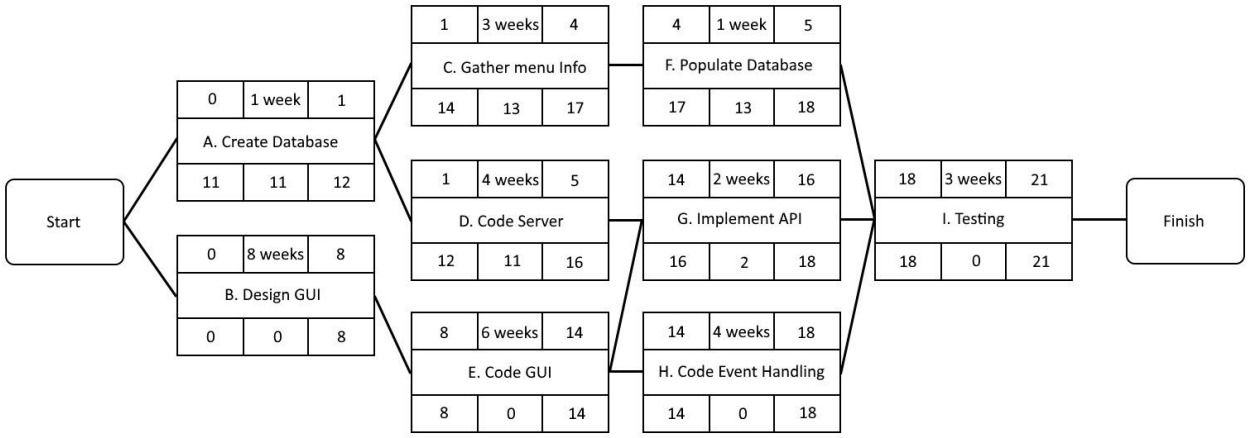
Faculty of Engineering and Applied Science

SOFE 3490U - Software Project Management

Software Project Management
Lab 4

Name	Student ID
Kyuhyun Abe Park	100592092
Rounald Riad Andrawes	100602857
Ernest Li	100658701



Activity A: Create Database
Risks: Overloading Database, Unscalable architecture, Data security
Addressing risks:

- A countermeasure to an overloaded database is to discuss with experts and estimate a reasonable projection of total number of customers and users. This will allow a rough measurement of the size of the database that will be required to prevent data overload. It can also be mitigated through scalable architecture, so that in the case of data overload due to unexpected increase in usage, that the data can be migrated to a larger database without issues.
- Data security is another issue that can occur during the creation and usage of the database. The database based on the scripting language may be vulnerable to a variety of attacks such as sql injection. In these cases the query string and form input must be sanitised to prevent a breach in data. The database should also be encrypted in the scenario that the data is leaked to mitigate some damage.

Activity B: Designing a GUI
Risks: Final product may not be to spec, GUI maybe too complicated,
Addressing risks:

- As a countermeasure to both the risks, the GUI should be designed alongside the clients/users of the application so that their inputs can be taken into consideration, as well as prototypes should be developed and tested by users so that the GUI will be what the user wants.

Activity C: Gather menu info
Risks: Changes in menu during usage and construction
Addressing risks:

- Get concrete items that will stay on the menu, rather than experimental ones, nothing that is subject to change such as seasonal menus
- Create a database that is able to change and drop items if needed

Activity D: Code Server
Risks: Security issues, overloaded servers, availability
Addressing risks:

- Encrypt handshake, verify users/customers, to prevent unwanted access of database from server from and unknown source
- Sanitize inputs, protects the database from SQL injections
- Add region/local servers, limits the amount of people that will be on a given server at a time, and stabilizes connections due to proximity.
- If choosing a third party server such as amazon, ensure third party liability in the case of server failure or service shutdown.
- Keep local server active in case of server failure

Activity E: Code GUI
Risks: Difficult to use, final design not up to spec, pricey to edit/change
Addressing risks:

- As a countermeasure against difficult to use and the final design not being to spec, a prototype should be created before release, this will ensure that the client/user will get a chance to test and give feedback to ensure quality on release
- The GUI should be coded to a standard, to allow for changes in the case of updating or adding new elements to the GUI

Activity F: Populating Database
Risks: Data entries are not standardized
Addressing risks:

- Ensure that data is inputted using a standardized format, i.e. all menu items should start with a capital, passwords should not contain non-standard characters, etc.

