VIPASSANA Course assignment – VIA

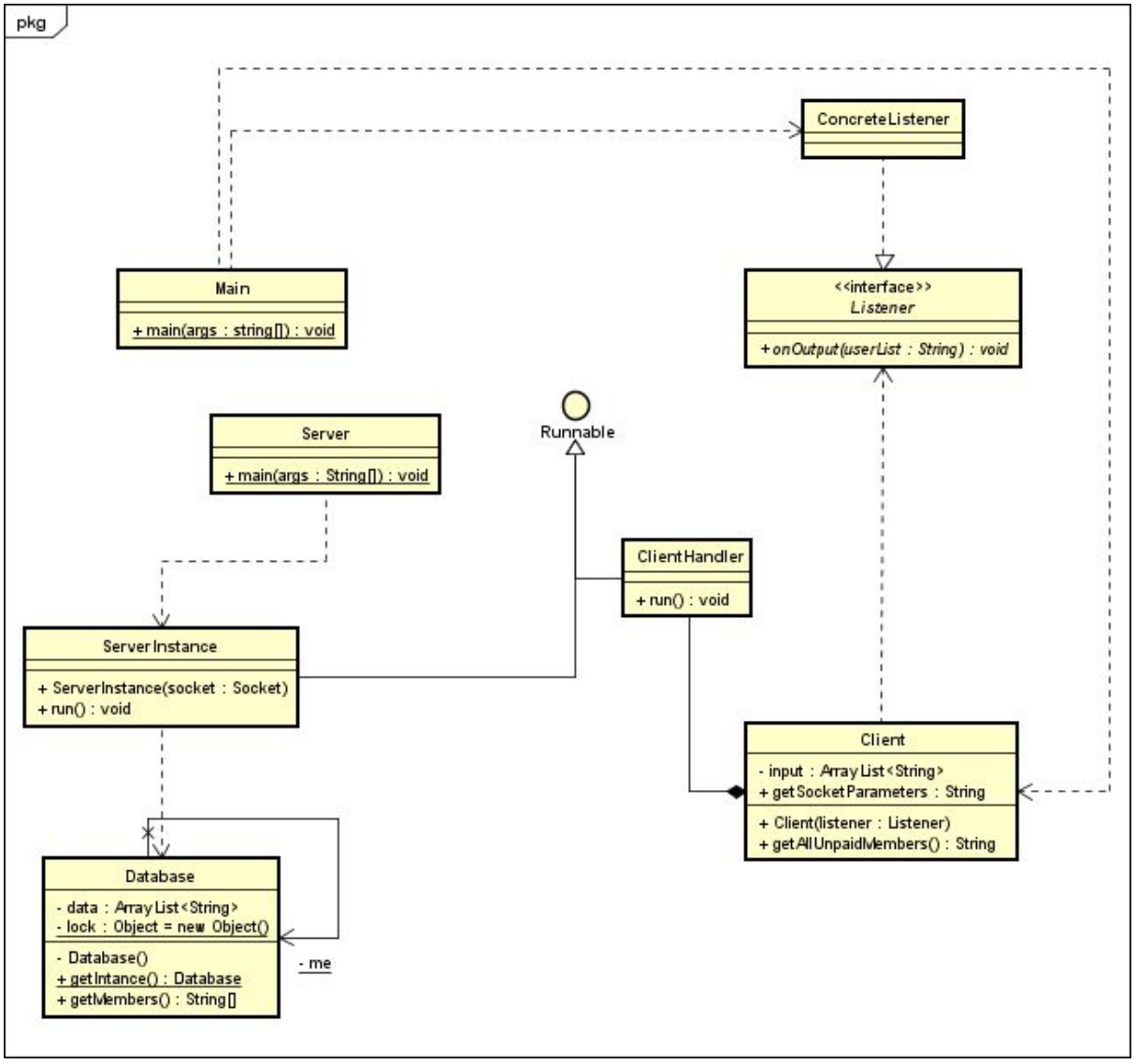
Students:

Angel Petrov	266489
Kenneth Petersen	269379
Diyar Hussein	266352
Daniel Bergløv	220951

Abstract:

System implementation is able to retrieve information from an already populated database simulator and present this information in console view where the server is responsible for fetching information from the database and sending the information back to the client side. The database inserted data is hard coded.

