Module Interface Specification for Flow

Team 9, min-cut
Ethan Patterson
Hussain Muhammed
Jeffrey Doan
Kevin Zhu
Chengze Zhao

January 28, 2026

1 Revision History

Date	Version	Notes
10/11/25	Rev -1	
21/01/26	Rev 0	

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/ritual-17/flow/tree/main/docs/SRS-Meyer

Contents

3 Introduction

The following document details the Module Interface Specifications for Flow

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/ritual-17/flow.

4 Notation

The specification of Flow uses several derived data types, including sequences, strings, and tuples. Sequences are ordered lists containing elements of the same data type. Strings are sequences of characters. Tuples represent a finite collection of values, potentially of different types. Functions are described by specifying the data types of their inputs and outputs, and local functions are documented using their type signatures followed by their behavioural descriptions.

Several modules, most notably the Shape Interface Module, operate on geometric objects that are placed on a two-dimensional canvas. Shapes are treated abstractly and are not tied to a specific rendering or user interface implementation. A shape is represented conceptually as a tuple consisting of a shape type and a finite set of parameters. The shape type is drawn from a predefined set of supported shapes (e.g., rectangle, circle, text box), while the parameters describe properties such as position, size, and orientation.

Positions are expressed using Cartesian coordinates in a two-dimensional plane, typically represented as (x, y) pairs of real numbers. The canvas may be conceptually overlaid with a uniform grid that provides a visual reference and optional alignment structure for placing and manipulating shapes. The grid does not impose semantic constraints on shape placement, but may be used to support consistent positioning, snapping behaviour, or alignment operations. Dimensions such as width, height, or radius are represented using real-valued quantities. Collections of shapes are represented as sequences of shape tuples. Operations on shapes, such as insertion, deletion, movement, resizing, or transformation, are described in terms of updates applied to these abstract representations. Any constraints on valid coordinates or dimensions are specified explicitly in the relevant module interfaces.

5 Module Decomposition

Level 1	**Level 2**
Hardware-Hiding	6 Display Interface Module 7 Input Interface Module 8 File Interface Module
Behaviour-Hiding	9 Geometry State Parser Module 10 Geometry State Converter Module 11 Commands Parser Module 12 Mode Commands Module 13 Shape Interface Module 14 User Preference Module
Software Decision	15 Text Buffer Module 16 Geometry State Module 17 Geometry State Mutator Module 18 Undo Redo Module 19 User Persistence Module

Table 1: Module Hierarchy

6 MIS of Display Interface Module

6.1 Module

Display for the system

This is the module that will handle the display for the system. This is a module that when called updates the display.

6.2 Uses

16 Geometry State Module

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
display	Shapes	Display to screen	-
resize	width, length	Resize the screen as needed	-
status		Show status	-

6.4 Semantics

6.4.1 State Variables

N/A

6.4.2 Environment Variables

- System Screen
- Rendering API

6.4.3 Assumptions

6.4.4 Access Routine Semantics

display(Shapes):

- output: Outputs Shapes visually on the screen

resize(length,width):

- output: Outputs resized version of the current screen

status():

- output: Outputs current status of the program e.g. waiting for input, rendering etc.

6.4.5 Local Functions

N/A

7 MIS of Input Interface Module

7.1 Module

Input Module. Deals with User inputs and passes them on to the 12 Mode Commands Module

7.2 Uses

12 Mode Commands Module

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_keyboard_inputs	-	keyboard_inputs	-
get_mouse_inputs	-	mouse_inputs	-

7.4 Semantics

7.4.1 State Variables

7.4.2 Environment Variables

- Keyboard API
- Mouse API

7.4.3 Assumptions

7.4.4 Access Routine Semantics

get_keyboard_inputs():

- output: Data containing all current keyboard inputs. All keys will be off if the user is not using a keyboard.

get_mouse_inputs():

- output: Data containing mouse location and inputs. All inputs are off if the user does not have a mouse.

7.4.5 Local Functions

N/A

8 MIS of File Interface Module

8.1 Module

9 Geometry State Parser Module

8.2 Uses

12 Mode Commands Module

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
open	file_path	opened_file	Invalid File
save	file_path, opened_note	success	Saving Error

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 Environment Variables

- File System

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

open(file_path):

- output: The opened file
- exception: Invalid File if file is missing or of an incorrect type.

save(file_path, opened_note):

- output: If the Module was successful in saving opened_note to file_path
- exception: No Space if there is not enough space to save the file.