Activity G: Implement API
Risks: Over secured, under secured, accountability
Addressing risks:

- When the API is created, and being implemented, it should not be over secured, have some security measures when being used but, not too many, especially redundant security measures i.e. logging in multiple times to verify the user
- When implementing the API, there should be security from both ends, such as sanitizing inputs, limiting what can be inputted into a form, such as non-standard characters
- Accountability will be held starting from the development side of the API, the API should be easy to use and secure from the backend. The implementation of the API should also be secure, if no effort is taken to prevent unwanted inputs then the fault lies with the frontend development

Activity H: Event handling
Risks: Insecure input, unscalable coding
Addressing risks:

- Ensure that the input only allows standardised characters, no special characters to prevent attacks.
- Make sure that during coding, that the event handling is not hard coded to each GUI element, but rather made so that more elements can be added to the GUI with ease.
- Train or hire staff to ensure that they code up to standards

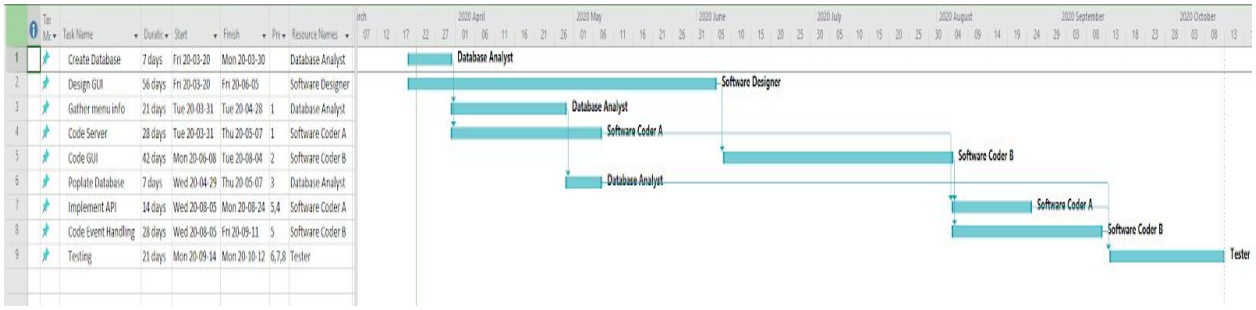
Activity I: Testing
Risks: client does not like final product
Addressing risks:

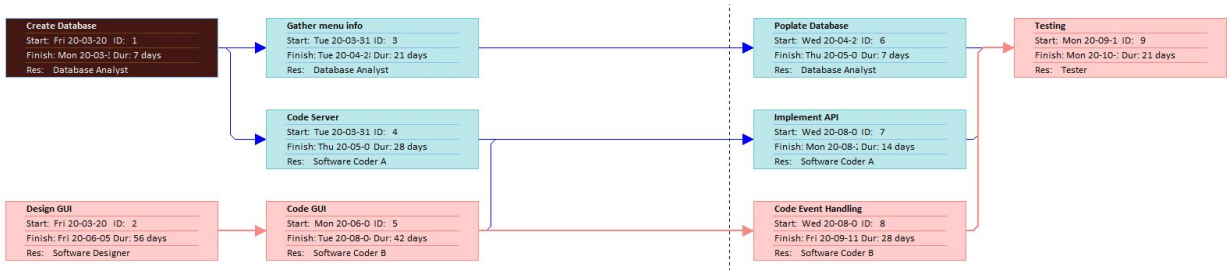
- Create prototypes during testing so that the client can criticize the product so that the final product is up to test.
- Ensure that the client has set up requirements, and that all the requirements are met

