Module Guide for Park'd

Team #29, caPstOneGroup
Albert Zhou
Almen Ng
David Yao
Gary Gong
Jonathan Yapeter
Kabishan Suvendran

April 5, 2023

1 Revision History

Date	Version	Notes
Jan 18, 2023	1.0	Revision 0
Apr 3, 2023	1.1	Revision 1

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

See SRS Documentation at Park'd Software Requirements Specification.

symbol	**description**
Park'd	Parking Lot Application
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Park'd	Explanation of program name
UC	Unlikely Change
SUV	Sports Utility Vehicle
RNN	Recurrent Neural Network
ID	Identification

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. We advocate a decomposition based on the principle of information hiding. 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, 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. 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 type of vehicle information, such as the vehicle's length and width. This may change to simply require the vehicle class, such as SUV or Utility vehicle, using which the system would retrieve the dimensions.
As of Revision 1, vehicle-related information is not used at all. In future revisions, this information will be collected and used to implement parking violation detection (single vehicle occupying multiple spots).

AC2: The source of the video footage (from a camera feed website or from a group member's device with a camera)

AC3: The implementation of machine learning model is subject to change(Different generations of YOLO model may be used)
As of Revision 1, YOLOV3 was eventually used in the project due to its robustness that able to identify parking spots in any different footage angles. In future revisions, different generations of YOLO architecture may be used to provide even more accurate results

AC4: The layout of the parking spaces and if they reflect the exact layout of the parking lot, or just locations relative to one another.

AC5: Expand the stats of a parking layout to include past stats and trends over a time period.

AC6: Extend the navigation to start from outside the parking layout.

AC7: Represent the parking layout in a more graphical way.

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 type of information that is requested from the user to be provided to the website in order to set tailored parking preferences (user ID, password and vehicle information). **As of Revision 1, vehicle information is not gathered from users. In future revisions, this information will be collected and used to implement parking violation detection (single vehicle occupying multiple spots).**

UC2: Parking data will continue to be generated from a remote back-end and relayed to the front-end.

UC3: The application's main goal is the efficient display of available parking spaces for a supported parking lot.

UC4: The organization and interfaces of the modules involved in representing a parking layout.

UC5: The parking status and layout data will be stored in an agreed upon structure in json files.

UC6: The Navigation Module will give users directions to their selected parking spot.

UC7: The Machine Learning Module will detect cars and determine if labelled parking spots are occupied.

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: Video Capture Module

M3: Admin Console Module

M4: Admin Module

M5: Parking Lot Layout Module

M6: Parking Layout Element Module

M7: Parking Spot Module

M8: Authentication Module

M9: Navigation Module

M10: Parking Stats Module

M11: Machine Learning Model Module

M12: User Module

M13: User Action Handler Module

M14: Vehicle Module

M15: View Module

M16: Database Module

Level 1	**Level 2**
Hardware-Hiding Module	None
Behaviour-Hiding Module	Video capture module Admin console module Admin module Parking lot layout module Parking layout element module Parking spot module Authentication module User module User action handler module Vehicle module View module Database module
Software Decision Module	Navigation module Parking Stats module Machine learning model 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 3.

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. *Park'd* means the module will be implemented by the Park'd 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 Video Capture Module (M2)

Secrets: The latest photo or video clip that was captured

Services: Captures a photo or video clip from a YouTube live stream

Implemented By: Park'd

Comments: The Camera capture module is changed to a video capture module due to the input source is changed to YouTube live stream, thus no hardware is required for our project and this module is moved under the Behaviour-hiding module category.

7.2.2 Admin Console Module (M3)

Secrets: Holds the set of admins.

Services: Authenticate admin user access.

Implemented By: Park'd

Type of Module: Abstract Object

7.2.3 Admin Module (M4)

Secrets: Holds an admin's credentials and the layouts of their parking lots.

Services: Draw a parking lot layout to the screen and a path from a point on the road to a parking spot. Modify existing parking lot layouts.

