**Troy High School**

**AppDossier: TheGoodEnoughPlanner**

****

The BBYs: Darin Kishore, Damon Lee, Justin Lee, Benjamin Lium, Patrick Rim

Period 4

**Defining the Problem**

|  |  |
| --- | --- |
| The client, Alex Liu, is a student that has trouble organizing his things-to-do and prioritizing tasks. It is also difficult for him to keep all of his events scheduled because he has so many of them. This is especially troubling for Alex because he has to stay on top of his workload and maintain healthy time management as a junior in high school.  Alex is currently using a planner to schedule, which is slow and does not provide a good visualization of what needs to be done and how much time is left to do it. This makes it harder to get a grasp on the urgency of certain tasks and overall is an insufficient solution.  Our group has reached out to Alex to provide a solution for him by creating a computer program. By conducting an interview (refer to evidence), we found the specific expectations that Alex has for the program. These include being extremely intuitive and easy to use, having a user-friendly and appealing design, and being able to create lists and recurring tasks. He also requested that the program could be used very quickly for efficiency purposes. | |
| Word count: | 190 |

**Rationale for Proposed Solution**

|  |  |
| --- | --- |
| Our program will use the Java programming language because of its ease of use and its ability to cover all of the functions needed for our program including the algorithm used to sort and order tasks, the graphics that we will need to create a visually appealing interface (most likely using JavaFX), and others.  We decided not to create a solution on a mobile app through a program such as AppInventor because it falls short of the capabilities of using Java. AppInventor provides limited design possibilities, is slower in function with long loading times and lagging issues, is harder to test as an emulation, and would simply not be able to meet the expectations determined by the client as thoroughly as a program made in Java would be able to.  Our program will provide an easy-to-use interface that allows the client to visually sort and prioritize his/her tasks as well as to easily modify the to-do list and schedule. The graphics will most likely consist of simple labels and text boxes organized in an appealing fashion. We will implement an algorithm that will sort added tasks and plans without the manual instruction of the user, thereby producing ease of use.  Ultimately, our solution will provide the client with an alternative to a handheld or even digital planner/list that provides greater accessibility, efficiency, and functionality. | |
| Word count: | 226 |

**Criteria of Success**

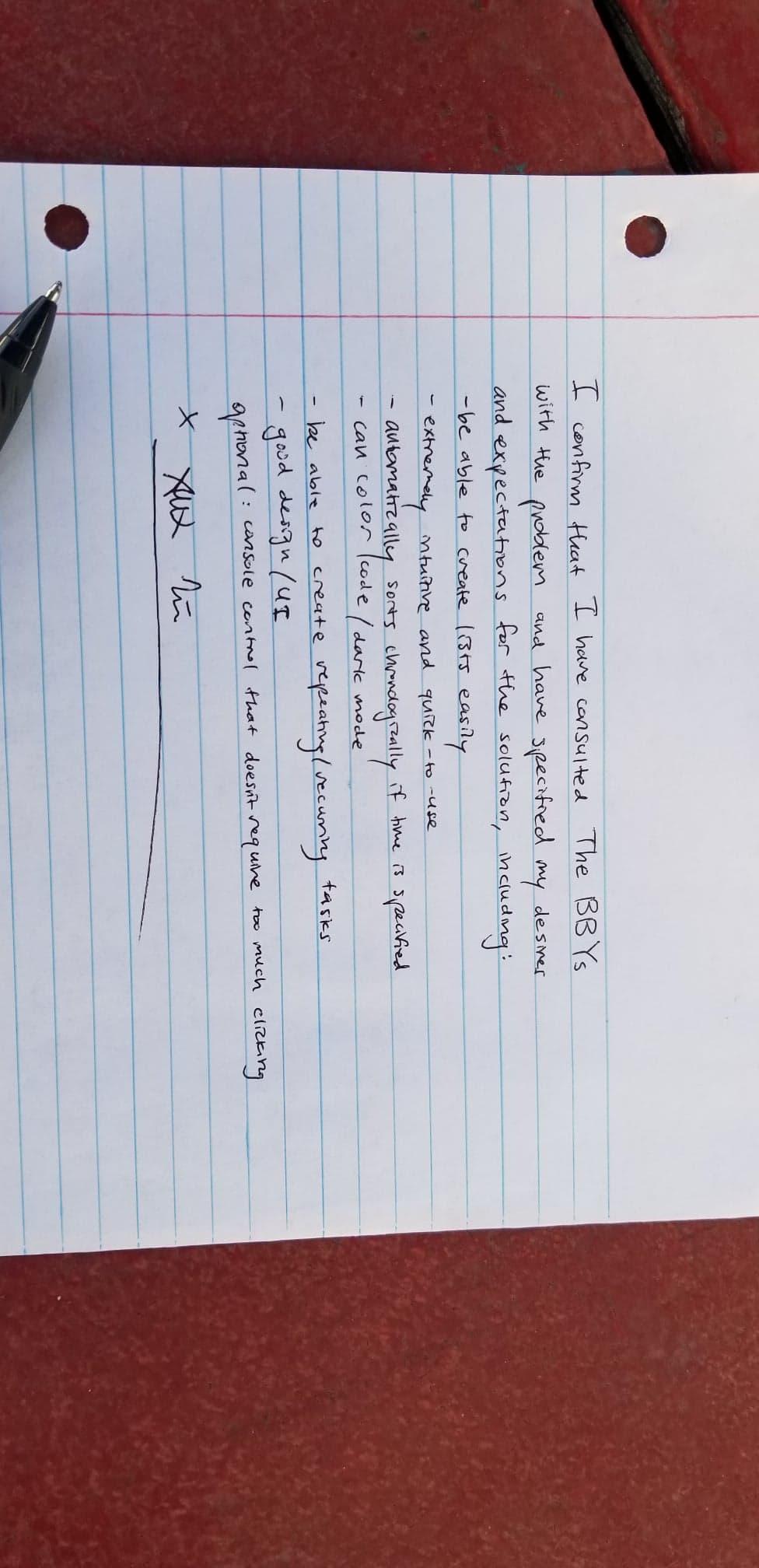
1. The program will be able to accept inputs to add events.
2. **The program will be able to order a series of events based on date and time.**
3. The program will be able to organize and view events both in calendar form and in list form.
4. **The program will have an intuitive and accessible design.**
5. The program will perform its tasks quickly and efficiently.
6. **The program will highlight tasks that the user designates as priorities.**
7. The program will allow the user to color code different tasks.
8. **The program will be able to display its information in both a dark and light mode.**
9. The program will allow for the creation of tasks that repeat themselves at preset intervals.
10. **The program will be able to report statistics regarding tasks.**
11. The program will be able to guide the user with its timer functionality.
12. **The program will support both GUI and CLI operation.**

