

UWA  
CITS1220  
Software Engineering  
**Programming Project**  
**Project Report**

20925931

Eric (Jun) Tan

## Description of my System Design

My system design was aimed at developing a simple yet aesthetically pleasing program that can manage a 'project' through a user interface. This program will keep track of the staff involved in a project and the project tasks that need to be completed. The base type (entity) classes were very simple to code, with the inclusion of a custom 'Date' type class that I self-made as I felt the library classes would not be adequate for the standard I was looking for.

In a 'Project', there exists 'Staff' and 'Tasks', both of which are entity classes. My system implements each staff member with a name, number, and role. Each task will have a name, start date, end date (of my custom Date class), description, assigned staff member and a completion Boolean variable. A project will then include a project name, collection of Staff (one of whom is the project manager) and a collection of Tasks. My user interface implements a graphical interface that will manage a single project, where a large portion of code has been dedicated to the robustness of the UI, rather than the simple logic behind it.

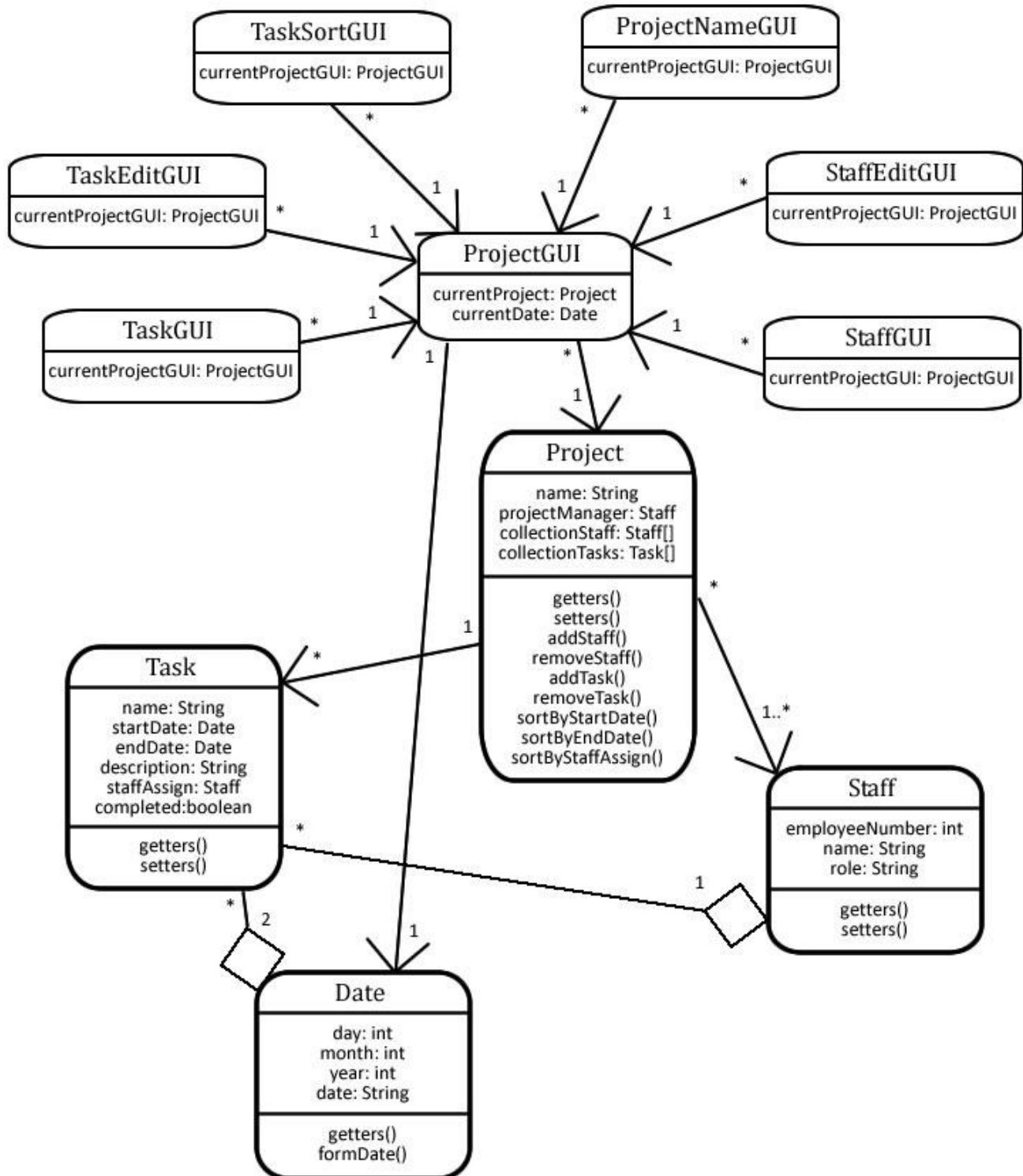
The basic entity classes were fairly straight-forward, with simple logic methods such as getters and setters being implemented. Checks are mostly made within these entity classes, throwing mainly illegal argument exceptions when incorrect data is passed into the constructors or setter methods. Most of the functionality is built in my project class, including adding and removing staff or tasks, finding staff or tasks and sorting the tasks in a chosen order. My graphical user interface class (ProjectGUI) then simply presents a user friendly and robust interface of a single project, together with many extra checks and functionality being implemented.

