Capstone 4G06: System Verification and Validation Plan for yoGERT GIS Toolbox

Team 19,
Smita Singh, Abeer Alyasiri, Niyatha Rangarajan,
Moksha Srinivasan, Nicholas Lobo, Longwei Ye

New VnV Template

November 2, 2022

1 Revision History

Date	Version	Notes
November 2, 2022	1.0	Longwei: Sec 3; Moksha: Sec 6,7; Smita: Sec 6,7; Abeer: Sec 3,4,formatting; Niyatha: 5.2; Nicholas: Sec 5.2,5.3

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

symbol	**description**
SRS	Software Requirements Specification
GPS	Global Positioning Systems
GIS	Geographical Information Systems
GERT	GIS-based episode reconstruction toolkit
Point	location coordinate with time stamp.
Session	Object activity history quantified by GPS points
Episode	Session
Segment	Group of GPS Points combined based on episode attributes.
Trip	GPS points represents an object moving to a different position.
Route	Object path to get from position A to position B
Mode Detection (MD)	Detection of type of transportation being used
Time Use Diary (TUD)	Time Use Diary are records of continuous events and actions through a particular period of time (usually 24 to 48 hours)
Route Choice Analysis (RCA)	Analyzes route selection from point a to point b
.shp	.shp are geospatial data format files
CSV/.csv	Comma Separated Values is a file type that contains large amounts of data separated by commas.
Potential Activity Locations (PALS)	PALS are potential trip stops
Activity Locations (ALs)	ALs are trip stops
FR	Functional Requirement
NFR	Non-Functional Requirement

This document provides the project's verification and validation plan for documentation phase and implementation phase. Also, it will include detailed testing decisions for system test cases and unit test cases.

3 General Information

3.1 Summary

The software being verified and validated is the *yoGERT* toolbox. The software general functions include:

- Process user's GPS files into compatible data types.
- Determine choice model estimations.
- Extract travel episodes variables.
- Extract segments from travel episodes and classify segments.
- Categorize movement and stop behaviour.

3.2 Objectives

The objectives of the verification and validation plan are:

- To build confidence in the toolbox correctness. By confirming the functions are executed as expected by the requirements.
- To demonstrate the fundamental functions meet the stakeholders goals.
- To demonstrate the project scope meets the capstone deadline.
- To build confidence in toolbox accessibility and transferability.

3.3 Relevant Documentation

The test plan are created follow the documents listed below:

- Software Requirements Specification.
- Module Guide.
- Module Interface Specification.

4 Plan

This section outlines verification and validation plan including details on possible testing approaches and division of resources. The section provides rationalized decisions making process for the verification and validation plan.

4.1 Verification and Validation Team

The testing team consists of all members of *yoGERT* team. All team members need to be actively aware of each testing plan and involved in all aspects of the testing process. All team members are involved because this is a capstone project that requires students to participate in all the project's stages. Therefore, the team aims to evenly split testing and preparing for automated testing. The table below shows the division of responsibilities. It assigns leaders for different testing milestones. This way the team is sure that every testing stage is on track.

Team Member	Testing Role	Responsibilities
Abeer Alyasiri	SRS and Design Verification and Acceptance Testing	Leads document and code walkthroughs and inspections. Leads user and business acceptance testing using black box techniques.
Longwei Ye	Integration Testing	Leads regression testing using black box techniques.
Moksha Srinivasan	Unit Testing	Leads implementation of white box testing techniques.
Smita Singh	Unit Testing	Leads implementation of white box testing techniques.
Nicholas Lobo	System Testing	Leads functional requirements testing.
Niyatha Rangarajan	System Testing	Leads non-functional requirements testing.

Table 1: Verification and Validation Team.

