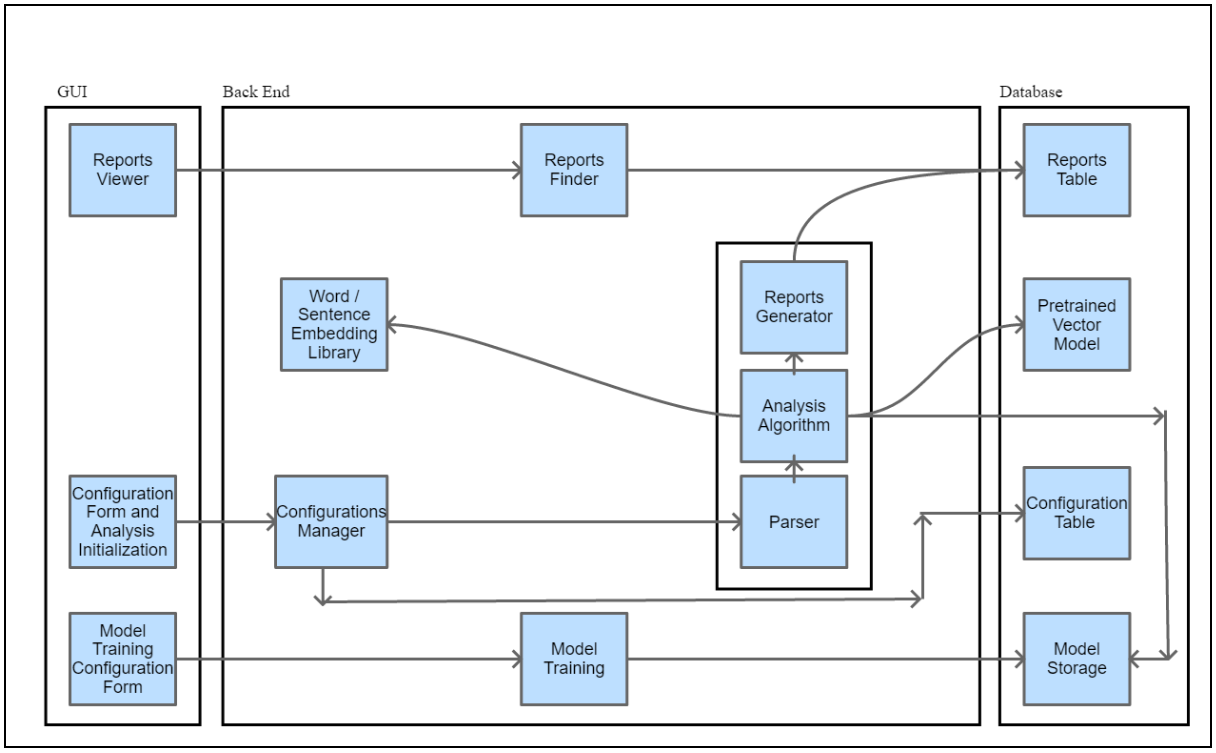


Figure 2: Block Diagram

### 2.5.1 Reports Viewer and Reports Finder:

The reports generated by this tool will be stored in the Reports Table in the database. These reports will be able to be viewed by the user via the Reports Viewer on the GUI. The user will go to the Reports Viewer and select a report they would like to view. The desired form will be sent to the Reports Finder module which will pull the report from the table and send the data back to be displayed in the Reports Viewer on the GUI. The user will also have the option to download the report into an excel file from this page.

### 2.5.2 Configuration Form/Analysis Initializer and  the Configuration Manager:

The tool will need information from the user in order to run the analysis process. The information will be provided by a webform on the GUI. This information will include the following:

- The input requirement documents will be selected to be upload to the server for analysis
- Information about which requirement file links to which other requirement file
  - This may be also be implemented by enforcing a file format which declares a column only if the files are linked
- The input document layout (i.e. the column names) so that our parser knows how to read the input files
  - For simplicity, the file layout may become strict requiring the user to follow it
- Whether to scan for possible missing links
  - Not scanning for these missing links will make the process run much faster because there will be a reduced number of comparisons between requirements
- Whether to display or download the report when analysis is complete
  - If not specified to do either of these options, the report data will still be stored in the database.

Once the user has specified all of the information that the tool requires to run, the user will have three options for proceeding. First, they may start the analysis process with the settings provided in the form they filled out. Otherwise they could simply save the settings without running the tool, or finally, they could both run the tool and save the configurations at the same time.

The user also will be provided with the alternative option of providing and submitting a settings file. The parser will be set to read either option so that we may also run the tool via command line effectively. The analysis module will just need to be called with the respective settings file and the process can then be called or even automated or scheduled.

### 2.5.3 Parser

The parser is the first component of the analysis process. Before our tool can analyze any requirements, the parser will extract the data from the requirements documents provided by the user.  The parser will receive parsing rules from either a GUI web form or a settings file specified by the user. From there, the parser will accomplish the following tasks:

- Check that the files contain the required columns which include:
  - Object ID
  - Requirement text
  - Derived
  - Object ID of linked requirement

- Replace any abbreviations with their full wording
- Extract the fields listed above into a dictionary which will be used by the analysis algorithm

### 2.5.4 Pretrained Vector Model

This represents the Google Word2Vec Model or the GloVe Vectors Model would be stored. It would live on the backend and allow for the Analysis Algorithm to query it to get the vectors for the sentences that are being analyzed.

### 2.5.5 Word/Sentence Embedding Library

This module was applied for three of the design approaches we considered: Universal Sentence Encoding, Word2Vec and GloVe. The module represents the Python library which is used to calculate the similarities between the embedded words/sentences. For example, in the Word2Vec implementation, the Gensim library would be included and used to calculate the Word Mover's Distance for similarity.

