**Validation Plan Document**

Project Title: Rules Based Decision Aid Framework

Distribution:

ASRC Federal Mission Solutions, Kimberly Davis

ASRC Federal Mission Solutions, Kevin Wainwright

ASRC Federal Mission Solutions, Christopher Barone

Rowan University, Professor Jack Myers

# 

# **Table of Contents**

Table of Contents..…………………………………………………………………..…………….2

Purpose…...…………………………………………………………………………………...…...3

Scope……………...…………………………………………………………………………...…..3

Environment……………………………………………………………………………………......4

Approach………………………………………...………………………………………………....4

Pre-requisites……………………………………………………………………………………….5

Acceptance Criteria……………………………………...………………………………………..6 Exception and Deviation Management…………………………………………..……………..8

Roles and Responsibilities………………………………………………...……………………..8

Terms and Definitions……………………………………………………………………………..9

Revision History……………………………………………………………………………...…...9

# 

# 

# 

# 

# 

# 

# **Purpose**

This document is the Validation Plan for ASRC Federal Mission Solution’s Rules Based Decision Aid Framework. The Validation plan’s goal is to describe:

● Quality Control

● IT Management

● The Wolf Team’s Testing Strategy

The Validation plan states how the Wolves development team maintains the standards in manufactured products by checking whether the product specifications capture the stakeholders’ needs. This document is also used to inform what forms of testing are being used, what sections of the product were tested, the extent of which they were tested and the environment being tested in.

# **Scope**

The scope of the RBDAF is meant to describe the extent of which the product will be tested and to ensure that the all of the specifications in the created solution conform to the original needs that were delivered by MSE. Examples of what is being tested include but are not limited to:

* The input and parsing of data; which can come from a .txt, .csv or .json format.
* The output of evaluated data
* Rule creation, and therefore the syntax of rules
* File creation and the retrieval of such files
* The activation of rules; being able to input all types of data and rule files into the KieFileSystem.
* Whole classes
* Individual Methods

**Environment**

The RBDAF is being developed and tested on computers owned by Rowan University students participating in Software Engineering in the Fall 2016 semester. The computers being used by these students include both PC’s and Macs with their respective operating systems; the software successfully runs on both. No machines are running any type of OS virtual machine. The product is currently running on hardware that should be similar to the specifications listed below:

* OS: Microsoft Windows 10 professional
* System Type: x64-based PC
* Processor: Intel Core i7-5500U CPU @ 2.40GHz, 2401 Mhz
* RAM: 8.00 GB
* 1x 1TB hard drive

The Software is being developed and tested in Java, using IntelliJ IDEA and the drools plugin. There is little detail to how MSE will be running this system/what hardware they will be using to execute it, so it is assumed that the software should be able to function on hardware with specifications as minimal as the ones listed above.

# **Approach**

This section is meant to describe the Wolf team’s approach to testing; this includes the types of tests being developed, and the extent of what will be tested. The following list shows the types of testing that are being implemented in the RBDE:

* Development Testing: This was the main form of testing used for the RBDE since it was a good way for each developer to assure that their methods and classes were functioning as intended while they were being developed.
* Unit Testing: Unit testing was necessary for the development in the RBDE. There are certain methods within the product that are not only needed to make whole classes work, but are also important to work in association with other classes.
* Object Testing: Object testing was frequently used in the development of the RBDE. In order for major components of the project to function properly, all of the methods within our classes have to work properly.

Since product functionality has been the main focus for previous sprints, product testing will now be one of the main focuses. More interface testing will be done since a lot of components use the functionality of other ones, however there is a large chance that not every pair of interacting objects will be tested. Systems testing will also be implemented as much as possible when all of our major product components are complete. It will also be a goal to make object testing more automated using JUnit tests. We will try to test as many of our classes and methods as possible. The only components that shouldn’t be tested are methods and classes that only use built in methods from an API, since it is already assumed they work.

# **Pre-requisites**

The following includes prerequisites necessary for validation to occur:

* The component being tested can be successfully compiled and built; it is a fully running component.
* The component being tested has been completed by the development team, in accordance to its acceptance criteria.
* Varying sample test data has been created based on the component being tested.
* If an automated unit test is being performed, a JUnit test class has already been created and implemented.
* If the component deals with importing data from external sources, the respective external files have already been created with data that follows the correct format.
* The developer or tester performing the tests is completely informed on how the component functions.

