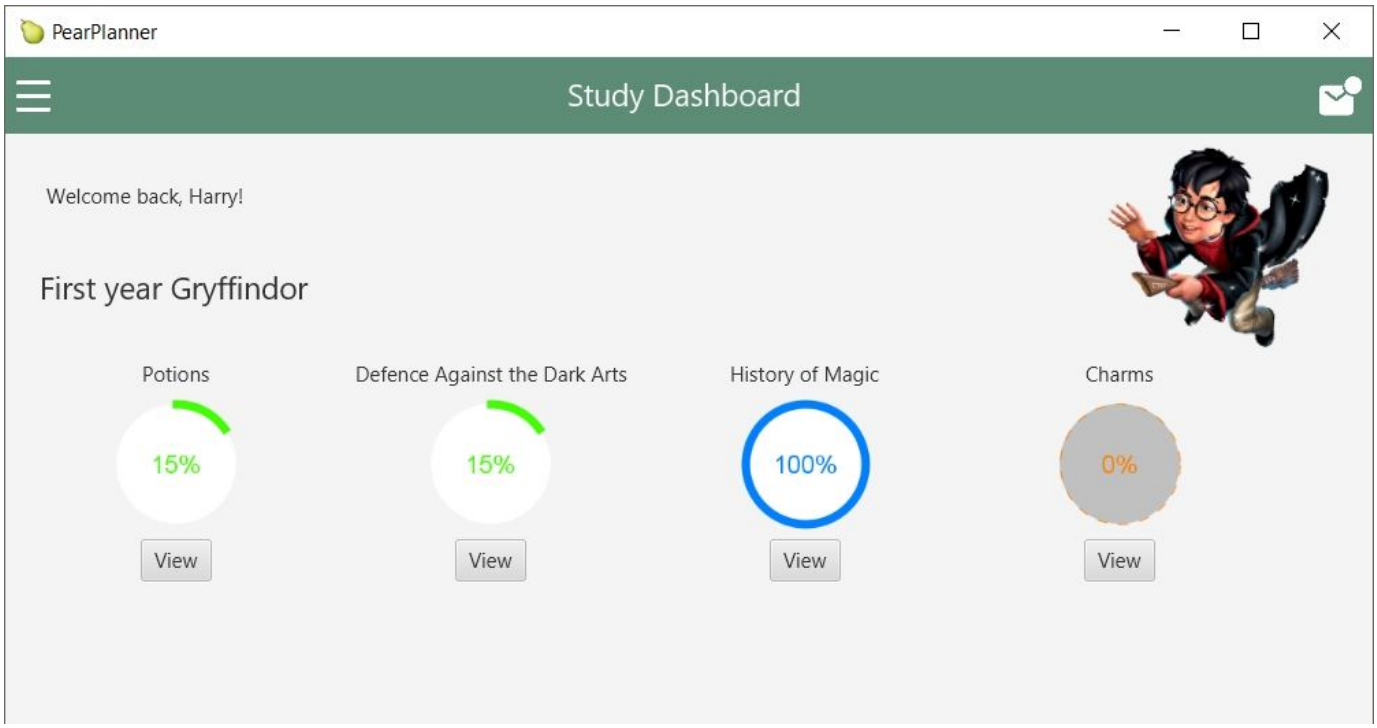
⌜ Planner



the best study planner since the Ganttish Chart

MoSCoW Analysis

Below we have documented the provided MoSCoW Analysis and how we feel that our prototype either satisfies the requirements or how it has the groundwork and frameworks in place to be developed further to meet them fully in the future.

Must

Be able to load a semester file containing module, coursework and deadline information, and exam information from a defined file format

- Our system uses an XML file format containing all of the modules, coursework and deadline information
    - a study profile
        - year/semester number
        - list of modules
            - timetable
            - assignments
                - coursework
                - exams
    - list of assets used above
        - people, rooms, buildings, timetable event types

Be able to define study milestones which must be attached to coursework or exams

- The user has the ability to define study milestones in the Milestones menu and attach Tasks to it. Since tasks are part of assignment (coursework or exam), therefore the milestone will be attached to an assignment as well, but the user will be able to have milestones that span multiple assignments.

Be able to define study tasks contributing towards specific coursework or exams

- User can add study tasks by navigating to a specific assignment. Tasks will contribute to the overall progress of that specific assignment.