**Test Plan**

**Dossier | Criterion B**

|  |
| --- |
| 1. Input an event into the program. 2. Input two events with different dates and see that they are sorted chronologically. 3. Toggle between the calendar and list forms of the program. 4. Navigate throughout the program clearly and efficiently. 5. See that the program runs smoothly without long delay, loading, or lag. 6. Mark a task as a priority and see that it is highlighted. 7. Set different tasks to different colors. 8. Toggle between dark and light modes effortlessly. 9. Create a task that recurs every day and see that it refreshes. 10. Look at the time left, amount left to do, etc. of a specific task. 11. See that the timer function is accurate for each task set with a time/date. 12. Operate the program using command line input with the same functionality. |

**Evidence of Interview**

**Dossier | Appendix**

|  |
| --- |
|  |

**Record of Task**

**Dossier | Criterion B**

*Record of Task must should be guided to explicitly address all 5 stages (plan, design, develop, test and implement). The record of task must be on attached Record of Task Form.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Task number | Planned action | | Planned outcome | | | | Time estimated | | Target completion date | | | Criterion |
| 1. | Brainstorming of possible project ideas | | Find a problem and client to work with and determine possible solutions. | | | | 1 day | | 01/22/2019 | | | A |
| 2. | Contact client | | Discuss interest that the client has in a computer program solution to the problem. | | | | 0.5 day | | 01/24/2019 | | | A |
| 3. | Conduct interview with client | | Determine the client’s desires and expectations for the product (take notes) and collect evidence of interview (signature and picture). | | | | 0.5 day | | 01/24/2019 | | | A |
| 4. | Create appropriate criteria of success | | Develop a range of criteria of success that are crucial to the product according to the client’s expectations | | | | 0.5 day | | 01/26/2019 | | | A |
| 5. | Create test plan | | Develop a list of specific tests to execute to verify that the program meets all the criteria of success | | | | 0.5 day | | 1/28/2019 | | | B |
| 6. | Create Use Case Diagrams | | Represent the user’s interaction with the system and establish relationship between user and different use cases | | | | 1 day | | 1/29/2019 | | | B |
| 7. | Create Input Process Output Chart | | Represent structure of information processing in the program | | | | 1 day | | 1/30/2019 | | | B |
| 8. | Create Context Data Flow Diagram | | Broadly represent the general flow of data through the entire system in relation to external entities | | | | 1 day | | 1/31/2019 | | | B |
| 9. | Create Level 1 Data Flow Diagram | | Represent a more detailed flow of data including subprocesses | | | | 1 day | | 2/1/2019 | | | B |
| 10. | Create Hierarchy of GUI/UX Chart | | Represent the user interaction and experience and how they can navigate the program | | | | 1 day | | 2/2/2019 | | | B |
| 11. | Create Event Flow Diagram | | Represent the flow of events and how the user interacts with the system | | | | 1 day | | 2/3/2019 | | | B |
| 12. | Create prototypes of product | | Develop a collection of radical prototypes to visualize ideas | | | | 1 day | | 2/4/2019 | | | B |
| 13. | Compare different prototypes and combine elements | | Select a final prototype with various features from different prototypes for optimal functionality | | | | 1 day | | 2/5/2019 | | | B |
| 14. | Create Chart of Build of Features for Product | | Show all the components of the product and what features they are needed for | | | | 0.5 day | | 2/7/2019 | | | B |
| 15. | Create System Flowchart Diagram | | Represent the flow of different types of data and processes in the entire system | | | | 0.5 day | | 2/7/2019 | | | B |
| 16. | Create Flowchart Diagram of Product | | Represent the flow of different types of data and processes within the product | | | | 0.5 day | | 2/8/2019 | | | B |
| 17. | Create Flowchart Diagram of Sub-processes | | Represent the flow of different types of data in a specific subprocess | | | | 0.5 day | | 2/8/2019 | | | B |
| 18. | Create Modular Diagram | | Represent the different modules and subdivisions of the system | | | | 0.5 day | | 2/9/2019 | | | B |
| 19. | Create Inheritance and Class Diagram, UML Diagram | | Represent the hierarchy of classes and their actions in the system | | | | 0.5 day | | 2/9/2019 | | | B |
| 20. | Create Data Dictionary, Variable Table | | Represent all the data variables that will be used and examined | | | | 0.5 day | | 2/10/2019 | | | B |
| 21. | Create Database Structure | | Represent all of the data storage in one structure | | | | 0.5 day | | 2/10/2019 | | | B |
| 22. | Create CRC Cards | | Represent the different classes’ responsibility and collaboration | | | | 0.5 day | | 2/11/2019 | | | B |
| 23. | Create list of features to build | | Have a set checklist of features that need to be implemented in the product in order to meet the criteria of success | | | | 0.5 day | | 2/11/2019 | | | B |
| 24. | Create a schedule for building the product | | Have a general schedule on what will be worked on and when, up to the deadline | | | | 1 day | | 2/12/2019 | | | B |
| 25. | Create the GUI design | | Design a user-friendly, appealing graphical user interface. | | | | 3 days | | 2/16/2019 | | | C |
| 26. | Code the task list | | Create the code for the task list function so that a user can add, edit, and remove tasks. | | | | 1 day | | 2/17/2019 | | | C |
| 27. | Code the calendar function | | Create the calendar display and code its interactive features. | | | | 3 days | | 2/20/2019 | | | C |
| 28. | Code the timer function | | Create the code for the timer function to allow the user to keep track of time periods. | | | | 1 day | | 2/21/2019 | | | C |
| 29. | Code the statistics function | | Create the code for the statistics function to allow the user to get different statistics about his or her tasks. | | | | 1 day | | 2/22/2019 | | | C |
| 30. | Implement the database for data persistence | | Have a database that keeps persistent data and functions correctly. | | | | 3 days | | 2/25/2019 | | | C |
| 31. | Test and debug the code | | Find as many errors in the code as possible and optimize different functions. | | | | 5 days | | 3/2/2019 | | | C |
| 32. | Implement proper documentation for the code. | | Make the code easy to read and clear to understand. | | | | 2 days | | 3/4/2019 | | | D |
| 33. | Record video of program | | Prepare a video that demonstrates our program, testing each of the functions | | | | 3 days | | 3/7/2019 | | | D |
| 34. | Begin evaluation of product | | Compare our final project with the established criteria of success and client feedback to determine possible improvements | | | | 2 days | | 3/9/2019 | | | E |
| 35. | Begin future developments | | List any possible future developments for the program | | | | 1 day | | 3/11/2019 | | | E |
| 36. | Add appendix | | Add a final evaluation of the project as a whole in light of client feedback | | | | 1 day | | 3/12/2019 | | | E |

