

Ren Hao – Project Portfolio for Gazeeebo

About the project

My team of 5 computer engineering students were tasked with developing a basic command line interface personal assistance desktop application, that can run on desktop with command line tool for everyone's computer, for our Software Engineering project. We chose to enhance it to become a specific assistance tool for NUS Computer Engineering (CEG) students. We call it Gazeeebo. This enhanced application enables CEG students to make a module plan for 4-years of study in NUS and to manage specialization courses; Meanwhile, they are also able to view their calendars, calculate their CAP results and record down notes.

This is what Gazeeebo looks like when you just entered:

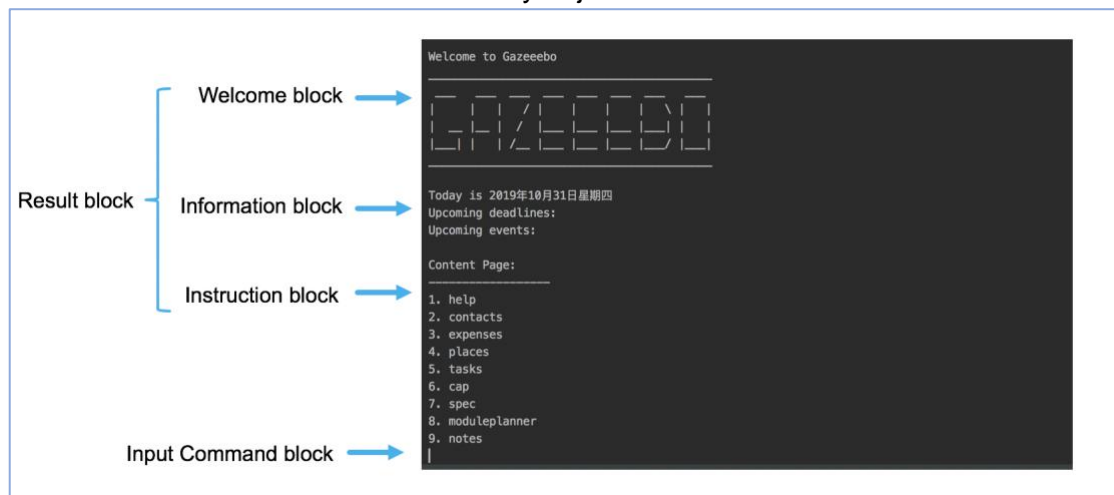


Figure 1. The User interface of Gazeeebo

My role was to design and write the codes for the 4-year module plan feature. The following sections illustrate these enhancements in more detail, as well as the relevant documentation I have added to the user and developer guides in relation to these enhancements.

Note the following symbols and formatting used in this document:



This symbol indicates important information.

`delete`

A grey highlight (called a mark-up) indicates that this is a command that can be inputted into the command line and executed by the application.

plan

Courier New format indicating that this is a sample input.

Summary of contributions

This section shows a summary of my coding, documentation, and other helpful contributions to the team project.

- Enhancement/Feature added:
 - ✓ I added the ability to plan a 4-year module plan
 - What it does: User could use `add` command to add module he/she wants to take into the specific semester column in the module plan table. Meanwhile, user could also delete existing module in the module plan or shift it into other semesters. With `plan` command, the whole plan table would be displayed on the screen. The system will also give comments and suggestions for the plan.
 - Justifications: `plan` command serves for display and showcase of the module plan table, while `add`, `delete` and `shift` command serves to modify the module plan.
 - ✓ I added the prerequisite feature.
 - What it does: User could check prerequisite modules of a certain module with `prerequisite` command, a tree of prerequisite relations will be drawn on the screen.
 - Justifications: This feature is based on Depth-First-Search algorithm. The algorithm was investigated by French mathematician Charles Pierre Trémaux, I adapt it from: [AdaptingAddress](#)
- Code contributed: I contributed 2328 lines of code to the project. Click here to view a sample of my code: [#Code1](#)
- Other contributions:
 - Other Enhancements and features:
 1. Trivia function: Automatically records down users inputs, using them to display suggestions if users do not know what to do. [Ex1](#)
 2. Sort function: Sort the list of tasks for user so that more urgent task or more prior task is shown first. [Ex2](#)
 - Documentations:
 1. Improve Javadoc and code quality by adjusting code with “check style” tool. [Ex3](#)
 2. Write Javadoc to explain usage and implementation of code. [Ex4](#)
 - Community:
 1. Report bugs and offer suggestions to other groups in class. [Ex5](#), [Ex6](#).
 2. Review Pull Requests (review teammates’ code and approve the code for merging into our main project). [Ex7](#).