Be able to store defined milestones and tasks on disk as part of the semester profile

- Milestones and tasks, together with all the other user added data, are always serialized and saved to a file upon exiting the system. Once the user loads up the system, the saved file will be automatically read and all the data processed to be displayed in the Study Planner.

Should

Be able to record study activities that are related to study tasks and contribute towards completing milestones

- Our system allows the user to record study activities and attach tasks to them. When

recorded, study activities automatically contribute to the attached tasks and tick of which requirements of that task have been completed. Study activities also contribute towards completion of milestones and assignment requirements.

Be able to store activities in the semester profile file

- Activities, together with all the other user added data, are always serialized and saved to a file upon exiting the system. Once the user loads up the system, the saved file will be automatically read and all the data processed to be displayed in the Study Planner.

Have a study progress dashboard that highlights upcoming deadlines, progress towards completing milestones and time spent for each module

- The main window of the system is a study dashboard which highlights progress towards each module.
- Upcoming deadlines are highlighted in the Calendar and the system also has the functionality to automatically notify the user about upcoming deadlines via the notification module, although this feature was disabled in the current release because of the lack of testing and not knowing whether it is robust enough to be released yet.
- Time spent for each module is also recorded and displayed in the Modules window, for each individual module.

Be able to visualise activities, dependencies, milestones and deadlines in a Gantt chart representation

- Our data structure meant writing an algorithm to render a Gantt chart would have proven very difficult to achieve by the deadline and at the expense of many other features. It is possible, as all tasks are assigned a deadline and all requirements have a required time value. Tasks also have dependent tasks listed, so our data structure has all the information needed in order to create a proper Gantt chart.
- For the demonstration, we created a visual representation that provided a similar and useful representation of the tasks. We informally called it a “Ganttish Chart” - although it was pointed out to us that is is similar to a Kanban diagram (although flipped left/right)
    - It shows, left to right:
        - Completed tasks
        - Tasks started
        - Tasks that can be started (ie dependencies completed)
        - Tasks that can’t be started (ie dependences not yet completed)
    - The left to right inversion to a Kanban style is to make it more similar to a Gantt chart, where earlier tasks are shown on the left and later tasks on the right.

Could

Be able to process updated semester files with new deadlines and updating the stored activities, tasks and deadlines

- We added the ability to update information in the study planner
- This is in the format of an xml file with a different schema
    - version
    - extensions (not implemented in prototype)
        - would contain a list of approved or denied extensions to assignments and

add this information to the appropriate data
    - updated items
        - person
        - building
        - room
        - module (not implemented in prototype)
        - exam
        - coursework
        - timeTableEventType
        - timeTableEvent
        - event (not implement in prototype)
        - deadline (not implemented in prototype)
        - examEvent (not implemented in prototype)
    - new items (not implemented in prototype)
- other than managing extensions, the underlying code for the update files has been written, as this is all part of our VersionControlEntity class. This allows objects to be updated safely

Be able to highlight the critical path of activities within the Gantt chart

- Until we implement a full Gantt Chart, this obviously cannot be met, and is the requirement least fulfilled in the prototype. However, as we have many methods within our controllers and model classes for determining completion, time taken, etc... again, we have most, if not all of the data needed to construct a critical path, however we admit we have not yet designed the algorithms for visually representing this on a screen.

Be able to visually highlight progress made in the Gantt chart based on study activities

- Although we don't have a Gantt chart, our Kanban style chart does visually highlight progress made in the form of what has been completed, started or not started.
- It also shows how how complete each task is, as a percentage, in the form of a circular dial.
-

Changes to Classes

Introduction

As the development of the software began the development team noticed a need of change to the structure of the system. It is crucial to document these changes and update the design documents to make our project easy to understand for further development by potential developers in the future. In this section, we have included classes that have faced changes in the implementation stage.