8.4.5 Local Functions

N/A

9 MIS of Geometry State Parser Module

9.1 Module

File parser takes the file input and parses it into data structure containing all shapes which is then stored by the 16 Geometry State Module which holds all shape data.

9.2 Uses

16 Geometry State Module

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
parse	opened_file	shapes	Invalid File

9.4 Semantics

9.4.1 State Variables

N/A

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

N/A

9.4.4 Access Routine Semantics

parse(opened_file):

- output: Data structure containing all shapes from the given file.
- exception: Invalid File: if the file given is able to be parsed.

9.4.5 Local Functions

10 MIS of Geometry State Converter Module

10.1 Module

Note converter that converts notes to files of different file types, sending it to the file interface module

10.2 Uses

8 File Interface Module

10.3 Syntax

10.3.1 Exported Constants

Legal file types

- PDF
- fnote

10.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
convert	opened_note, file_type	file	Invalid file_type

10.4 Semantics

10.4.1 State Variables

10.4.2 Environment Variables

10.4.3 Assumptions

10.4.4 Access Routine Semantics

convert(opened_note, file_type):

- output: file that is ready to be saved on the computer.
- exception: Invalid file_type if the given file type is not compatable with the ones implemented

10.4.5 Local Functions

convert_*(opened_note) functions that do the converting once the file type has been determined. (ex convert_pdf)

11 MIS of Commands Parser Module

11.1 Module

Command Parser, converts keyboard inputs into commands and passes them onto the shape mutator.

11.2 Uses

17 Geometry State Mutator Module 14 User Preference Module

11.3 Syntax

11.3.1 Exported Constants

Command names or ids e.g. ctrl + s -> <C-s>, d + d -> "dd"

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
parse_commands	keyboard_inputs, mouse_inputs	parsed_input	-

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Environment Variables

N/A

11.4.3 Assumptions

Inputs have been already cleaned by the other modules.

11.4.4 Access Routine Semantics

parse_commands(keyboard_inputs, mouse_inputs):

- output: commands, which would be used by the Geometry State Mutator Module 17

11.4.5 Local Functions

N/A

12 MIS of Mode Commands Module

12.1 Module

Main Module contains the commands usable in the current mode.

12.2 Uses

N/A

12.3 Syntax

12.3.1 Exported Constants

Modes:

- normalMode
- insertMode
- visualMode
- commandMode
- lineMode
- textMode

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_commands	mode	commands	-

12.4 Semantics

12.4.1 State Variables

N/A

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

Mode given will always be a valid mode.

12.4.4 Access Routine Semantics

get_commands(mode):

- output: commands that can be run in the given mode

12.4.5 Local Functions

N/A

13 MIS of Shape Interface Module

13.1 Module

Module that contains information about shapes.

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

Shapes: Types of shapes

- Rectangle
- circle
- Triangle
- Text Box

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
create_shape	shape_type, points	shape	Invalid Points
move_shape	shape, new_points	-	Invalid Points

13.4 Semantics

13.4.1 State Variables

N/A

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

N/A

13.4.4 Access Routine Semantics

create_shape(shape_type, points):

- output: Shape with type shape_type using points for location and size.
- exception: Invalid points if the points are not possible for the type of shape.

move_shape():

- transition: changes the points of the shape to the new points
- exception: Invalid points if the points are not possible for the type of shape.

13.4.5 Local Functions

N/A

14 MIS of User Preference Module

14.1 Module

User preferences, provides an interface for creating and manipulating user preferences in the system.

14.2 Uses

6 Display Interface Module 19 User Persistence Module

14.3 Syntax

14.3.1 Exported Constants

N/A

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_command_prefrences	mode	commands	-
modify	-	-	-
get_theme_prefrences	-	theme	-

14.4 Semantics

14.4.1 State Variables

Command preferences: users defined commands Theme preferences: user theme prefrences

14.4.2 Environment Variables

14.4.3 Assumptions

14.4.4 Access Routine Semantics

get_command_prefrences(mode):

- output: Any user defined preferences for the current mode

modify():

- transition: Has the user input their preferences where the moudles saves it to a file.

get_theme_prefrences():

- output: Theme preferences for the system

14.4.5 Local Functions

15 MIS of Text Buffer Module

15.1 Module

TextBuffer

15.2 Uses

Geometry State Module (16)

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
createText	location, text, formatting	TextID	Invalid Location
modifyText	TextID, text	-	Text Not Found
moveText	TextID, newLocation	TextObject	Text Not Found
deleteText	TextID	-	Text Not Found

15.4 Semantics

15.4.1 State Variables

text : Map<TextID, TextObject>
Where TextObject contains:

- position : location
- content : string

15.4.2 Environment Variables

Screen (for rendering text)

15.4.3 Assumptions

TextIDs are unique and immutable.

