Computer Science Capstone

CS 486 - Requirements Engineering

SmartState

Project Title: Event-driven MachinE Learning, Intelligent Assessor (EMELIA)

Software Testing Plan

<u>Overview:</u>

The purpose of this document is to present this project at a more technical level, in addition to showcasing our software design and project architecture. This document will describe the components and modules that will define our strategy for ensuring our project passes necessary unit, integration, and acceptance testing.

Team Members:

David Rodriguez
Andrew Hurst
Reed Hayashikawa
Jianxuan Yao
Jesse Rodriguez

Clients:

Jon Lewis
Aaron Childers

Company Sponsor:

General Dynamics Mission Systems

Capstone Mentor:

Fabio Marcos De Abreu Santos

Northern Arizona University
School of Informatics, Computing, and Cyber Systems

Version: 1.0
Date: 3.29.2020

1. Introduction

General Dynamics Missions Systems is a global aerospace and defense company that develops solutions in all areas such as land, air, and cyber domains. General Dynamics develops and maintains an advanced command, control and direction-finding communications system to execute search and rescue missions. The current communications system used by General Dynamics relies on a ticket system to generate reports regarding system or hardware failures. These tickets contain data such as a hex code representing the error, a unique ID, location for where the error occurred, and many more.

Each category such as the ones previously mentioned must be classified according to the type to assist system engineers at General Dynamics. System engineers manually parse through the tickets in order to classify the type of error reported. This process is inefficient and reduces our client's ability to respond to system failures. In order to properly address the problem at General Dynamics, we will create an Event Driven MachinE Learning Intelligent Assessor (EMELIA) that will effectively classify these reports and automate this process.

In general, software testing is done to assure the user will have the optimal experience and that the application is functioning correctly. Independent components of the software are isolated and put through rigorous tests to correctly analyze its behavior and effectiveness. These tests which are noted as unit tests, analyze each individual component's functionality and correctness. Integration tests check the interaction between major components by analyzing the passage of data and locate where faults can be detected and resolved. Lastly, usability tests are performed in order to analyze if the software has a good user interface or if the user experience is optimized.

Before the software is released, each component of this application will be put through extensive tests. Many independent methods within the application will be tested using unit tests. This is confidently said for the methods that exist in the data processing component because of the amount of data that is modified and transferred through this component. Any information that undergoes any modification will be analyzed to assure the desired outcome is returned. The application is also heavily dependent on multiple components communicating in order to function; so extensive integration testing will be held in order to assure proper functionality. Lastly, usability tests are crucial in order to ensure that the engineers at General Dynamics can confidently execute our program with minimal hassle. We will invite participants who are both familiar and unfamiliar with our product to initiate various tasks within the system and to record their feedback. After obtaining their feedback, the systems interface will undergo a redesign in order to properly serve the user and to make this system more user friendly. After extensive testing and research, we are confident that the application will serve as a fully functioning prototype and is ready to be released.

2. Unit Testing

This section of the document will focus on unit testing and our team's process for ensuring that important functions are performing as we expect them to in order to ensure the quality and robustness of our code. The first section will primarily focus on reviewing what unit testing is and why it is important to include in our software. The second section will discuss the overall process for conducting the tests by describing the libraries and tools that our team is utilizing for testing. The third section will describe the test related metrics that we are determining and provide the units that we will be testing for EMELIA. For our software, we will be focusing on processing the data for the training of the learning model since our software deals with machine learning and we utilize a pipeline of functionality into the rest of the software. Unit testing should primarily focus on the tests that have the most influence on the overall behavior of our system, so ensuring that the client's data is processed correctly before it is utilized by the training model will ensure that the results of the training model are useful to our client. The last section will outline our plan for the testing of our code, describing each unit of code that we are testing, while including faulty tests to prove that each function works the way they are intended to.

2.1 Unit Testing Explained

Unit Testing is a form of testing in software engineering that is primarily focused on the individual components of our system. Unit testing is typically performed by software engineers that prepare test cases, review the performance of individual test results, and refactor the code if necessary. Unit tests are used to detect if any parts of the components are defective or do not behave as they are designed to, allowing developers to improve the quality of their code. In order for unit tests to be effective, the code must be modular so that it is easy to use and update if any problems are detected. The overall goal of unit testing is to reduce the amount of time spent by software developers fixing bugs and improve the maintainability of the code.