### 2.5.6 Classification Model Training

From the Model Training Configuration Form, the user will specify requirement documents to upload for training a new model. These documents will be similar to real documents that are used in analysis, except they will have an extra column which specifies whether a link is good or bad. This column paired with output for the distance between requirements (similarity) will be what the model uses to learn.

Our classes are not as separated in our experiments as this example below, but in an ideal world the similarity data from one of the three approaches would have looked similar to the diagram below:

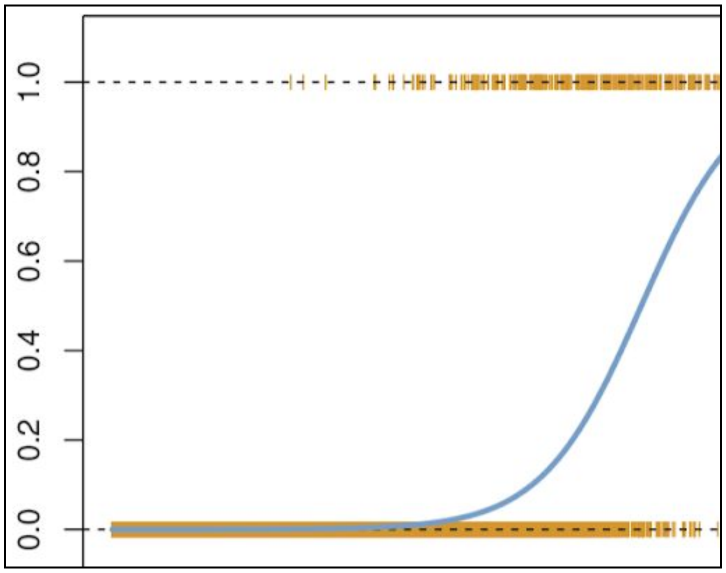


Figure 3: Logistic Regression Example

The true classification of a link should either be good or bad. In the graph above, the y axis represents the class probability and the x axis represents the similarity output by either Universal Sentence Encoding, Word Mover's Distance, or Cosine Similarity. These classifications can be seen in the figure above where the yellow marks where Y = 0 are the good links plotted at their similarity values, and the marks on the top are bad links plotted at Y = 0. The blue line is a sigmoidal function that fits the probability that any given point is either good or bad.

Say we wanted to classify a point using the graph above. We would plug in our similarity value and see what the Y value is. Let's say Y = .4 at similarity D. We would see that Y < .5 and say that the probability leans towards the link being a good link. Now if Y = .8, we would predict that the link was bad, because the probability leans towards the link being a bad link. What we can proceed to do is define a range where we classify the link as suspicious. To do this we may say that if .4 < Y < .6, then we cannot accurately predict the links class because it's nearly a 50% chance that the link is either good or bad. By following this format, we can read a similarity value and return a classification as good, bad or suspicious. To store the model in the database, we may simply store the coefficients of the sigmoidal function.

### 2.5.7 Analysis Algorithm

The analysis algorithm will bring together everything discussed so far. The algorithm will be given the dictionary of requirement data generated by the parser, will use either Universal Sentence Encoding, Word2Vec with Word Mover's Distance, or GloVe with Cosine Similarity, and will query for the desired classification model. For each link, the algorithm will use the similarity value to evaluate the similarity of the requirements. The output will then be run through the classification model which determines if the link is good, bad, or suspicious. Each of the classifications will be stored and sent to the report generator once analysis is finished.

After primary functionality is implemented, we will begin designing and expanding this algorithm to suggest possible missing links, analyze links from code to requirements, and analyze links from tests to code.

### 2.5.8 Report Generation

When the analysis algorithm is finished, it will send all of its classification decisions to this report generation module. This module will store all the findings of the analysis into the Reports Table. If specified by the user, the user will be sent to the Reports Viewer page to review the report that has been generated.

### 2.5.9 Database

The structure of the database will include three tables: Settings, Reports, and Models. Whenever a user trains a new model, the coefficients of the models will be stored in the Models table, so that they may be accessed again later. If the user specifies that they would like to save their settings, the settings will be stored in the settings table. After an analysis has been completed, the report that is generated will be stored in the Reports table so that all past reports that have been generated can be accessed and viewed by the appropriate users.

### 2.5.10 User Interface

The tool will include a simple and clean web GUI that allows users to configure the tool to their needs. There will be four pages for the user to access which are as follows:

1. Reports Viewer
2. Configurations Form/Analysis Initiation
3. Model Training
4. About/Help Page

The functionality of the first three pages have been discussed above. The fourth page will be a reference for users when there are any questions or help needed. We will provide instructions on using the tool, and explanations of how process work so that the user will be able to use the tool to its full potential.