15.4.4 Access Routine Semantics

createText(location, text, formatting):

- transition: Text box at location with text and formatting
- output: TextID of the created text box
- exception: Invalid Location

modifyText(TextID, text):

- transition: Text in text box is modified to now contain text with formatting
- exception: Text Not Found

moveText(TextID, newLocation):

- transition: text box is moved to location.
- exception: Text Not Found

deleteText(TextID):

- transition: text box is removed from the text buffer
- exception: Text Not Found

15.4.5 Local Functions

generateTextID() : TextID

16 MIS of Geometry State Module

16.1 Module

GeometryState (16)

16.2 Uses

None

16.3 Syntax

16.3.1 Exported Constants

ShapeType = Circle, Rectangle, Triangle, TextBox
LineType = Line, Arrow

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
getShapes	-	Set<Shape>	-
getText	-	Set<TextBox>	-
getLines	-	Set<Line>	-

16.4 Semantics

16.4.1 State Variables

- Shapes : Set<Shape>
- TextBoxes : Set<TextBox>
- Lines : Set<Line>

16.4.2 Environment Variables

None

16.4.3 Assumptions

Geometry is immutable except via Geometry State Mutator Module (17).

16.4.4 Access Routine Semantics

getShapes():

- output: Data Structure containing all shapes

getText():

- output: Data structure containing all text boxes

getLines():

- output: Data structure containing all lines

16.4.5 Local Functions

None

17 MIS of Geometry State Mutator Module

17.1 Module

GeometryStateMutator (17)

17.2 Uses

Undo Redo Module (18), Geometry State Module (16)

17.3 Syntax

17.3.1 Exported Constants

N/A

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
addShape	shapeSpec	GeometryID	Invalid Spec
addLine	lineSpec	GeometryID	Invalid Spec
delete	GeometryID	-	Not Found
resize	GeometryID	-	Not Found
move	GeometryID, location	-	Not Found, Invalid Transform

17.4 Semantics

17.4.1 State Variables

None

17.4.2 Environment Variables

None

17.4.3 Assumptions

Mutations must be recorded by Undo Redo Module.

17.4.4 Access Routine Semantics

addShape(shapeSpec):

- transition: Adds the shape to the canvas
- output: GeometryID of the added shape
- exception: Invalid Spec if the shapeSpec is invalid

addLine(lineSpec):

- transition: Adds the line to the canvas
- output: GeometryID of the added line
- exception: Invalid Spec if the lineSpec is invalid

delete(GeometryID):

- transition: Deletes the shape/line/textbox with the given GeometryID
- exception: Not Found if the GeometryID does not exist

resize(GeometryID):

- transition: Resizes the shape/line/textbox with the given GeometryID
- exception: Not Found if the GeometryID does not exist

move(GeometryID, location):

- transition: Moves the shape/line/textbox with the given GeometryID to location
- exception: Not Found if the GeometryID does not exist
- exception: Invalid Transform if the location is invalid

17.4.5 Local Functions

None

18 MIS of Undo Redo Module

18.1 Module

UndoRedoModule (18)

18.2 Uses

Geometry State Module (16)

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
undo	-	-	Cant Undo
redo	-	-	Cant Redo
addCommand	command	-	-

18.4 Semantics

18.4.1 State Variables

- undoStack : Stack<Command>
- redoStack : Stack<Command>

18.4.2 Environment Variables

None

18.4.3 Assumptions

Commands fully capture inverse operations.

18.4.4 Access Routine Semantics

undo():

- transition: Pops command from undoStack, executes its inverse, and pushes it onto redoStack
- exception: Cant Undo if undoStack is empty

redo():

- transition: Pops command from redoStack, re-executes it, and pushes it onto undoStack
- exception: Cant Redo if redoStack is empty

addCommand(command):

- transition: Pushes command onto undoStack and clears redoStack

18.4.5 Local Functions

N/A

19 MIS of User Persistence Module

19.1 Module

UserPersistenceModule (19)

19.2 Uses

User Preference Module (14)

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
save	FilePath	-	IOError
load	FilePath	User Themes and Saves	IOError

19.4 Semantics

19.4.1 State Variables

None

19.4.2 Environment Variables

File System

19.4.3 Assumptions

File format is versioned and backward compatible.