The result of unit testing will grant feedback on the functionality of individual components and will let the team know if the function needs to be refactored if the result is not what is expected. The team will then use the results of the tests to refactor parts of the code to ensure that the code functions properly when used in EMELIA. Unit Testing will be important for the development of EMELIA as it will allow the team to prove that our software is effective in solving the clients' problem.

2.2 Process of Unit Testing in EMELIA

We use the unit test package and coverage.py to test the single function in the main file. At the same time, we introduce "test alarm" and "test ticket" files as the parameters of the whole test function to provide them with testable values. Unittest was chosen as a test tool because it focuses on a small number of features and proves it to be right. And the test function, as a testing tool of python, uses it to test, which costs far less than the workload required by the same function as the unit test module. They will or should find the most difficult hot spots and corner situations. If they have to add some features, the first step should be to add a test to ensure that the new feature is not a working path that has not yet been inserted into the interface. Each function is independent. Since they are all fetching values from data, we set some edge values, give the length of data, as well as the maximum data and the minimum data to determine the test range. Giving the boundary range helps us to reduce the training time and reduce the probability of a wrong test. Specifically, we use the "self" method to call out each function Function to successfully test HEX value and ID.

2.3 Unit Testing Metrics

Each function has a test metric, depending on the length and size of the data they call. This measurement setting will help test accuracy and usability. However, we divide it into three parts to determine and implement the coverage accurately and ensure that the edge amount is correct. The first part, execution, first determines the size and total amount of each function call data to ensure that the set parameter is greater than or equal to this value, and then sets an empty array with the value of "null", which is convenient for us to operate the tested function, so as to avoid unexpected problems and program errors. Then, some files, such as "hex value" or "ID", which are called from the data file, are parameterized and changed into static ones. Next, the unit is used to test functions one by one. The second part, analysis, we need to analyze the test time and the return value of the test. At the beginning, we test each function one by one. The running time of each function is very short. Then we need to set a return value, such as "1", set "1" as the correct value, and "0" as the wrong value. So we can determine whether the test is successful as long as we determine whether it is "1" or "0", which will greatly reduce the time we spend on testing Between. Finally, after testing all functions without errors, you can delete the return value, which not only reduces the running time of the program, but also enhances the readability of the code. The third part, report, once we have a set of executed rows and missing rows, report is a problem of formatting this information in a useful way. Each report method (text, HTML, JSON, annotated source, XML) has a different output format, but the process is the same: write information in a specific format, possibly including the source code itself. After reading the reported questions, you can give feedback or modify the code, and then make notes, which can be used by other team members as reference notes. At the same time, the test coverage is also a very important part. We use coverage.py as a tool to measure the code coverage of Python programs. This tool monitors the entire program, pays attention to which parts of the code have been executed, analyzes the original code to determine the code that can be executed but not executed, and shows which parts of our code are executed through the test and which are not.

2.4 Unit Testing Strategy

2.4.1 Test Master Set Creation

When we retrieve the master set of Hex values and Incident Ids, we must ensure that the resulting lists satisfy certain conditions. The equivalence partitions range between the smallest Hex code and Incident Id/Ticket Id to the largest Hex code and Incident Id/Ticket Id in the client data. In order for the resulting lists to be valid, the length must range between 0 and the total number of unique hex codes and incident Ids present in the file. Given a set of all possible hex codes, the resulting lists must be equal in order to ensure that the function works correctly. Finally, a NULL value should be included in our test set to ensure that the methods correctly remove erroneous input.

2.4.2 Test Data Structure Formation

The next unit that our team will test is the methods involved in the formation of the data structure that we use to store important information from our client data. The equivalence partitions for these methods are dependent on the length of the resulting list ranging from 0 to the total number of tickets with associated hex values from the client data. Since the result of these methods is used in the next step, it is very important that the lower boundary of 0 tests for the length of the resulting list fail. Another test case for these methods is that the resulting lists must be sorted to ensure that the tickets from the client data stay in the same order at all times to prevent incorrect results from the learning model.

