Module Guide for TPG

Team 3, Tangle
Calvyn Siong
Cyruss Allen Amante
Edward Gao
Richard Li
Mark Angelo Cruz

January 15, 2025

1 Revision History

Date	Version	Notes
01/15/2025	0.0	Revision 0

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
DNN	Deep Neural Network
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
TPG	Tangled Program Graphs
UC	Unlikely Change

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: Support for additional MuJoCo environments.

AC2: Simulation support for all types of operating systems.

AC3: Comprehensive logging across all of TPG's components.

AC4: Extensive testing and checks for newly published code.

AC5: Refactoring old and unstructured components.

AC6: Format of inputting experimental parameters.

AC7: Shift to a different dependency management tool such as CMake.

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: Migration of TPG's implementation to a different language.

UC2: Refactoring the flow of task initialization and generation.

UC3: Transitioning away from Github Actions to other CI/CD platforms.

UC4: Replacement of MPI as the distributed computing framework.

UC5: Changing of communication protocol between different components such as gRPC.

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: Classic Control Module

M3: MuJoCo Module

M4: Visualization Module

M5: Logging Module

M6: TPG Experiment Module

Level 1	**Level 2**
Hardware-Hiding Module	N/A
Behaviour-Hiding Module	Classic Control Module MuJoCo Module Visualization Module Logging Module
Software Decision Module	TPG Experiment Module Framework 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.

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. *TPG* means the module will be implemented by the TPG 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 Classic Control Module (M2)

Secrets: Classic Control's data structures (such as disReset, actionTrace, etc. . .).

Services: Provides an interface between TPG and established for the classic control environments to allow for simulation and comparisons.

Implemented By: Existing TPG framework

Type of Module: Abstract Object

7.2.2 MuJoCo Module (M3)

Secrets: MuJoCo's data structures (such as mjModel, mjData, etc...).

Services: Provides an interface between TPG and MuJoCo to allow for simulation within MuJoCo-based experiments.

Implemented By: TPG

Type of Module: Abstract Object

7.2.3 Visualization Module (M4)

Secrets: Output logs from tpg*.std files.

Services: Provides the visualization of the experiment being run. Extract metrics from logs, aggregate the metrics, plot, and combine them into a singular PDF.

Implemented By: TPG

Type of Module: Abstract Object

7.2.4 Logging Module (M5)

Secrets: Generate logs containing experiment execution data, performance metrics, and debug information.

Services: Provides comprehensive logging functionality across all TPG components to be utilized by the visualization module

Implemented By: TPG

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 TPG Experiment Module (M6)

Secrets: Task parameters, team structures and decision making policies.

Services: Provides functionality to manage and evolve policies, evaluation of different tasks, and track on-going experiments

Implemented By: TPG

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**
FR-1	M3
FR-2	M4
FR-3	M5
FR-4	M1, M5, M6
FR-5	M2, M3, M5, M6
FR-6	M2, M3, M4, M5, M6
FR-7	M5, M6
FR-8	M2, M3, M4

Table 2: Trace Between Requirements and Modules

Anticipated Change	Modules	Module Numbers
AC1	MuJoCo	M3
AC2	Visualization	M4
AC3	Logging	M5
AC4	Framework	M??
AC5	Classic Control, TPG Experiment, Framework	M2, M6, M??
AC6	TPG Experiment, Framework	M6, M??
AC7	Framework	M??

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.

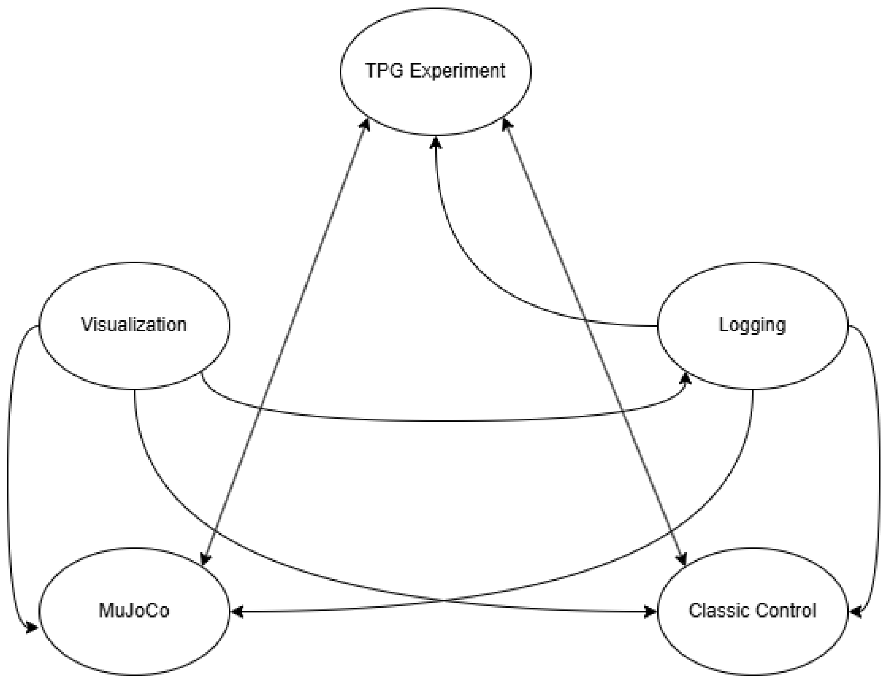

Figure 1: Use hierarchy among modules

10 Timeline

The capstone project's timeline will correspond to the main goals that have been outlined. The work has been split between the different modules with respective deadlines so that the team can work on separate modules concurrently. The deadlines have been set to help track progress and establish expectations.

Module	**Related members**	**Deadline**
Classic Control (7.2.1) - Develop the core control functionalities	Whole Team	Jan 15, 2024
MuJoCo (7.2.2) - Integrate the TPG framework with MuJoCo environments, ensuring compatibility and validation.	Whole Team	Jan 31, 2024
Visualization (7.2.3) - Implement visualization functionality with tangible data such as graphs, charts, video playback,realtime simulation to view results.	Edward	Jan 25, 2024
Logging (7.2.4) - Implement functionality to capture key info and debug information throughout framework	Whole Team	Jan 31, 2024
TPG Experiment (7.2.5) - Design and execute experiments to test the TPG framework performance across various scenarios	Edward, Cyruss	Jan 20, 2024
Framework (7.2.6) - should create functionality to authenticate a user using auth0	Cyruss, Mark	Jan 25, 2024
Continuous Integration (7.2.7) - Establish a CI pipeline for automatic linting, project building, and unit testing to ensure code quality and integration stability	Calvyn, Richard	Jan 31, 2024

Table 4: Timeline for each module

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.