4.2 SRS Verification Plan

The SRS verification plan consists of three parts. The main objectives is to check if the SRS document was completed according to the Volere Template Standards and if the requirements address the project goals. All parts will utilize a static testing technique that includes structured and unstructured reviews, walkthroughs or checklists.
Part one is the unstructured feedback received from classmates in the form of GitHub issues. These reviews provide technical improvements on the document from outside the team. It is helpful because it improves the document's information flow to professional individuals in the field, such as software engineers.
Part two is the structured review received from the TA. The TA follows a checklist in the form of a rubric. The feedback is beneficial because the SRS is reviewed from an industry standard perspective. Hence, the document

will be closer to implementing best-practices techniques throughout the document. This increases the productivity of using the SRS during the design stage.
Part three is a structured review with the supervisor. The review will be a combination of SRS walkthrough and modified task based inspection. The objectives of the walkthrough is to introduce the supervisor to the team's documentation and receive general feedback on the scope and clarity of the documentation. On the other hand the task based inspection will analyze both functional and non-functional requirements in depth. The inspection will consist of questions to motivate the supervisor to think about the relationship between system goals and the formulated requirements. This is helpful with removing ambiguities of the requirement's relevance to the desired final system. Also, the inspection will focus on problem categorization. These will represent the talking points of the task based inspection with the supervisor. The categorizations are clarity of requirements, conflicting requirements, and unrealistic requirements problems.
All reviews collected from the SRS verification plan will be applied to the document before the design document deliverable.

4.3 Design Verification Plan

Design verification will be similar to the SRS verification part one and part two. In addition it will include a formal review by teammates using a checklist. The checklist will consist of Dr. Spencer's MG and MIS checklists and the following points:

- Each design decision maps to one or more requirements.
- Each design specification has one output.
- Each function decomposition follow top to down design model.
- Design specification connect functional processes logically for the user to carry out tasks.
- Design specification does not include implementation details.
- Design specification describe inputs, logical operations, and output.
- Design specification outputs are consistent across a division of input cases.

- Design specification outlines error responses for unexpected behaviour.

The team will conduct the verification against the checklist using static and dynamic testing techniques. Static testing will involve a walkthrough to proof traceability and accuracy of the system architectural model. Dynamic testing will include unit testing, white box testing, black box testing, and integration testing.

4.4 Verification and Validation Plan Verification Plan

Verification and validation plan will be verified through reviews. It is important to highlight that it is difficult to proof the correctness of the test cases. Therefore, the combination multiple verification techniques induct that the verification and validation plan approximately tested critical points of the system.
First, it will be verified against Dr. Spencer's VnV-checklist by *yoGERT* team members. It will verify the completeness of the test cases. It is done by tracing at least one requirement to a test case and examining the requirement across different types of inputs.
Second, it will be reviewed by classmates in an informal way. This feedback is beneficial because it is outside the team professional opinion on what information is missing from the document.
Third, it will be reviewed by the TA in a standard way using the rubric as a checklist. The review will focus on the accuracy of information used to formulate the plan and if the plan is appropriate to the project. The plan is appropriate if it is feasible within the capstone timeline and test cases are complete within its requirement scope.

4.5 Implementation Verification Plan

The implementation verification plan includes both system test cases and unit test cases listed in this document. The verification plan is a combination of different testing techniques to start with testing the building blocks of the system up to testing structural interaction between theses components.
In the early stages of the implementation verification plan, the team will conduct static verification techniques. It includes code inspections to test code readability and code walkthroughs to verify implementation meets that design specification.

The other stages of the implementation verification plan will rely on dynamic testing techniques. These tests will be driven by white box testing and integration testing techniques. Both these techniques are focused on proving that the system follows the design specification and is consistent with the addition of new components. Also, The verification plan must encapsulate testing scenarios of how the system reacts to faulty inputs. It can be tested by inputting irrational data points and observing if a safe output will be produced instead of system failure. Therefore, it is important that the testing cases will include boundary and edge inputs to the system's safe outputs and consistent behaviour.

4.6 Automated Testing and Verification Tools

The section was done in the development plan document Sections 6 and 7.

The details of this section will likely evolve further in the project. Currently the plan is to only use automation testing for unit tests.