## 2.6 Proposed Design for Standard Enforcing Requirement Hosting Platform

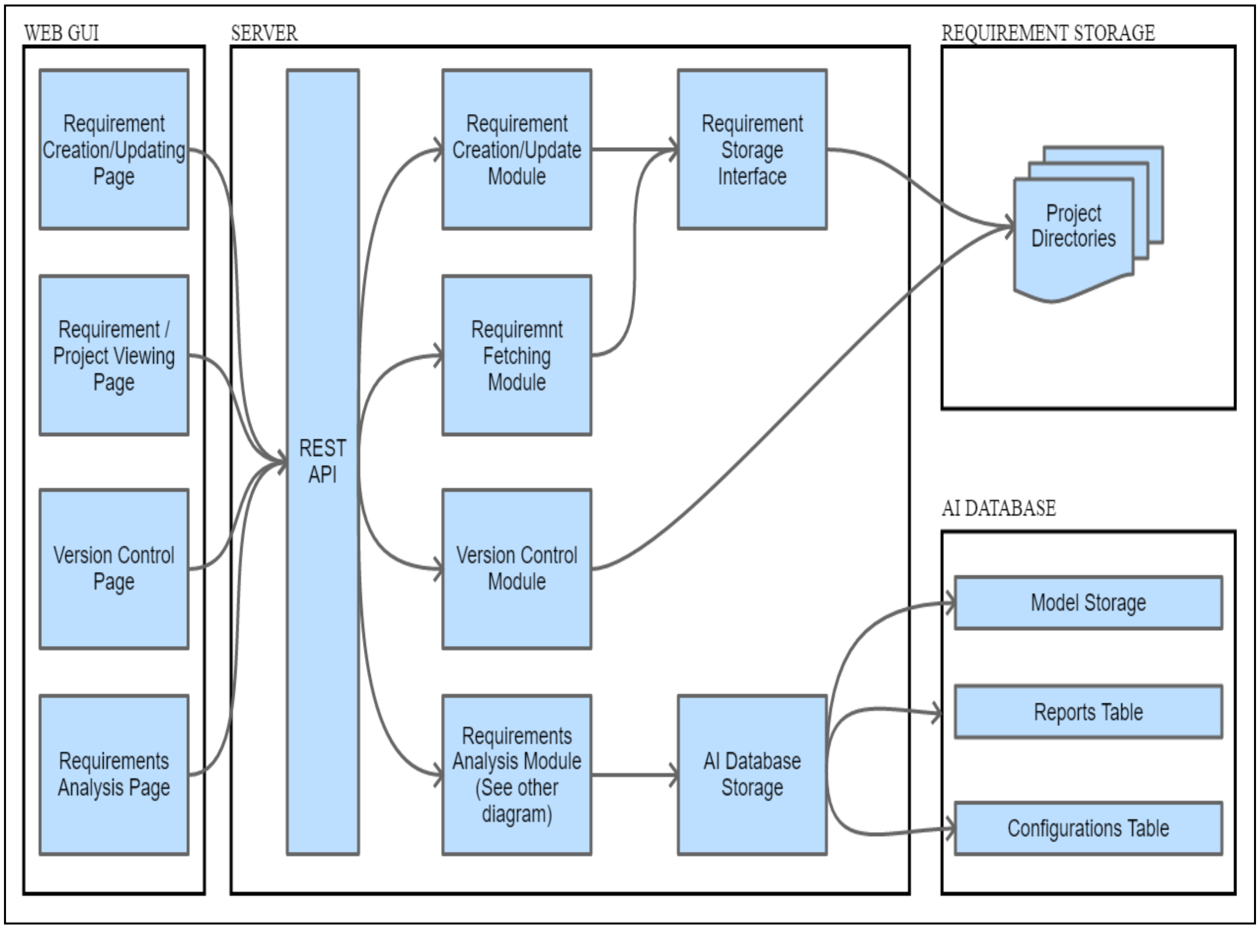

WEB GUI
SERVER
REQUIREMENT STORAGE
Requirement Creation/Updating Page
Requirement / Project Viewing Page
Version Control Page
Requirements Analysis Page
REST API
Requirement Creation/Update Module
Requirmnt Fetching Module
Version Control Module
Requirements Analysis Module (See other diagram)
Requirement Storage Interface
AI Database Storage
Project Directories
AI DATABASE
Model Storage
Reports Table
Configurations Table

Figure 4: Requirement Hosting Tool Design

### 2.6.1 Requirement Creation/Updating

When a user would like to create/update requirements, they will access the Requirement Creation/Updating Page. The page would include a form for creating new requirements which would have enforced standards that we would have created for formatting or including characteristics that would have allowed us to make a simpler classification method. Once the form is submitted, then the requirement would run through the Requirement Creation/Updating Module which calls the Requirement Storage interface to insert the requirement data into the appropriate Project Directory.

### 2.6.2 Requirement Storage

Each project would have its own directory in the Requirement Storage System. The Requirements for the project will be stored in xml format which model the standards and characteristics that were being enforced to ease the process of classification. Along with modeling the previously mentioned characteristics, the xml will also contain information about which other requirements are related, and their hierarchical status within the system.

### 2.6.3 Requirement Viewing

When a user would like to view requirements, they will navigate to the Requirement Viewing Page. Here there will be a form to specify which requirement directory to view. Once the user submits the form, the request will go through the Rest API and to the Requirement Fetching Module. This module will fetch the requested requirements and return them to the page for viewing.

### 2.6.4 Version Control

The tool will allow for version control for the requirements. If the user would like to perform version control tasks on a project directory, they will navigate to the version control page. Here there will be a form with options such as branching or rolling back a specified project directory.

### 2.6.5 Requirement Analysis

Finally, the user will be able to run an analysis on the Project Directories. With the enforced standards, an artificially intelligent analysis procedure will be in place. The Requirements Analysis Page will be the users method of running these analysis processes on their requirement directories stored within the tool. The backend for this would be designed more efficiently once the AI analysis process is defined.

