System Verification and Validation Plan for CVT Simulator

Team #17, Baja Dynamics
Grace McKenna
Travis Wing
Cameron Dunn
Kai Arseneau

November 1, 2024

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations, and Acronyms

acronym	definition
CD	Continuous Development
CI	Continuous Integration
CVT	Continuous Variable Transmission
GUI	Graphical User Interface
IM	Instance Model
MG	Module Guide
MIS	Module Interface Specification
MSE	Mean Squared Error
NFR	Nonfunctional Requirement
PR	Pull Request
RPM	Revolutions Per Minute
SRS	Software Requirements Specification
VnV	Verification and Validation

Table 1: Verification and Validation Acronyms

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Dunn et al., 2024) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]
This document serves as a guide for how we plan to verify and validate the CVT Simulator system. Within is a detailed plan of all the tests that will be conducted to ensure that the system is functioning as intended. The document is divided into several sections going from the general plan to the specific tests that will be conducted. On the verification side, the document lists all all the tests that will be conducted to verify the performance of the system. The tests are divided into functional and nonfunctional requirements and then separate system and unit test for each. There are also multiple checklist in place to verify that all the requirements both within and outside the document are met. On the validation side, the document outlines all the plans in place to validate the results of the system. This includes how we plan to compare our results against real world data and how we plan to validate the usability of the system. Overall, this document comprehensively outlines all the plans in place to verify the system and validate its results.

2 General Information

This section serves as an overall summary of the verification and validation plan. It includes a summary of the project and the overall objectives of the VnV plan. The relevant documentation and other specifications are also listed. In summary, this section provides a high level overview of the VnV plan.

2.1 Summary

The software that will be tested is the CVT Simulator, this is a system that simulates the Continuously Variable Transmission (CVT) system of a Baja vehicle. The system has three components, a user interface which will allow the user to input the parameters of the CVT system and visualize the output of the simulation. A mathematical component that will take the input parameters and simulate the CVT system. There is also a 3D model component that will display the some models of the CVT system. Each of these components covered will be verified and validated to ensure that the system is functioning as intended.

2.2 Objectives

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: “build

confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

The objectives of testing our CVT system are to ensure the accuracy and reliability of our outputs. This includes evaluating that our mathematical model provides correct outputs based on provided input parameters specifically looking at the calculations of RPM, velocity, acceleration, distance, torque of the engine, clamping forces, CVT ratio, Torque and Belt slippage. We aim to emphasize the description over specification to make the CVT's system's behavior understandable by demonstrating how input changes influence the outputs. We will verify that our 3D model and user interface allow users to easily input parameters to the system and interpret meaningful results. Our validation tests will focus on graphical models based on current available data to describe how the outputs should behave. We will aim to quantify these results by analyzing the difference between expected versus actual output. By referencing our outputs to real world data we can validate the accuracy of our mathematical model. We will conduct useability testing to ensure that the interface is intuitive and allows users to easily input parameters, visualize results and understand the performance of the CVT.

For our project attempting to optimize our systems outputs will be considered out of scope. Additionally, we will not verify any third party libraries used in our software as we assume that this has been previously validated.

2.3 Challenge Level and Extras

The challenge level of this project is general.

The extras for this project are the following:

Validation Report

- The system will undergo validation testing to verify that accuracy of the simulation.

- The validation report will be included in the final documentation.

Usability Testing

- The system will undergo usability testing by members of the McMaster Baja team to ensure that the user interface is intuitive and easy to use.
- We will compile the results of the usability testing into a report that will be included in the final documentation.

2.4 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

Software Requirements Specification Dunn et al. (2024)
This document is relevant because it it list all the requirements that the system should have and specifies how they should be achieved. Requirements serve an important role in the verification process as the metric for determining if the system is functioning as intended. When we do the verification of the system we will compare the system's capabilities and outputs to the requirements from the SRS document. This will tell us what parts of the system are working as intended and which need to be changed or updated. For validation, the specification serves and equivalent role as the metric for determining if the system is implemented correctly. The specifications will be compared with the implementations of the system to see if they correspond. If they do not, then the system will need to be updated so the the implementations match with the specifications. Overall, the SRS document is important for verification and validation as it provides the metrics for determining their success.