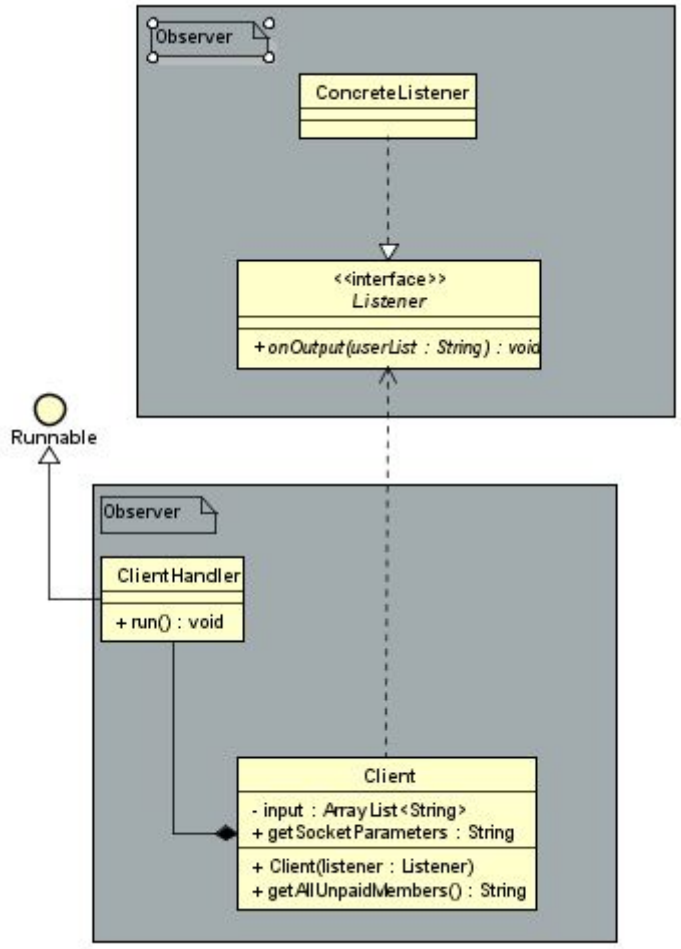
System Class Diagram:



The system implementation integrates the following four design patterns:

- Observer Design Pattern:

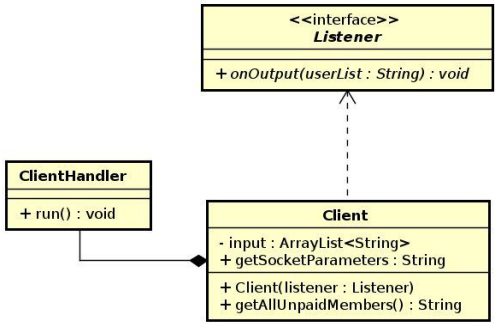
The observer pattern implementation makes use of two classes: ClientHandler is the observer and the Client is the observable which implements the Listener interface. There is a second instance of the pattern which contains the Listener and ConcreteListener which listen for an update. The ClientHandler calls the onOutput method in the Listener which receives stores the results to an arraylist.



- Proxy Design Pattern:

For the purpose of this small assignment, we decided to make use of a small proxy class (Client), that is used to retrieve data from the server. The proxy requires a listener to be passed during instantiation that will