19.4.4 Access Routine Semantics

save(commands,theme):

- transition: Saves user preferences to file at FilePath
- exception: IOError if there is an issue writing to the file

load(FilePath):

- output: User Themes and Saves loaded from file at FilePath
- exception: IOError if there is an issue reading from the file

19.4.5 Local Functions

None

References

20 Appendix

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

    - Chengze - While we got back on MG and MIS document after some of the implementation, we had a better understanding of the overall design of the project. This allowed us to better fill in the details of each module and their services. In particular, reviewing the specifications helped us identify missing requirements and confirm that our current code matches the intended module responsibilities. Writing this deliverable also made it easier to see how the system should be organized, ensuring that each component has a clear purpose and well-defined interface moving forward.
    - Ethan - In this deliverable we got a better idea of how the app would look and work from an architecture standpoint. It let us get a more concrete understanding of how components would be broken down in the code and how they would work together. Additionally, it helped us to write out some scaffold code for our rev 0 and do some work in parallel to better visualize module interactions and dependency.
    - Hussain - In this deliverable, we were able to begin thinking more on the development of Flow. By discussing and outlining the different modules and their services, I believe we were able to get a much better understanding of how we can put together a successful application for this project. We were assigned this document close to the due date of Rev0 which is our first real revision of the finished project. This also pushes us to use the decisions made in this document directly in the development of our project.
    - Jeffrey - During this deliverable, we began to visualize and actually understand how we wanted our application to work/function. Additionally, as stated in a further section, none of the existing documents required major changes which

helped speed the process of this deliverable. Moreover, with a well written MIS, we are now able to proceed with development with a clearer image and better instructions.

- Kevin -What went well in this deliverable was using our previous work on the MIS made implementing the skeleton or our project much simpler as it was already defined in the design docs. Additionally, having a skeleton to base the docs off of made it easier to find sections where we should have explained more to make implementation easier.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   - Chengze - The pain point I experienced during this deliverable was the lack of clarity on some of the module services. Since some of the modules were not fully implemented yet, it was hard to determine what services they should provide. To resolve this, we had to rework on MIS to include more details and make sure the services aligned with the overall design of the project.
   - Ethan - Overall I think this was the most challenging deliverable so far. This was where we had to go from ideas to a concrete system design. I have more experience in web development and backend system design, but not as much in designing something like a text editor or geometry manipulation program. It was difficult to think about all of the modules and how they would interact with eachother without writing any code. I helped resolve this by writing out some potentially throw-away code to play with while designing.
   - Hussain - A couple pain points I experienced during this deliverable had to do with the poor delegation of work and slower restart into working on the design document. Our group had decided to split the work for the design doc in a same way we did for Rev -1, but this gave some of us more work than others as this time around, the MIS required more work than the MG. In addition to this, some of us were a bit slow (potentially because of the break) in getting back to working on the project. Thankfully, I was able to resolve this by re-delegating some of the work and helping set up a meeting with the TA to discuss the items for the deliverable.
   - Jeffrey - The biggest pain point I had during this deliverable was trying to keep up with my peers when conceptualizing ideas for our concrete system. Since I only have a handful of experiences with app development, I was having a hard time helping my teammates come up with modules. Luckily, one of the team members Ethan created throw-away code which helped me visualize the system better.
   - Kevin - The main pain point while working on this deliverable was getting back into the swing of things after the break. This made our team less organized and communication slower than we should have for this deliverable, meaning that work discussion and delegation were slower. This problem was solved by ensuring

that all work and dates were posted on the group discord ensuring that we can still communicate and finish our work even if our communication is slower.

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

    - Most of our design decisions stemmed from the consulting the primary stakeholders of the project, namely ourselves as students and end users of the system. Since the software is intended to be used and evaluated by students with similar technical backgrounds and needs, our group discussions acted as a proxy for client consultation. Particularly, decisions regarding the overall modular decomposition were based on stakeholder discussions. As a group, we agreed to keep the higher-level module structure provided in the template, as it aligned well with the expectations for separation of concerns and maintainability. We then refined this structure by collectively defining the responsibilities and services of each module, which is reflected in the Module Guide.
    - Many of these stakeholder-driven decisions were mentioned as part of the functional and non-functional requirements outlined in the SRS. The design of individual modules and their interfaces was guided by these requirements, ensuring clear traceability from stakeholder needs to the system-level design.
    - Some design decisions did not stem directly from stakeholder input but instead came from practical constraints and other experiences. For example, user interface decisions were informed by common design patterns observed in modern drawing and canvas-based applications, helping minimize the learning curve and aligning with user expectations. Similarly, the project timeline was determined based on course deadlines and workload considerations, rather than client input, as these constraints were externally imposed by the course structure.