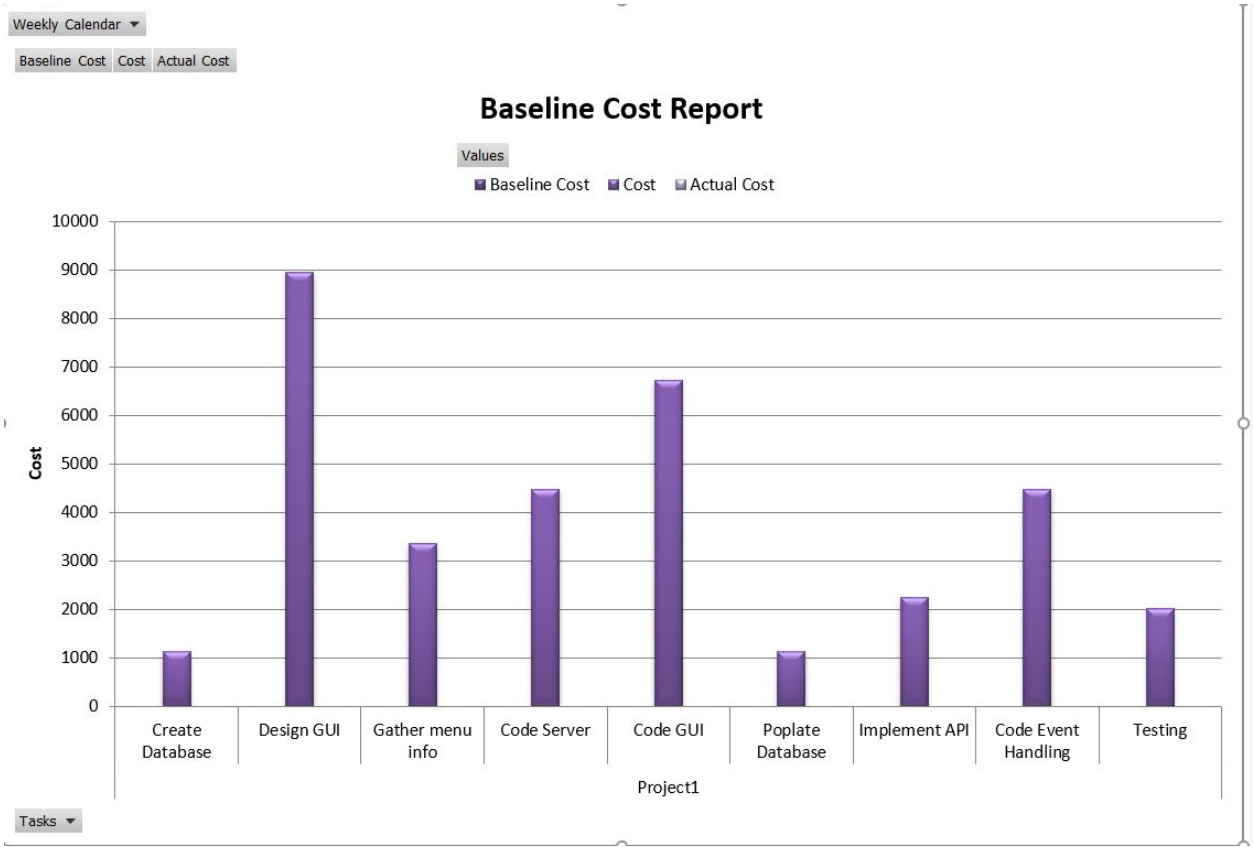
Assign Resources
Software Designer $20/hour
Software Coder A $20/hour
Software Coder B $20/hour
Tester $12/hour
Database Analyst $20/hour
Total Resource Usage