The user interface was designed to be visually pleasing, with design elements and principles such as colour schemes and balance taken into consideration. The interface itself is quite small, fitting easily in less than a quarter of a standard computer screen at 800x430 pixels. I decided to keep the GUI small as I felt this would be an appropriate size relative to the amount of functionality it provided and space it needed. The two main tables (staff and tasks) are easily readable and contain enough space for plenty of staff members and tasks, together with a scroll pane if ever needed. The general look and feel of the GUI was to keep it smart and simple, with many cool features such as a dedicated section at the bottom left to display information regarding the selected staff member from the table.

The many buttons (13 in total) in the interface are evenly positioned and grouped relative to each other, buttons implementing functionality towards the staff table being grouped, as buttons implementing functionality towards the task table were grouped, etc. These buttons would cater for the basic logical needs of the project, including adding new staff, new tasks, removing staff, editing staff, etc. The other buttons provided extra functionality which includes editing the project name, sorting the tasks, saving and loading projects and exiting the interface.

The GUI has also added robustness and user-friendliness, with checks on all (thought of) possible errors that may occur and checked exceptions ready to be thrown when these are found. Many labels on all the components in the interface are placed, making it very easy to read which component is meant for which functionality. The design itself is very balanced, aimed to be as aesthetically pleasing as possible, to present a nice look and feel to its users.

## UML Class Diagram



## Design Decisions

As mentioned in my description of the system design, much consideration was put into the designing of the graphical user interface itself, with the aim of being as aesthetically pleasing as possible and to provide a nice look and feel for the users. Many design principles and elements were implemented into the design, including use of colour schemes, typography and consistent balance. Below is general GUI shown at first launching of the program.

Project Manager

Manager: N/A

## New Project

Today's date is: 30/10/2011

Staff:

Name
------

Tasks:

Task Name	Start Date	End Date	Description	Staff Member	Completion
-----------	------------	----------	-------------	--------------	------------

Number: \_\_\_\_\_

Name: \_\_\_\_\_

Role: \_\_\_\_\_

\* indicates task is overdue

© Eric Tan 2011

Through use of the many buttons and functionality they provide, the user is able to add in unlimited staff members and tasks into the project, which will update the UI to represent the new data:

Project Manager

Manager: Bob

## Generic Software

Today's date is: 30/10/2011

Staff:

Name
Bob
Alice
Charlie

Tasks:

Task Name	Start Date	End Date	Description	Staff Member	Completion
Requirements Anal...	1/9/2011	1/10/2011*	Analysis of th...	Bob	<input type="checkbox"/>
System Design	15/9/2011	15/10/2011*	Designing th...	Alice	<input type="checkbox"/>
Code (A)	1/10/2011	1/11/2011	Code (A)	Charlie	<input type="checkbox"/>
Code (B)	1/10/2011	1/11/2011	Code (B)	Alice	<input type="checkbox"/>
Testing	15/10/2011	1/11/2011	Testing	Bob	<input type="checkbox"/>
Documentation	15/9/2011	1/11/2011	Documenting	Charlie	<input type="checkbox"/>

Number: 2

Name: Alice

Role: Developer

\* indicates task is overdue

© Eric Tan 2011

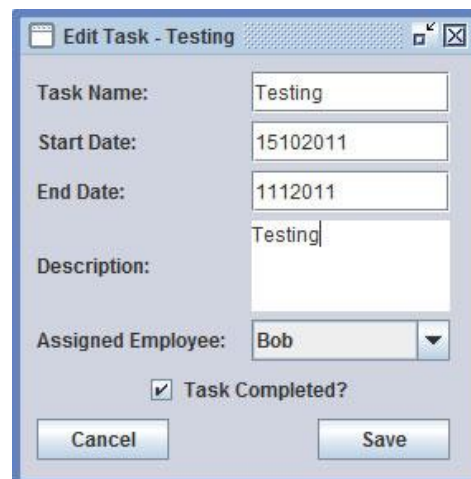
The first design decision made was the general layout of how my user interface would look. I decided to have a very large task table, as it would be the main focus of the interface that would hold the large majority of information required. A brief look at the sample solution provided saw equal sized tables for both the staff and task table, which I thought had actually wasted a large amount of space as the staff table had much less information to present. Further building off this observation was the idea that the staff table was not required to showcase the entirety of the information for each staff member (their name, number and role). I decided a cool feature to make would be to implement a little section at the bottom left dedicated to showing the information of the currently selected staff member in the table, thus only needing the single column in the staff table.

Other decisions included the placing of all the many buttons within the interface, which I decided would fit nicely if they were grouped and spaced evenly across the bottom and right strips or panels within the frame. Each button within its respective group was to have the same width, to provide the element of unity and enhance the look and feel of the UI. A large label for the project name was set as well as many other labels including the manager, current date and overdue indications were appropriately placed within the frame, making sure to keep the overall look as balanced as possible.

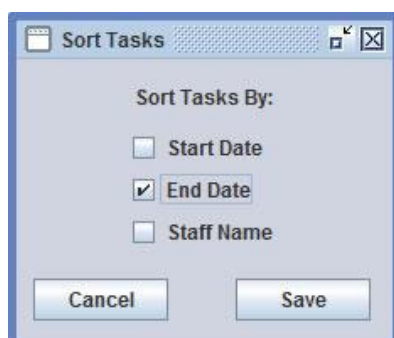
Lastly the graphical user interface needed to provide much functionality, such as adding and editing new tasks and staff. For these requirements I decided to implement extra pop-up JFrames (which are each an individual class on its own), to maintain the balance and unity of the original design. Each respective button would open a new frame (dialog box if you may), which would allow all the functionality required within the project. Each of the new frames was designed specifically to be as small as possible (mimicking a pop-up window) and coincide with the look and feel of the original project UI. Below show the possible pop-up windows and functionality, keeping in line with the design elements as used before.



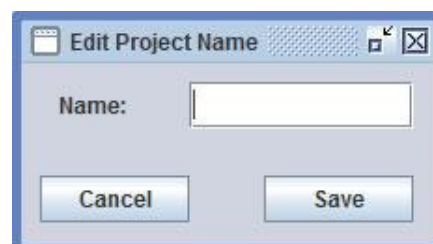
A dialog box titled "New Staff" with a standard Windows window border. It contains three text input fields: "Employee Number:" with the value "0001", "Name:" with the value "Bob", and "Role:" with the value "Manager". Below these fields is a checkbox labeled "Make Manager?" which is checked. At the bottom are two buttons: "Cancel" and "Save".