**External Design**

**Dossier | Part 2 | Criterion B**

**Use Case Diagrams**

**Dossier | Criterion B | External Design**

|  |
| --- |
|  |
| The user opens the application to access the home screen. From there, he can open either the calendar view or immediately create a new task. When a new task is created, it is stored in the Task Database. The user can then choose a task to begin timing. While timing, the user can abort or finish the task. In both outcomes, the database is updated to reflect the completion status of the task. |

**Input Process Output Charts**

**Dossier | Criterion B | External Design**

|  |
| --- |
|  |
| The Input Process Output diagram shows how our application (the process) can help solve the problems (input) to create a solved outcome (output). The three main problems the application aims to tackle are forgetting tasks, disorganization of recorded tasks, and the poor scheduling of tasks for completion. The application allows the user to record tasks, which gives them a framework to record all their assignments. Next, the user can organize the tasks by adding details, most importantly being the due date of said task. Finally, the timer function allows the user to hold himself accountable when he is attempting to accomplish his task. |

**Context Data Flow Diagram**

**Dossier | Criterion B | External Design**

|  |
| --- |
|  |
| Very simply, the way the data moves and is stored is completely dependent on the responses of the user. These responses will be stored onto the blank calendar that can be later accessed to retrieve the schedule. |

**Level 1 Data Flow Diagram**

**Dossier | Criterion B | External Design**

|  |
| --- |
|  |
| The data flow can be looked at more specifically. The user’s input of a task is created and stored as an event in the program. Based on the details that the user inputs, the task is scheduled and automatically sorted into chronological order (by due date). This is used by the system in order to give reminders to the user about upcoming deadlines. This flow of data makes up the foundation of the interaction between the user and the program. |

**Hierarchy of GUI/UX Chart**

**Dossier | Criterion B | External Design**

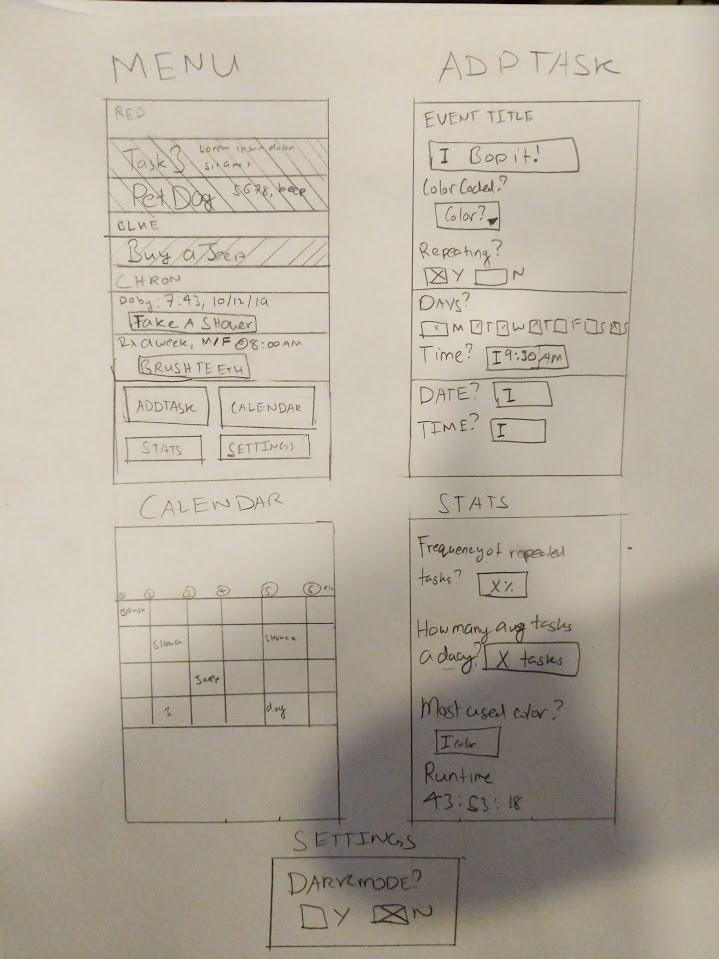
|  |
| --- |
|  |
| First, we can see that the default screen is the “week view”. From here we can either migrate to the year view, the month view, or the day view, which is where the task planner is. From each of these three screens we can switch easily back to the week view with an easy button click. From the day view, we can access the settings screen, which is where you can adjust settings such as notifications, the task limit, and others. From the setting screen, we can go back to the day view screen with a button click. From the day view screen, we can click the main “add task” button to access the “add task” screen. On this add task screen, we can ask tasks to our lists and pick settings such as whether or not we want a reminder for the task at a certain time on a certain day. After adding a task, the screen will automatically revert back to the day view screen. |