## 2.7 Proposed Design For Topic Modeling

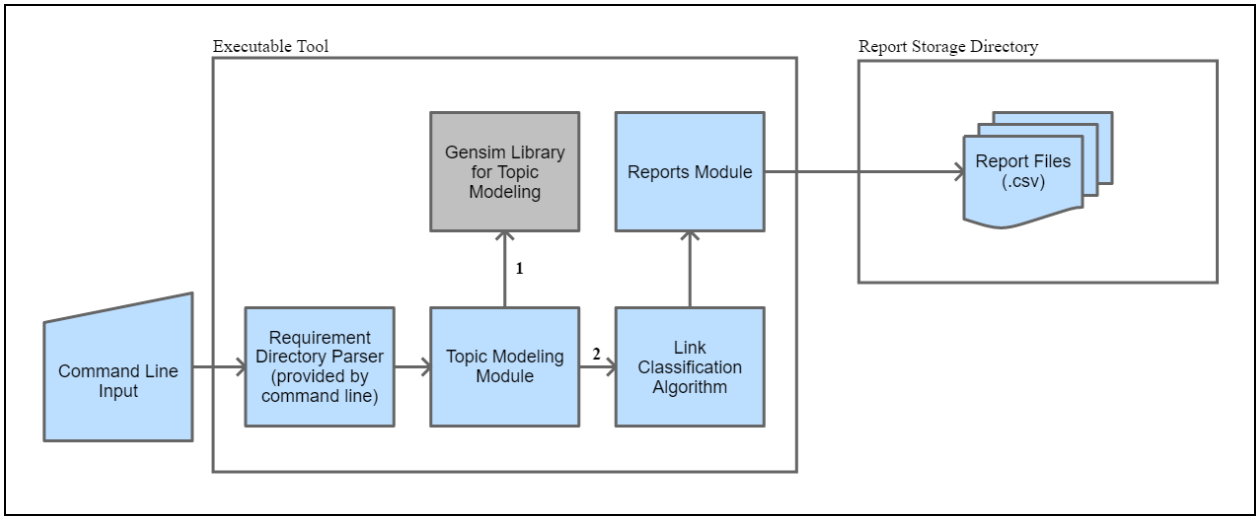

Executable Tool
Report Storage Directory
Gensim Library for Topic Modeling
Reports Module
Report Files (.csv)
1
Command Line Input
Requirement Directory Parser (provided by command line)
Topic Modeling Module
2
Link Classification Algorithm

Figure 5: Topic Modeling Design

### 2.7.1 Command Line

At the point this design was created, the web GUI was not of major concern. The tool would have been run as a command line tool. The tool would be called along with parameters which include the project directory, and any configurations need to run the analysis.

### 2.7.2 Requirement Directory Parser

This module would receive the parameters from the command line. It would find the appropriate project directory and parse it according to the provided configurations to be sent for analysis.

### 2.7.3 Topic Modeling Module

The parsed requirement links will be taken in as input for this module. The Topic Modeling Module will send the necessary configurations and parsed requirements to the Gensim library to be modeled. The return data will be passed along to the analysis algorithm.

### 2.7.4 Gensim Library for Topic Modeling

This python library will be used for implementing the process of topic modeling. It will receive all its parameters from the Topic Modeling Module and return the topic distribution once it has completed.

### 2.7.5 Link Classification Algorithm

Using the topic likelihood distribution generated previously, the Link Classification Algorithm will iterate over every link and analyze the likelihood that all of its linked requirements are good links. After analyzing existing links, the algorithm will then suggest similar requirements which are not linked already. All of the findings from this algorithm will be saved and sent to the reports module.

### 2.7.6 Reports Module

The reports module will simply receive all the findings from the analysis algorithm and generate/save a csv file containing all the discovered information. For example, the algorithm may discover two requirements as being a bad link. The reports generator will output a line including the requirement ID's, the requirement texts, and the likelihood that was produced for whether the requirements should be linked.

### 2.7.7 Report Storage Directory

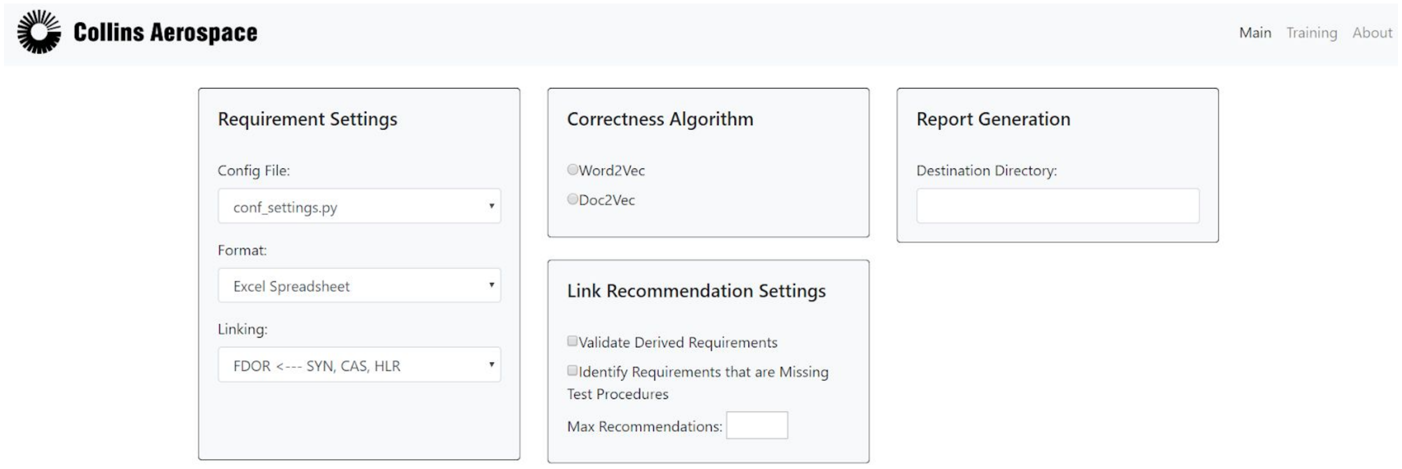
All of the reports would be stored in this directory. Each project will have its own directory within it where the reports for the given project will be stored. This way, the users may see a history of reports that were generated by the tool.