4.7 Software Validation Plan

Software validation plan will be divided into two parts. Part one will involve walkthrough and task based inspection with the supervisor similar to the SRS verification plan section. It will be conducted, for the same reasons from before, to flush out any problems with the SRS requirements. This standardized review will be conducted prior to implementation. Part two will involve walkthrough and demonstration to the supervisor to validate that the system behaves as the primary stakeholder expected. The formal walkthrough ensures validation of the design implementation functionality. On the other hand the demonstration validate the system's usability and response to user inputs. The supervisor will be able to provide feedback as he understand the GIS toolbox functionality and he represents a typical user for the *yoGERT* toolbox. If time permits, external data can be used for validation. The external data will be ARC GIS outputs to the same inputs fed into the *yoGERT* toolbox. The objective of this validation is to show the consistency between the *yoGERT* toolbox and the current available toolbox. This form of validation need to use external data with exact method applied to since the *yoGERT* toolbox is implementing parts of the ARC GIS application. Hence, not all outputs of the ARC GIS application are the expected outputs from the *yoGERT* toolbox.

5 System Test Description

5.1 Tests for Functional Requirements

The testing of functional requirements will be divided into two sections. One to test user functionality and one to test system functionality.

5.1.1 User Functionality Tests

This will test all functionality from the users perspectives. How data is being read will be the main focus of these tests.

User Input Testing

1. test-UT-1

   Control: Manual

   Initial State: The application has been loaded onto the computer

   Input: User loads in a CSV file of GPS data

   Output: The system saves the CSV file of GPS data

   Test Case Derivation: The user wants the application to read the given CSV file

   How test will be performed: Different sets of valid CSV data will be uploaded by the tester to see if the computer reads the values correctly

   Associated Functional Requirement: FR1

2. test-UT-2

   Control: Manual

   Initial State: The application has been loaded onto the computer

   Input: The system has a loaded file of GPS data

Output: The system saves the values found in the CSV file as latitude longitude and time variables

Test Case Derivation: The user wants to use the software to save the given CSV files into variables that can be manipulated

How test will be performed: Different sets of valid CSV's of GPS data will be uploaded by the tester to see if the computer reads the values correctly

Associated Functional Requirement: FR2,FR5

3. test-UT-3

   Control: Manual

   Initial State: The application has been loaded onto the computer

   Input: User loads in a CSV file of time use diaries (TUD)

   Output: The system saves the the the CSV file of TUD's

   Test Case Derivation: The user wants to use the software to read the given CSV file and save it

   How test will be performed: Different sets of valid CSV's of TUD data will be uploaded by the tester to see if the computer reads the values correctly

   Associated Functional Requirement: FR3

4. test-UT-4

   Control: Manual

   Initial State: A CSV of GPS data has been inputted to the application

   Input: User downloads the file that it uploaded to the system

   Output: The system gives the user the data in a CSV format

   Test Case Derivation: The user wants to use the software to read the given CSV file and save it to attributes that can be

   How test will be performed: Different sets of valid CSV's of TUD data will be uploaded by the tester to see if the computer reads the values correctly

   Associated Functional Requirement: FR4

5.1.2 System Functionality Tests

This will test all functionality from the system perspectives. How data is being process and outputted will be the main focus of these tests.

System Output Testing

1. test-ST-1

   Control: Manual

   Initial State: CSV of GPS data has been inputted to the application

   Input: The user types a function to call for the system to organize the inputted data into episodes

   Output: The system returns a report of categorized data points such as speed, duration, distance, and change in direction.

   Test Case Derivation: The system needs to be displayed in a way that the user can read easily

   How test will be performed: The tester will use a variety of CSV files filled with valid GPS data and use the function call to see if valid reports were generated

   Associated Functional Requirement: FR6

2. test-ST-2

   Control: Manual

   Initial State: A CSV of TUD data has been inputted to the application

   Input: The user types a function to call for the system to organize the inputted data into episodes

   Output: The system returns a report which contains a list of episodes that have categorized data points such as speed, duration, distance, and change in direction.

   Test Case Derivation: The system needs valid GPS points

How test will be performed: The tester will use a variety of CSV files filled with valid GPS data and use the function call to see if valid reports were generated

Associated Functional Requirement: FR7

3. test-ST-3

   Control: Manual

   Initial State: CSV of GPS data has been inputted to the application

   Input: The user types a function to call for the system to organize the inputted data into episodes

   Output: The system returns a report of episodes categorized by different methods of transportation(walk, car, bus).