**Event Flow Diagram | UX Flow Diagram**

**Dossier | Criterion B | External Design**

|  |
| --- |
|  |
| The main screen functions to both display the tasks and allow access to different screens, each of which are accessible via the buttons at the bottom. The “ADD TASK” screen allows users to add a task and customize it to their liking, giving them the pick of color, title, repeating, and date/time. The “CALENDAR” screen functions to view tasks (including repeating ones) in a grid mode, allowing the user to contextualize the different tasks and how they fit into the user’s schedule. The “STATS” screen provides facts about the user’s usage of the app in hope of the user understanding their own usage of the app better. The “SETTINGS” screen simply allows for users to toggle dark mode. |

**Prototypes | Layout Diagrams**

**Dossier | Criterion B | External Design**

|  |  |  |
| --- | --- | --- |
| Darin Kishore  The main screen functions to both display the tasks and allow access to different screens, each of which are accessible via the buttons at the bottom. The “ADD TASK” screen allows users to add a task and customize it to their liking, giving them the pick of color, title, repeating, and date/time. The “CALENDAR” screen functions to view tasks (including repeating ones) in a grid mode, allowing the user to contextualize the different tasks and how they fit into the user’s schedule. The “STATS” screen provides facts about the user’s usage of the app in hope of the user understanding their own usage of the app better. The “SETTINGS” screen simply allows for users to toggle dark mode.    Damon Lee  A very clean interface, minimalist, to ensure that the user does not get distracted or lost. The Menu is simply the calendar displayed for the current month with each date being an interactive button. Dates will have bullets under them if a task is noted for that particular day. After pressing a date, the user will be taken to a screen where they can see their tasks for the day, the tasks they’ve completed so far, and have the option of adding more tasks. When the Add Task label is clicked the user is taken to the final screen where they can create a title, set a specific deadline or reminder for the task, and set repeating patterns. Both the Click on Date screen and Click on Add Task screen have back arrow buttons on the bottom left corner that allows the user to return to the previous screen.    Justin Lee  This design has everything centralized on a main page so that information can quickly be accessed and added. It is very concise and would promote efficiency. One can add more details to a task if need be which would allow them to create a list, set a due date, and make it recurring. Each item could be selected to look at it in more detail and the calendar view could also be expanded.    Patrick Rim  This design is very efficient and aims to provide a clean and easy-to-use interface for our users. The first screen is the menu screen which offers four different views: a day view, a week view, a month view, and a year view. The default view is the week view because most people like to plan for about a week in advance. When you click on each day (the day view), the selected day is displayed in large font at the top of the screen for easy visibility and the rest of the screen is divided into “to-do” and “do's”. The “to-dos” will move over into the “do's” section once the task is completed and indicated by checking the bubble next to each task. On the bottom, is an extremely useful progress chart that uses an algorithm to calculate the percentage of the work is already done. When you click the “add task” button, you will find a new interface. You will be able to add tasks by specifying its name, its date, and optionally its time and whether or not you want a reminder at this time. You can also at a motivation tagline that will help to motivate you to do the task.   |  |  | | --- | --- | |  | Benjamin Liu  This breaks up the many components of the application into different screens to reduce clutter and improve ease of use for our users. When the app is opened, the user is immediately greeted with your upcoming tasks. From that screen, the user can choose to add a task or jump to calendar or start timing a task. The add task page allows the user to specify a title, due date, category, urgency, and additional notes. The application also features a timer screen which allows the user to keep track of what they’ve done and how long it took. They can analyze their performance on the statistics page. The calendar page is organized so the user can see up to a whole month at a time. They can choose to give an upcoming tasks preview to hint at their most urgent tasks. The GUI can be customized so it can suit even more people. |   We decided to combine different elements from each prototype in order to create the most optimal and successful program. One feature that stood out to us was a concise and centralized view that would allow for a lot of information to be viewed quickly when the program is opened.  Another feature that was crucial for our design was the ability to shift between simple views and more detailed looks of a scheduled task.  This allows our user to view lots of information while keeping each screen view clean and uncluttered. Finally, we all agreed upon the necessity of a screen where the user could change settings for the functionality and different feature of the program. Some of these custom personalizable settings would be a dark mode and other GUI color settings. |

**Chart of Build Of Features for Product**

**Dossier | Criterion B | Internal Design**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Feature** | **Components Used** | **Explanation** | | Interactive squares (in calendar) | mouse listeners, buttons | Each day of the calendar would be able to be interacted with and expanded/viewed for more details. | | Adding Tasks | buttons, text fields, accessing database | The text fields are used to input information about the task as prompted and the button is used to submit the information underneath one task. | | Automatic Sorting | ArrayLists, sorting algorithms, database | The sorting would be based on user input of time and deadline to determine the position/priority in the list of tasks. | | Editing Tasks | buttons, text fields, accessing database | The text fields are used to input modifications by accessing the database, and the buttons are used to save said modifications. | | Finished Tasks | buttons | A button can be used to denote a task completed. | | Button Based Navigation | buttons, screen changes | The user can navigate through the program by using buttons that go to different screens. | | Timer | timing algorithm, display, buttons, alarm | A timing algorithm helps keep track of time spent on a particular task and can be started by the user at their discretion. | | Color Code (Priority) | graphics, sorting | The user can color code certain tasks to mark them as a priority. | | Light/Dark Mode | graphics, toggling buttons | The user can toggle between light and dark displays according to their preference. | | (optionally requested by the client)  Command Line Interface | text prompts, panels, keyboard listeners | The user can use the program using a CLI in addition to a GUI. | |
| This chart outlines the main features for the program and gives us an idea of the components required to build the program. This helps organize how the program will be created, giving us an easy to look at checklist that adequately addresses the desired features and allows the group to take a modular approach to coding. |

**System Flowchart Diagram**

**Dossier | Criterion B | Internal Design**

|  |
| --- |
|  |
| The user first inputs a task into the program, specifying certain details such as if it is a priority, if it should be a recurring task, etc. The task is then sorted chronologically and stored into a larger database. Then, it is displayed in either a list or calendar view, which can be toggled between. The program will give alerts and reminders based on the time left to do a task. The user can also decide if the display is in a light or dark mode. |