## 2.8 Current Status

### 2.8.1 Conclusion of Work Done so Far

At the current time, our team has experimented with multiple approaches for creating this artificially intelligent tool for Collins Aerospace. We have experimented with word-embedding approaches toward natural language processing by implementing Word2Vec with Word Mover Distance, GloVe, Topic Modeling, Universal Sentence Encoder and Doc2Vec. We performed experiments based on the following approaches. As a conclusion, following approaches has failed to provide us with the expected result.

Our team has developed the Graphical User Interface based on the sketches provided by our client at Collins Aerospace. As shown in the below screenshot, our team has manage to complete the front end part of the GUI which includes main, training and the about page. However, in the later semester this Graphical User Interface is subject to change if required by our client.



Requirement Settings

Config File: conf_settings.py

Format: Excel Spreadsheet

Linking: FDOR <--- SYN, CAS, HLR

Correctness Algorithm

Word2Vec

Doc2Vec

Report Generation

Destination Directory:

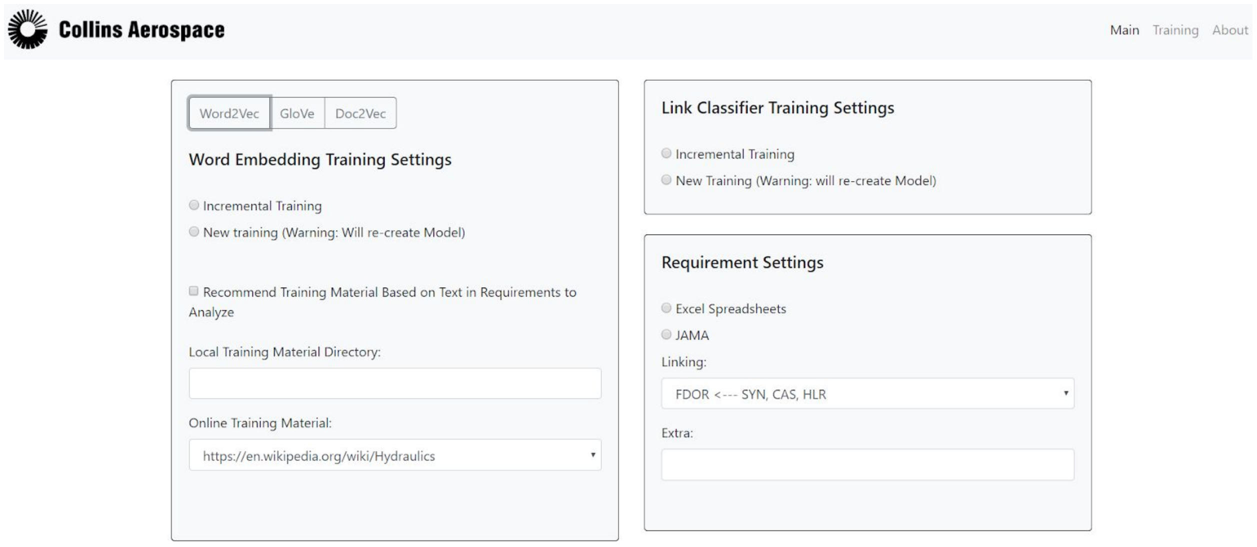
Link Recommendation Settings

Validate Derived Requirements

Identify Requirements that are Missing Test Procedures

Max Recommendations:

Figure 6: Main Page of GUI



Word Embedding Training Settings

Incremental Training

New training (Warning: Will re-create Model)

Recommend Training Material Based on Text in Requirements to Analyze

Local Training Material Directory:

Online Training Material: https://en.wikipedia.org/wiki/Hydraulics

Link Classifier Training Settings

Incremental Training

New Training (Warning: will re-create Model)

Requirement Settings

Excel Spreadsheets

JAMA

Linking: FDOR <--- SYN, CAS, HLR

Extra:

Figure 7: Training Settings Page of GUI

As the approaches above has failed to show us the expected result. The team has started to look into other approaches such as swarm intelligence and neural network for requirement traceability. In the later semester, we will be focusing on experimenting the following approaches, and moving forward to primary and secondary requirements.

## 2.8.2 Implementation Issues and Challenges

One of the main challenges we faced over the course of the semester was to obtain sufficient amount of requirement data from Collins Aerospace in order to train our machine learning models. Since many of the projects developed at Collins Aerospace are government related, their requirement data contains sensitive information and is classified to the public. We compensated for this by training these models on other sets of data, such as Google News. Once we were able to obtain sample data from Collins, our biggest challenge was the inconsistencies of formatting of the requirement data. These inconsistencies led to the data being very difficult to parse and analyze, and led to our proposal of creating a requirement hosting tool that enforced a standard format for requirements. This proposal was rejected, due to the fact that Collins had previously tried to implement a similar tool and had difficulty managing it. So, this challenge remains.  We still have struggled to find an algorithm that can accurately classify good, bad, and suspicious requirement links. Moving into the next phase of this project, we hope that our experiments with swarm intelligence and neural networks will yield more promising results than our previous experiments.

# 3 Testing and Implementation

## 3.1 Interface Specifications

The user will interact directly with one of two components to run the tool; either a command line, or a graphical user interface. The components that the user have access to will call the components on the server via an API that will be defined for communication between client and server. From there the server will communicate with the database via the standard MySQL interfacing.

## 3.2 Hardware and software

We will be using python's built-in module "unittest" for testing parser, trained mode, report generator, or any other modules written in python. We will use unittest for testing the databases as well.

A test program called Postman will be used to manually test the REST API we will develop for the UI. For automated testing, we will simply use python's "requests" module and "unittest".

For UI, we will use mocha and selenium. Mocha is for unit-testing the functions used inside the UI. Selenium is used to test the UI such as simulating the click, input data, scroll etc.