A dialog box titled "Edit Task - Testing" with a standard Windows window border. It contains five text input fields: "Task Name:" with the value "Testing", "Start Date:" with the value "15102011", "End Date:" with the value "1112011", "Description:" with the value "Testing", and "Assigned Employee:" with a dropdown menu showing "Bob". Below these fields is a checkbox labeled "Task Completed?" which is checked. At the bottom are two buttons: "Cancel" and "Save".



A dialog box titled "Sort Tasks" with a standard Windows window border. It contains a label "Sort Tasks By:" followed by three checkboxes: "Start Date" (unchecked), "End Date" (checked), and "Staff Name" (unchecked). At the bottom are two buttons: "Cancel" and "Save".



A dialog box titled "Edit Project Name" with a standard Windows window border. It contains a single text input field labeled "Name:". At the bottom are two buttons: "Cancel" and "Save".

## Testing Strategy

A large proportion of the construction of my system design was dedicated towards testing and checking all boundary and exceptional cases. To test the logical functionality of the basic entity classes (Task, Staff, Project and Date classes), a JUnit 4 test class was written for each of these, aimed to test not only the normal cases but also any thought of boundary and exceptional cases. These mainly included testing appropriate construction of the objects, as well as checking all types of illegal inputs would throw exceptions. The project class provided much more functionality to be tested, such as adding and removing of tasks and staff members. Also included was finding specific task and staff members given their name or index as well as sorting the task list. The appropriate JUnit test class would take into account all this extra functionality and make checks for each individual method, with the exception of all the getter and setter methods (assumed to work).

For the project class, at first with the 'collection of staff' and 'collection of tasks', I implemented a simple array of each. Very quickly I realised this did not provide nearly enough functionality as I would regularly need to make changes to these arrays, such as adding and removing tasks or staff which proved to be very difficult using a simple array. I immediately decided to change from using a basic array to the more enhanced ArrayList class, which provided me with exactly the right amount of functionality I required. Consideration was made to make use of the custom HashMap class, but I felt I would not need all the extra functionality it provided and it would essentially make things more complicated than they really needed to be.

While coding the basic classes, I would consistently check it through the extensive JUnit test cases relative to it, making sure that each method would provide the exact functionality it required and nothing more. This was the main strategy behind coding all the entity classes, together with regular checks of CheckStyle to make sure I had conformed to the rules and regulations of the coding style. At times I had copy-pasted the code into BlueJ (another IDE similar to EasyEclipse), which allowed me to manually check each object and method and visually tell whether or not each method would work appropriately. This helped greatly with such methods related to the sorting of the task table.

Only after I was completely satisfied with the functionality of all the entity classes had I made a start on the graphical user interface class. At first I had no real experience with coding user interfaces so I looked at using an IDE specific to helping me create GUI's. I tried using NetBeans to create the overall design of the GUI, but I quickly found myself not liking the automatically generated code it gave me. However using this IDE did help me with designing the layout of the GUI itself, as described in my description of design decisions.

I decided to manually code the entire GUI class using Eclipse while running it through CheckStyle regularly (a huge hassle with GUI code). I used the week\_05 lab class CustomerDetails.java as a very helpful basis to help me being the GUI code, such as construction of the JFrame and initialising all of the components. Horrible amounts of CheckStyle errors were found but each was eventually fixed which created a great learning experience for me. The main strategy behind testing of my GUI code was to do each component one by one, checking that it as well as all my previously coded components had still worked. This was a very long and tedious method, but it proved to be successful and I did not believe there would be a better way to do it.