**Flowchart Diagram of Product**

**Dossier | Criterion B | Internal Design**

|  |
| --- |
|  |
| The product prompts and accepts the task input from the user. If the user specifies certain details, the product will make sure to mark the task with certain conditions. The task is added to storage which is sorted automatically based on chronological order. Based on the database, the product updates the display and then sets up the statistics and notifications for the various tasks. The product is then reprompts for inputs. |

**Flowchart Diagram of Sub-process**

**Dossier | Criterion B | Internal Design**

|  |
| --- |
|  |
| This is a flowchart of the sorting subprocess that is used to prioritize and order tasks in the list. First, the program checks if a time was even indicated; if not, the event is simply added to the end of the list. If there was an indicated time, then that time is temporarily stored for processing purposes. The sort then finds the correct chronological position for the task and inserts it into that position. |

**Modular Diagram**

**Dossier | Criterion B | Internal Design**

|  |
| --- |
|  |
| The graphical user interface will be built primarily with the Java Swing GUI toolkit, which expands upon the Abstract Window Toolkit (AWT) to allows for more comprehensive and user-friendly GUI design. The four main components of our application (timer, calendar, to-do list, statistics) will all be operated through the GUI and access various APIs and libraries to accomplish their task. The timer and calendar components elements are based around the time API along with other bits of code. For example, proper stopwatch and calendar date picker code will need to be implemented for the temporal components of our application to work as intended. The to-do list will rely on ArrayLists to enter, order, and reorder tasks as the user adds tasks and arranges as they find necessary. The statistics screen will have to utilize a library such as JFreeChart or charts4j (undecided) to plot and graph the data captured from the user’s task completion. The calendar, to-do list, and statistics components will all access the database created using Java DB. By using a local database, the user can keep persistent data that will remain even after the application is closed and reopened. |

**Inheritance and Class Diagram | UML Diagram**

**Dossier | Criterion B | Internal Design**

|  |
| --- |
|  |
| This UML diagram shows an outline of the classes that will be used in our program and their attributes and functions. This is just a general outline of how the classes and code will be organized; coding has not begun yet. The Screen is the superclass and has subclasses Task, Calendar, Graphic, and Timer. These classes are key to the features of the program and carry out the main functions. The Interactive class is a subclass of Calendar that allows for the Calendar’s individual squares to be interacted with and expanded. |

**Data Dictionary | Variable Table**

***(identify names | types of data | inputs outputs | range of data\* )***

**Dossier | Criterion B | Internal Design**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Name | Data Type | Class/Function | Input | Output | Range of Data | | currentScreen | String | Screen, to describe the currentScreen on the interface | Screen number | Depends on the screen the user is on | normal: specific names of the screens, anything else is abnormal | | time | int | Task, to describe the amount of time to do a task | User inputs the time needed to complete a certain task | An int is displayed on the screen showing the time | normal: any positive number, extreme would be too high/low, abnormal would be negative | | deadline | String | Task, to describe the deadline of a task | User inputs a personal deadline | A String reminder that is displayed on the screen | normal: any date or time classifier, anything else is abnormal | | recurring | boolean | Task, to describe if the task is recurring or not | The user marks whether or not the event is repeating. | The event is duplicated corresponding to the days marked for repeating. Nothing if false. | normal: true or false | | priority | Color | Task, to describe if the task is a priority or not | The user marks whether or no the event is a priority event. | The event is marked as priority or non priority through a visual cue. | normal: differing colors based on priority level, anything else is abnormal | | tasks | ArrayList | Task, to serve as a database for all existing tasks | The user inputs a number of tasks that need to be completed. | The tasks are displayed in a list for the user to view. | normal: tasks of differing specifications, only accepts tasks | | month | int | Calendar, to describe the current month being displayed | The user inputs the month for the calendar. | The month is displayed on the screen. | normal: values from 1-12  extreme, abnormal: <1, >12 | | day | int | Interactive, to describe the current day being displayed | The user inputs the starting day for the calendar. | The day is displayed on the screen. | normal: values from 1-31  extreme, abnormal: <1, >31 | | isLight | boolean | Graphic, to describe if light or dark mode is currently toggled | The user marks whether light mode or dark mode is used. | The background of the screen changes accordingly based on the user’s changes. | normal: true or false | | timeLeft | int | Timer, to describe the amount of time left on the timer | The user sets the amount of time to start with. | The time remaining for the task is displayed. | normal: any positive number, abnormal would be negative | |
| All of these variables are examples of what would be used in our product. They are temporary and could be set/accessed based on the need to edit or change certain specifications. There are not many extreme or abnormal data that can be entered because most of the data is limited to a certain type anyways. |

*\*Range of data should callout or identify the what would be normal, extreme, abnormal data*

**Database**

**Dossier | Criterion B | Internal Design**

|  |
| --- |
|  |
| The main data storage part of the database is used to store task information, such as categories (color coding) and task information (metadata). This would work by storing Three lists- a list of categories, a list of the tasks inside them, and a list of data for each task.This way, it’s easy to pinpoint a particular piece of data from any task simply by selecting three pieces of information. |

**CRC Cards**

**Dossier | Criterion B | Internal Design**

|  |
| --- |
|  |
| As of now, there are 6 CRC cards that encompass our program. The Screen Class is the superclass that serves as our superclass and contains the Task, Calendar, Graphics, Interactive, and Timer subclass. The Task class encompasses relevant task processes that allow for task modification and greater utility, adding options such as Add Tasks, Edit Tasks, Modify Tasks, Remove Tasks, Specific Details. The Calendar subclass contains the Display Tasks and Visualize Deadlines methods and works with the Task and Graphics subclasses to create the visual displays. The Interactive Class allows the user to view specific details on the Calendar through expanding the tasks on the calendar. As such, it works with the Task and Graphics classes. The Graphics subclass will be responsible for the color coding of our program and also the Display and Navigation of our program. As such it works with the Screen, Task, Calendar, and Timer classes. The Timer class allows the user to set Alerts, Reminders, Alarms, and assign specific Timings. This clase works very closely with the Task, Graphics, and Calendar subclasses. |