## 3.3 Functional Testing

### 3.3.1 Testing Model Accuracy

The most important testing that we will need to do is testing how accurately our model can classify links. We would like to shoot for a minimum of 70%-80% classification accuracy. If our model has accuracy under this range, then we will need to either rethink our classification model, or come up with a requirements formatting standard that will help us capture similarity more accurately.

### 3.3.2 Testing Algorithm Under Configurations

Another major area for functional testing will be testing whether the analysis process will break under any combination of settings. The tool will be able to be configured in many different ways so we need to make sure that the tool won't break down and handles any poor configuration attempts.

### 3.3.3 Testing Parser with Broken Input

We will need to test our parser and make sure that it will not break if the tool was misconfigured or the input data is improperly formatted. It is very possible that these situations occur due to user error or updates in the company processes. We need to test the parser to ensure that it will be able to respond appropriately under all circumstances.

### 3.3.4 Testing Methods

Our team decided that using the standard python testing library unittest will be sufficient to test all of our projects components. The library is both well received and trusted so we will have no conflicts in our decision of testing library.

One component will be tested at a time. Once a two communicating components are individually tested, they will be put together and tested as a whole to make sure they work as expected. This process will continue until all the components have been tested and implemented together.

### 3.3.5 Traceability Testing:

#### 3.3.5.1 Forward Traceability Test

As forward traceability test, the software needs to verify if it could map the higher level requirement only to a lower level requirement. Test cases will be created keeping in mind the example show below:

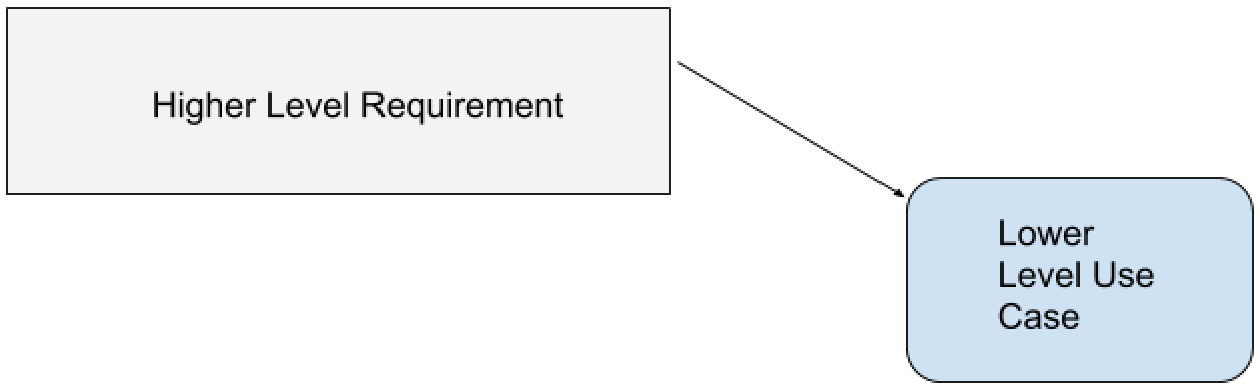


Figure 8: Test Case for Forward Traceability

Thus, it will test if the higher level is been correctly mapped with a lower level use case.

### 3.3.5.2 Backward Traceability Test

As backward traceability test, the software will need to verify if it could relate the lower level use case with its higher level requirement efficiently. Test cases needs to be created keeping in mind the example shown below:

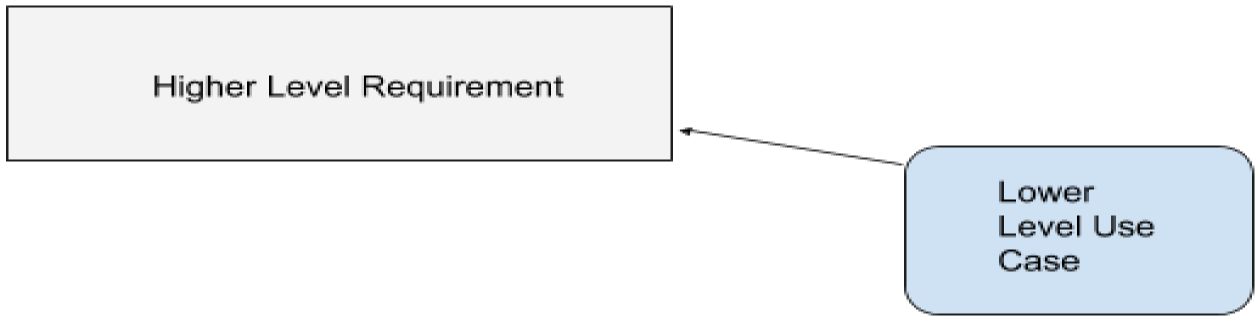


Figure 9: Test Case for Backward Traceability

Thus, a test case will be created to check if lower level use case traces back to its desired higher level requirement.

