Module Guide for ESF

Alaap Grandhi

March 22, 2025

1 Revision History

Date	Version	Notes
Mar 21	1.0	Initial Draft
Mar 22	1.1	Added Anticipated and Unlikely Changes

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
ESF	Equivariant Sensor Fusion
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 laid 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 of the input images and LiDAR.

AC3: The loss function used for training.

AC4: The specific equivariant layer implementation used.

AC5: The specific version of ADAM used for optimization.

AC6: The choice of training datasets (expanding beyond just NuScenes and Waymo).

AC7: The metrics collected for evaluation (adding more than just mAP).

AC8: The choice of library for logging.

AC9: The configuration file format.

AC10: The choice of library for generating 3D visualizations of predicted bounding boxes.

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: The use of PyTorch in the construction of many of the modules.

UC2: The use of a deep-learning based model.

UC3: The use of supervised learning techniques to train the model.

UC4: The use of OpenPCDet as a foundation for the code.

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: Config Module

M3: Data Module

M4: Model Module

M5: Checkpoint Module

M6: Training Module

M7: Inference Module

M8: Loss Module

M9: Evaluation Module

M10: Optimization Module

M11: Data Processing Module

M12: Equivariant Layer Module

M13: OpenPCDet Layer Module

M14: Plotting Module

M15: PyTorch Module

M16: Logging Module

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.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Config Module
	Data Module
	Model Module
	Checkpoint Module
	Training Module
	Inference Module
	Loss Module
	Evaluation Module
	Optimization Module
	Data Processing Module
	Equivariant Layer Module
	OpenPCDet Layer Module
Software Decision Module	Plotting Module
	PyTorch Module
	Logging Module

Table 1: Module Hierarchy

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.
ProgName means the module will be implemented by the ProgName 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

4

display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Modules

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: –

[Record, Library, Abstract Object, or Abstract Data Type]

7.2.1 A (M??)

Secrets:

Services:

Implemented By:

Type of Module:

7.2.2 Config Module (M2)

Secrets: The format and structure of the input configuration.

Services: Loads the configuration from a predefined file and makes it accessible to other modules.

Implemented By: ESF

Type of Module: Abstract Object

7.2.3 Data Module (M3)

Secrets: The format and structure of the training and evaluation datasets along with their respective loaders.

Services: Loads the input data and packages it in a form (a dataloader) that facilitates batched training.

Implemented By: ESF

Type of Module: Abstract Data Type

7.2.4 Model Module (M4)

Secrets: The format and structure of the deep learning fusion model.

Services: Forms a learnable and differentiable mapping from input images and LiDAR to predicted bounding boxes.

Implemented By: ESF

Type of Module: Abstract Data Type

7.2.5 Etc.

7.2.6 Checkpoint Module (M5)

Secrets: The methods for saving and loading a model from the file system..

Services: Allows for the saving of trained models to the file system and the loading of model weights from the file system.

Implemented By: ESF

Type of Module: Library

7.2.7 Training Module (M6)

Secrets: The process by which a model is trained on the input dataset.

Services: Trains the model on the input dataset using the loss function and the optimizer specified in the configuration file.

Implemented By: ESF

Type of Module: Library

7.3 Software Decision Module

7.3.1 Inference Module (M7)

Secrets: The process by which a model is run in inference on some input scene.

Services: Uses a trained model to produce a visualization of predicted bounding boxes on an input scene.

Implemented By: ESF

Type of Module: Library

7.3.2   Loss Module (M8)

Secrets: The method by which a loss value is produced from a given input.

Services: Compares a model's predicted bounding boxes to the ground truth bounding boxes for a given scene in order to produce a loss value.

Implemented By: ESF

Type of Module: Abstract Data Type

7.3.3   Evaluation Module (M9)

Secrets: The method by which a model is evaluated.

Services: Compares a model's predicted bounding boxes to the ground truth bounding boxes on some evaluation set to produce an mAP metric for the model.

Implemented By: ESF

Type of Module: Library

7.3.4   Optimization Module (M10)

Secrets: The method by which a model's parameters are updated.

Services: Updates a model's parameters according to the gradients stored on them using the ADAM algorithm. These gradients themselves are generated by the Loss module.

Implemented By: ESF

Type of Module: Abstract Data Type

7.3.5   Data Processing Module (M11)

Secrets: The method by which input data is converted into a usable format.

Services: Converts the input images and input LiDAR into a PyTorch Tensor format that can directly be used for training.

Implemented By: ESF

Type of Module: Library

7.3.6   Equivariant Layer Module (M12)

Secrets: The format and the structure of the equivariant layers that will be used to enhance fusion object detection performance.

Services: Creates equivariant layers that can then be put together to comprise the model.

Implemented By: ESF using escnn (Cesa et al. (2022)) and equivariant transformer (Tai et al. (2019))

Type of Module: Abstract Data Type

7.3.7   OpenPCDet Layer Module (M13)

Secrets: The format and the structure of the existing OpenPCDet layers.

Services: Creates layers that can be put together to form the base BEVFusion model and that will be used for parts of the model.

Implemented By: OpenPCDet (Team (2020))

Type of Module: Abstract Data Type

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: –

7.3.8   Plotting Module (M14)

Secrets: The method by which to generate 3D visualizations of bounding box predictions

Services: Provides methods which allow for LiDAR pointcloud scenes to be plotted in 3D alongside ground truth and predicted bounding boxes.