**List of Features to Build**

**Dossier | Criterion B | Internal Design**

|  |
| --- |
| Calendar   * Interactive squares   List of Events Tab   * Add New Task   + Automatic Sort * Edit Task * Finished Task   Add New Task/Edit tab   * Repeating Event box * Timer   + Alarm * Color code (Priority)   Settings   * Light/Dark Mode   Command Line Interface   * Text Prompt/Input   Navigation |
| One of the important features in our program is the calendar, which would display show a holistic view of a user’s schedule. The user can then see specific tasks for each day by pressing that day and transitioning over to another tab. While viewing these tasks, the user can either mark as complete, edit, or add tasks to the list. Marking a task as complete would simply cross it out, though a copy would be saved on the bottom of the page to ensure nothing is completely forgotten, while the edit and add new task options would take our user to another page where they can set whether or not this event is repeating, if they need an alarm to remind them of a specific task, or a timer to ensure they are on task. There will also be settings that the user can configure, such as deciding between a light or dark mode. There will also be a CLI feature where the user can control the program through text input. All pages will be equipped with buttons so that the user can easily navigate between them. |

**Schedule for building the product[[1]](#footnote-0)**

**Dossier | Criterion B | Internal Design**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Task Name** | **Duration** | **Start** | **Finish** | | Plan out GUI design | .5 days | 02/15/2019 | 02/15/2019 | | Code GUI | 2.5 days | 02/15/2019 | 02/17/2019 | | Code To-do List | .5 day | 02/17/2019 | 02/17/2019 | | Code Calendar | .5 day | 02/18/2019 | 02/18/2019 | | Implement task visibility and functionality on calendar view | 2 days | 02/18/2019 | 02/19/2019 | | Code Timer | .5 day | 02/20/2019 | 02/20/2019 | | Code Statistics Screen | 1 day | 02/21/2019 | 02/21/2019 | | Implement Database for Persistent Data | 4 days | 02/22/2019 | 02/26/2019 | | Testing and Debugging | 5 days | 02/27/2019 | 03/04/2019 | |
| The building of this application will require careful planning and time management to ensure we meet the deadline of 03/05. FIrst, the GUI design must be planned out so the coding of the GUI can be as streamlined as possible. A approximately 2.5 days will be spent on the development of the GUI because this is, superficially, the most important aspect of the application. Without a user-friendly, intuitive GUI, the app will be ineffective at accomplishing its goal of helping to organize the user. Next, coding the to-do list and calendar parts should be relatively simple. The next longer task is implementing the tasks that will be seen on the calendar view. This task is allotted more time because there may be difficulties encountered when attempting to incorporate the two pieces. The timer will also be another simple task, but the statistics may take longer to troubleshoot. Internally, the most important part of the application is the database, which is responsible for keeping data persistent between opening and closing of the application. Without this functionality, the application would be obsolete. That is why it has been allotted a large segment of time in the building process. Finally, the application will be tested and debugged for many days to ensure there are no flaws, or at least the most obvious flaws are patched. |

**Lists of Techniques | Algorithms**

*Candidates should aim to explain their understanding of and the use of the techniques involved and to showcase their algorithmic thinking and creativity.*

**Dossier | Criterion C | Development**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name of Technique | Algorithm | | Explain the technique application in the product |
| 1 | ArrayList to Store Master Task List | | Tasks made are put onto a master task list and are passed on to each object/updated across the different classes. This is essential to the product because when a task is made in a specific section, it has to be updated across all the different methods of viewing the tasks. |
| 2 | ListModels/Lists to Display Tasks | | ListModels are used in order to update the different task lists. This is similarly important because all of the different displays of the tasks must be updated whenever one of them is. |
| 3 | Passing of Objects Between Classes | | The fluidity of different objects between the classes is important because of how everything is synced together and updates all at once. |
| 4 | “Refresh” Listener | | The refresh listener is the trigger to update/synchronize all of the tasks and displays. A manual button to refresh the tasks allows the user to control the synchronization. |
| 5 | “Complete Tasks” with Loops | | The CompleteListener goes through the tasklist and decides which one to remove based on user input. It uses loops to go through the entire ArrayList and find the completed task to eliminate. |
| 6 | Changing Themes | | The ListListener is used to select the themes, which are imported from java libraries. This allow the user to customize how their experience looks based on their own aesthetic preferences. |
| 7 | Loops to Add Tasks to Day/Week/Month Views | | The loops used to add the tasks to certain days/weeks/months are essential because they make sure that the tasks are added to the correct time slots. |
| 8 | Create Appropriate Week/Month Using Created Time Class | | The loop is used to properly create grids that display tasks for each month. It is combined with a switch case to let the program react to what month the user inputted. |
| 9 | XStream to Create Persistent Data | | Use a third-party library to convert Task objects to XML that can be easily stored and accessed. |
| 10 | Timer to Time Tasks | | The timer function uses a listener in combination with a timing counter in order to allow the user to time themselves on a certain task. |

**Evidence of Complexity #1**

**ArrayList to Store Master Task List**

|  |
| --- |
|  |
| Tasks made are also put onto the master task list and are passed on to each object/updated across the different classes. The adequate technique for this was using a button and listener in combination with parsing for user input and storing that into both the ListModel and the ArrayList. This functions because of the existence of the master task ArrayList that allows for the collection of all the created tasks. |

**Evidence of Complexity #2**

**ListModels/Lists to Display Tasks**

|  |
| --- |
|  |
| ListModels are used in order to update the different task lists. This proves complexity because all of the different displays of the tasks must be updated whenever one of them is. This code is adequate for this technique because it error-checks for duplicates and then updates the ListModels to include the new tasks. |

**Evidence of Complexity #3**

**Passing of Objects Between Classes**

|  |
| --- |
|  |
| The synchronous nature and fluidity of the different objects contributes to the program’s complexity- everything is synced together and updates all at once. This code functions adequately because of the ability to access different views from the tabs of the calendar. |