Module Guide
The Module Guide will serve as a reference for the components of the system. Each module listed in the MG will have a corresponding test case in the VnV plan.

Module Interface Specification
The Module Interface Specification serves as a reference for the functions and methods of the system. Each function and method listed in the MIS will have a corresponding test case in the VnV plan.

Verification and Validation Report
The Verification and Validation Report will be referenced to ensure that all the components of the system have been verified and validated.

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

This section outlines the overall plans for the verification and validation of the system. This includes our approaches, techniques and tools that will be used. Our approaches our outlined and each person's role is specified. There is also a plan listed for both the verification and validation along with an implementation specification. Any tools that we plan to use to automate the process are also listed in this section. Overall this section details how we plan to verify and validate the system.

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

Name	Role	Responsibilities
Kai Arseneau	Data Verification Lead	Analyzes data outputs to ensure trends align with expected performance and validates consistency across simulation runs.
Travis Wing	GUI Verification Lead	Ensures user interface functionality and usability, verifying that GUI elements accurately reflect simulation data and respond as intended.
Cameron Dunn	Experimental Data Comparison Lead	Compares simulation outputs to real-world data and scripts, verifying the simulation's accuracy and adjusting model parameters as needed.
Grace McKenna	Quality Assurance Lead	Conducts comprehensive testing of simulation processes and components, identifying and documenting issues, and ensuring overall software reliability.
Ariel Wolle	Team Lead	Supports the project by overseeing resources and approving initiatives that facilitate enhanced data acquisition or other verification efforts.
Daksh Mathur	Data Acquisition Lead	Acts as the primary contact for real-world data acquisition, providing essential data for validation and supporting data needs beyond the project group.
Benjamin Waldie	Mechanical Advisor	Collaborates with the team to validate mathematical models within the simulation, ensuring mechanical accuracy and guiding technical correctness.
Duncan Shearer and Leonardo Ansari	Primary User	Utilizes the simulation tool to tune CVTs, providing feedback on tool usability and functionality to ensure it meets real-world tuning needs.
Dr. Spencer Smith	Project Advisor / Supervisor	Reviews and provides final feedback, ensuring verification meets project standards and advising on methodology improvements.

Table 2: Verification and Validation Team for CVT Simulation Project

3.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates (like your primary reviewer), or you may plan for something more rigorous/systematic. —SS]

[If you have a supervisor for the project, you shouldn't just say they will read over the SRS. You should explain your structured approach to the review. Will you have a meeting? What will you present? What questions will you ask? Will you give them instructions for a task-based inspection? Will you use your issue tracker? —SS]

[Maybe create an SRS checklist? —SS]

Do a quick intro for this section, stating that all issues with the SRS will be brought onto the github issue tracker. For all below sections, there is also a checklist that can be used as follows:

- Review the SRS document
- Check the theoretical models
- Check the data definitions
- Check the instance models

This checklist is to be used by the team as well as our supervisor.

Review Meetings

This section discusses sitting down with Ben and Ariel to review the SRS document, but specifically the theoretical models, data definitions and instance models. Alumni as well?

Practicality check on the requirements to ensure they are all feasible and measureable metrics. Assumptions check to ensure they will not have a massive impact on our simulations behaviour.

Team Wide Review

Talk here about sharing the document to the team as a whole, include a checklist and receive issues on github.

Us review

Perform unit analysis / dimension analysis inspections at each stage of mathematical models, ensuring consistency.

Traceability matrix will be verified and use to make sure requirements map logically to our goals. Further using our asumptions in this will be sure they are all valid assumptions and are used somewhere.