Contributions to the User Guide

This section shows my contribution in our team's **Gazebo User Guide**, taking those additions I have made for the 4-year module plan feature as an example.

Module Planner Section

Feature Description: The Module Planner block mainly helps users to create and manage a 4-year module plan for studying life in NUS

Example:

Lets say you have already logged in to our gazeebo main page! And on that page there are a lot of instructions.

How to get to the Module Planner page?

- From main page, type in command `moduleplanner`, and then system will drop into Module Planner session



You could only enter the module planner section from the main page

Following images shows when you type in `moduleplanner`.

```

Welcome to Gazeeebo

-----

  _  _  _  _  _  _  _  _  _
 |  |  |  | /  |  |  |  \  |
 |  _  |  | /  |  _  |  _  |
 |  _  |  | /  |  _  |  _  |
 |  _  |  | /  |  _  |  _  |

-----

Today is 2019年10月31日星期四
Upcoming deadlines:
Upcoming events:

Content Page:
-----
1. help
2. contacts
3. expenses
4. places
5. tasks
6. cap
7. spec
8. moduleplanner
9. notes
moduleplanner

```

Figure 2. How to get into the Module Planner Page

```

moduleplanner
Welcome to Module Planner!

1. Add module to your plan: add CSXXXX to n(Semester number)
2. Delete module from your plan: Delete CSXXXX from n(Semester number)
3. Shift module to other semester: shift CSXXXX to n(Semester number)
4. See your Study Plan: plan
5. See your Prerequisite of a module: prerequisite CSXXXX(module code)
6. Exit Module Planner page: esc

```

Figure 3. The start page of Module Planner Section

Scenario 1:

Let's say without doing anything, now you just want to check what we have!

Display the module plan table and showcase suggestions: `plan`

Description: showcase the module plan table

Format: `plan`



If it is the first-time user is using this feature, the system showcases default module plan table which is the CEG recommended module plan for AY18/19 intake.

```

plan
+-----+
| Sem 1 | Sem 2 | Sem 3 | Sem 4 | Sem 5 | Sem 6 | Sem 7 | Sem 8 |
+-----+
gazeebo.exception.DukeException: We cannot find the MC of this module: PC1231
| CS1010E | EE2026 | CS2101 | CG2023 | CP3880 | CG4002 | CG3207 | |
+-----+
| MA1511 | CG1112 | CS2113T | ST2334 | EG2401A | EE4204 | | |
+-----+
| MA1512 | MA1508E | CG2271 | CG2027 | PC1231 | | | |
+-----+
| CS1231 | CS2040C | GER1000 | CG2028 | | | | |
+-----+
| CG1111 | GEQ1000 | GET1013 | GEH1036 | | | | |
+-----+
| ES1103 | | | | | | | |
+-----+
| MCs:22 | MCs:22 | MCs:20 | MCs:16 | MCs:14 | MCs:12 | MCs:4 | MCs:0 |
+-----+
* Note: You haven't reach the graduation requirement! *
* To meet the graduation requirement, you have to take following modules: *
EG3611A
CS3230
CS1010
* Note: You need to have at least 20 MCs of Technical Elective Modules! *
* You need 16 MCs More. *

```

Figure 4. Example of displaying module plan using `plan` command

Scenario 2:

Now you want to add something and start planning your university!

User adds module to specific semester column: `add`

Description: Add a new module to any column of the study plan table.