**Evidence of Complexity #4**

**“Refresh” Listener**

|  |
| --- |
|  |
| The refresh listener will act as a trigger to synchronize ALL of the tasks/displays, showing complexity. The manual button is provided for users if they would like to refresh themselves. The RefreshListener was necessary to constantly repeat the action being performed, which is to check for tasks that pertain to the current day. This code functions adequately with error checking as well as display updates in combination with the use of a button and listener. |

**Evidence of Complexity #5**

**“Complete Tasks” with Loop**

|  |
| --- |
|  |
| The CompleteListener iterates its way through the tasklist and selects which task to remove based on user input. This is all that is necessary to properly remove a task. The ListModel simply removes the specified task. |

**Evidence of Complexity #6**

**Changing Themes**

|  |
| --- |
|  |
| The ListListener is used to select the themes, which are imported from java libraries. This allow the user to customize how their experience looks based on their own aesthetic preferences. This code is adequate because it error checks as well as provides a drop down menu from which the user can select. |

**Evidence of Complexity #7**

**Loop to Add Tasks to Day/Week/Month Views**

|  |
| --- |
|  |
| The loops used to add the tasks to certain days/weeks/months are essential because they make sure that the tasks are added to the correct time slots. We needed loops because they are scalable to each month- we don’t have to code the months individually. This code is adequate because it error checks and adds tasks based on numeric value corresponding to different times. |

**Evidence of Complexity #8**

**Create Appropriate Week/Month Using Created Time Class**

|  |
| --- |
|  |
| The loop is used to properly create grids that display tasks for each month. It is combined with a switch case to let the program react to what month the user inputted. This code is adequate because it makes sure to account for all the different month lengths and creates the displays based on that. |

**Evidence of Complexity #9**

**XStream to Create Persistent Data**

|  |
| --- |
|  |
| This technique was used so the user can store their tasks properly. The application would be inconvenient or even useless if the user-inputted tasks were to be cleared whenever the user exited the application. To solve this problem, the tasks are stored as Task objects. Each Task object has various fields: the name, start time, and end time. Each task can be stored as XML. XML is characterized by its tag format, with each tag followed by the information the tag specifies. The information within each Task object, the fields, very logically translates to tags (sub-elements). Each time a Task is created, it is added to the tasks list. The list can be converted to an XML element which contains Task sub elements, and each of those Tasks contains sub elements that hold the data of each field. By using the XStream library, the conversion of objects such as a Task to XML is drastically simplified.  In Image 1, it is shown that when the application is launched, previously stored tasks (in XML format) are read in from tasks.txt into the storedTasks variable using a FileInputStream. String storedTasks is then converted to a list of Task objects using the fromXML method.  In image 2, it is shown that after the creation of a new Task, it is added to the tasks list. Then xstream.toXML(tasks) is called and the output is printed to tasks.txt using PrintWriter. |

**Evidence of Complexity #10**

**Timer to Time Tasks**

|  |
| --- |
|  |
| The timer function uses a listener in combination with a timing counter in order to allow the user to time themselves on a certain task. This code is adequate because it correctly prompts the user for which task they want to be timed on and can start/stop the timer based on user designation. |

**Flowchart Diagram of a Feature**

**Dossier | Criterion C | Development**

|  |
| --- |
| Feature 8: Create Appropriate Week/Month Using Created Time Class |
| The feature that creates the grid display of the calendar is quite simple. The user inputs the current month and the program uses a switch case to distinguish the different day lengths of the months. By choosing the upper bounds of the iterations of the loop from either the set 28, 30, or 31 days, it then goes through the loop to create the individual grids. These are all combined in order to form the full calendar display. |

**Proof of Functioning Database**

**Dossier | Criterion C | Development**

|  |
| --- |
|  |
| After the creation of a new task object, the tasks list is successfully converted to XML and written into tasks.txt in the directory the program is running in.  In our program, the tasks.txt acts as the database for the storage of Task object data. The FileWriter acts as accessor to our database. |

**Proof of Persistence of Data**

**Dossier | Criterion C | Development**

|  |
| --- |
|  |
| To test persistence, the tasks.txt file is filled with one Task entry in XML form. When the program runs, void main(String[] args) runs first, which invokes the code to read in from tasks.txt. The XML text is converted to an ArrayList of Task objects. This allows for the application to use the Task objects that had been stored in our task.txt “database”. When the Day/Week/Month view is refreshed, the task will appear under its start date. This has all been accomplished without having to create a new task in this run of the application, which proves the persistence of data through our database. |

**Bibliography[[2]](#footnote-1)**

**Dossier | Criterion C | Internal Design**

|  |
| --- |
| Java Swing: <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>  Swing is a package that provides many GUI components.  This package allows for the most important parts of the application to be created, such as tabs, buttons, text boxes, dropdown lists, labels, etc.  XStream: <http://x-stream.github.io/>  XStream is a simple library that easily converts Java Objects to and from XML data.  The tags used for identification in XML are easily adapted to store information about the the fields of Objects.  Swing Timer: <https://stackoverflow.com/a/33488613>  The timer of the application’s timer screen is adapted from this StackOverflow answer to fit the application’s needs.  A button was added to allow the timing session to be linked to a task.  The timer was reformatted to fit the application. |

**Documentation**

Extensibility is straightforward due to the detailed design overview and to proper attention to programming style.

**Dossier | Criterion D | Extensibility**

|  |
| --- |
|  |
| All of the methods have concise and useful explanations; comments have also been added throughout code to provide clarity to those reading the program. Formatting and indentation makes the code easy to read and whitespace separation by sections of code allows for a person reading it to be able to follow the flow of logic. |

**Programming Style**

Extensibility is straightforward due to the detailed design overview and to proper attention to programming style.