   Test Case Derivation: The user wants to understand the methods of travel used from the set of data points given

   How test will be performed: The tester will use a variety of CSV files filled with valid GPS data and use the function call to see if valid categories are found in the reports generated

   Associated Functional Requirement: FR8,FR20

4. test-ST-4

   Control: Manual

   Initial State: CSV of GPS data has been inputted to the application and a report of episodes was generated

   Input: The user selects one of the episodes generated from the report

   Output: The system returns the segments of the episodes into type stop and trip

   Test Case Derivation: The user wants to understand the behaviour of the object given an episode in the report

   How test will be performed: The tester will use a variety of generated reports to see if valid episode segments were created

   Associated Functional Requirement: FR9,FR19

5. test-ST-5

   Control: Manual

   Initial State: CSV of GPS data has been inputted to the application

   Input: The report of episodes and segments are generated

   Output: The system generates the trip trajectory values based on the given segments

   Test Case Derivation: The system needs trip trajectory values for route choice analysis

   How test will be performed: The tester will validate the trajectory values based on the given CSV GPS data

   Associated Functional Requirement: FR10

6. test-ST-6

   Control: Manual

   Initial State: CSV of GPS data has been inputted to the application and the report is generated

   Input: The report of episodes and segments are generated

   Output: The system generates and stores activity locations for each of the episodes in the report

   Test Case Derivation: The system needs to generate high and low activity locations

   How test will be performed: The tester will validate a sample reports activity location matches with a curated list of episodes with known activity locations

   Associated Functional Requirement: FR11,FR16,18

7. test-ST-7

   Control: Manual

   Initial State: CSV of GPS data has been inputted to the application and the report of episodes and their segments are generated

Input: The trajectory values are calculated by the system

Output: The system creates RCA variables based on the trip trajectory

Test Case Derivation: The system needs the RCA variables to define route choice behaviour data set.

How test will be performed: The tester will generate multiple RCA datasets from different reports and check the validity of them

Associated Functional Requirement: FR12,FR13

8. test-ST-8

   Control: Manual

   Initial State: A RCA dataset has been generated by the software

   Input: The user request a route from two GPS points A and B

   Output: The system generates a mapped route from position A and position B

   Test Case Derivation: The user needs requested routes given two GPS points

   How test will be performed: The tester will request for multiple routes to be created from a generated RCA dataset

   Associated Functional Requirement: FR14,FR17

9. test-ST-9

   Control: Manual

   Initial State: A RCA dataset has been generated by the software

   Input: The user request a route from two GPS points A and B with selected constraints

   Output: The system generates a mapped route from position A and position B with selected constraints

   Test Case Derivation: The user wants customized routes based on selected constraints

How test will be performed: The tester will request for multiple routes with selected constraints be created from a generated RCA dataset

Associated Functional Requirement: R15

5.2 Tests for Nonfunctional Requirements

5.2.1 UI/UX components

The information must be presented to the user in readable format. Hence, there will be UI/UX related tests.

1. test-id1

   Type: Regression testing

   Initial State: Word document is present in the directory. Various functions to perform on the GPS data are shown to the user.

   Input/Condition: The user types a function to call for the system to organize the inputted data (word document) into episodes giving a word document as input.

   Output/Result: An error stating that the input file provided is not of the correct format.

   How test will be performed: Since, different functions require different inputs, it is important to see if the current modules functionality changes when the format of the input file changes.

   Associated NFR: 11

2. test-id2

   Type: Manual testing

   Initial State: CSV of GPS data has been inputted to the application

   Input: The user types a function to call for the system to organize the inputted data into episodes

   Output: The system returns a possible set of input data types if the function matches a stored function in the system

How test will be performed: This works like VS code, were as you type a function, a function description hovers over the function call, depciting the required user input for that function. One can try this test with different function calls to check its validity.

Associated NFR: 1, 2, 3, 4, 5, 6, 7

5.2.2 Memory and performance issues

The toolbox must work with large sets of data, hence test must consider the edge cases of data size and its relevant processing time.