3.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

To ensure that the CVT system is functional and accurate, a design verification plan will be conducted on the system. The components of the system that will be tested are the graphical user interface, data output and visualization and the mathematical model. The design verification plan will include, reviews from fellow classmates, stakeholder reviews and specification testing based on the functional and nonfunctional requirements of the system. The system will receive Stakeholder review providing us with relevant feedback on the functionality and interface of our design. The system will undergo reviews from fellow classmates, providing our team with quality feedback from an outside perspective. A checklist has been designed below to which fellow classmates and stakeholders can use to verify the system.

1. For each functional requirement there is a at least one corresponding system test.

2. For each non functional requirement there is a at least one corresponding system test.

3. The system tests contain tests for boundary cases for user inputs, empty inputs and edge cases.

4. Each instance model (IM) within the SRS document shall correspond to a unit test.

5. Each function written in our mathematical python model shall correspond to at least one unit test.

6. There exists system tests for each component: user interface, mathematical model, data visualization and 3D model.

3.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

To ensure that the Verification and Validation Plan is sufficient, this document will undergo a verification plan. This plan will review the contents of the document to make sure that all necessary components are included. It will be reviewed by our team, supervisor, stakeholders and classmates. Below is a checklist to use as reference for the verification of the Verification and Validation Plan.

Overall

- The document is well organized and easy to follow
- The document is free of spelling, grammatical and formatting errors
- The document is consistent with all other documentation and implementation
- The document is complete and has all components filled out
- References are included and correct

Section 1

- List of Tables included and correct
- List of Abbreviations included and correct
- List of Acronyms included and correct
- Introduction sufficiently introduces the document

Section 2

- Roadmap effective summarizes the section
- Summary effectively summarizes the document
- Objectives are clear and concise
- Challenge Level correct and Extras are appropriate

- Relevant Documentation is matching and descriptive

Section 3

- Roadmap effective summarizes the section
- Team roles allocated properly and effectively
- SRS Verification Plan includes all necessary components in the checklist
- Design Verification Plan includes all necessary components in the checklist
- VnV Plan Verification Plan includes all necessary components in the checklist
- Implementation Verification Plan includes all techniques that are used
- Automated Testing and Verification Tools includes all tools that are used
- Software Validation Plan includes all external data that is used

Section 4

- Roadmap effective summarizes the section
- All functional requirements have corresponding system tests
- All nonfunctional requirements have corresponding system tests
- Traceability table between test cases and requirements is clear

Section 5

- Roadmap effective summarizes the section
- Unit Testing Scope is correctly defined
- All functional requirements have corresponding unit tests
- All nonfunctional requirements have corresponding unit tests
- Unit tests are properly grouped into their respective modules
- Traceability table between test cases and modules is clear and includes all modules

Section 6

- Roadmap effective summarizes the section and includes all additions
- Additional information is relevant and necessary
- Included sections are informative and properly filled out

3.5 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walkthrough. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

For the implementation verification plan, we will be using many techniques to ensure that the system is functioning as intended. We will use the unit tests listed in the this document as well as other common testing techniques. It will be required that all unit tests are run and pass for any PRs to be merged into their parent branch. We will be doing code inspections during development by ensuring that all code is reviewed by at least one other team member before merging. Static analyzers will also be used throughout development to ensure a high quality codebase and will be integrated into our CI/CD pipeline. Before major milestones, full code walkthroughs will be conducted which is enforced by all team members needing to approve merges into main.

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

We will be using a wide variety of tools for automated testing and verification. We will use separate tools for our Unity C# frontend and python backend.

For the python backend we will be using the following tools:

flake8 - A Python linter that checks for PEP8 compliance.

black - A Python code formatter that will ensure consistent code style.

unittest - Python's built-in testing framework that will be used for unit testing.

coverage - A testing framework for Python that will be used for code coverage.

For the Unity C# frontend we will be using the following tools:

SonarLint - C# linter that checks for code quality and security vulnerabilities.

StyleCop - C# linter that checks for code style and formatting.

UTF - A testing framework for C# that will be used for unit testing.

UTR - A testing framework for Unity that will be used for unit testing and code coverage.

Additionally, we will be using GitHub Actions to automate our testing process. This will be done to ensure CI/CD and that our tests are run automatically on every push to the repository.