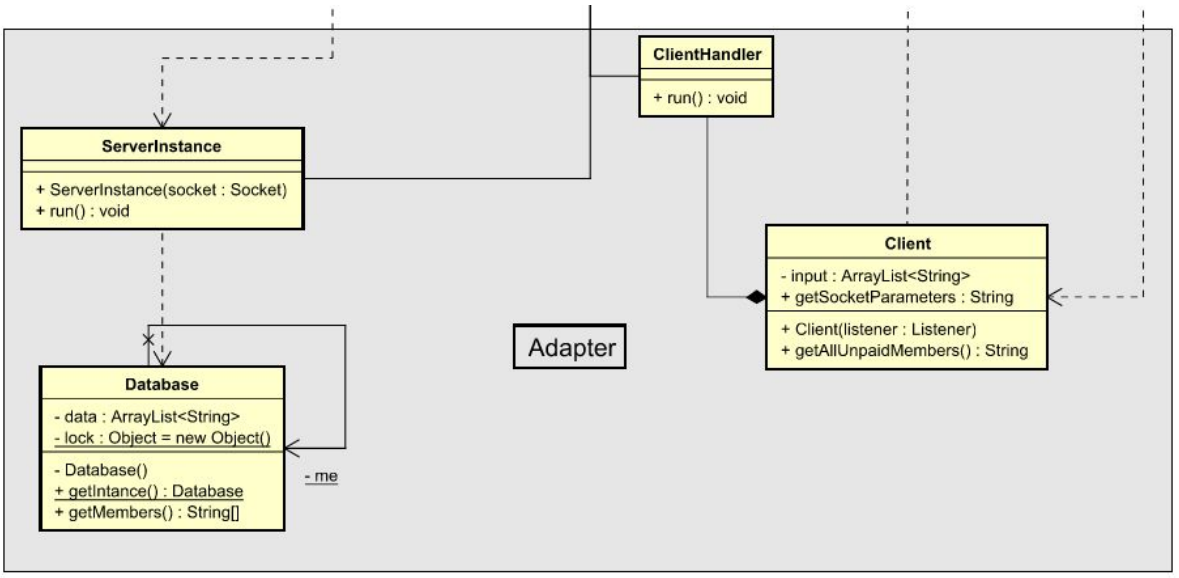
receive the data once it has been retrieved. The communication between the proxy and server is done via an internal handler that works in a separate thread as not to block the main thread.



This works on this project due to it's small size. In a real world scenario this would not be the best option. For starters it would be a good idea to add some sort of timeout feature in cases where the network connection would take to long, and also deal with network dropout. Another problem with this apprough is the fact that all data from the server is cached by the proxy before passing it to the listener. If we are dealing with a large amount of data, this could result in memory crashes. In a real world scenario, this would be better served using an iterator object that retrieves new data for each iteration, optionally with a small caching feature to limit I/O between client and server.

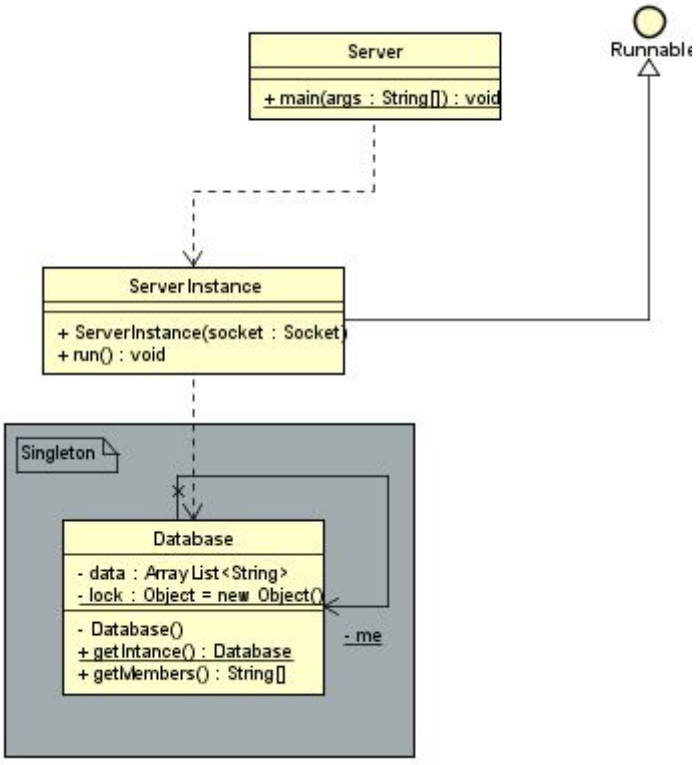
- Adapter Design Pattern:

Originally we intended to implement MVC, but since there was no need for our ServerInstance class to actually do anything except parse retrieved information from the database to the client, it can not really be described as a controller. Instead it is an adapter, translating a request from the client to a database call and translating the database response back to the client. By not having direct access to the database the client is restricted in his interaction with the database and the database is not vulnerable to malicious interaction.

