SE 3XA3: Test Report
Random Flag Generator

Team #2, Team Jakriel
Nathaniel Hu, hun4
Ganghoon Park, parkg10
Akram Hannoufa, hannoufa

April 12, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2022-04-11	1.0	Initial document creation
2022-04-12	1.1	Added sections 1, 3, 4, 5 and 6
2022-04-12	1.2	Cleaned up formatting of revision history, tables of abbreviations and definitions and document introduction
2022-04-12	1.3	Added initial draft of section 2 non-functional requirements evaluation, as well as symbolic parameters subsection and uses
2022-04-12	1.4	Added initial draft of section 7 trace to requirements
2022-04-12	1.5	Added initial draft of section 8 trace to modules
2022-04-12	2.0	Completed Test Report documentation for Revision 1 submission

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
FR	Functional Requirement
GUI	Graphical User Interface
LF	Look and Feel test
NFR	**Non-Functional Requirement**
PAGAN	**Python Avatar Generator for Absolute Nerds**
PE	Performance test
RFG	**Random Flag Generator**
RGB	Red Green Blue
SRS	Software Requirements Specification
UH	Usability and Humanity test
UI	User Interface

Table 3: **Table of Definitions**

Term	**Definition**
Gallery	Collection of previously generated flags.
Graphical User Interface	A form of UI that allows users to use electronic devices using interactive graphics.
Hashing	Algorithm that converts input data to a fixed-size value. A hashing function usually outputs a string or hexadecimal value.
Input String	The input of type string from the user.
Pytest	Python testing tool that allows testers to write test code and create simple and scalable test cases.
Python	The programming language used in this project.
Software Requirements Specification	A document that details what the program/software will do and how it will accomplish the expected performance/tasks.
System/Program	Collection of instructions or components that tell a computer how to operate.
Tester	An individual testing the software via the user interface or the code/test cases.
Test Case	a specification of inputs, execution conditions, producedure and expected results for testing a program's behaviour.
Typeform	Website that is a software as a service that specializes in creating and building online surveys.
User	Person who uses or operates a computer program.
User Interface	Where interactions between machines and humans occur.

This document details the complete testing process for Random Flag Generator, as laid out in the project test plan. It contains an evaluation of the project's functional and non-functional requirements that are defined in the **Software Requirements Specification**, the changes made due to testing, and an analysis of the traceability between requirements and modules.

1 Functional Requirements Evaluation

The following are the test cases that were evaluated for testing the functional requirements of this system.

1.1 User Interface Testing

Test #1:	**FR-01**
Description:	UI initialization
Type:	Functional, Dynamic, Manual
Initial State:	UI is uninitialized
Input:	command to initialize and run the UI
Output:	UI is initialized
Expected:	main menu UI should open and allow the user to enter text into the input string field. The main menu should also present buttons for the instructions, flag gallery and settings menus
Result:	PASS

Test #2:	**FR-02**
Description:	UI handles button clicks
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	mouse clicks on the instructions, flag gallery or settings menu buttons, or the appropriate keyboard strokes assigned to open each menu
Output:	the instructions, flag gallery or settings menu will be pulled up and can be viewed in its entirety
Expected:	From main menu UI, button clicks pull up appropriate menu
Result:	PASS

Test #3:	**FR-03**
Description:	UI handles user text input
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	keyboard strokes to (re)enter text into the input string field
Output:	text appears in the input string field
Expected:	User is able to enter and re-enter input strings and it shows up in the input string field
Result:	PASS

Test #4:	**FR-04-08**
Description:	flag is generated and saved
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	(text in input string field and) mouse click on generate flag button
Output:	flag will be generated using the input string and saved on the local machine (in generated_flags directory)
Expected:	Clicking the generate button, generates and saves the flag image
Result:	PASS

Test #5:	**FR-05-09**
Description:	flag is displayed
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running, a flag has already finished generating
Input:	(text in the input string field and) mouse click on the display flag button.
Output:	generated flag with be displayed to the user through the UI
Expected:	Clicking the display button, displays the most recently generated flag image
Result:	PASS