2.4.3 Test Encoding Functionality

The last unit that our team will test is the methods involved in one hot encoding the alarm hex values for each ticket that will be used as the input for the training model. Like the previous units, it would be problematic if the end result was an empty list as we cannot use that to train the learning model. So the equivalence partition would range from 0 to the number of tickets with associated hex values in the client's data. Another requirement is that each item in the resulting list must be a list with a length between 0 and the number of unique hex values and contain only ones or zeros. An example of an erroneous test case would be a list of lists that contain values other than one or zero and a list of lists with different lengths. These test cases will ensure that the functionality of the one-hot encoding methods are behaving as expected.

3. Integration Testing

In this section, different segments regarding this test will be outlined and organized into multiple subtopics. The first topic that will be discussed is the overall reasoning behind integration testing. This topic also includes the beneficial outcome our team will achieve by conducting this test. The following section will discuss the main goals of this test and overall approach. Lastly, the components that are being tested will be listed along with the corresponding results our team has received.

3.1 Integration Testing Explained

Integration testing is the next iterative step used to ensure proper operation and quality. This step is responsible for testing the functionality and interaction between major components. By analyzing the passage of data between components and methods, faults can be detected and resolved. As unit testing focuses on individual components and validates proper functionality of those units, integration testing verifies the correct functionality when those units are integrated together into a coherent system.

3.2 Goals of Integration Testing

In a machine learning pipeline, changes to one component can cause errors in the further components. We will implement tests that will run the entire pipeline from end-to-end. Our goal for integration testing is to ensure all essential technologies and dependencies are in order and working properly when combining different components of the project.

3.3 How it Works

Our system consists of four main modules, Data Processing, Training Model, Model Classification, and Output Metrics. These are the main components of the pipeline of our prediction model architecture. The system has four lines of communication where data is sent between modules:

- Data Processing → Training Model
- Data Processing → Prediction Model
- Training Model → Prediction Model
- Prediction Model → Output Metrics

Establishing testing of each transmission of data from one module to another with the correct range of form is necessary to ensure safe integration of the system's modules and to guarantee reliability of the software. The following sections illustrate how each situation will be addressed.

3.3.1 Data Processing to Training

The Data Processing module is used in two cases, the first, formating data to be used to train a Neural Network model object in order to predict future ticket data. In order to train, a set of features and labels must be of the same length and order to ensure the model is being trained by a ticket with its correct corresponding information.

Specifically, we will test that these sets match dimensions before passing through as parameters of the training function. Any other case will result upon failure. Subsequently, to test the second step we will run the training function with said parameters and expect success.

3.3.2 Data Processing to Prediction

The second case in which the Data Processing module is used is formatting new, unclassified tickets to be passed through the Neural Network model object to be classified. The sets of tickets to be passed through can range from one to very large. Realistically, the ticket set should not be null in order to function properly.

We will test that the ticket set passed from the Data Processing module is not null. If the set is null or empty, failure will be triggered before attempting to pass to the prediction module. Just as the training integration testing, the second step will run the prediction function with ticket parameters and expect success.

3.3.3 Training to Prediction

In order for the system to make predictions upon the given data, the systems will take the trained model that was generated using Keras high-level API and load the model into the prediction module. Being trained on an existing dataset, this model's logic and patterns will be applied to new data when forecasting the likelihood of a particular outcome. The prediction module will then take in parameters representing as input and link this input with the limited selection of output using confidence values. This is done using the model's logic and previous knowledge gained from the previous set.

In order to properly execute integration tests with these components, quality and compatibility will be evaluated to determine the efficiency of the model. Integration tests will be run when pushing new models and new versions of the software. Sudden degradation, a bug that could cause significantly lower quality in the prediction, and slow degradation, a detection in the models quality that has been lowered over various versions, will be key errors that our team will look for when performing these tests. Solutions for these two errors consist of validating the newer versions of the model by ensuring the quality is at a higher standard when compared to the previous version, and constantly updating the validations sets when training in order to ensure that the model still meets that same quality threshold.

3.3.4 Prediction to Output Metrics

This specific component in the machine learning pipeline monitors the model's accuracy and performance. After the model has made its predictions based on the training and input data, the results are then exported and written to a comma-separated values file which is done in the output metrics module. In order to test the model's accuracy, our team will revert back to the original dataset containing the input's actual classifications and check to see if the model has classified each input correctly.

