Module Guide for CVT Simulator

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

January 14, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
CVT Simulator	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The format and structure of the initial input data.

AC3: The engine simulator implementation could be updated or replaced as new models or technologies become available.

AC4: External forces considered in the simulation could vary, depending on changes in design parameters or environmental considerations.

AC5: The CVT simulator implementation is likely change during the validation and verification (VnV) process.

AC6: The visualizations used and their implementation could be updated to improve the user experience or to accommodate new features.

AC7: Constants used in the software could be updated based on changes to the car's physical design.

AC8: Input parameters for the system could change to include more parameters based on new requirements or to improve the accuracy of the simulation.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: Constraint parameters in our model/system will remain fixed and will not change.

UC3: The simulation will always be an acceleration run, i.e the car will always start from rest and accelerate.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Engine Simulator Module

M3: External Forces Module

M4: CVT Simulation Module

M5: Input Module

M6: ODE Solver Module

M7: Main Module

M8: Playback Module

M9: Visualizer Module

M10: Constants Module

M11: State Module

M12: Backend Controller Module

M13: GUI Module

M14: File Output Module

M15: Communication Module

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Engine Simulator Module
	External Forces Module
	CVT Simulation Module
	Input Module
	ODE Solver Module
	Main Module
	Playback Module
	Visualizer Module
	Constants Module
	State Module
	Backend Controller Module
Software Decision Module	GUI Module
	File Output Module
	Communication Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. —SS]

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the

module is provided by the operating system or by standard programming language libraries. *CVT Simulator* means the module will be implemented by the CVT Simulator software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 Input Format Module (M5)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: CVT Simulator

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.2 Engine Simulator Module (M2)

Secrets: The relationship between torque, RPM, and load in the engine's performance curve.

Services: Simulates engine behavior by generating torque based on input RPM, throttle position, and load conditions.

Implemented By: CVT Simulator

Type of Module: Abstract Object

7.2.3 External Forces Module (M3)

Secrets: The equations for calculating gravity, air resistance, and rolling resistance forces acting on the vehicle.

Services: Computes the net external forces acting on the vehicle for given conditions like incline, speed, and drag coefficient.

Implemented By: CVT Simulator

Type of Module: Abstract Object

7.2.4 CVT Simulation Module (M4)

Secrets: The dynamics of the CVT, including the relationship between clamping forces, belt ratios, and load transfer.

Services: Simulates the CVT's behavior under changing conditions by calculating primary and secondary clamping forces and belt ratios.

Implemented By: CVT Simulator

Type of Module: Abstract Object

7.2.5 ODE Solver Module (M6)

Secrets: The numerical methods and parameters used by SciPy to solve ordinary differential equations.

Services: Solves the system of differential equations governing the CVT system's dynamics over time.

Implemented By: SciPy

Type of Module: Library

7.2.6 Main Module (M7)

Secrets: The orchestration of various modules to achieve the simulation flow.

Services: Manages the simulation's execution by calling the ODE solver, handling inputs/outputs, and coordinating between modules.

Implemented By: CVT Simulator

Type of Module: Abstract Object

7.2.7 Playback Module (M8)

Secrets: The format of simulation data and its mapping to Unity playback mechanisms.

Services: Provides playback functionality by reading simulation data and rendering it in Unity.

Implemented By: CVT Simulator

Type of Module: Abstract Object

7.2.8 Visualizer Module (M9)

Secrets: The graphical representation and rendering techniques used in Unity.

Services: Visualizes simulation results through charts, graphs, and 3D models, integrating data from various modules.

Implemented By: Unity

Type of Module: Abstract Object

7.2.9 Constants Module (M10)

Secrets: The storage and retrieval of constant values like engine torque curves, car weight, and drag coefficients.

Services: Provides a centralized location for retrieving predefined constant values used across the simulation.

