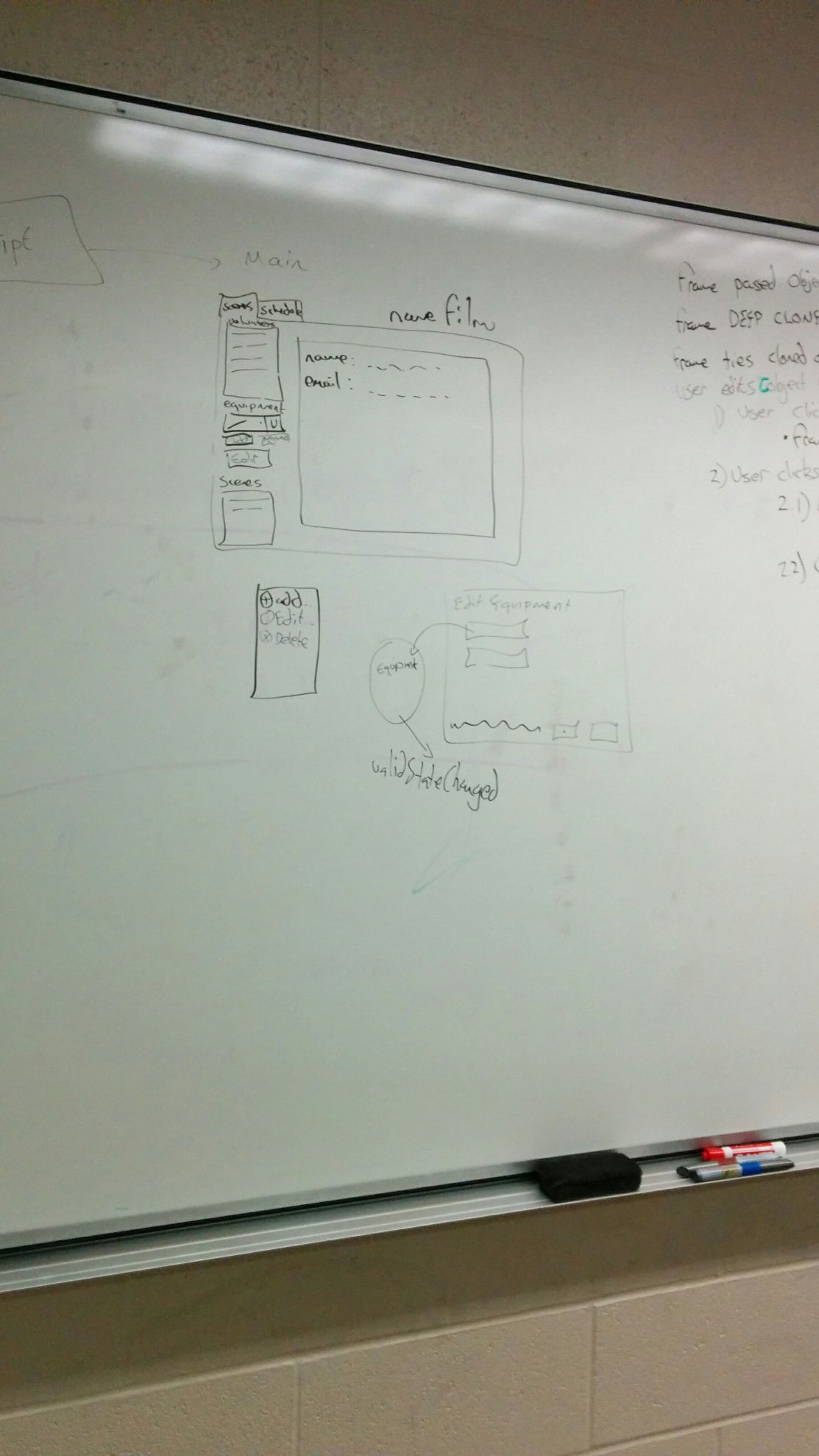
Meeting: Iain/Ryan

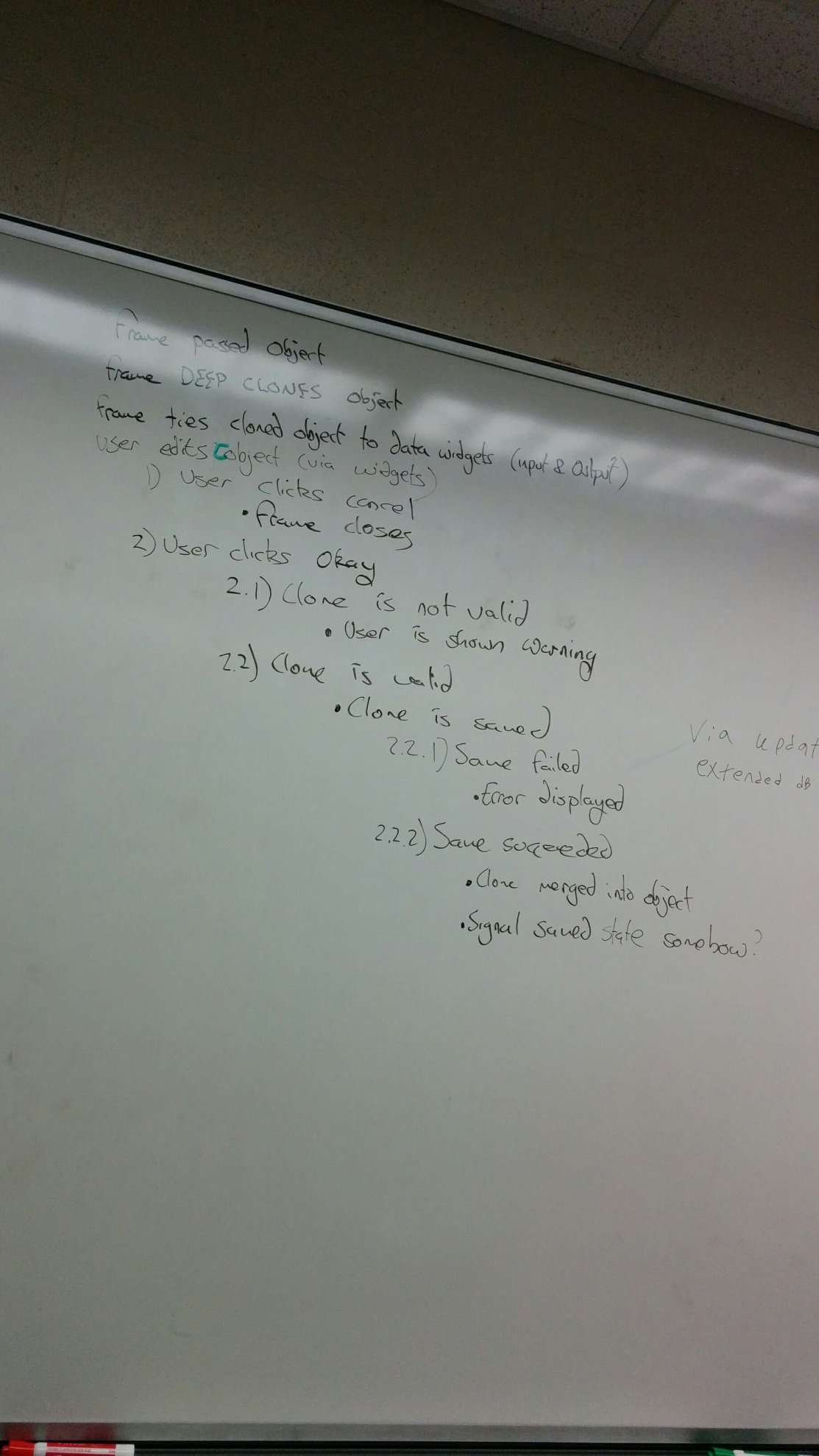
6:00-8:00.

Topics of discussion:

Inserting the volunteer availabilities into the database:  
solution: before inserting new availabilities, purge the database of all availabilities for that specifics volunteer and reinsert the new ones. This helps ensure data integrity.   
The idea is to always insert the entire list of availabilities at once, even if the old availability was in the database.

Alterations to the database class  
We need to ensure that the database handles everything as an atomic action. That is, either it happens or it doesn’t. This means that the begin, rollback, and commit statements should be used in the database class. This ensures database integrity.

Alterations to the main menu to increase usability.  
Iain has proposed an overhaul of the main menu. Some of its features and a picture is listed below.  
-There will be three lists on the left hand side of the screen.   
-There will be a large description box on the right hand side.  
-The user would right click the components in the list and select the operation to perform.  
  
  
  
  
  
  
  
  
  
  
Incorporate a loadAll action for the script at the beginning of the application.  
This action would load all volunteers, equipment, scenes and so on into the script. Every other component in the application can then pull the necessary information from the Script.

A systematic way of using all of our ui components.   
With this new main menu would come a new step-by-step process of interacting with the main menu.  
The steps are listed in the picture below.  
  
  