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

3.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]
We plan on using data collected from the McMaster Baja team to validate the accuracy of our simulation. Some of the values we are able to collect include:

- Primary RPM
- Secondary RPM
- CVT Ratio (from Primary and Secondary RPM)
- Wheel Speed
- GPS Speed
- GPS Position
- GPS Time
- Acceleration (from Speed and Time)

To validate this data against our simulation we plan on creating a script to compare them. This script will compute the MSE between the two datasets and also plot that error over time. Plotting the error will allow us to see in what scenarios the simulation is least accurate. This will allow us to better diagnose the issues and allocate our resources to fix them.

4 System Tests

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

Standard Simulation Inputs

The standard simulation inputs for the CVT system are as follows.

1. Primary Pulley System:
    - Flyweight
    - Ramp Geometry
    - Spring rate
    - Spring pretension
2. Secondary Pulley System:
    - Helix Geometry
    - Torsional spring rate
    - Compressional spring rate
    - Spring pretension
3. Vehicle Characteristics:
    - Car weight
    - Driver weight
    - Traction
    - Angle of incline

Standard Simulation State

The standard simulation state for the CVT system is as follows.

- Vehicle is stationary.
- The engine's angular velocity is at ω_{idle}.
- The CVT system is in a neutral state, unshifted.
- The belt is stationary.

4.1.1 Vehicle Dynamics

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Position

1. test-1

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: The input values from Section 4.1.

   Output: The resulting graph should show a position that is a monotonically increasing function over time, starting from 0. There is no upper bound. A typical output graph will look similar to the one shown in Figure **??**.

   Test Case Derivation: As long as the vehicle is able to overcome the initial force of the slope, the vehicle should continue to move forward indefinitely.

   How test will be performed: The script will run with the provided inputs. After execution is complete, the output position over time will be visualized as a graph which should show a smooth upward trend as time progresses. Its slope should increase over time as well. Visual inspection will follow against Figure **??** for confirmation of expected behaviour.

2. test-2

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: Provided experimental data along with the input values from Section 4.1, matching those of the vehicle during experimental data collection. Sample experimental data is located here: Latitude.csv, Longitude.csv.

   Output: Mean squared error between the simulated position and the experimental position.

   Test Case Derivation: Data from the simulation should closely match the experimental data collected from the physical car.

How test will be performed: The script will run with the provided inputs. After execution is complete, the output of the simulation will be compared to experimental data from the physical car to ensure that the simulation is accurate. The experimental data will be configured to show the absolute distance travel since the start of the experiment.

Velocity

1. test-1

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: The input values from Section 4.1.

   Output: The resulting graph should show a velocity that is a monotonically increasing function over time, starting from zero and remaining below $v_{\max}$, which reflects the vehicle's physical limits. A typical output graph will look similar to the one shown in Figure **??**.

   Test Case Derivation: Based on assumption REF HERE that the vehicle should not move backward under normal conditions. Derived from the vehicle's geometry and the CVT configuration, limiting the velocity to a defined top speed based on the system's constraints.

   How test will be performed: The script will run with the provided inputs. After execution is complete, the output velocity over time will be visualized as a graph which should show a smooth upward trend that begins at zero and approaches $v_{\max}$ as time progresses. Visual inspection will follow against Figure **??** for confirmation of expected behaviour.

2. test-2

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: Provided experimental data along with the input values from Section 4.1, matching those of the vehicle during experimental data collection. Sample experimental data is located here: Speed.csv.

   Output: Mean squared error between the simulated velocity and the experimental velocity.

Test Case Derivation: Data from the simulation should closely match the experimental data collected from the physical car.

How test will be performed: The script will run with the provided inputs. After execution is complete, the output of the simulation will be compared to experimental data from the physical car to ensure that the simulation is accurate.

Acceleration

1. test-1

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: The input values from Section 4.1.

   Output: The resulting graph should show an acceleration that is a smooth monotonically decreasing function over time, starting from $a_{\max}$ and remaining above $a_{\min}$, which reflects the engine's limits along with the maximum and minimum transmission ratio. A typical output graph will look similar to the one shown in Figure **??**.

