StudyPlanner Phase 1
Design Document, Specification, and Progress Report

by Group dinosaurs

Updated Specification

StudyPlanner allows a student to track various checklists of tasks, and to organize their active study time and break time into study blocks. Students can create Tasks of different weights (what each Task is worth as a percentage of their grade), importance (a 5 point system of assigning priority), due dates, and lengths (time to complete). Students also create one or multiple Checklists, which automatically sort tasks.

Once the Student has added some of their open Tasks, they can create a StudyBlock that will give them a detailed breakdown of how to organize their time while completing the Tasks. The StudyBlock assigns active time and break time until time runs out or all tasks are completed. The Student can save their preferences for how they would like to divide active and break time. For convenience, StudyBlocks are exportable in a common calendar format so that Students can access them on the go.

Major Design Decisions

We expanded on the original specification by allowing the user to implement multiple Checklists, and from this they can create StudyBlocks from one or more Checklists. This allows for flexibility in the program. For example, some group members expressed interest in being able to use one checklist per day, while others preferred to have one combined checklist of all tasks. The TempCreator function allows for both styles of usage to be incorporated. We have also added a data persistence functionality where the user is able to store their locally created Checklists and Study Blocks onto their device and load them in when the program opens.

In addition to the features we had in our original specification, we have added a GUI interface which allows the user to view and manipulate their Tasks, Checklists, and StudyBlocks. The GUI was much larger in scope than we anticipated for this phase of the project, and we were not able to implement all the features that we laid out. However, most of the GUI design decisions have been made now, so we can move ahead quickly in Phase 2.

We made a large change from our initial design where we anticipated having to build our own calendar representation. Since calendars are quite complex and we needed to guarantee the accuracy of our schedulable blocks, we shifted to a Schedulable interface that can export the StudyBlocks our program creates (using the Biweekly library https://github.com/mangstadt/biweekly.git.) so that users can import them into their preferred calendar software.

We also switched from the ZonedDateTime format to a LocalDate format. This decision was somewhat controversial, as one of the benefits of ZonedDateTime is the incorporation of a time (which would be very useful for assignment deadlines -- right now every deadline is 12:00am on the due date) and the benefit of adapting to different time zones around the world. However, in an effort to use a cleaner input system which is more user friendly (e.g. calendar drop down versus typing out time, timezone etc.) we decided to switch to LocalDate and give up the extra functionality.

How the Project Adheres to Clean Architecture

We placed a high priority on ensuring our inner layers were never dependent on outer layers. For example, we can consider creating a new Task. The interface of our GUI is always separate from the Controller that it depends on. The Controller only interacts with our Task and Checklist entities through the TaskManager use case class (in this example, addTaskHelper() and addTask()). After creating the Task, the TaskManager is used again to add it to a Checklist in the Data class. Once in Data, the GUI can access the Checklist and Task to display them, or they can be saved to disk when the program exits. This example demonstrates that data from user inputs flows inwards to our entities and then back out to our controllers and drivers, without any skipping of layers.

How the Project Adheres to SOLID Principles

The Task, Checklist, Student and StudyMethod classes from the entities package follow the single responsibility principle; Task just saves all the information like weight, etc. of the Task created, Checklist stores Tasks based on priority and completion, Student stores the Student's information and StudyMethod stores the Student's preferred study method. Hence, all 4 of these classes perform only 1 function. The saver and manager classes also perform just one function, following the single responsibility principle.

The Open/Closed principle is shown by the Schedulable interface as the methods for that are implemented in StudyBlock not directly in the interface. Moreover, the managers for Checklist, Task, the various GUI controllers, and the savers for Studyblock and Checklist are all examples of this principle as well because they do not add directly to the classes they are managing, just control/organize them.

The Schedulable interface follows the Liskov Substitution Principle as no matter what type of event it is, a calendar will need to be created, so will an event and it will need to be written up as an ICS file. Hence even substituting the type or event or studyblock will still result in the interface working.