Implemented By: Park'd

Type of Module: Abstract Data Type

7.2.4 Parking Lot Layout Module (M5)

Secrets: Holds the sequence of elements representing a parking lot.

Services: Create/modify a parking lot layout.

Implemented By: Park'd

Type of Module: Abstract Data Type

7.2.5 Parking Layout Element Module (M6)

Secrets: Holds the id and type of a parking layout element.

Services: Access attributes of a parking layout element.

Implemented By: Park'd

Type of Module: Record

7.2.6 Parking Spot Module (M7)

Secrets: Holds the properties of a parking spot.

Services: Access attributes of a parking spot.

Implemented By: Park'd

Type of Module: Record

7.2.7 Authentication Module (M8)

Secrets: Algorithm to permit users to sign in and access more personalized parking suggestions

Services: Compares the user's input information to the User information stored on the database and determines if there is a match between the information. If there is a match, the user is allowed to login and view personalized parking suggestions. Otherwise, the user will not be allowed to login and they can use the software as a guest.

Implemented By: Park'd

7.2.8 User Module (M12)

Secrets: Holds information that uniquely identifies a user and their needs.

Services: Provides a User object that can be stored in the database and later retrieved for making decisions about which parking spaces to show.

Implemented By: Park'd

Type of Module: Abstract Data Type

7.2.9 User Action Handler Module (M13)

Secrets: Handler the incoming user requests

Services: Act as a controller and handle user incoming requests and distribute them to the corresponding module that provides related service.

Implemented By: Park'd

7.2.10 Vehicle Module (M14)

Secrets: Holds information that uniquely identifies a user's vehicle.

Services: Provides a Vehicle object that can be stored in the database and later retrieved for making decisions about which parking spaces to show.

Implemented By: Park'd

Type of Module: Abstract Data Type

This module will be implemented in future revisions.

7.2.11 View Module (M15)

Secrets: Describe how components on the screen are placed.

Services: Provide graphical output.

Implemented By: Park'd

Type of Module: Abstract Object

7.2.12 Database Module (M16)

Secrets: Holds the data of the parking lots

Services: Provide parking lot layout information and availability information to user, as well as annotation information for the machine learning model through a set of database queries.

Implemented By: Park'd

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 Navigation Module (M9)

Secrets: Calculate a path from a starting point to a parking spot following the road using a path finding algorithm.

Services: Provides a sequence of parking layout elements from a starting point to a target parking spot.

Implemented By: Park'd

7.3.2 Parking Stats Module (M10)

Secrets: Calculate the current statistics of a parking layout.

Services: Get the number of parking spots with a given set of properties.

Implemented By: Park'd

7.3.3 Machine Learning Model Module (M11)

Secrets: Determine if the mid point of the bounding boxes of a detected vehicle predicted by machine learning model is inside the bounding boxes of the parking spots

Services: Provides users with the real time parking-lot availability information

Implemented By: Park'd

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**
FR1	M8, M9, M16
FR2	M8, M9, M16
FR3	M2, M11
FR4	M3, M4, M5, M6, M7, M8, M11,M16
FR5	M3, M4, M5, M6, M16,M7, M8
FR6	M9, M16, M13
FR7	M3, M16
FR8	M6, M16,M7, M13
FR9	M5
FR10	M6, M16,M7, M13
FR11	M9
FR12	M5, M13
FR13	M9
FR14	M7, M13
FR15	M7, M13
FR16	M9
FR17	M5, M7, M11
FR18	M13
FR19	M9
FR20	M7, M11
FR21	M13
FR22	M13
FR23	M13
FR24	M9, M13
FR25	M9
FR26	M3, M8
FR27	M3, M8
FR28	M4, M8
FR29	M8, M10
FR30	M4, M8
FR31	M4, M8
FR32	M4, M8, M15

Table 2: Trace Between Requirements and Modules

Req.	Modules
FR33	M4, M8, M15, M11, M16
FR34	M4, M8, M11, M16
FR35	M4, M8, M11, M16
FR36	M16
FR37	M16

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M14
AC2	M2
AC3	M11
AC4	M5
AC5	M10
AC6	M9
AC7	M4

Table 4: 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.