Implemented By: CVT Simulator

Type of Module: Library

7.2.10 State Module (M11)

Secrets:

Services:

Implemented By: CVT Simulator

Type of Module: Record

7.2.11 State Module (M12)

Secrets:

Services:

Implemented By: CVT Simulator

Type of Module: Abstract Object

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 GUI Module (M13)

Secrets: The graphical representation techniques and rendering optimizations used in Unity.

Services: Displays simulation outputs to the user, enables interaction (e.g., input adjustments, playback controls), and visualizes simulation results in 3D and 2D formats.

Implemented By: CVT Simulator

Type of Module: Abstract Object

7.3.2 File Output Module (M14)

Secrets: The format of the output data and the methods used for file serialization.

Services: Saves simulation results (e.g., graphs, raw data) into user-friendly formats like CSV or JSON for external use and analysis.

Implemented By: CVT Simulator

Type of Module: Library

7.3.3 Communication Module (M15)

Secrets: The protocol and serialization methods used for data exchange between Unity and the Python backend.

Services: Facilitates data transfer between the Unity-based GUI and the Python-based simulation modules, ensuring synchronization and integrity of inputs/outputs.

Implemented By: CVT Simulator

Type of Module: Abstract Object

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M4, M6, M10
R2	M3, M10
R3	M3, M10
R4	M3, M10
R5	M4, M10
R6	M4, M10
R7	M4, M10
R8	M4, M10
R9	M2, M10
R10	M13, M5
R11	M13, M5
R12	M13, M5
R13	M9, M8
R14	M9, M8
R15	M9, M8
R16	M1
R17	M1

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2AC8	M5
AC3	M2
AC4	M3
AC5	M4
AC6	M9
AC7	M10

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

Figure 1: Use hierarchy among modules

10 User Interfaces

Link to Figma: https://www.figma.com/design/REbQeg2EuDxgU07C6aS5Os/Capstone?node-id=0-1&t=0Ay7pkvwNLkEgVxE-1

Note that the screenshots of each interface can be seen in the Appendix of this document.

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS]

11 Design of Communication Protocols

There are two main communication protocols used in the system. One is for going from Unity to Python and the other is for going from Python to Unity. Outside of these two instances, the other modules communication is handled through standard Unity or Python internal communication.

11.1 Unity to Python

The Unity from Python communication is handled through passing parameters. Unity opens a terminal which calls the main Python script. This call also includes all of the parameters that are specified by the user. These parameters are then received by the Python script and are used to run the simulation.

11.2 Python to Unity

The Python to Unity communication is handled through a CSV file. After the simulation is ran, the Python script writes the results to a CSV file. Unity then reads from this CSV file and gets the results of the simulation. This data is used to display the results of the simulation to the user.

12 Timeline

Rev 0 of the product is due by February 3rd, 2025. To ensure adequate time for any last-minute adjustments or unforeseen issues, the goal is to complete all tasks by February 1st, 2025.

Backend and Mathematical Model

The backend and mathematical model are primarily the responsibility of Kai, with support from Cameron. The team aims to complete this work by January 27th, 2025. This includes finalizing the CVT mathematical modules, which are critical to the product. Additionally, the engine and external forces modules, which are already complete, will be integrated into the system.

Initial validation and verification will also be conducted. These checks will include ensuring that values increase monotonically where expected, verifying the realism of generated values within the context of the problem, and performing quick automated tests that do not rely on realistic data. These steps will help ensure the robustness of the backend before integration.

Unity and User Interface (UI)

The Unity and user interface components will be developed collaboratively by Travis and Grace. The deadline for this section is also January 27th, 2025. The team will work to build the UI prototype based on the Figma design provided in Section 10 of the documentation. This prototype will handle user input effectively and include visualizations such as 3D models, graphs, gauges, and other essential elements. These features are intended to provide an intuitive and visually engaging user experience.

Connecting the Components

The integration of the backend and mathematical model with the Unity and UI components will involve all team members and will be completed by February 1st, 2025. This phase will ensure that the backend models connect seamlessly with their respective visualizations. Full playback functionality will also be implemented, allowing the system to operate as intended.