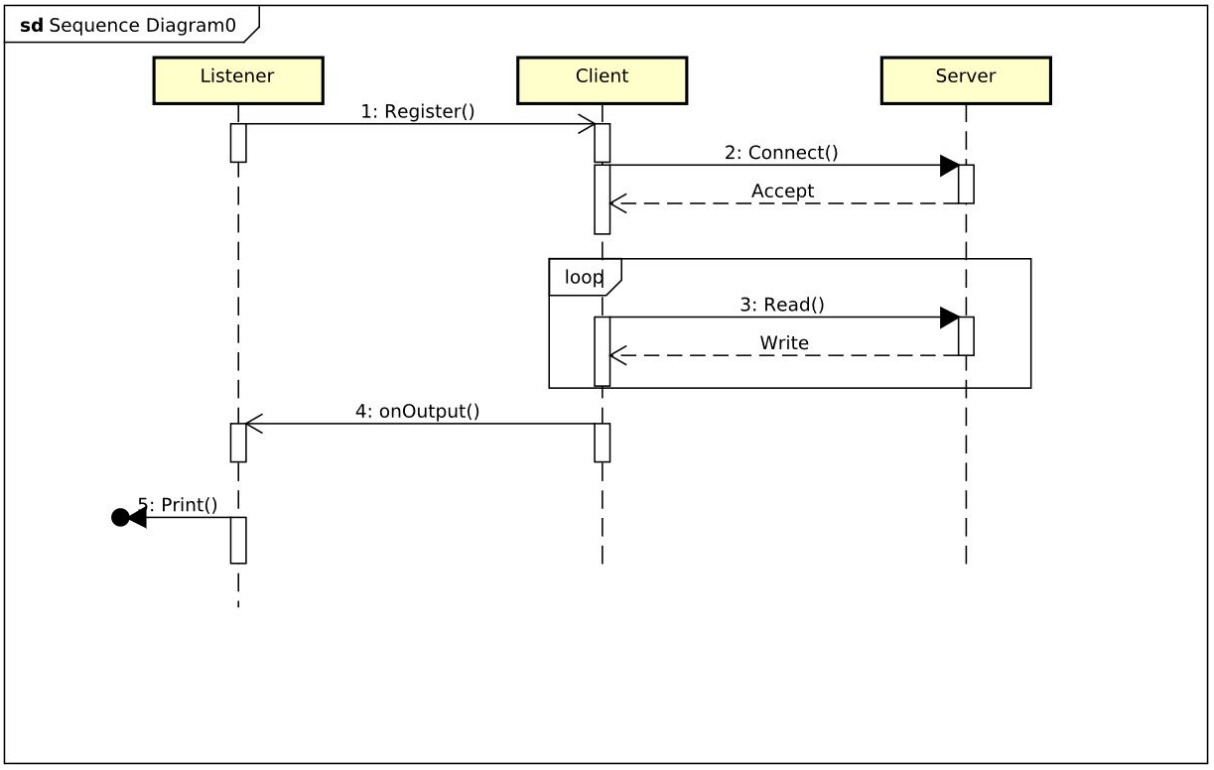


- Singleton Design Pattern:

The Database class is applying Singleton to ensure that there is only one instance of it in the program. It has a private static instance of itself, private constructor, and a public getInstance method.