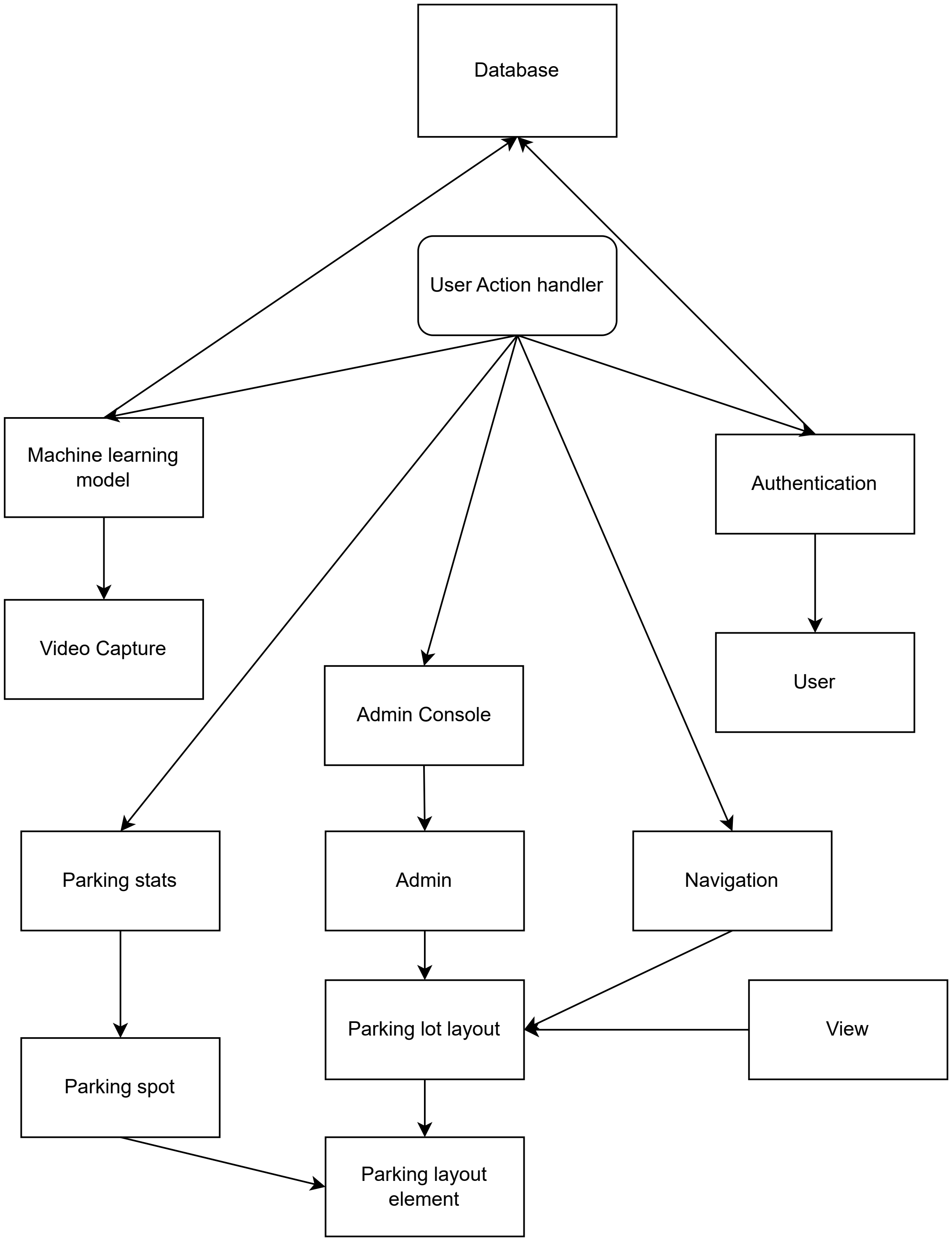

Figure 1: Use hierarchy among modules

References

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.