Our team will then compile these results and take the average of the correctly classified predictions versus the incorrectly classified predictions. In addition, to test the model's accuracy for the purposes of this integration test of this component, multiple trial runs could be made and a metric such as the average of correct predictions over the number of trials can be calculated in order to get the true accuracy.

In order to track the system's performance and to see if these two components are compatible, our team will test the prediction steps per second for a model and track the number of input parameters answered per second. To ensure the compatibility and quality of the integration between the two components, our team will write tests to check for null and not-a-number values (NaN) which should not be present in the results.

4. Usability Testing

This section of the document will be organized into subtopics regarding usability testing and the testing strategy to be used by the development team for EMELIA. The first section will describe what usability testing is and its importance for this project. The next section will describe the main goals of usability testing and the questions that must be answered at the conclusion of this testing stage. A section regarding ‘how it works’ will provide insight regarding the usability testing process and the manner in which it is conducted. The last section of the document will be based on usability testing strategies and methods used to test the viability of EMELIA. Usability testing strategy decisions will be rationalized according to the needs of the users described by the use cases in the requirements document.

4.1 Usability Testing Explained

Usability Testing is a form of testing in software engineering that is primarily focused on user interaction with the product. The user is able to interact with a prototype of a product in order to determine if the system is usable, as described by the system requirements agreement between the development team and the client.

The result of the usability testing will often yield feedback regarding the positive and negative aspects of the current system design. The development team will then take the assessment of the system, as described by the user, and make improvements to the system.

Usability testing will be vitally important for evaluation of the performance and effectiveness of EMELIA as a machine learning classification system. The client’s ticketing system is proprietary, so the system is going to be designed specifically to accommodate the needs of the client. EMELIA will utilize usability testing to ensure the fulfillment of all functional and nonfunctional requirements, and performance once in production as a ticketing classification system.

4.2 Goals of Usability Testing

The goal of usability testing is to obtain measurable and clear feedback regarding the performance of the system. There are three main questions for usability testing that concern the development team:

- Is the system easy to use?
- Is the system flexible for the users?
- Is the system able to meet its original objectives?

Since the client/user are directly responsible for providing feedback in this stage of testing, the answers to these questions should be conclusive regarding the current state of the system and its ability to meet user needs.

The goals for EMELIA are to be accurate during ticket classification, while being an easy to use system for domain experts and system engineers. EMELIA is currently being developed as a prototype with these questions in mind. There is a specific focus to robust functionality that is tightly bundled into a single executable function. This will allow for the system to be neatly wrapped for the user, and provide flexibility regarding current or future user actions that are to be part of the system.

4.3 How It Works

The main components of conducting usability testing are as follows:

1. Invite participant to test the product
2. Welcome participant and make them feel comfortable prior to testing
3. Present the test to the participant → Follow the testing plan
4. Observe the success or failure of actions performed by the user
    a. Real observation is not just documenting success or failure of actions, it requires:
        i. Length of time needed to perform action
        ii. Ease of use regarding completion of a task such as number of command or number of clicks that are needed

iii. Built-in help instructions within the system so the user can remain independent during task completion. If they have to ask you, it needs to be made more obvious for the user

5. Analyze the results of the testing
6. Review results and make improvements to the system

Typically, usability testing requires the development team to recruit users for the product that is in development. These users should ideally be part of the user base that has been identified for the product. This is an obstacle that does not have to be addressed for EMELIA. Since the target users are restricted to the clients that we are developing the system for, we already have identified the user base for our system. This will be detailed in the Usability Testing Strategy section of the document, since this is referred to as Expert Review.

Regarding usability testing, it is important to identify key components that will be tested by a client or user. This is called the testing plan. The testing plan will be predicated on the use cases that have been identified for the system in the earlier stages of planning. This allows the development team to focus testing on the actions that are going to be used regularly by users. By focusing the testing on the key behavior of the system, the development team can apply changes directly to exposed bugs, poor design, or lack of features needed for the system to fulfill its intended purpose.

The main focus for our usability testing will be the accuracy of our system, the ease of use in order to perform actions, and the speed by which all use cases can be completed. EMELIA is a system that is designed to reduce work and error for our client’s ticketing system classification process. That is our value proposition for this project. If the user cannot accomplish each use case within a relatively short amount of time, the usability test for that use case shall be considered a failure and improvements must be made.