1. test-id3

   Type: Regression testing

   Initial State: CSV of 47.3 million data points of GPS data has been inputted to the application

   Input/Condition: The user types a function to call for the system to organize the inputted data (word document) into episodes giving a word document as input.

   Output/Result: The system returns a report of categorized data points such as speed, duration, distance, and change in direction within 6000 seconds upon request

   How test will be performed: We perform edge case tests to see if performance and capacity requirements are met.

   Associated NFR: 9, 11, 12

2. test-id4

   Type: Unit testing

   Initial State: CSV of GPS data has been inputted to the application

   Input: The user types a function to call for the system to organize the inputted data into episodes

   Output: The system returns a possible set of input data types if the function matches a stored function in the system

How test will be performed: We perform a unit test to see if the outputted data matches the expected episodes we require from the system. This is helpful for precision requirements.

Associated NFR: 11

5.2.3 Security of user information

Since the information inputted will be used by APIs online, the system must ensure protection of user information.

1. test-id5

   Type: Manual

   Initial State: CSV of GPS data has been inputted to the application

   Input/Condition: The user types a function to call for the system to organize the inputted data into episodes

   Output/Result: No data seen at API endpoint.

   How test will be performed: We must make sure we use online APIs like pandas, geopy, etc. does not store any user inputted information.

   Associated NFR: 19

2. test-id6

   Type: Dynamic testing

   Initial State: CSV of GPS data has been inputted to the application

   Input: The user types a function to call for the system to organize the inputted data into episodes

   Output: Inputted data has not changed once the episodes are created.

   How test will be performed: Black box testing using Finite state machines. If there is no change of state for the input, then the test succeeds.

   Associated NFR: 10

5.2.4 Environment issues

For the functioning of the application, it must have certain prerequisite software like Python installed and the environment it is run on like Mac, Windows, etc. must be accounted for.

1. test-id7

   Type: Unit testing

   Initial State: CSV of GPS data has been inputted to the application

   Input/Condition: The user types a function to call for the system to organize the inputted data into episodes.

   Output/Result: Error is outputted stating that Python must be installed in the system.

   How test will be performed: Python is not installed in the system before inputting the data.

   Associated NFR: 15

2. test-id8

   Type: Unit testing

   Initial State: CSV of GPS data has been inputted to the application

   Input/Condition: The user types a function to call for the system to organize the inputted data into episodes.

   Output/Result: The system returns a report of categorized data points such as speed, duration, distance, and change in direction

   How test will be performed: Python2 is installed in the system before inputting the data. Since, Python can only be installed on a valid OS, we simultaneously test for the operational environment.

   Associated NFR: 14,15,16

5.2.5 Accessibility issues

3. test-id9

Type: Manual testing

Initial State: CSV of GPS data has been inputted to the application

Input/Condition: The user typed a function to call for the system to output the inputted data into episodes with input to specify file output.

Output/Result: The system returns a report of categorized data points such as speed, duration, distance, and change in direction in a csv file format to allow output to be saved

How test will be performed: The GPS data points are inputs and a function is ran on them with the option to open the output in csv file format.

Associated NFR: 8.

5.2.6 Scalability issues

4. test-id10

   Type: Manual testing

   Initial State: application has a pre-existing data points as inputs

   Input/Condition: The second user inputs a different set of GPS data points.

   Output/Result: The system returns a successful message of the accepted inputs and maps them as a continuation of the initial data points.

   How test will be performed: The toolbox initialized with an input then an new input is loaded on the toolbox.

   Associated NFR: 13.

5.2.7 Security issues

NFRs 17,18 are not applicable to the *yoGERT* toolbox as it was mentioned in the Hazard Analysis document. After the Hazard Analysis NFR changes are applied to the SRS document then appropriate test cases will be described here.

5.3 Traceability Between Test Cases and Requirements

	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13	FR14	FR15	FR16	FR17	FR18	FR19	FR20
test-UT-1	X				X															
test-UT-2		X																		
test-UT-3			X																	
test-UT-4				X																
test-ST-1						X														
test-ST-2							X													
test-ST-3								X												X
test-ST-4									X										X	
test-ST-5										X										
test-ST-6											X					X		X		
test-ST-7												X	X							
test-ST-8														X			X			
test-ST-9															X					

Table 2: Traceability Matrix Showing the Connections Between Functional Requirements and their test.