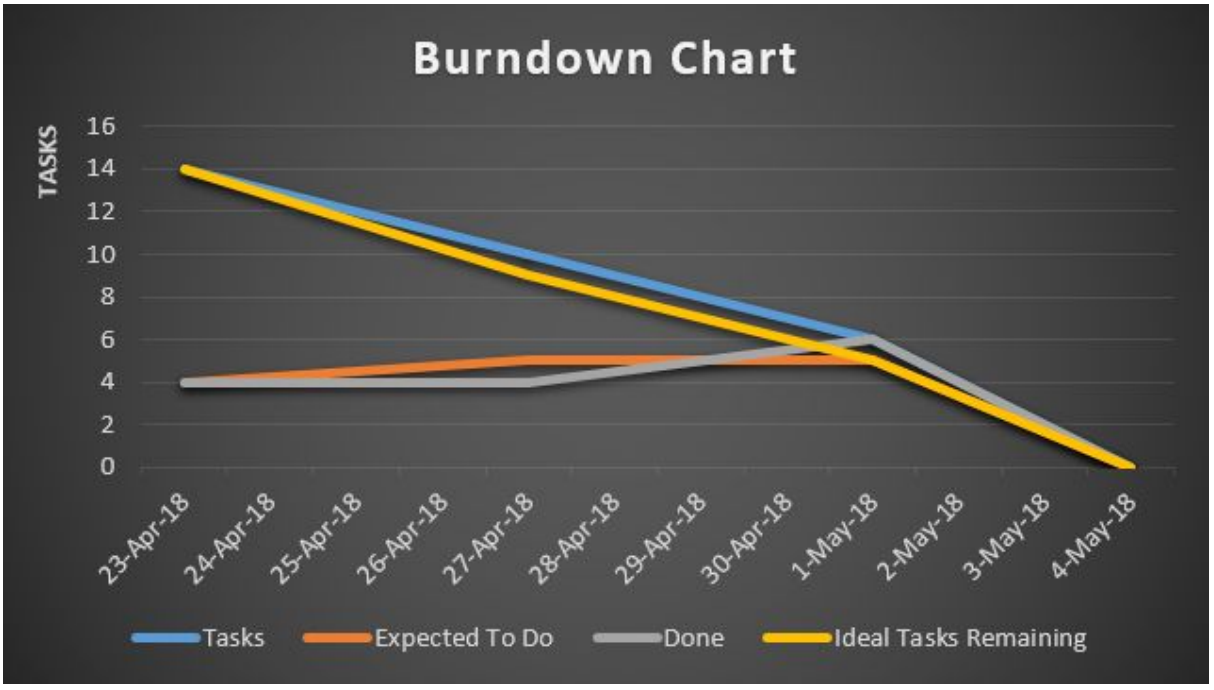
Sequence Diagram:



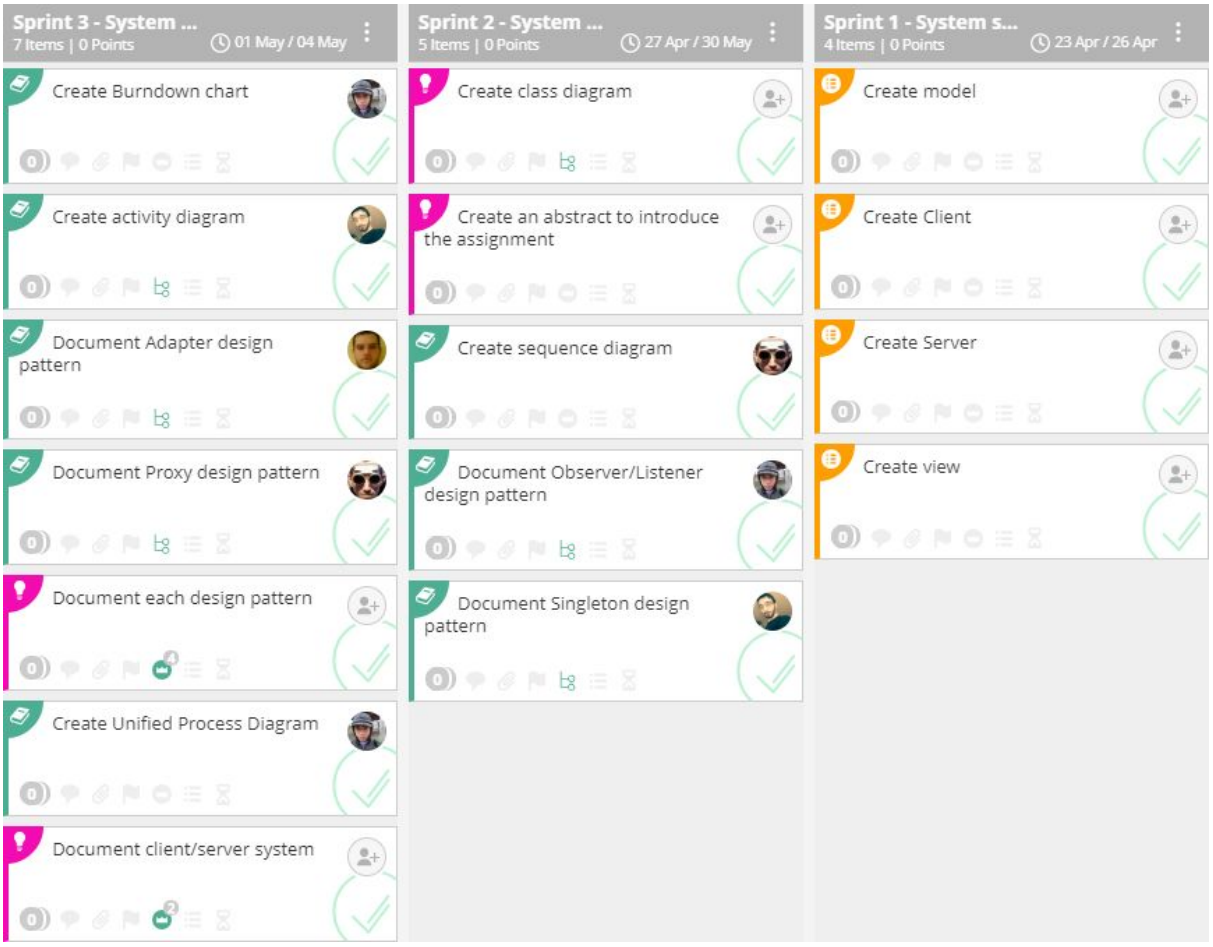
The client has to be registered with a listener in order to create a connection with the server. When the connection is created between the client and the server, the server starts writing and the client starts reading throw the socket in a loop, until the clients receive all the names in the database. When the loop ends, the client sends a message to the listener with the list of members in form of a string. The listener then prints the list out.