Test #6:	**FR-06-10**
Description:	flag settings overrides the most recent display
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running, a flag has already finished generating
Input:	(text in the input string field,) saved changes to the settings and mouse click on the display flag button
Output:	flag will be regenerated using the input string and changed settings, and saved on the local machine (in generated_flags directory)
Expected:	User selected settings overrides the most recently generated flag
Result:	PASS

Test #7:	**FR-07-11**
Description:	newly generated flag should appear in flag gallery
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	(text in the input string field and) mouse click on the generate flag button
Output:	flag will be generated using the input string and saved on the local machine (in generated_flags directory) using the input string as its (default) name
Expected:	User's newly generated flag should appear in flag gallery
Result:	PASS

Test #8:	**FR-12-13**
Description:	UI displays instructions menu
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	mouse click on the instructions button, or the appropriate keyboard stroke assigned to open the instructions menu
Output:	the instructions menu will be pulled up and can be viewed in its entirety, and a button will be displayed that the user can click on to close the instructions menu and return to the main menu
Expected:	UI displays instructions menu after clicking instructions button and can return to main menu
Result:	PASS

Test #9:	**FR-14**
Description:	UI displays settings menu
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	mouse click on the settings button, or the appropriate keyboard stroke assigned to open the settings menu, and clicks/dragging to change settings
Output:	the settings menu will be pulled up where settings can be changed
Expected:	UI displays settings menu after clicking settings button and can return to main menu
Result:	PASS

Test #10:	**FR-15**
Description:	UI displays version number
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	mouse click on the settings button, or the appropriate keyboard stroke assigned to open the settings menu
Output:	the settings menu will be pulled up where the flag generator version will be displayed
Expected:	UI displays version number on settings menu after clicking settings button and can return to main menu
Result:	PASS

Test #11:	**FR-16**
Description:	Close settings menu to view main menu
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	mouse click on the settings button, or the appropriate keyboard stroke assigned to open the settings menu
Output:	the settings menu will be pulled up where a return to main menu button will be displayed that the user can click on to close the settings menu and return to the main menu
Expected:	From the settings menu, clicking return to main menu closes settings and returns to main menu
Result:	PASS

Test #12:	**FR-17**
Description:	Open flag gallery menu from main menu
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	mouse click on the flag gallery button, or the appropriate keyboard stroke assigned to open the flag gallery menu
Output:	the flag gallery menu will be pulled up where a list of all flags and the input strings used to generate them will be displayed
Expected:	From the main menu, clicking on the flag gallery button brings up the flag gallery menu
Result:	PASS

Test #13:	**FR-19**
Description:	Return to main menu from flag gallery menu
Type:	Functional, Dynamic, Manual
Initial State:	UI is initialized and running
Input:	mouse click on the flag gallery button, or the appropriate keyboard stroke assigned to open the flag gallery menu
Output:	the flag gallery menu will be pulled up where a button will be displayed that the user can click on to close the flag gallery menu and return to the main menu
Expected:	From the settings menu, clicking return to main menu closes settings and returns to main menu
Result:	PASS

1.2 Output Testing

Test #14:	**test_get_hash_algo**
Description:	getting the correct hashing algorithm
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	input string of hashing algorithm name
Output:	hashlib hashing algorithm
Expected:	returns the correct hashing algorithm based on the given input
Result:	PASS

Test #15:	**test_get_hash_hex**
Description:	getting the correct hash digest from input
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	input string to be used to generate flag, hashlib hashing algorithm
Output:	hash digest of input string in hexadecimal form
Expected:	returns the correct hash digest in hexadecimal form based on the given input
Result:	PASS

Test #16:	**test_hash_generator**
Description:	getting the correct hash digest from input
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	input string to be used to generate flag
Output:	hash digest of input string in hexadecimal form
Expected:	returns the correct hash digest in hexadecimal form based on the given input
Result:	PASS

Test #17:	**test_pad_hashcode**
Description:	padding hash digest to minimum length
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hash string to be used to generate flag
Output:	output hash string padded to minimum hash length
Expected:	returns a modified hash digest of at least minimum length
Result:	PASS

Test #18:	**test_choose_from_list**
Description:	choose element from list
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	list of elements, float value to determine an index to be used to select an element from the list
Output:	element from the list with index larger than and closest to input float value
Expected:	returns the correct value from the list
Result:	PASS