   Test Case Derivation: Based on the engine's output limits, going no lower than ω_{idle} and no higher than $\omega_{\max}$. Based on air resistance, which increases with the square of velocity, alongside the diminishing torque output of the engine.

   How test will be performed: The script will run with the provided inputs. After execution is complete, the output will be visualized as a graph. Visual inspection will follow against Figure **??** for confirmation of expected behaviour.

2. test-2

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: Provided experimental data along with the input values from Section 4.1, matching those of the vehicle during experimental data collection. Sample experimental data is located here: Accel X.csv, Accel Y.csv, Accel Z.csv.

Output: Mean squared error between the simulated acceleration and the experimental acceleration.

Test Case Derivation: Data from the simulation should closely match the experimental data collected from the physical car.

How test will be performed: The script will run with the provided inputs. After execution is complete, the output of the simulation will be compared to experimental data from the physical car to ensure that the simulation is accurate. The experimental data will be configured to display the acceleration in the direction of travel of the vehicle.

4.1.2 CVT Dynamics

Clamping forces

1. test-1

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: The input values from Section 4.1.

   Output: The resulting graphs should show a clamping force against engine angular velocity beginning from $f_{\text{clamp\_min}}$, which should then rise to a peak before falling during the shifting phase. Suboptimal parameters may lead to a clamping force that is too low or too high, but a similar shape nonetheless. A typical output graph will look similar to the one shown in Figure **??**.

   Test Case Derivation: This behaviour is resulting from ideal tuning parameters, and may vary depending on inputs. Ramp and helix geometry will both have a major impact on the shape of these graphs, while weight and spring constants will affect the magnitude.

   How test will be performed: The script will run with the provided inputs. After execution is complete, the output will be visualized as a graph. Visual inspection will follow against Figure **??** for confirmation of expected behaviour.

Shift

1. test-1

Control: Manual

Initial State: The state values from Section 4.1.

Input: The input values from Section 4.1.

Output: The resulting graphs should show the shift distance against the engine's angular velocity. An ideal graph will look very similar to the leading edge of a square wave. A typical output graph will look similar to the one shown in Figure **??**.

Test Case Derivation: This behaviour is resulting from the CVT's clamping forces overcoming the resistive forces just enough such that the CVT begins shifting. Once this has begun, ramp geometry is expected to cause shifting rather than the engine increasing in speed, maintaining a constant power output. With suboptimal tuning parameters passed in, one might observe a slanted shifting portion.

How test will be performed: The script will run with the provided inputs. After execution is complete, the output will be visualized as a graph. Visual inspection will follow against Figure **??** for confirmation of expected behaviour.

2. test-2

Control: Manual

Initial State: The state values from Section 4.1.

Input: Provided experimental data along with the input values from Section 4.1, matching those of the vehicle during experimental data collection. Sample experimental data is located here: Primary RPM.csv, Secondary RPM.csv

Output: Mean squared error between the simulated shift against engine velocity compared to experimental data. Differences of shape and magnitude will be considered.

Test Case Derivation: Data from the simulation should closely match the experimental data collected from the physical car. Key points of interest include the shift point and the slope during the shifting phase.

How test will be performed: The script will run with the provided inputs. After execution is complete, the output of the simulation will be compared to experimental data from the physical car to ensure that the simulation is accurate. The experimental data will be configured to show the primary RPM

divided by the secondary RPM on the y-axis, and the primary RPM on the x-axis.

1. test-1

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: The input values from Section 4.1.

   Output: The resulting graph show show the engine's angular velocity against the vehicles velocity. The output should show a smooth trend upwards starting at ω_{idle} until reaching the shift point, in which it then moves horizontally until finally reaching full shift and bringing the engine's angular velocity to ω_{max}. A typical output graph will look similar to the one shown in Figure **??**.

   Test Case Derivation: This behaviour results from the CVT's shifting behaviour as a whole. An in-depth explanation of the phases can be found beneath Figure **??**.