Format: `add module_code to semester_number`

Example: `add CS4223 to 5`

Steps to add a module:

1. Type in command `add module_code to semester_number` and press ENTER
2. System will showcase the module is successfully added.



If adding is not successful, an IOException message will be displayed

Scenario 3:

Let's see what you can do if you suddenly want to remove this module!

Delete module from specific semester column: **delete**

Description: delete an existing module from any column of the study plan table

Format: `delete module_code from semester_number`

Example: `delete CS4223 from 5`

Steps to delete a command:

1. Type in command `delete module_code from semester_number` and press ENTER
2. System will showcase the module is successfully deleted.



If deleting is not successful, an IOException message will be displayed

Scenario 4:

Now what if you want to shift some module to other places?

Shift module from specific semester column to another column: **shift**

Description: shift an existing module from any column to another column of the study plan table

Format: `shift module_code to semester_number`

Example: `shift CS4223 to 5`

Steps to add a command:

1. Type in command `shift module_code to semester_number` and press ENTER

2. System will showcase the module is successfully shifted.



If shifting is not successful, an IOException message will be displayed

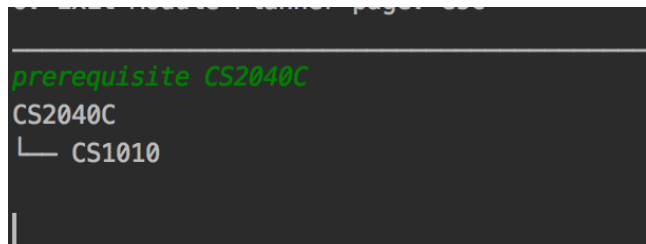
Scenario 5:

Show a prerequisite tree of a module: `prerequisite`

Description: showcase all prerequisite requirements for a specific module.

Format: `prerequisite Module_Code`

Example: `prerequisite CS2040C`



```
prerequisite CS2040C
CS2040C
├── CS1010
```

Figure 5. Example outcome of prerequisite command

Scenario 6 [Future feature coming in v2.0]:

Auto adjust the table base on prerequisites: `adjust`

Description: Users could type in adjust command; the system will check whether modules inside the study plan have met their prerequisites. If not, the system will then auto fill in prerequisite module

Contributions to the Developer Guide

This section shows my contribution to our team's Gazebo Developer Guide, taking those additions I have made for the 4-year module plan feature as an example.

Module Plan Feature

The Module Planner is mainly facilitated by `StudyAssistCommandParser`, it extends `Command` and stores information internally in data structure of an `ArrayList<ArrayList<String>>` and externally in `Study_Plan.txt` file.

`StudyAssistCommandParser` takes care of the logic for the module plan page and `StudyAssistCommandParser` implements the following operations:

- `StudyAssistCommandParser#add()` — Add module to specified semester in the module plan.
- `StudyAssistCommandParser#delete()` — Delete module from specified semester in the module plan.

- StudyAssistCommandParser#shift() — Shift module from existing semester to another semester in the module plan.
- StudyAssistCommandParser#plan() — Print out existing study plan and display reminder/suggestions in the command line interface.
- StudyAssistCommandParser#prerequisite() — Check prerequisite of certain module.
- StudyAssistCommandParser#undo() — Undo previous commands.

Here is a class diagram which shows the structure of the classes that implement the module planner feature:

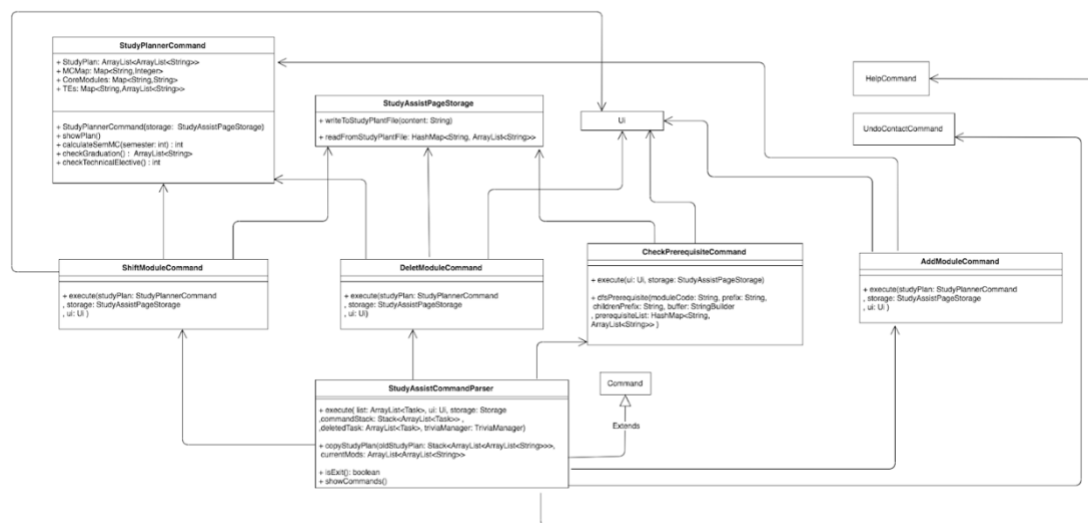


Figure 6. Class diagram indicates the structure of the module planner feature.

The following steps show a usage scenario:

Step1. The user enters the application for the first time. **StudyAssistCommandParser** is initialized by calling **execute()** method

Step 2. The user inputs a command, add, delete, shift, plan undo or prerequisite.

Step 3. The respective command will be initialised.

Step 4. The user exits by inputting "esc".

The following sequence diagram shows the initialization step, when **execute ()** method of **StudyAssistCommandParser** is called.

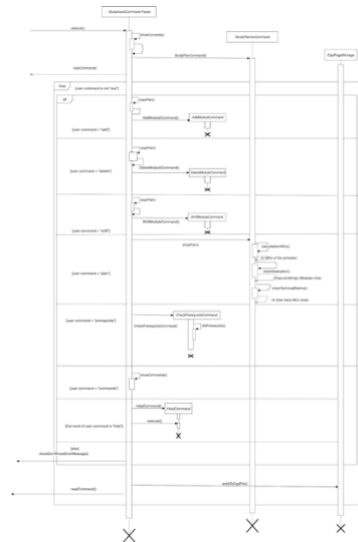


Figure 7. UML Sequence diagram of initialization step.

When an **AddModuleCommand** is called:

- Step 1. The user types in add command to add a module to a specified semester.
- Step 2. The add command calls addModuleCommand, insert a new module into the StudyPlan, modifying in internal ArrayList StudyPlan.
- Step 3. writeToStudyPlan() method is called to store updates in external Study_Plan.txt.

When a **DeleteModuleCommand** is called:

- Step 1. The user types in delete command to delete certain module from specific semester of the existing Study Plan.
- Step 2. The delete command calls deleteModuleCommand to delete module from the StudyPlan, modifying in internal ArrayList StudyPlan.
- Step 3. writeToStudyPlan() method is called to store updates in external Study_Plan.txt.

When a **ShiftModuleCommand** is called:

- Step 1. The user types in shift command to shift certain module from its belonging semester to another semester of the study plan.
- Step 2. The shift command calls shiftModuleCommand to delete module from the StudyPlan, modifying in internal ArrayList StudyPlan.
- Step 3. writeToStudyPlan() method is called to store updates in external Study_Plan.txt.

When a **StudyPlannerCommand** is called:

Step 1. User inputs plan into the command line.

Step 2. The “plan” command calls showPlan() method of StudyPlannerCommand, displays study plan in table format in the command line interface.

Step 3. Then, calculateSemMCs(), checkGraduation() and checkTechnicalElectives() methods of StudyPlannerCommand are called to show some comments and suggestions of the plan.

When a **CheckPrerequisiteCommand** is called:

Step 1. User inputs the module code that he wants to find prerequisite for.