Test #19:	**test_map_decision**
Description:	generating an float index value
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	three numerical values representing the maximum possible option, the number of possible decisions and the digit to map within the possible options
Output:	float index value to be used to decide which element to select (presumably from a list)
Expected:	returns the correct float value representing an index
Result:	PASS

Test #20:	**test_split_sequence**
Description:	split string into substrings
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	input string, integer value specifying the length of the substrings to be split from the input string
Output:	list of substrings from the input string with as many substrings of the specified length as possible
Expected:	returns a list of substrings of the corrent length
Result:	PASS

Test #21:	**test_grind_hash_for_colors**
Description:	generate colors from hash digest
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hash string to be used to generate flag
Output:	list of five colors' RGB values (R, G, B)
Expected:	returns a list of 5 correct RGB values
Result:	PASS

Test #22:	**test_grind_hash_for_base_stripe_style**
Description:	generate stripe style from hash digest
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hash string to be used to generate flag
Output:	output string of base stripe style to be used to generate flag
Expected:	return correct base striple style value
Result:	PASS

Test #23:	**test_grind_hash_for_overlay_stripe_style**
Description:	generate stripe style from hash digest
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hash string to be used to generate flag
Output:	output string of overlay stripe style to be used to generate flag
Expected:	return correct overlay striple style value
Result:	PASS

Test #24:	**test_grind_hash_for_stripe_number**
Description:	generate stripe number from hash digest
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hash string to be used to generate flag
Output:	output string of stripe number to be used to generate flag
Expected:	return correct stripe number value
Result:	PASS

Test #25:	**test_grind_hash_for_symbol_locations**
Description:	generate symbol location from hash digest
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hash string to be used to generate flag
Output:	output string of symbol location to be used to generate flag
Expected:	return correct symbol location value
Result:	PASS

Test #26:	**test_grind_hash_for_symbol_number**
Description:	generate symbol number from hash digest
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hash string to be used to generate flag
Output:	output string of symbol number to be used to generate flag
Expected:	return correct symbol number value
Result:	PASS

Test #27:	**test_grind_hash_for_symbol_types**
Description:	generate symbol type from hash digest
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hash string to be used to generate flag
Output:	output string of symbol type to be used to generate flag
Expected:	return correct symbol type value
Result:	PASS

Test #28:	**test_hex2rgb**
Description:	generate RGB value from hexadecimal input
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	output hexidecimal color code string to be used to generate flag
Output:	output tuple of RGB values for color derived from heximdecimal color code string to be used to generate flag
Expected:	return corresponding RGB tuple values based on hexadecimal input
Result:	PASS

Test #29:	**test_diff**
Description:	absolute difference of float values
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	two float values
Output:	absolute value of the difference between the two given float value
Expected:	return correct absolute difference of the two input floats
Result:	PASS

Test #30:	**test_generate_flag**
Description:	flag is generated correctly
Type:	Functional, Dynamic, Automated, Manual
Initial State:	
Input:	input string, hashing algorithm name string and dictionary of settings (i.e. chosen colours, elements)to be used to generate flag
Output:	generated flag image file
Expected:	Manual visual comparison of generated flag agrees with expected output, flag pixels match expected pixels
Result:	PASS

Test #31:	**test_generate_flag_data**
Description:	flag data is generated correctly
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	input string, hashing algorithm name string to be used to generate flag
Output:	tuple consisting of list of 5 tuples of RGB values, tuple of stripe style, number and tuple of symbol location, number and type
Expected:	All generated outputs match their expected output values
Result:	PASS

Test #32:	**test_parse_jka_file**
Description:	asset files are properly read
Type:	Functional, Dynamic, Automated
Initial State:	
Input:	input string of flag asset (.jka) file name to be parsed for pixel data
Output:	list consisting of tuples of (x, y) coordinates of the position of all filled pixels that compose the selected flag asset
Expected:	All generated pixel coordinates match their expected values
Result:	PASS

2 Nonfunctional Requirements Evaluation

The following are the test cases that were evaluated for testing the non-functional requirements of this system.