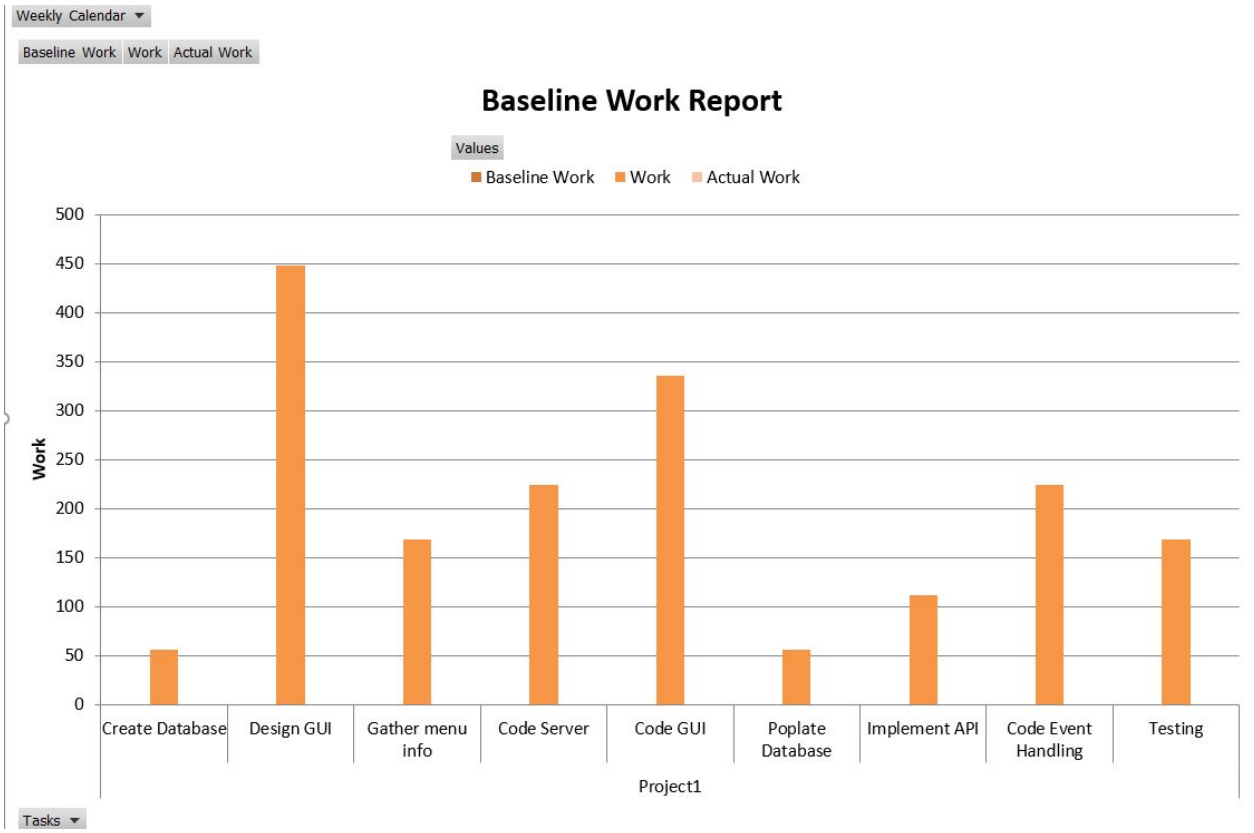
Resource Name	Work
Software Designer	448 hrs
Design GUI	*448 hrs*
Software Coder A	336 hrs
Code Server	*224 hrs*
Implement API	*112 hrs*
Software Coder B	560 hrs
Code GUI	*336 hrs*
Code Event Handling	*224 hrs*
Tester	168 hrs

Testing	*168 hrs*
Database Analyst	280 hrs
Create Database	*56 hrs*
Gather menu info	*168 hrs*
Populate Database	*56 hrs*






Weekly Calendar
Baseline Cost
Cost
Actual Cost
Baseline Cost Report
Values
Baseline Cost
Cost
Actual Cost
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Cost
Create Database
Design GUI
Gather menu info
Code Server
Code GUI
Poplate Database
Implement API
Code Event Handling
Testing
Project1
Tasks


Weekly Calendar
Baseline Work
Work
Actual Work
Baseline Work Report
Values
Baseline Work
Work
Actual Work
500
450
400
350
300
250
200
150
100
50
0
Work
Create Database
Design GUI
Gather menu info
Code Server
Code GUI
Poplate Database
Implement API
Code Event Handling
Testing
Project1
Tasks