·

**Acceptance Criteria**

The Rules Based Decision Engine requires a large amount of acceptance criteria in order to ensure that the product being delivered meets its original requirements. The following includes the most important requirements necessary for the project and can be referenced on the Wolf team Trello board:

|  |  |  |
| --- | --- | --- |
| **PBI #** | **PBI Title** | **PBI Acceptance Criteria** |
| 48 | Import Data | -The standard format that the data will be imported as has been declared  -Data can be imported and parsed from an external text file  -Imported data can be stored in an object to use in other parts of the program  -Also see PBI 68 |
| 68 | JSON to Data Object | -The program will check the selected data file for its file extension  -The program should detect the json extension  -The program will be able to import the information from the JSON file  -That imported information can be used to create a data object |
| 40 | Create Data Object | -Prerequisite: PBI # 48 has been completed  -The imported and parsed data can be used to create a Data object  -Reference PBI #90 for changes to system since this was implemented |
| 76 | Finish Rule Class | -Have a series of enumerations that reflect operators (ex: less than, equal to, etc.). This enum goes into the constraints class.  -Create a constraint class that will go in between a set of parentheses. The constraint will consist of at least two operands and the operator enum.  -Have a condition class that will be the whole “when” part of a rule. Consists of a constraint.  -Create an action class that will be the whole “then” statement for a rule  -Have collection services for all of the above so the user can choose which parameters they want to use. |
| 82 | Make Rule Class serializable | -Prerequisite: The rule class is in a completed state.  -The rule class can be serialized into a .ser file for storage.  -a .ser file can be retrieved from the user’s file system  -The .ser file can be used to deserialize the rule class. |
| 53 | Design product GUI | -Have drawings/figures of main GUI components.  -Have the design narrowed down to basic components |
| 83 | Make File Location configurable | -Have the ability for the user to be able to search through their file system  -Have the ability for the product to use the search stated above to retrieve the filepath.  -This method should have the ability to be used for numerous classes. |
| 85 | Investigate Parenthesising | -Have the ability for the user to be able to and/or any number of constraints in the condition part of a rule  -Automatically add the correct number of parentheses in the correct placement for the criteria above.  -Also see #’s 91 and 92 |
| 81 | .drl File to Rule Object | -Have the ability to import a .drl file into the program.  -Split and convert the contents into the parameters for new Rule objects  -(See PBI #91) |
| 91 | Infix to Postix | -Have the ability to take the condition for a drl as a string, and convert its contents to postfix notation for easier object storage |
| 92 | Postfix to Infix | -Refer to 91; have the ability to take a postfix notation condition string, and convert it to infix for .drl creation. |
| 39 | Associate Data with Rule | -The program has a list of imported data  -The program has a list of rules  -Using the kiefilesystem and associated Drools components, the program can associate data and rules together. |
| 87 | Dynamically Create Rules | -The user has lists of all available components they can use to create a new rule  -The program can use these components to create a new Rule object.  -The program can convert the newly created rule object into strings for the creation of a new rule file. |
| 42 | Activate New Rules from Existing Ones | -Precondition: Have a list of already created rules  -Have a list of configurable rules  -Be able to chain new rules from already existing ones. |

# **Exception and Deviation Management**

Exception handling for the RBDE will be necessary, especially given the nature of the project. In order for the rules based decision engine to work, almost every component has to work. For example, if the user fails to select the right file or data to input in the rule activation class, the program will give an error rather than associating the two. This is why the Wolf team will complete as much error handling as possible with the limited amount of time left in the development phase.

Error handling will primarily be dealt with using custom exceptions or Java’s built in exceptions if applicable. If an exception is caught, the Wolf team will do as much as possible to keep the program from crashing. One of the main uses for the RBDE will be to check constantly updating data feeds, so downtime should be minimal. If there is user error, the program should receive their input a second time rather than just failing. As stated above, the development team will catch as many exceptions as possible, however actually finding all of the problems and correcting them might not be possible with the given remaining time.

**Roles and Responsibilities**

The table below describes the roles and responsibilities for developing, validating, and implementing a Solution:

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Responsibilities** |
| Klaydon Balicanta | Scrum Master | Enacting the scrum principles and rules; Handles any problems occurring within the group. |
| Michael Moscariello | Product Owner | The contact between the Scrum team and stakeholders. |
| Trae Lewis | Development Team | Delivering potentially releasable increments of “done” at the end of each sprint. |
| Ian Markind | Development Team | Delivering potentially releasable increments of “done” at the end of each sprint. |
| Shiv Patel | Development Team | Delivering potentially releasable increments of “done” at the end of each sprint. |
| Michael Crinite | Development Team | Delivering potentially releasable increments of “done” at the end of each sprint. |
| Professor Jack Myers | Project Manager | Works closely with the scrum master to offer the team guidance. |

# **Terms and Definitions**

|  |  |
| --- | --- |
| **Term or Acronym** | **Definition** |
| RBDAF | Rules Based Decision Aid Framework |
| RBDE | Rules Based Decision Engine |
| PBI | Product Backlog Item |
| GUI | Graphical User Interface |
| MSE | Mission Solutions Engineering |

# **Revision History**

|  |  |  |
| --- | --- | --- |
| **Version** | **Version Date** | **Revisions** |
| 1.0 | 12/2/2016 | Initial Release |