2.1 Look and Feel Requirements

Test #33:	**LF-01**
Description:	Tests that the user interface makes it easy for users to navigate the program using developer judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #34:	**LF-02**
Description:	Tests that the image quality and aesthetics of the generated flags are aesthetically pleasing using a rating survey with options on a scale of 1-10
Type:	Manual, Functional, Dynamic
Tester(s):	Testers
Pass:	Average survey score of at least Θ_{mid}
Result:	PASS with an agreement of 95%

Test #35:	**LF-03**
Description:	Tests that the colours used in the generated flag are within certain colour ranges using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 90%

Test #36:	**LF-04**
Description:	Tests that all flag components are visible in the generated flag image using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{min}
Result:	PASS with an agreement of 100%

2.2 Usability and Humanity Requirements

Test #37:	**UH-01**
Description:	Tests that user interface components are placed in a logically flowing manner using developer judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 95%

Test #38:	**UH-02**
Description:	Tests that the user must not have to jump between interfaces to accomplish a task using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #39:	**UH-03**
Description:	Tests that the program is easy-to-use for people aged MIN_AGE or older using tester judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Testers
Pass:	Average tester consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #40:	**UH-05**
Description:	Tests that user is able to select output specifications (type of hashing type of image file etc.) using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 90%

Test #41:	**UH-07**
Description:	Tests that the user is able to use the program with no prior experience using developer and tester judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Developers, Testers
Pass:	Average developer/tester consensus of at least Θ_{max}
Result:	PASS with an agreement of 90%

Test #42:	**UH-08**
Description:	Tests that the user is able to access a brief instructions blurb using tester judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Testers
Pass:	Average tester consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #43:	**UH-09**
Description:	Tests that the user is able to use the program by following the main user interface instructions only using developer and tester judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Developers, Testers
Pass:	Average developer/tester consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #44:	**UH-10**
Description:	Tests that the program uses consistent language throughout using developer judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #45:	**UH-11**
Description:	Tests that the program uses simplified terminology wherever possible using developer judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #46:	**UH-12**
Description:	Tests that the program uses easy-to-read fonts and font sizes using tester judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Testers
Pass:	Average tester consensus of at least Θ_{max}
Result:	PASS with an agreement of 95%

2.3 Performance Requirements

Test #47:	**PE-01-02**
Description:	Tests that the program minimizes the time taken to generate and process the downloading of an image using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #48:	**PE-03**
Description:	Tests that the program minimizes the time taken to load in a user's gallery (i.e. average gallery load time is below MAX_LOAD_TIME) using timing tests and developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Program loads in the user's gallery with an average load time less than MAX_LOAD_TIME and an average developer consensus of at least Θ_{max}
Result:	PASS with an average load time within MAX_LOAD_TIME and an agreement of 100%

Test #49:	**PE-04**
Description:	Tests that the different hashing systems all deliver consistent and precise outputs using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #50:	**PE-05**
Description:	Tests that the generated images all have accurately placed components (i.e. matching the templates) using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #51:	**PE-06**
Description:	Tests that the colours in the generated flag image are precise (i.e. the hexadecimal value produces the correct colour) using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #52:	**PE-07**
Description:	Tests that the program is available to run anytime in the day using developer judgement and consensus
Type:	Manual, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #53:	**PE-09**
Description:	Tests that the program limits the number of users to one per machine using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #54:	**PE-10**
Description:	Tests that the program allows for the addition of other hashing functions (relatively easily) using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #55:	**PE-11**
Description:	Tests that the program allows for the addition of other flag components (relatively easily) using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 95%

2.4 Operational and Environmental Requirements

Test #56:	**PE-13**
Description:	Tests that the program does not require an internet connection to function using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

Test #57:	**PE-14**
Description:	Tests that the program can run on any computer that can support the Python language using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

2.5 Interfacing with Adjacent Systems Requirements

Test #58:	**PE-15**
Description:	Tests that the program does not alter files outside the working directory using developer judgement and consensus
Type:	Manual, Functional, Dynamic
Tester(s):	Developers
Pass:	Average developer consensus of at least Θ_{max}
Result:	PASS with an agreement of 100%

3 Comparison to Existing Implementation

The original source project, PAGAN only had about 15 test cases spread throughout the module files, and did not do a thorough job testing the program. There was no additional documentation provided in the source code with any reference to tracing these tests to specific requirements. For Random Flag Generator multiple documents were produced that help trace between requirements and testing. These documents include: a software requirements specification, a test plan, and additional design documentation. In all, with the added traceability through the documentation, over 60 test cases were developed and performed for automated functional testing, and multiple non-functional requirements tests were also performed to ensure program robustness and correctness before finalizing the project.