4.4 Usability Testing Strategy

4.4.1 Methodology

To begin discussion on the usability testing strategy for EMELIA, the usability testing methodology should be discussed first. The usability testing for EMELIA's machine learning capability will use a remote 'Expert Review' method. This method uses domain experts to test the viability of the product. Expert Review will be essential to determine the functionality of the system as was intended. Again, this is essential for our project due to its proprietary nature and its specific application to our client's ticket classification business process. This decision has been determined to yield pertinent results and provide feedback for overall system improvement. For ease-of-use testing, the team will utilize a moderated approach to gather information related to how the user reacts to a system they are unfamiliar with. This will simulate the experience of the client using the product for the first time. The goal is that if users with little or no expertise in our client's business can use the product, domain experts and system engineers will be able to use the tool with relative ease. These two methods should provide a wide coverage regarding total system capability and usability.

4.4.2 Strategy

The following section will detail the use cases for our system according to the listed functional and nonfunctional requirements. These use cases will provide direction regarding the usability tests and the actions the user should be able to perform during a single testing session.

The testing strategy is to prove that our product can perform the following for our participant:

- Minimize the time needed to learn and use the user interface
- Allow the user to perform tasks with minimal effort
- Perform safety checks that prevent errors in the system when the user performs an undesired action
- Contain a user interface with actions that are easy to recall after extended time of nonuse

- Make the user confident they can perform essential functions with the program

The section will be split according to the system's need to be reviewed by experts and non-experts alike. Usability tests on both of these user groups will be needed to reach the goals for usability testing.

Need for Remote Expert Review:

Expert review will be necessary to validate the functional requirements of EMELIA. An expert will be able to provide insight regarding the accuracy of the system and the type of failure behavior that should be expected for our client's ticketing system. Their domain knowledge will help the team determine where the product is lacking in error checking or useless processing of redundant data points. They may also provide suggestions regarding the construction of the learning model or the structure of the data that is passed as input to EMELIA. All of the technical aspects of machine learning and ticket data can be thoroughly assessed by a field expert. The use of Expert Review is an important part of our usability tests because of the impact it will have on the functional requirements of this project.

Need for Moderated Review by Non-Experts:

The need for non-experts to test the system will be necessary to validate the nonfunctional requirements of EMELIA. This will test the correct use of the following nonfunctional requirements:

- Use of the correct user interface
- Contribution to system performance by packages installed in the Conda environment
- Availability of actions that the user can perform in the system
- Flexibility offered by the system to accommodate various ticket data and alarm files
- Help options that assist the user with performing actions

A non-expert can provide general feedback regarding system use and how a new user may perceive the system. This methodology provides coverage of nonfunctional requirements in the system and can provide feedback on system design that technical experts may overlook.

The use cases to be tested by each of the users will need to be categorized as functional and nonfunctional prior to testing. It is important to note that expert reviewers must also test the nonfunctional requirements of the system. This is to ensure that the intended user base finds the interface and overall design to be efficient and reduce redundancy. Although expert reviewers will be required to test nonfunctional requirements, non-experts may not test the functional requirements of EMELIA.

The methodology and choice of usability testing participants should provide the feedback to address system deficiencies or make improvements prior to production.

5. Conclusion

In summary, we are confident that our three-part testing plan can significantly decrease the possibility of failure. Unit Testing assures the performance of low-level functionality and key methods within EMELIA. It also provides a detailed outline of how the code operates on this level with expected operation. While confident the system's source code produces what it is intended to on a modular level, it is equally important that these modules operate together; hence the value of Integration Testing. Testing of compatibility proves functionality as a holistic software and the flow of information from component to component is seamless and safe. Lastly, the final step is Usability Testing. The most valuable evaluation consists of the usability of the system by the end-users. With this step, we can demonstrate as a team we have created a well-thought-out system, not only from the perspective as engineers but also from that of the client. This provides us feedback to improve and ultimately prove that EMELIA will meet the end goal, to provide a usable, efficient, and powerful system to integrate within General Dynamics' current infrastructure.

Producing software without testing and proving functionality is essentially a useless endeavor. Outlining a software testing plan as such provides reassurance to our client and our team EMELIA will meet requirements and function to best possible capabilities. Bugs in software are virtually unavoidable but with thorough and robust testing methods we can confidently avoid failure and maximize client satisfaction.