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

    - While creating the design document, none of the existing documents required major changes. The SRS served as a foundational input to the design process, and the module decomposition and interfaces were created specifically to satisfy the functional and non-functional requirements defined. As a result, the design document did not introduce new requirements or require modifications to the SRS, but instead validated that the requirements were implementable and well-scoped.
    - Similarly, the Hazard Analysis did not require updates as no new hazards were identified during the design process. However, the design helped clarify how previously identified hazards would be addressed through module separation, controlled state mutation, and undo/redo mechanisms, reinforcing the effectiveness of existing mitigation strategies.

- The design document does, however, influence how the Verification and Validation (VnV) Plan would be interpreted and later refined. By clearly defining module responsibilities and interfaces, the design gives more precise unit and integration testing strategies, which would be reflected in future updates to the VnV plan.

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

    - One of the biggest limitations of our solution is that we are priotizing rapid development and usability over mximum performance and architetural flexibility.To clarify, given our time and resource constraints, we chose technologies that allowed us to build and iterate quickly; however, this would obviously introduce trade offs.

      For instance, our application uses Electron, which is relatively resource-heavy when compared to native solution. We chose this as it enables cross-plateform development and faster prototyping. The cross-platform development is very important to use as the team is spread across multiple platforms. While this seems like a great choice, it also has it's cons such as higher memory usage and reduced performance. With unlimited resources, we could reimplement the application using a more performant, lower-level stack to improve effciency, speed, and responsiveness. For example, we could reimplment our application using Rust or C++.
    - Another limitation is that currently our code is coupled between frontend and backend. If we had unlimitd time we would completely isolate the backend and business logic behind a well-defined API. This would allow alternative frontends, mobile or web, to be developed independently and would allow us to improve the application's scalability and maintainability.
    - Additionally, if we were truly given unlimited resources, we would contemplate the idea of adding end-to-end encryption for notes. This would also include stronger authentication mechanisms, secure key management, and regular security audits. While this may seem excessive for a note taking application, the idea of large scale companies using our application might require such measures.
    - Next, our projet relies on third-party libraries which we use for rendering and geometry-related operations. While the libraies we use are robust and trusted, they are general-prupose tools and not optimized for our application specifically. If we were given an unimited resource such as time, we would like to design and implement custom solutions/libraries tailored to our application's specific requirements. By doing this, we believe it would help improve performance, flexibility, and control over rendering behaviour.

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)

- Other design solutions we considered was using a primarily text based note taker or using alternative input methods in combination with a keyboard. As our problem is pretty constrained we did not consider many other options and the solutions proposed by our team were very similar.
- The benefits of a primary text based note taker was that it would be easier to code and much more intuitive to users who have used similar software such as notepad or word. However, making it too text focused makes it too similar to the above mentioned programs making ours not valuable for those that can access these other common actions.
- Another solution we considered was using alternative input methods such as the microphone to gather input along with the keyboard. It could transcribe the lecture while giving the user the ability to input. While it would be helpful in gathering information for the note taker, it would also fall outside of our primary goals for the project and could only be considered a stretch goal that could be pursued after the primary goal of the project is properly implemented.

7. (After you have implemented another team's module, which means this isn't filled in until after the original deadline). What did you learn by implementing another team's module? Were all the details you needed in the documentation, or did you need to make assumptions, or ask the other team questions? If your team also had another team implement one of your modules, what was this experience like? Are there things in your documentation you could have changed to make the process go more smoothly for when an "outsider" completes some of the implementation?

   Implementation: https://github.com/SpaceY-Labs/RoCam/pull/262

   The other team's module was well specified and straightforward to implement. We learned how helpful and important clear documentation is for collaboration. We were able to implement the module with little knowledge of the project as a whole and without needing external context for how this module would fit into the larger system. Almost all the details we needed were in the documentation. A couple areas where we made assumptions included the format of the logging files as well as assuming the referenced "OSDData" object structure was JSON. We definitely could have added more detail to our documentation to make it easier for the other team to implement. Writing out steps in plain english made it clear for us what to implement. We also could have specified the exact file path for them to implement. Additionally, the module we were assigned was rather small and easy to understand in isolation. We could have done a better job of breaking up our modules to allow for this type of isolated knowledge requirements during implementation.