	NFR1	NFR2	NFR3	NFR4	NFR5	NFR6	NFR7	NFR8	NFR9	NFR10	NFR11	NFR12	NFR13	NFR14	NFR15	NFR16	NFR17	NFR18	NFR19
test-id1											X								
test-id2	X	X	X	X	X	X	X												
test-id3									X		X	X							
test-id4											X								
test-id5																			X
test-id6										X									
test-id7															X				
test-id8														X	X	X			
test-id9								X											
test-id10													X						

Table 3: Traceability Matrix Showing the Connections Between Non Functional Requirements and their test.

6 Unit Test Description

The pytest library will be used to complete unit testing for this toolbox. To develop unit tests for the internal functions of the program, we will be creating a corresponding test file for each module. Each test file will contain unit tests for each function within the module. These tests contain a variety of inputs, including those which output the correct transformation as well as inputs that generate errors and exceptions.

We will be using coverage metrics to determine how well-tested our code is. This will be determined through the use of coverage.py, a python library that quickly analyzes code coverage of all modules within a project. We will be aiming for 90% code coverage per module, ensuring that we adequately test all functions.

6.1 Unit Testing Scope

Route choice analysis variable modules will be verified for correct functionality (correct sample inputs output correct sample outputs), but logic of previously existing modules will be assessed for correctness by our supervisor, Dr. Paez.

6.2 Tests for Functional Requirements

This section will be completed once the MIS has been updated and there is greater clarity on specific modules.

6.3 Traceability Between Test Cases and Modules

This section will be completed once the MIS has been updated and there is greater clarity on specific modules.

7 Appendix

7.1 Symbolic Parameters

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

7.2 Usability Survey Questions?

This is a section that would be appropriate for some projects.

Appendix — Reflection

General Team:

To implement the verification and validation plan within our project our team will have to learn a few new skills. The team will have to create a standard testing suite and develop a standard testing method that each member of the team will follow. This will be to ensure that all tests are understandable and readable by all members of the group.

The team will also familiarize themselves with the pytest framework which will allow us to create consistent, efficient tests that will test each function in our program. As well as learn system testing techniques such as —. The team will also need to create a testing strategy that is appropriate and feasible for the project.

Smita Singh:
Smita will be responsible for creating unit test for Route Choice Analysis. She will need understand the inputs and outputs of each of the methods that will be required to perform that specific module. Smita will also be leading the creation of a test strategy.

Moksha Srinivasan:
Similar to Smita, Moksha will also be responsible for creating tests for Route Choice Analysis. Moksha will also be responsible for helping set up the standard test suite and implementing CI/CD into our git repository by following the tutorial given by Chris Shankula.

Longwei Ye:
Longwei will be responsible for creating unit testing for trip trajectory. Longwei will be responsible for learning about best testing practises through research and will ensure that the team sticks to those practises.

Niyatha Rangarajan:
Niyatha will be creating unit testing module for travel episode verification and categorization. Niyatha has been passionate about end to end system testing. She will be researching how to perform relevant system testing by researching industry standards and then be responsible for informing the rest

of the team.

Abeer Alyasiri:
Abeer will be responsible for learning about different file formats (CSV, XML, JSON, SHP) and the most efficient ways of parsing through and transforming data. This will ensure that modules are well designed and time efficient. She will ensure test cases include all relevant input file formats and malformed data inputs as well. She will learn about these best practices through the use of data parsing python tutorials online as well as researching libraries/prior implementations of open source GIS analysis tools. Through her co-op position, Abeer has significant experience working with various types of data and hence is the most suited member of our team for this task.

Nicholas Lobo:
Nicholas has always been interested in learning about data analysis and normalization. Within the scope of this project, he has taken on the responsibility of learning about GPS data standards to help provide a wide range of inputs for all tests. He will ensure that the preprocessing unit can handle various types of GPS data as well as inform decisions about edge cases related to incorrect data. He can complete this task by consulting academic papers that detail the characteristics of and how to parse GPS data.