We have designed our program so that inner layers are never dependent on outer layers, so we haven't implemented many interfaces between layers or needed dependency inversion. However, we could create more interfaces to better define the layers themselves, instead of simply relying on grouping them into the appropriate packages. Moreover, since we haven't used a large amount of interfaces and our interfaces are not very large, the interface segregation principle has not been required thus far.

Design Patterns Implemented (or Planned)

In the GUI, we did not implement any design patterns ourselves. However, we do use Observers in the case of ObservableLists that update the various ListViews throughout the program. We also implemented Iterable in our Checklist class, and rely on its implementation in a number of ArrayLists that store instances of Checklists and StudyBlocks.

Command could be implemented in the StudyMethod class to schedule study and break times and log what the student is doing during that scheduled study block. The StudytimeStartCommand will implement command and start study time. The same thing can be done to end it or start and end break times in StudyBlock. It can also be used to change study methods during the session or record the one currently being employed.
Alternatively, when students make a request i.e. add a task in this case, the command class will call the execute method and store the details of the task. These details like weightage, estimated time to complete, due date, etc. will be passed into client as parameters. Finally the receiver will create the task. This way tasks will be created cleanly and stored in TaskManager.

The Factory design pattern could be used for Checklist and StudyBlock saver and read writer classes as ChecklistSaver and StudyBlockSaver are very similarly written save for what they are saving and so are ChecklistReadWriter and StudyBlockReadWriter. Instead of having separate classes like this, we could create Saver and ReadWriter superclasses with subclasses for the type of object they are saving.

Use of GitHub Features

During this phase, we switched our github branching naming pattern from using our usernames to descriptions of the area and feature we were implementing on that branch. However, we did not use a single style convention, so while the names were more descriptive, they were not unified. We also began to push and pull branches between various users before merging them. During Phase 0, the complexity of the program was low enough that we were satisfied by reviewing a branch's code on github before merging. However for Phase 1, especially after the GUI was rolled out, pulling branches between one another allowed us to better review and test them before they were merged into main. We currently do not use any of github's notification features, instead we rely primarily on our group WhatsApp where it is easier to reach each other quickly.

Code Style and Documentation

Our code is generally well-documented and clear, and follows Java naming conventions.

Testing

We try to write test cases in parallel with writing the code because it helps us debug when needed. In this phase of the project we were able to have the following tests; ChecklistTest, ReadWriterTest, StudyBlockTest, TaskMangerTest.

ChecklistTest checks if a new checklist can be made. TaskManagerTest checks for the organizations of tasks in a checklist based on Length, Importance, and Weight. ReadWriterTest assures that checklists and StudyBlocks can be saved in the program. While StudyBlockTest tries different possibilities of tasks, and their lengths, along with different StudyBlock lengths. It also tests for a different studyMethod.

Refactoring

Our entire user interface was rebuilt in JavaFX, replacing our old Main file with a new MainGUI and various controllers. However, there is currently a fair amount of duplicate code in the GUI controllers for buttons and pop-up windows that are repeated in multiple scenes. Now that we have locked down our GUI design and have more experience with JavaFX, in Phase 2 we can refactor to eliminate the duplication. Many of the helper methods like the comparators and read writers have very similar code so these could be refactored into subclasses with a comparator or read writer superclass. This can also be done with the saver classes as much of the code is duplicated.

Code Organization

Packaging Strategies
We packaged our project according to clean architecture to ensure we were adhering to it with all entities in one package, all use cases in one package and so on. This strategy also enabled us to easily check our imports for each class to gauge whether it was accessing or dependent on any classes in a way that violated the principles of Clean Architecture.

Functionality

We have implemented almost all of the functionality that we set out in our specification, although there are still areas of code (like creating and exporting StudyBlocks) that have not been linked into the GUI. We are confident that once everything has been implemented in the GUI we will have a strong base from which to expand our scope during Phase 2.

Progress Report

Open Questions:

What database functionality would be appropriate for scope? Student/Tasks/Schedule? History of tasks and/or StudyBlocks?
How should we store local memory in order for it to adhere to clean architecture? Right now, we have a controller class Data which stores all of our information, but should this instead be moved into Entities and then manipulated by the controllers using use cases?