### 3.3.5.3 Bi-Directional Traceability Test

Bi-directional traceability test will test if both higher level requirement and the lower level requirement traces back to each other. Thus, test cases will be created keeping in mind the following scenario as shown below:

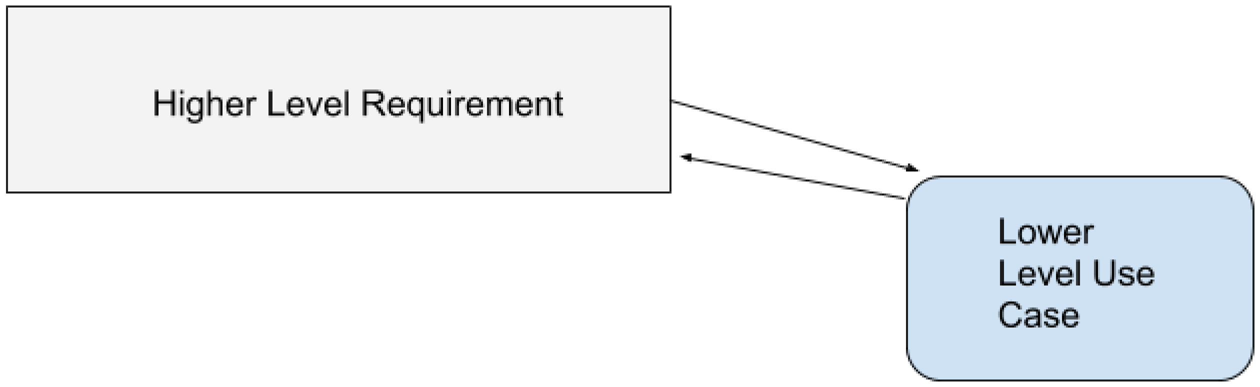


Figure 10: Test Case for Bi-Directional Traceability

Thus, a test case will be created to confirm that bi-directional traceability functionality.

### 3.3.6 Classification and Prediction Testing:

We will test the classifier by doing a series of 80/20 training/test data splits and generating confusion matrix on each run. An 80/20 training test split means that we will choose 80% of the data to train the model, chosen at random. Once the model is trained, the model will attempt to classify the rest of the 20% of data. We will monitor how the model classifies the 20% of test data and generate confusion matrix on it which gives us statistics on how the model is performing, including accuracy statistics

## 3.4 Non-Functional Testing

### 3.4.1 Security

Security is important for this tool as the data deals with information provided by U.S Defence Division through Collins Aerospace. Following are the testing that needs to be performed before the end product is handed over to Collins Aerospace:

- Vulnerability Scanning

  The software needs to be scanned for any vulnerability that may cause any security holes.

- Security Scanning

  Perform testing to identify weakness related to network and system and reduce the risk of such security holes using manual or automated system that are already available.

- Penetration Scanning

  Testing will be performed on the software to evaluate if there are any changes of malicious hacker to hack the system easily. In order to reduce the risk of such attacks, analyses needs to be performed on the system related to security.

- Posture Assessment

  Security scanning related to ethical hacking and risk assessment needs to be performed in order to verify that there are no risk related to the software that could cause loss of data as the data is highly classified.

### 3.4.2 Portability

The tool will be used by various different departments within Collins Aerospace, and therefore will need to be deployed on many machines across the organization. The software needs to be easily deployable on new machines, as well as easily configurable in order to perform its duty as per each department requirement. Finally, we must test that the tool can be executed from the command line.

### 3.4.3 Performance

Application will be forming machine learning algorithm on large number of data provided by each department of Collins Aerospace. Thus, it needs to be done as effectively as possible so that the processes are not time consuming and inefficient. Thus, performance is another important factor that needs to be initiated while designing the software.

### 3.4.4 Modifiability

As the application is going to be used by different department of Collins Aerospace. Each department has an option to implement their own machine learning algorithm. Thus, it needs be easily modifiable so that no other section of the software breaks on a new updated version of the software, and machine learning models should be able to be easily re-trained whenever necessary.

## 3.5 Process

Section 1: Training Module

- Test the accuracy of trained model using test data (we will have both train data and test data).
- Statistically show how accurate it is (e.g. xx %). Since there is a trade-off between train data accuracy and test data accuracy, choose the configuration that creates the max value of the sum of both accuracies.

Section 2: Analysis Algorithm

- Core of our application and the most important. We need to thoroughly test this module. Again, we will use python unit test.
- There are two important aspects: Accuracy and Efficiency. The accuracy can be tested by randomly choosing input with known results and making a prediction. Then check how accurate the prediction is.
- The efficiency can be tested by using the timer. Ideally this should be in a matter of seconds since developers need to use our tool very often.
- Now check to see if the algorithm raises any exceptions. The most common exception would be out-of-vocabrary exceptions. We will check this will not occur for all the data we are provided so far.

Section 3: Parser

- Given data with enough edge cases, check if the parser gives the output with desired format.
- Check if all acronyms are replaced.
- Check if all flag type codes are replaced with English texts that hold some meaning.
- We will use python unit test.

Section 4: Web User Interface

- Both manual testing and automated testing will be performed. Test for both Firefox and Chrome.
- Open a webpage with a browser
- For reports viewer, with given report key in the input box, find the corresponding report

- if the report shows up, then check if the report has all the information we need
- For configuration UI, check if the UI have all necessary inputs
- Then check if the inputs a user provides are actually sent to the server by checking the server log.
- If the input is sent to the server, check if the data is in the correct format.
- (Optional) From the security perspective, check if all forms on the UI are resilient to XSS and Infection attacks. This can be done by using Kali and the associated tools.

Section 5: Configuration Manager

- First, We will check if it contains all necessary configuration options.
- Then check if the configurations are properly stored in the database.
- It needs to endure the concurrency problems since multiple users can edit the configuration at the same time.

Section 6: Report Module

- This is an intermediate step from analysis algorithm output to the database.
- Unit test if the data is properly received
- Unit test if the output is stored properly in the database by using raw SQL *statements.*

Section 7: The whole report generating process

- This process has to be able to handle multiple requests at the same time. So input multiple data and see if it doesn't stop working.
- Anticipate the hardware failure. Occasionally store the intermediate results into the disk to mitigate hardware failure. During the testing, we will force shutdown the system and see if it recovers.