Team-wide testing will take place during this phase to identify and resolve any integration issues. Clear communication and collaboration between sub-teams will be crucial to ensure that all components function harmoniously.

Additional Notes

All updates and task tracking will be managed through the project GitHub repository. The repository can be accessed at https://github.com/gr812b/CVT-Simulator/issues. Between January 27th and February 1st, 2025, effective communication between sub-teams will be vital to ensure the integration process proceeds smoothly. Team members are encouraged to check in regularly to verify that progress aligns with the established timeline.

[Schedule of tasks and who is responsible —SS]

[You can point to GitHub if this information is included there —SS]

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

Appendix

Figure 2: Home Page

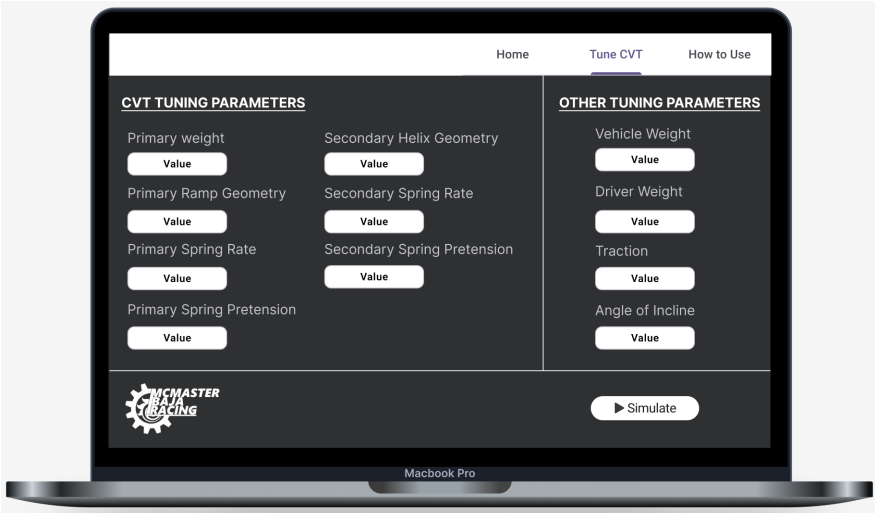

Figure 3: Inputs Page

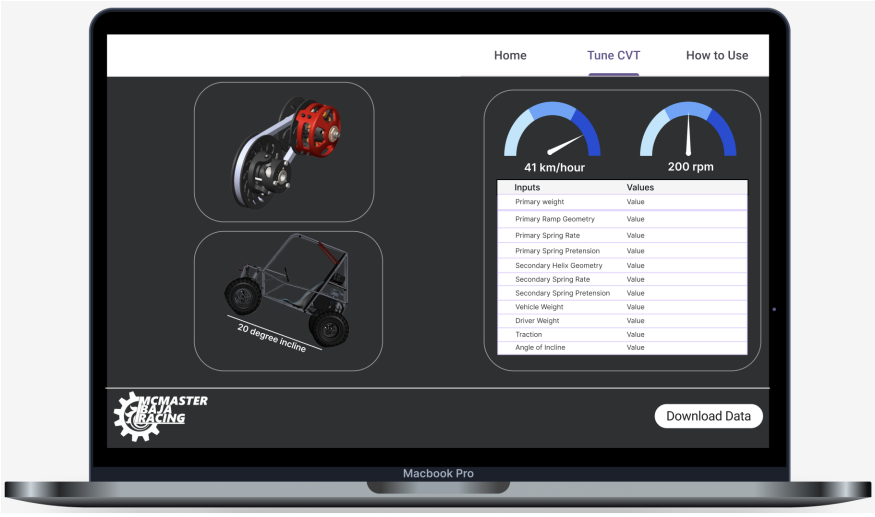

Figure 4: Simulation Page