   How test will be performed: The script will run with the provided inputs. After execution is complete, the output will be visualized as a graph. Visual inspection will follow against Figure **??** for confirmation of expected behaviour.

2. test-2

   Control: Manual

   Initial State: The state values from Section 4.1.

   Input: Provided experimental data along with the input values from Section 4.1, matching those of the vehicle during experimental data collection. Sample experimental data is located here: Primary RPM.csv, Secondary RPM.csv

   Output: Mean squared error between the simulated shift curve (see Figure **??**) and the experimental data. Differences of key points, shape and magnitude will be considered.

   Test Case Derivation: Data from the simulation should closely match the experimental data collected from the physical car. Key points include the shift point, slope during shifting phase, engagement point and the final shift point.

   How test will be performed: The script will run with the provided inputs. After execution is complete, the output of the simulation will be compared

to experimental data from the physical car to ensure that the simulation is accurate. The experimental data will be configured to show the primary RPM on the y-axis and the secondary RPM on the x-axis.

4.1.3 User Interface

1. test-1
   Initial State: Default input parameters are loaded

   Input: Change the input parameters with new values

   Output: Input parameters have been updated

   Test Case Derivation: Based on FR10 and FR11, the user should be able to adjust each of the input parameters (within the limitations of the input itself). This includes CVT parameters such as weight, ramp geometry, spring rate and spring pretension as well as other parameters such as vehicle weight, driver weight, traction and angle of incline.

   How test will be performed: Upon a fresh launch of the application, manually change each of the input parameters from the default values to a new value.

2. test-2

   Control: Automatic

   Initial State: Application has not been launched

   Input: Launch application

   Output: User interface for changing the input parameters is displayed

   Test Case Derivation: Based on FR12 validate that the system provides the user with a UI to change the input parameters.

   How test will be performed: Upon a fresh launch of the software, the system will assert whether the input parameter page is displayed or not.

3. test-3

   Control: Automatic

   Initial State: Simulation pre loaded with input parameters

   Input: Run simulation

   Output: Graphical representation of simulation data is displayed to the user

Test Case Derivation: Based on FR13, validate that the simulation produces output in the form of graphs.

How test will be performed: The test will run the simulation with a set of pre made input parameters, then it will verify that whether or not a graph has been outputted.

4. test-4

   Control:

   Initial State:

   Input:

   Output:

   Test Case Derivation:

   How test will be performed:

5. test-5

   Control: Manual

   Initial State: Simulation has completed

   Input: Select export data option

   Output: simulation data exported successfully

   Test Case Derivation: Based on the FR15 validate user can export simulation data after simulation has completed.

   How test will be performed: After simulation has completed successfully export the data and verify that it was exported in the correct format to the right location.

6. test-6

   Control: Manual

   Initial State: Application is not installed

   Input: Install application files

   Output: Applications installs and functions as expected

Test Case Derivation: Based on FR16 and FR17, validate that the application can be installed and ran on different operating systems and types of computers.

How test will be performed: This test will be performed on two types of computers, a personal laptop and a desktop. On each of those types of computers it will be ran on each of the operating systems, Windows, Linux and MacOS to verify compatability with each OS and type of computer.

...

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

The nonfunctional requirements we will be testing for are accuracy, useability, maintainability, verifiability, understandability and reusability. Referencing the SRS document Nonfunctional Requirement 1 (NFR1: Accuracy) and Nonfunctional Requirement 4 (NFR4: Verifiability) will reference (LINK TO TEST ABOVE). The Nonfunctional requirements related to useability and understandability (NFR2 and NFR 5 respectively) will reference the useability/understandability survey linked in the appendix.

4.2.1 Accuracy

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2.2 Useability

1. test-id1

   Type: Manual

   Initial State:

   Input/Condition: Users within the Primary User role as well as Baja team members are asked to rate how simple the navigation process of the main interface. They are asked to rate this on a scale of (1-5) 1 being extremely difficult and 5 being extremely easy with the other options being 4: somewhat easy, 3: neutral and 2: somewhat difficult.