## 3.6 Results

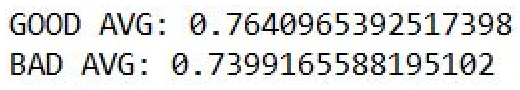
### 3.6.1 Word2vec & Word Mover Distance Approach

This approach failed us for two major reasons. First off, Collins requirement data contains several acronyms and flags. Whenever a new acronym or flag is seen, the model would need to be retrained to account for the new words. The problem is that there must be extensive data to accurately capture the context/meaning of the flag, which was not available. Secondly, requirement data at Collins is not written in such a way that the algorithm will succeed. The algorithm thrives with sentences which are rephrasings of each other, but requirement texts varied too much for the algorithm to pick up similarity regardless of whether the links were classified as good or bad.

### 3.6.2 GloVE and Cosine Similarity

This similarity comparison approach is very similar to Word2Vec (which is discussed in section 4.7.2). Though GloVe is very effective in calculating the similarity between two words, it is not nearly as effective in computing semantic similarity of sentences due to the fact that it computes similarity based solely on the similarity between words in a sentence but does not take into consideration the context in which those words are being used.

Studies comparing Word2Vec and GloVe ability to perform sentence comparisons have been conducted by Yves Piersman[5], and have concluded that Word2Vec is actually the better option when comparing sentence similarity, even though it is still not very good. The results showed that the average similarity of good links was very close to the average similarity of bad links, which would not allow us to effectively distinguish whether a link was good or bad in order to officially classify it. A screenshot of the results from an experiment we ran on a small set of requirement data can be seen below, displaying that the average similarity for good links (the total sum of the similarity / the total number of good links) and the average similarity of bad links only vary by 0.02418, making the two classifications indistinguishable from each other. After this experiment failed, we decided that we needed to take a different approach towards natural language processing, and try to find a method to compare sentences based on the topic of the content rather than strictly on the similarity of the text.



```
GOOD AVG: 0.7640965392517398
BAD AVG: 0.7399165588195102
```

Figure 2: Results from GloVe experiment

### 3.6.3 Universal Sentence Encoder

Universal Sentence Encoder showed somewhat similar result as obtained in Word2Vec with Word Mover Distance and GloVe and Cosine Similarity algorithm. It was able to separate some of the good and bad link classification. However, most of them were overlapping with each other. Thus, we decided to move on from Universal Sentence Encoder algorithm.

### 3.6.4 Topic Modeling

Topic modeling showed the highest potential so far with the accuracy of 57% on our chosen dataset. The result of this experiment implies that the significant amount of preprocessing is necessary to compare the text similarity of requirements with high accuracy. In other words, directly comparing text similarity is unlikely to work. We will continue to apply this approach on bigger datasets and see if the accuracy would increase. If that is the case, this approach could be quite reliable since the bigger data size, the higher the accuracy is. During this experiment, we learned several techniques such as filtering out all words nouns to increase prediction accuracy and these techniques could be used in other approaches as well.

# 4 Closing Material

## 4.1 Conclusion

Artificial Intelligence for Requirement Analysis Tools will be delivered successfully along with a GUI that contains a command line application as well as other visual and form-based application as asked by the client. The software will be focused on analysis of links between the platforms, domains and subdomains as per the client's requirement. Analyzing will be performed using machine learning algorithms. The software will meet the non functional requirements related to security, performance, portability, and modifiability. These project will help the company to analyze the system links, and would reduce human errors.

So far, our team has contributed their time in researching and implementing machine learning algorithms such as word2vec, word mover distance, universal sentence encoder, doc2vec and topic modeling.

Future goals include researching and experimenting on swarm intelligence, neural network and few more machine learning algorithms for requirement tracing. Apart from that we will be focusing on implementing our primary requirement for the software as well as determining the secondary requirement. We will be focusing on creating test modules in small chunks and for the primary and secondary requirement. Last but not the least, we will hand over the deliverables to the collins aerospace that contains an instructional manual, a video demonstration of the software along with the reference research and documentation.

## 4.2 References

[1]“gensim: topic modelling for humans,” *Radim ÅehÅ¯Åek: Machine learning consulting*. [Online]. Available: https://radimrehurek.com/gensim/models/word2vec.html. [Accessed: 26-Mar-2019].

[2]“gensim: topic modelling for humans,” *Radim ÅehÅ¯Åek: Machine learning consulting*. [Online]. Available: https://radimrehurek.com/gensim/models/doc2vec.html. [Accessed: 26-Mar-2019].

[3] “Confusion Matrix.” *Wikipedia*, Wikimedia Foundation, 4 Feb. 2019, en.wikipedia.org/wiki/Confusion_matrix.

[4] “Latent Dirichlet Allocation.” *Wikipedia*, Wikimedia Foundation, 24 Apr. 2019, en.wikipedia.org/wiki/Latent_Dirichlet_allocation.

[5] Peirsman, Yves. “Comparing Sentence Similarity Methods.” *NLP Town Blog | Comparing Sentence Similarity Methods*, 2 May 2018, nlp.town/blog/sentence-similarity/.

[6] “A Beginner's Guide to Word2Vec and Neural Word Embeddings.” *Skymind*, skymind.ai/wiki/word2vec.

[7] Ma, Edward. “Word Distance between Word Embeddings.” *Towards Data Science*, Towards Data Science, 25 Aug. 2018, towardsdatascience.com/word-distance-between-word-embeddings-cc3e9cf1d632.

[8] “Gensim: Topic Modelling for Humans.” *Radim ÅEhÅ¯ÅEk: Machine Learning Consulting*, radimrehurek.com/gensim/models/ldamodel.html.

[9] PyOhio. “Natural Language Processing in Python.” *YouTube*, YouTube, 29 July 2018, www.youtube.com/watch?v=xvqsFTUsOmc.

[10]G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning: with applications in R*. New York: Springer, 2017.