4 Unit Testing

Unit testing was a large part of the testing done for Random Flag Generator. All functional modules had unit tests written and performed. Unit testing is crucial to this project, as there are many small specific functions that perform calculations or make deicisons that get used by other modules. All modules must produce the correct output to be taken in by the next module as input, to ensure a correct and complete final generated image. If at any point, any of the modules' functionality is incorrect, the final flag generation cannot be guaranteed. Additionally, given the random nature of the program(ie. decisions using hashing), it was important to make sure most if not all of the possible options were able to be selected, ensuring the widest array of flag options possible.

5 Changes Due to Testing

No major changes occurred due to the testing that took place. Small changes occured in individual modules to better handle boundary cases. For example, the Hash Generator was not able to handle certain hashing algorithm type inputs like SHAKE, as it required additional user input. A workaround was implemented for this algorithm type, when this bug arose during testing.

6 Automated Testing

All of the non-GUI functionality of the program was tested automatically using the pytest unit test framework. A suite of unit tests were developed and then ran for each functional module. The testing of the GUI was not automated, as ad-hoc user input was better for GUI testing. Additionally, image generation (and correctness) were able to be tested using pixel by pixel comparisons (done in a unit test format).

7 Trace to Requirements

Traceability matrices documenting the traceability between the test cases and the various functional and non-functional requirements of this system are shown on the next few pages.

Table 4: **Traceability Matrix for Functional Requirements I**

	Requirements													
Test Cases	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13	FR14
FR-01	X													
FR-02		X												
FR-03			X											
FR-04-08				X				X						
FR-05-09					X				X					
FR-06-10						X				X				
FR-07-11							X				X			
FR-12-13												X	X	
FR-14														X

Table 5: **Traceability Matrix for Functional Requirements II/Non-Functional Requirements I**

	Requirements											
Test Cases	FR15	FR16	FR17	FR19	LF1	LF2	LF3	LF4	UH1	UH2	UH3	UH5
FR-15	X											
FR-16		X										
FR-17			X									
FR-19				X								
LF-01					X							
LF-02						X						
LF-03							X					
LF-04								X				
UH-01									X			
UH-02										X		
UH-03											X	
UH-05												X

Table 6: **Traceability Matrix for Non-Functional Requirements II**

	Requirements											
Test Cases	UH7	UH8	UH9	UH10	UH11	UH12	PE1	PE2	PE3	PE4	PE5	PE6
UH-07	X											
UH-08		X										
UH-09			X									
UH-10				X								
UH-11					X							
UH-12						X						
PE-01-02							X	X				
PE-03									X			
PE-04										X		
PE-05											X	
PE-06												X

Table 7: **Traceability Matrix for Non-Functional Requirements III**

	Requirements						
Test Cases	PE7	PE9	PE10	PE11	PE13	PE14	PE15
PE-07	X						
PE-09		X					
PE-10			X				
PE-11				X			
PE-13					X		
PE-14						X	
PE-15							X

8 Trace to Modules

The traceability table documenting the traceability between the functional requirements and the system module(s) that implement them is shown below.

Req.	**Modules**
FR1	M7
FR2	M7
FR3	M7
FR4	M1, M2, M3, M4, M5, M6, M7
FR5	M8
FR6	M1, M2, M3, M4, M5, M6, M11
FR7	M6
FR8	M1, M2, M3, M4, M5, M6, M7
FR9	M8
FR10	M1, M2, M3, M4, M5, M6, M11
FR11	M6, M7, M9
FR12	M10
FR13	M10
FR14	M11
FR15	M11
FR16	M11
FR17	M9
FR19	M9

Table 8: Trace Between Requirements and Modules

9 Code Coverage Metrics

9.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

$MIN\_AGE = 7$
$MAX\_FIND\_TIME = 5$
$MAX\_LOAD\_TIME = 2$
$\Theta_{min} = 80$
$\Theta_{mid} = 90$
$\Theta_{max} = 100$