Implemented By: ESF using Open3D

Type of Module: Library

7.3.9 PyTorch Module (M15)

Secrets: The set of data structures and algorithms contained within the PyTorch library

Services: Allows for the creation of datasets, dataloaders, and models for the purposes of Deep-Learning training

Implemented By: PyTorch

Type of Module: Records, Libraries, Abstract Data Types

7.3.10 Logging Module (M16)

Secrets: The method by which to log statistics about model training and evaluation.

Services: Logs loss during training and mAP during evaluation to a file where they can be easily visualized later.

Implemented By: ESF using a wrapper around Tensorboard

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	M11, M3
R2	M11, M3
R3	M1, M2, M3, M4, M5, M6, M8, M9, M10, M11, M12, M13, M15, M16
R4	M1, M2, M4, M5, M7, M11, M12, M13, M14, M15

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M11
AC3	M8
AC4	M12
AC5	M10
AC6	M3
AC7	M9
AC8	M16
AC9	M2
AC10	M14

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.
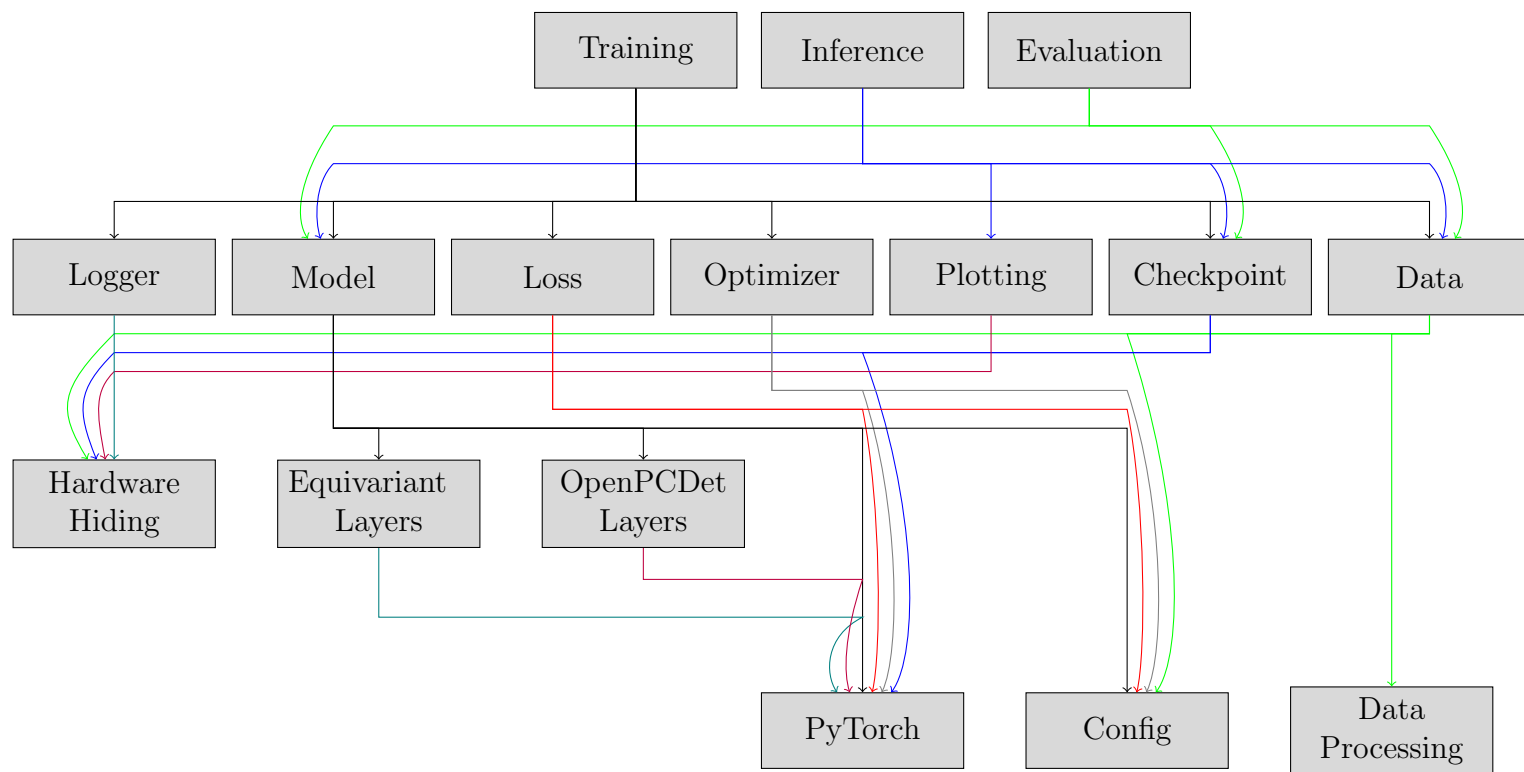
Figure 1: Use hierarchy among modules

10   User Interfaces

This section may be added in the future.

11   Design of Communication Protocols

N/A

12   Timeline

At this point in the course, it does not really make sense to do this section.

References

Gabriele Cesa, Leon Lang, and Maurice Weiler. A program to build E(N)-equivariant steerable CNNs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=WE4qe9xlnQw.

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.

Kai Sheng Tai, Peter Bailis, and Gregory Valiant. Equivariant Transformer Networks. In *International Conference on Machine Learning*, 2019.

OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet, 2020.