Burndown Chart:

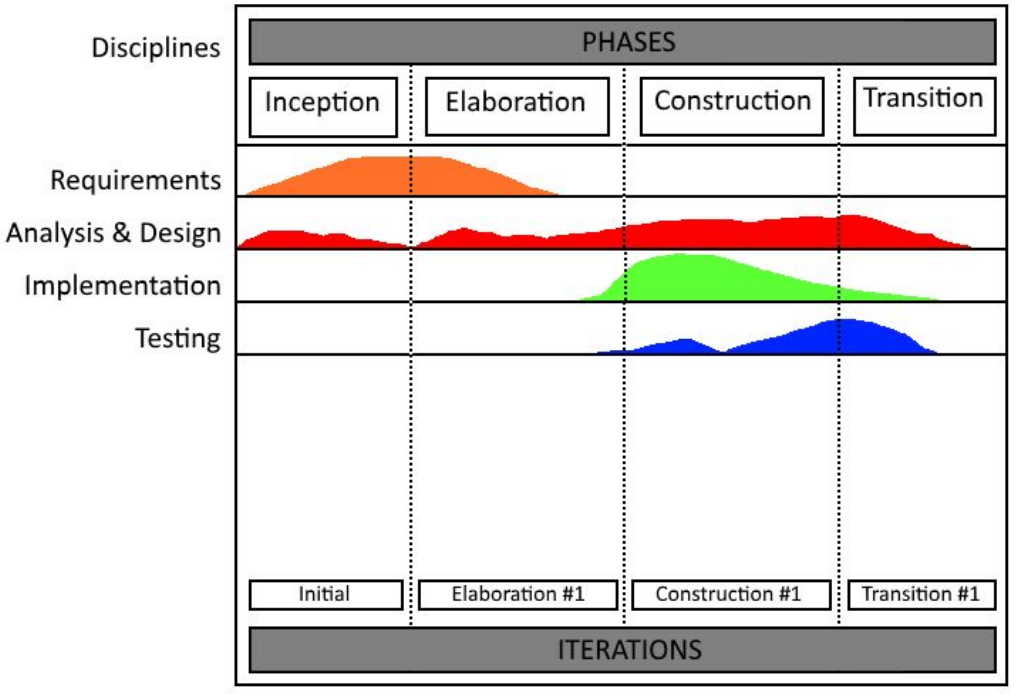
Sprint #	Date:	Tasks	Exp. T. D.	Done	Ideal T. R.
1	23-Apr-18	14	4	4	14
2	27-Apr-18	10	5	4	10
3	1-May-18	6	5	6	5
N/A	4-May-18	0	0	0	0



Sprint task roles:



Unified Process Diagram:



Appendixes:

Appendix **A**:

System implementation code and documentation (zipped).