Deleted Classes

- FileSystem – From: Data I/O
- UIEvent – From: View
- UIElement – From: View
- UIView – From: View
- EventController – From: View

Replaced Classes

- ExtentionApplication – From: Controller To: Model
- HubFile – From: Controller To: Model
- Notification – From: View To: Model

Added Classes

- MultilineString – To : Model
- AccountController – To : Controller
- ActivityController – To : Controller
- MenuController – To : Controller
- MileStoneController – To : Controller
- RequirementController – To : Controller
- StudyProfileController – To : Controller
- TaskController – To : Controller
- XMLController – To : Controller

- ConsoleIO – To : View
- GanttishDiagram – To: View

Testing Report

Introduction

Testing is a very crucial part of any software development project; it is at this stage that a development team makes sure that the functionalities are working as they are expected to. While developing the study planner, the development team used a variety of tools to test the product while developing it.

Java Unit testing

Our have implemented wide range of test for the study planner. Those tests are covering all the software functionality aspects. We are using JUnit as a testing framework.

JUnit

JUnit performs series of tests covering most of the code and ensures that all the classes are behaving in expected way.

Java UI testing

We have implemented some basic UI tests using automated user interface testing framework TestFX. It allows programmatic simulation of user interaction with the UI as a part of JUnit testing routine.

Travis CI

Our project is taking a full advantage of Travis CI. The system is building and running tests on every git commit and pull request on disposable virtual environment. It helps us to find any possible bugs associated with running a build on a different environment (e.g missing libraries). Travis is taking advantage of Apache Ant library to build the project using build.xml file provided. Travis CI is also capable of performing UI test using X Virtual Frame Buffer.

Test items

The main items that were considered to be critical for testing are as follows:

- Model
    - Account
    - Assignment
    - Event
    - HubFile
    - ModelEntity
    - Module
    - MultilineString
    - Person
    - Requirement
    - StudyPlanner
    - StudyProfile
    - Task
    - VersionControlEntity
- View
    - CreateAccount
- Controller
    - MainController
    - StudyPlannerController

Features to be Tested

The development team considered the limited time for developments as well as the scale of the product and decided on the features that needed to be tested. The features that were decided to be tested were the core functions of the software that were used constantly by other parts of the software. These features should not be too simple and impossible to fail.

Features not to be Tested

Based on the limited time for development and the scale of the product the development team needed to decide on features that were not as critical for testing. These features were too simple and impossible to fail such as getters and setter with no processing. Some of these features were also dependent on future development to be ready for testing.

Approach

The development team divided the team members into two groups, one for implementing the software, and one for testing the written functions. After each short iteration by the implementation team the testing team would test the written functions by unit testing using Junit and the following stage would be integration testing which requires all the tests written for modules to be tested together to confirm that they work together as a whole. At the end of each testing iteration the new build would be pushed to the GitHub account and a system testing would be done by Travis CI.

Item pass/fail criteria

The pass/fail criteria for each unit of the software that was needed to be tested are as follows. Each test item was expected to work as needed for the system and the pass criteria for testing that unit was that if the function works as expected. Furthermore, the unit would be tested to make sure that it would not process any data that would be inconsistent to the rest of the software which were considered to be the fail criteria for the test.

Responsibilities

The development team responsible for testing the software divided the responsibilities between the two members with one responsible for unit testing and integration testing and the other responsible for system testing to make sure the software has a successful build and works properly.

Coverage Matrix

Element	Class, %	Method, %	Line, %
com			
Content	100% (0/0)	100% (0/0)	100% (0/0)
Controller	11% (4/34)	3% (9/226)	2% (37/1698)
css			
font			
fxsampler			
java			
javafx			
javax			
jdk			
jfxtras			
junit			
META-INF			
Model	34% (12/35)	16% (64/380)	16% (227/1395)
netscape			
oracle			
org			
roboto			
sun			
View	16% (1/6)	1% (1/58)	0% (3/576)

Conclusion

In conclusion, the implementation and testing parts of the development were dependent on each other to deliver a final finished product. Testing stage helped the development team to make sure the system works as expected, an example of this would be a bug that was found in a get method in MultilineString class regarding casting an array of objects to an arrays of Strings.

Implementation Report

During the study planner development process, two main iterations was established

First iteration

During the first iteration the base class skeleton of the program was created and filled with the naive method implementations. UI haven't been created at this point, console I/O was used instead to develop the main functionality of the software.

Second iteration

During the second iteration we were focusing on building a comprehensive UI and making the software more stable. Naive method implementations were replaced with a proper safe implementations. Unit testing verifying the behaviour of the software and automatic deployment using Travis-CI were implemented at this stage.