   Output/Result: The average output rating from all users is greater than or equal to a 4(somewhat easy or above expectations).

   How test will be performed: Each user in the test group will be provided with a survey which provides a series of questions and a scale for each option where 1 represents Poor, 2 represents below expectation, 3 represents satisfactory, 4 represents above average and 5 represents excellent. The average rating will then be calculated and must be above or equal to 4 for each criteria.

2. test-id2

   Type: Manual

   Initial State:

   Input/Condition: Users within the Primary User role as well as Baja team members are asked to rate the features inputting parameters, adjusting parameters, viewing data outputs and saving and exporting data on how easy it was to use each feature. They are asked to rate this on a scale of (1-5) 1 being extremely difficult and 5 being extremely easy with the other options being 4: somewhat easy, 3: neutral and 2: somewhat difficult.

   Output/Result: The average output rating from all users for each listed feature is greater than or equal to a 4(somewhat easy or above expectations).

   How test will be performed: Each user in the test group will be provided with a survey which provides a series of questions and a scale for each option where 1 represents Poor, 2 represents below expectation, 3 represents satisfactory, 4 represents above average and 5 represents excellent. The average rating will then be calculated and must be above or equal to 4 for each criteria.

4.2.3 Maintainability

1. test-id1

   Type: Manual

   Initial State: Completed system which includes, the finalized mathematical model, GUI and 3D model.

   Input/Condition: The 2023 McMaster Baja CVT's configuration.

   Output/Result: The amount of time taken and number of lines of code modified using the 2023 CVT.

   How test will be performed: The test will be manually completed by the teams Quality Assurance Lead, where the amount of time and number of lines of code will be recorded when implementing the changes to correspond to the 2023 CVT.

4.2.4 Verifiability

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

4.2.5 Understandability

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition: Users within the Primary User role as well as Baja team members are asked to rate how clear they found the features and functions within the system. They are asked to rate this on a scale of (1-5) 1 being extremely unclear and 5 being extremely clear with the other options being 4: somewhat clear, 3: neutral and 2: somewhat unclear.

   Output/Result: The average output rating from all users for each listed feature is greater than or equal to a 4(somewhat clear or above expectations).

   How test will be performed: Each user in the test group will be provided with a survey which provides a series of questions and a scale for each option where 1 represents Poor, 2 represents below expectation, 3 represents satisfactory, 4 represents above average and 5 represents excellent. The average rating will then be calculated and must be above or equal to 4 for each criteria.

2. test-id2

   Type: Manual

   Initial State:

   Input/Condition: Users within the Primary User role as well as Baja team members are asked to rate their understanding of the simulation results and

outputs. They are asked to rate this on a scale of (1-5) 1 being extremely unclear and 5 being extremely clear with the other options being 4: somewhat clear, 3: neutral and 2: somewhat unclear.

Output/Result: The average output rating from all users for each listed feature is greater than or equal to a 4(somewhat clear or above expectations).

How test will be performed: Each user in the test group will be provided with a survey which provides a series of questions and a scale for each option where 1 represents Poor, 2 represents below expectation, 3 represents satisfactory, 4 represents above average and 5 represents excellent. The average rating will then be calculated and must be above or equal to 4 for each criteria.

4.2.6 Reusability

1. test-id1

   Type: Manual

   Initial State: Completed system which includes, the finalized mathematical model, GUI and 3D model.

   Input/Condition: The 2023 McMaster Baja CVT's configuration.

   Output/Result: The amount of time taken and number of lines of code modified using the 2023 CVT.

   How test will be performed: The test will be manually completed by the teams Quality Assurance Lead, where the amount of time and number of lines of code will be recorded when implementing the changes to correspond to the 2023 CVT. This will indicate how long it will take for the system to be adapted to future CVT configurations.

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Cameron Dunn, Grace McKenna, Kai Arseneau, and Travis Wing. System requirements specification. https://github.com/gr812b/CVT-Simulator/blob/develop/docs/SRS/SRS.pdf, 2024.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

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

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?