What We've Been Working On: Oct. 25 - Nov 1st

We all met on Friday afternoon to review our presentation and plan next steps, assign work equally, and brainstorm ideas for extending functionality. We considered redesigning the calendar portion of the program to work with ICS file import/export so that it would be able to work with users' existing calendar apps and also be more portable.

Eric: Redid the implementation of Checklist and TaskManager in order to reduce cohesiveness (i.e. created checklist to be more entity-like). I plan on moving ahead with Kenneth in order to implement the main study block method so it can take into account how long the User has to study for, as well as what to do when a task is finished.

Jeb: Researched basic Java GUI implementations and Swing/JavaFX. UI's require a different way of thinking about control flow that has to be reflected in our program's design. Aiming to implement a button and text field-based GUI for Phase 1, but first steps for this week are to get JavaFX up and running, get comfortable with its architecture and research UI/UX basics that are applicable to our program.

Paridhi: Researched on various Study Methods which we wanted to implement. But then we decided that we only wanted to keep 3 methods, and rather have users be able to customise their study hours and break hours. So I made the necessary changes to StudyMethod.

Zoya & Pooja: Researching how to do the ics exporting. We went to multiple different websites including git to try and download and run ical4j and understand how it works. Tried adding different jars and dependencies to get it functional. Went to office hours to ask about how to make it work after adding the jars for it and conversed with a TA about it as well.

What We've Been Working On: Nov 1 - 8th

Eric: Worked on the implementation of study block
Zoya & Pooja: Worked on figuring out the ics exporting, moved from trying to use ical4j to using biweekly. Added a method that saves comments/ notes so they appear once the export is opened in ical.
Ken: Added a new variable to the assignTasks method which allows us to keep track of how much time is left in a students task after a study period.
Paridhi: Started learning JavaFX
Jeb: Learning JavaFX, troubleshooting maven/lib issues
Paridhi: Made changes to Study method

What We've Been Working On: Nov 8 - 15th

Eric: Made much of the underlying business data serializable, intend to keep implementing load and save functions within the mainGUI program. Also worked to make the temp creator helper function and added an Iterable function onto checklist.
Zoya & Pooja: Made schedulable an interface (it was previously a class with a main method) through the use of biweekly and information from the Studyblock class which implements the interface. Added instance variables to the instantiation of studyblock and added methods related to the schedulable interface. Also installed JavaFX.
Jeb: Completed layouts of all scenes in program, navigation between scenes. Working with Paridhi to add all button methods and string displays of program output.
Ken: Finished assignTask(), which allows for the studyblock to be filled with the tasks in a checklist based on given time and how the user wants to divide up their study time (ie. Pomodoro and Desktime).
Paridhi: Beginning work on study block

What We Will Work On Next

Pooja: Continue working with the schedulable interface - try to add it to the GUI and get ICS exports to work in the main GUI.
Paridhi: Plans to work on StudyBlock next.
Ken: Plans to add new ways to represent a studyblock. This new studyblock takes in a starting time and creates the block using that time.
Eric: Continuing to implement the rest of the GUI buttons, and work on streamlining the serialization loading process. Right now, I have the idea of creating a folder for checklist and one for studyblocks and just having the entire folders loaded in instead of selecting multiple files.
Jeb: Continuing to refine and implement all features of the GUI, and add database support for our persistent data. Testing edge cases to ensure that all elements of the GUI work appropriately, and do not cause exceptions or other display issues.

What Worked Well?

We have strong communication among the group members which is working really well in organising the group meetings, coordinating and updating one another about the progress on the project. Our communication especially on the group whatsapp worked very well as it was easy to get in touch with and discuss the project with any group member.
Specifically in the project, the MainGUI is working quite well as it opens a new app and allows us to look at tasks and multiple different checklists. There is still scope of improvement which we plan to cover up in the following few weeks. We were skeptical about adding multiple checklists initially but the decision to add multiple checklists has worked in our favour, with users getting the option to manage them and add or remove them so users can organise their work however they want.