For everyone:

Everyone’s classes needs to implement a toString() method to put their information into an easily readable format. There may be two toString methds. One to simply identify the particular object, and another to list every detail of that object in a nice way.   
  
This would increase usability as well as adaptability(the current main menu is rather strict).

We need everyone’s classes to be able to perform a deep clone.  
 This requires one to copy all of the attributes, as well as copy any object that is an attribute of the parent object. This is to allow us to implement the above steps in our ui implementation. The whole purpose of this is to be able to modify an object in real time, report or highlight errors as soon as they occur, and be easily able to revert back to the previous object.  
  
We need everyone’s classes to implement a merge function.  
This function takes two objects of the same type as the class that is implementing this function.   
The function’s purpose is to copy all of object b’s attributes into object a. That is, overwrite all attributes of object a with object b’s attributes. This makes the above ui implementation easier.

Proposed implementation summary- to be discussed In the next meeting.

|  |  |
| --- | --- |
| Implementation | Who’s responsible |
| Inserting volunteer availabilities into database | Matt/John |
| Alterations to database class | Ryan |
| Alterations to main menu | Iain -everyone |
| Load all action | Ryan/Iain |
| Systematic UI approach | everyone |
| toString() methods | John/Ryan |
| Perform a deep clone | John/Ryan/Iain |
| Perform a merge function | John/Ryan/Iain |