Step 2. execute () method will be called, inside execute () method, it runs dfsPrerequisite() method to go through a depth-first search in the data structure.

Step 3. Tree of prerequisites is printed out

The following activity diagram summarizes what happens when execute () method of **StudyAssistCommandParser** is called.

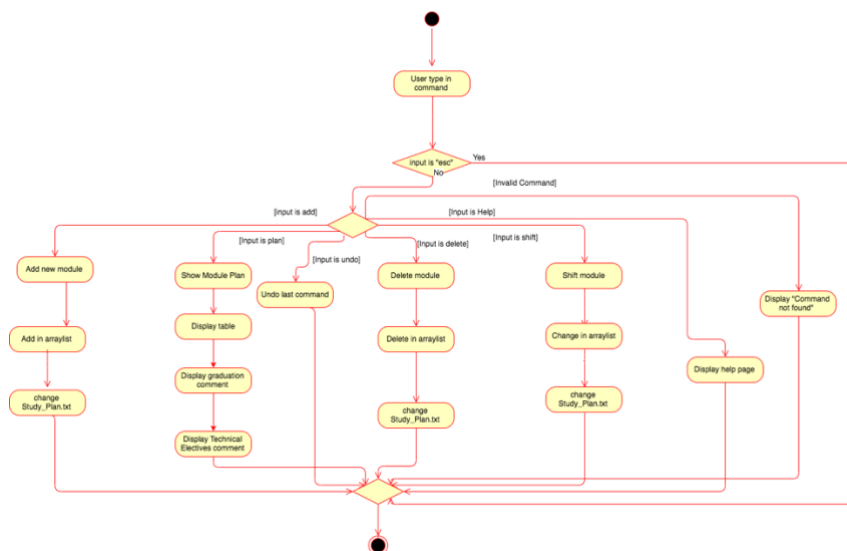


Figure 8. UML Activity diagram StudyAssistCommandParser

Design Considerations

When designing the module plan functions, I had to make decisions on how best to execute the commands and what data structure to support the commands. The following is a brief summary of my analysis and decisions.

Aspect	Alternative 1(Current Choice)	Alternative 2
How to store the Study Plan internally and externally	<p>Using an ArrayList of ArrayList of String to store modules of the Study plan, According to semesters.</p> <p>Also stores the ArrayList line by line in a txt file.</p> <p>■ Pros: Easy to implement</p>	<p>Using HashMap to store modules and use json file to store study plan externally.</p> <p>■ Pros: HashMap 's time complexity of delete and insert is $O(1)$, when size increases it performs</p>

	<ul style="list-style-type: none"> ■ Cons: ArrayList has a time complexity of $O(n)$ when insert or delete an element of it, it may become time consuming when ArrayList size increases. Also, when there is huge data, txt file may become large in size and causing the programme size to increase. <p>I decided to proceed with this option because for our feature we need to frequently access elements with its index, ArrayList is more efficient in this way.</p>	<p>faster. Json file is more compact in size also.</p> <ul style="list-style-type: none"> ■ Cons: HashMap's time complexity for access certain element in the HashMap is $O(n)$, it has to check previous elements before accessing the exact element. Json file may be different to implement.
How to check total MCs of the modules	<p>Using static HashMap to record MCs of all modules, module code is the key, MC number is the value.</p> <ul style="list-style-type: none"> ■ Pros: The MC HashMap is static and can be accessed globally; HashMap find's time complexity is $O(1)$ make it fast to search and find MC of certain module. ■ Cons: It takes up a lot of memory usage. <p>I decided to proceed with this option because it is easier to implement, and same file could be used for other needs in the future as well.</p>	<p>Create a module class, store all information (MCs, Description, Code and so on) in the class. When checking total MCs, just traverse the whole ArrayList and access each element's MC and count them up.</p> <ul style="list-style-type: none"> ■ Pros: It convenient not only MC calculation but also benefit other operations of modules; ■ Cons: It needs to change other methods and Command to cooperate with the new module class, thus may be troublesome to implement.