Development Process

Development Method

Our team undertook the implementation stage in an agile manner, with quick and short iterations followed by testing. The members of the team would attend meetings two times a week to discuss about the changes that the software had faced and decide on the new tasks of each member, since face to face conversations between the members of the development team would help to clarify on requirements and tasks. The development team would often communicate with the client to get clarifications on requirements since it is a crucial part of the development of a system.

We decided to use GitHub for our Version Control as all of us had used it before.

We also decided to switch from NetBeans to IntelliJ as this was a more modern IDE and we felt we would benefit from many of its new features.

Allocating and Prioritising Tasks

The development team allocated members tasks based on two essential development stages, implementation and testing, two members of the team undertook the responsibility to implement the functionality of the software and the other team members undertook the testing responsibility while helping with some functionality implementations. The development team prioritised the implementation of the system in such a way that in each short iteration the system was testable, starting with the Model classes to create the structure and then implementing the Controller and View classes.

We decided to use GitHub for our Version Control as all of us had used it before.

We also decided to switch from NetBeans to IntelliJ as this was a more modern IDE and we felt we would benefit from many of its new features.

Ben

- Project Team Leader
- Implemented version control and file importing/processing
- Implemented the Ganttish Diagram
- Created sample XML files and came up with their internal structure
- Created a Console IO system for testing and debugging
- Worked on implementing individual models and controllers
- Documented the development process in the final report

Andrew

- GitHub Co-ordinator
- Managed GitHub (implemented Travis CI, automatic binary deployment)
- Implemented the Calendar module and made it interface with our current data structure
- Improved the aesthetics of the UI design
- Reviewed JavaDoc
- Implemented TestFX for testing UI windows

Bijan

- Unit Testing Coordinator
- Implemented JUnit and wrote tests for various modules of our software
- Kept the previous diagrams up to date and remade some to reflect all changes that were made during the development process
- Planned the initial structure of the final report
- Created additional XML files for testing purposes

Zilvinas

- Java FX Team Leader
- Implemented JavaFX view files (.fxml)
- Worked on implementing individual models and controllers
- Wrote the UI controllers and the UI manager
- Wrote Java Doc and documented the code
- Created additional XML files for presentation purposes
- Wrote additional methods for Models to interface with the Controllers

Future Improvements

While writing the software we came across many better ways of doing things that weren't in our original plan. Where these were discovered early or when they were simple to change, we implemented the improved ideas.

Where they proved more difficult to change, due to the amount of work needed, we had to make a decision on whether to continue with our original, less efficient solution, or change to the more optimal idea, or whether to compromise.

Also, because of the fact we were using a lot of technologies for the first time, such as implementing a GUI in java or processing XML files, we had to balance the time required to learn all the features versus implementing with more limited knowledge.

There were also cases where we hard coded data into the source code, where in a final release it would be saved in a settings file. We have documented in the code where this is the case.

A good example of this is our file import system.

We decided to use XML for our files. Being familiar with basic XML file structures, and the file format being easy to edit in a text editor we felt it was ideal for creating study profiles with detailed data without needing to creating software to create the files.

However, taking the XML data and creating all of the associated Java classes was a new skill, and our first approach is not the method we would have implemented if starting again, however changing to that system would have proven too expensive in terms of time. So it was more important to have a working system, particularly as it was robust, it was just less efficient in terms of code and more difficult in terms of maintainability (although this was improved towards the end as we used generics to write methods for doing repeated processes on classes extended from VersionControlEntity.

The solution we would have implemented in the end, would have been to have created a constructor for an object based on an XML NodeList, for example Person(Nodelist fromXML) -

Instead, we have a method that parses the entire XML file, extracts the data and calls the regular constructor.

The reason we did not change to this for the prototype is because it would have required writing and testing many new constructors for all of our objects, and we had a parser that worked correctly. Instead it was easier to optimise the parser using generics.

We also wrote our our methods for validating the XML, and extracting the data (using the methods written in XMLcontroller) and hardcoded schemas using these methods in the HubFile class.

While we appreciate that there are probably already many inbuilt methods to do this, it was more efficient for us to write our own code as we only needed a limited number of specific features.

For long term maintainability we would update the code to use official libraries where possible as this would give code that is easier for other programmers to modify.