The bulk of the design and layout coding of the GUI was done at first, with checks relative to the

design being made only to make sure I conformed to the design I had decided on. Implementing the functionality behind all the buttons as well as the two JTables was where the actual testing strategies were used. Firstly I would implement both the 'new task' and 'new staff' buttons, which would simply add their respective fields to the array tables. Removing the tasks and staffs was a similar, removing it from its respective array deal (in actual fact the remove code would throw exceptions, so I decided to remove the entire table and recreate it with the new array list). Editing the fields was nearly identical to creating a new field, however instead of 'adding' in the new element I would set all the new data to the old element.

This testing strategy was straight forward, I would make sure the adding new tasks and staffs would work, then removing them, and then editing them, making sure that each bit of functionality would not break the others. This took a large amount of time however in the end was well worth it as everything came together. Finally the last pieces of functionality were implemented including showing all the tasks in the task table, loading a new project, saving, and exiting the current UI.

Outlines of all the unit and acceptance tests I have applied include:

Date:

- Testing constructing a new date initialises values correctly
- Forming a human-readable string in dd/mm/yyyy form is correctly build from 3 integers
- Exceptions are thrown if the date is negative
- Exceptions are thrown if the date is zero
- Exceptions are thrown if the month is illegal, or the day is outside the bounds of the current month

Staff:

- Testing constructing a new staff member initialises values correctly
- Exceptions are thrown if the name or role is empty
- Exceptions are thrown if the employee number is zero, or taken
- Exceptions are thrown if the name is not in correct form ("[A-Z][a-z]\*")

Task:

- Testing constructing a new task initialises values correctly
- Exceptions are thrown if either date is invalid (the date will throw an exception when made)
- Exceptions are thrown if the name or description is empty
- Exceptions are thrown if the end date is before the start date

Project:

- Testing constructing a new project initialises values correctly
- Testing adding and removing of tasks and staffs work correctly
- Testing sorting the tasks sorts correctly
- Testing removing non-existent staff or tasks throws exceptions

The ProjectGUI was run continuously against the client use case acceptance test (Generic Software).



## Extension Functions

My program includes the implementation of two of the three extension functions, (1) allowing the tasks to be sorted by start date, end date or assigned staff member, and (3) using the actual date to highlight which tasks are overdue. I have included in my GUI the prototype for the extension function (2) allowing a user to save and load projects from file, but I did not get around to actually implementing the correct functionality behind the code. I left the prototype design included within the GUI to showcase how it would work had I coded it correctly.

### 1) Allow the tasks to be sorted by staff date, end date or assigned staff member

This function was implemented entirely in my project class, making use of an insertion sort method (the most efficient behind quick sort – quick sort being a little too much of a hassle to implement in such a small array) to sort the respective task and staff ArrayLists by start date or end date (in YYYYMMDD form – a custom method in my Date class) or lexicographically by assigned staff member names. The insertion sort algorithm was learnt and taken from CITS1200 (which I completed last semester), changing from the int values to the respective dates or strings.

The GUI then implemented a new button (“Sort”) which would open a new JFrame dialog box (an individual class – see design decisions) where the user may select up to one sorting method. Exceptions are thrown if no method is chosen and each time a single check box is clicked, the other boxes deselect. Choosing the sorting method would then sort the task array and update the table in the GUI.

### 3) Use the actual date to highlight which tasks are overdue

This function is purely implemented within my ProjectGUI class, being completely within the scope of the user interface. The GUI class has a variable of the current date (using the computers date calendar), extracting it using the library class Calendar within the API. I then extract the day, month and year from the current date and transform it into my custom Date object, which allows me to check whether one date is larger than the other. With this current date variable, at each allocation of an end date (either creating a new task or editing one), a check is made for if the current date is after the end date. If it is the table will set the end date value with an \* appended to the end of it, indicating the task is overdue (a label is placed in the table to describe this). At first I wanted to physically ‘highlight’ the task, ie fill in the cell background with colour, but this proved to be too difficult in implementation so I settled for a string indication.

This function was thoroughly tested with the acceptance tests to make sure each implementation of a new task table (removing tasks, selecting staff members) would also run this check to see if each task was overdue and then update the table.