**Dossier | Criterion D | Extensibility**

|  |
| --- |
|  |
| Each component of the application is encapsulated within its own class. The main process creates JTabbedPane which contains the Home, Calendar, Timer, and Settings Panels, which are each contained in their own class. Similarly, the Calendar Panel is also a JTabbedPane that contains more detailed panels, each a separate class. Paired with clean documentation of code, this encapsulation allows for understandability of code, even for people that have not seen the code before. The code  Again, the indentation and whitespace separation of the code makes it easy to read and edit when testing and debugging the program.  Indentation allows for processes such as if-else statements, switch cases, documentation, etc. to be easily identifiable and distinguishable.  The addition of whitespace and lines helps create the natural separation of processes into its component steps. For example, the main method is responsible for setting up the majority of the GUI components, but it is not overwhelming to read and analyze because of its formatting.  The abundance of listeners in the programs ensures that the product runs based on user input and control. The listeners in each class all intuitively attach the corresponding actions to the correct objects in order to achieve this. |

**Functionality | Evaluation | Future**

**Dossier | Part 4 | Criterion D & E**

**Evaluation of Product**

**Dossier | Criterion E | Evaluation of Success Criteria**

*Evaluate the effectiveness of the product based on feedback from the client/adviser. Direct references to the success criteria identified in criterion A. Candidates are expected to provide a detailed evaluation (not just whether or not success criteria were achieved) including a discussion of significant client feedback. The assessment will be based only on material presented in section* ***below*** *with the exception of evidence of client feedback in Appendix E.*

|  |  |
| --- | --- |
| # | Effectiveness of the product based on the success criteria identified in criterion A | Evaluation |Based on feedback from the client |
| 1 | **The program will be able to accept inputs to add events.** |
| **The method was able to add tasks.** |
| “I was able to add new tasks” |
| 2 | **The program will be able to order a series of events based on date and time.** |
| **The method was able to add different events that had specific dates and times.** |
| “I was able to order the tasks based on their date and time. |
| 3 | **The program will be able to organize and view events both in calendar form and in list form.** |
| **The program organized tasks in both the list and calendar, and were able to be viewed.** |
| “I was able to view the tasks as a single list as well as in a calendar” |
| 4 | **The program will have an intuitive and accessible design.** |
| **The program was easy to use even without instruction** |
| “The interface was very intuitive and I was able to make my way around the app very easily with help from the command line input” |
| 5 | **The program will perform its tasks quickly and efficiently.** |
| **The program transitioned from one code to the next without delay.** |
| “The program was very quick and there were no issues with speed” |
| 6 | **The program will highlight tasks that the user designates as priorities.** |
| **Specific tasks were able to be highlighted** |
| “If I click I task it will designate it as a priority” |
| 7 | **The program will allow the user to color code different tasks.** |
| **The visual display of the program was able to be changed.** |
| “I can make the tasks different colors by clicking on them” |
| 8 | **The program will be able to display its information in both a dark and light mode.** |
| **Program had both light and dark mode working properly.** |
| “There are many different settings for the color scheme of the app” |
| 9 | **The program will allow for the creation of tasks that repeat themselves at preset intervals.** |
| **repeated tasks had appeared at the rights times over the correct intervals** |
| The code checks for duplicate tasks by checking if they are the same name and same time |
| 10 | **The program will be able to report statistics regarding tasks.** |
| **The program accurately recorded the statistic regarding the user’s task usage,** |
| You are able to see the current time and the tasks currently still left to do. |

|  |  |
| --- | --- |
| 11 | **The program will be able to guide the user with its timer functionality.** |
| **The timer allows the user to start the time and time their tasks.** |
| The timer function works and I am able to time my tasks. |

|  |  |
| --- | --- |
| 12 | **The program will support both GUI and CLI operation.** |
| **The program allowed the user to navigate the app using written prompts** |
| I can navigate the app through the GUI as well as the CLI. |

**Future of Development**

**Dossier | Criterion E | Future Improvement**

*The recommendations for improvement are specific and non-trivial. A recommendation for improvement should be justified by client feedback. The assessment will be based only on material presented in section below with the exception of evidence of client feedback in Appendix E.*

|  |  |
| --- | --- |
| # | Area of Improvement | Proposal |justification from feedback of the client |
| 1 | Organization of text |
| The size of the text boxes for each of the calendar sections cannot fit the length of some of the recorded events. As a result the square for each date has a drag bar that allows the user to scroll and read the task/reminder in its entirety. However, this looks rather unprofessional compared to the text just moving into the next line. As such a future improvement might be testing the length of each task/reminder and checking if it fits the size of the reminder in each section (whether it is being displayed in the month tab, week tab, or day tab) and have it move to a next line if necessary. |
| “I thought the program worked fine and did everything I wanted it to do, but it was awkward moving the little bar to see the entire hw assignment.” |
| 2 | User-friendliness |
| While the program is functional and fulfills all of the client’s requested features, the color scheme of the program is in shades of gray with simple black font and text. So while client might be able to use this program, it’s standard font and lack of color might hinder a user’s experience with the program. Additionally, client’s might enjoy features that allow them to personalize how their calendar looks like, such as through changing the color scheme, the font, etc. We tried to make up for this lack of personalization by adding default themes for the user to switch to, however it is still limited to the options we provide. |
| “It's kind of bland. There's not really a lot of color and I can’t control the customization. It’s just set to your options for you. Also the size of the text boxes don't fit the size of the text and it looks unprofessional.” |

|  |  |
| --- | --- |
| 3 | Online database |
| This program, although useful, is limited to the medium the user uses in through. When opening this app from another new phone, none of the previously set schedule and tasks will be preserved. As such one future development might be an online database with a user login and password that allows users to be able to preserve their schedules over multiple phones or even a computer. So they can access their schedule from multiple locations rather than just being limited to one device. |
| “The program is nice in that I am able to save my tasks and have it there when I re-open the app. However, I would prefer that my tasks are secured in an account with a personalized username and password.” |

**Appendix E[[3]](#footnote-2)**

**Dossier | Criterion E | Client feedback**

*A proper evaluation should include a discussion of success criteria based on significant client feedback (added in an appendix, but referenced in the evaluation) and recommendations for improvement should be justified by client feedback. Provide evidence of extensive client feedback.*

|  |
| --- |
|  |

1. The requirements for this worksheet will depend on the product being developed, the development environment, and development paradigm. [↑](#footnote-ref-0)
2. All sources (code, images, templates, and algorithms) are identified. [↑](#footnote-ref-1)
3. [↑](#footnote-ref-2)