Joshua Van Deren, Jacob Lai, Grant Savage, Joy Tan

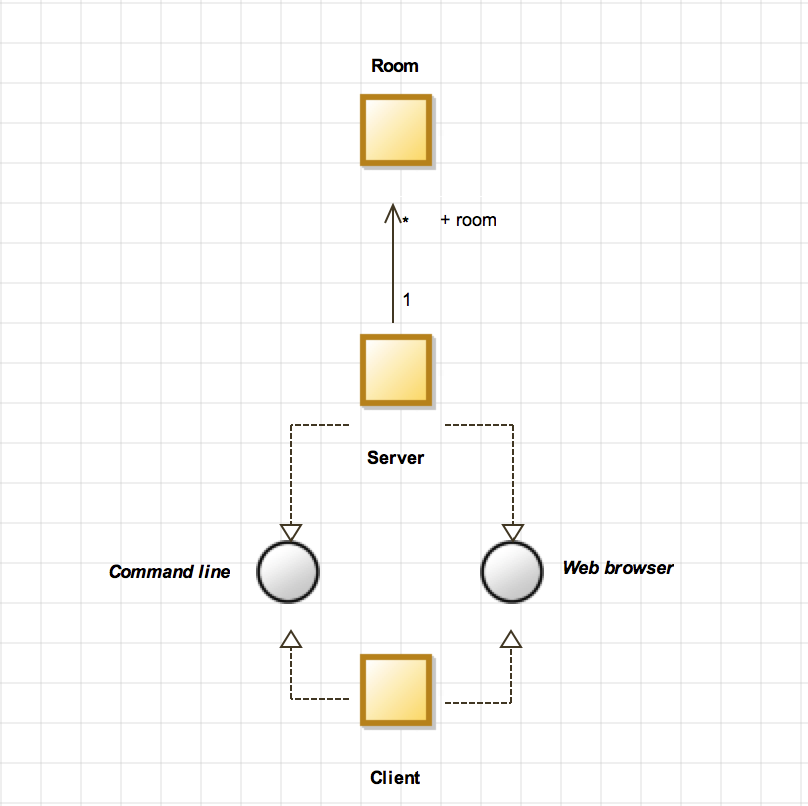
April 2, 2018

SE420

Siewert

**SE420, SQA:**

**Exercise #5 – System/Sub-system Prototype and Regression Testing**

1. We are using GitHub issues to document and assign our bugs. This can be found at [https://github.com/jvanderen1/cli\_chat/issues](https://github.com/jvanderen1/cli_chat)/issues . Below is a list of our issues ordered from highest priority to lowest. Please look at our project on GitHub for further details about each issue.
   1. Defect: The array storing the list of connected users is not updating.
   2. Defect: The array storing the list of existing rooms is not updating.
   3. Defect: 'Successful event' message prints after prompt.
   4. Defect: If user tries to create a new room and names it "NONE", the program prints out an error message but does not re-print the prompt to let the user continue using the application.
   5. Defect: Prompt prints out if user is interrupted, even in middle of question.
   6. Bug: When user receives message, while in a question, the content of the message is not pushed through.
   7. Bug: Prompt still continues even after incorrect input.
   8. Defect: Created rooms do not appear to users who connect to the server after rooms have been made.
   9. Enhancement: If a client attempts to connect to a non-existent room, they should be alerted with a message.
   10. Defect: Users are not supposed to be able to send messages to themselves, but they currently are allowed to.
   11. Bug: Group message sends to users who used to be in room but are not currently in room (they left the room but still receive group messages from room).
2. Product requirements, define the product better
   1. Re-write specific requirements
      1. 5 major function and feature requirements
         1. Every user that connects to the server must have an assigned universally unique identifier (UUID) which is randomly generated upon connection.
         2. Every user must have the ability to create a chat room if they are not already in one.
         3. If there is more than one user connected to the server, every user must have the ability to send a message.
         4. A user can only be in one chat room or in none (in the default room) at any given moment.
         5. Private messages sent to a specific user can only be seen by the receiver and the sender.
      2. 5 performance requirements
         1. The server application must be deployed on a Virtual Private Server that has at least 1GB of RAM, 1 vCPU, and 25 GB of disk space.
         2. The time it takes between the sending of a message and the receiving of a message between two users must be as close to real-time (1/1) as possible.
         3. The client must be able to connect to the server within 10 seconds of startup. Otherwise, the client will abort.
         4. Upon client disconnection the client shall attempt to reconnect to the server within 10 seconds. Otherwise, the client will abort.
         5. The delay time between message sending and receiving in a room shall be the same across all clients regardless of how many clients are connected to a room.
      3. 5 error handling, recovery, or ease of use features
         1. (Recovery) When there’s a lost connection, there should be a time frame to reconnect.
         2. (Recovery) If the server crashes, there should be a time frame to reboot.
         3. (Error Handling) Check for non-empty input when sending a message.
         4. (Error Handling) Check if user is already in room before letting them create or join a room.
         5. (Error Handling) Check if there is more than one user connected to the server before letting users send messages.
   2. Update acceptance test outline
      1. 5 major tests for function and features
         1. Emit a ‘connect’ event channel call to admit a user into the server with a unique UUID.
         2. First check if user is not in a room, then emit a ‘create room’ event channel call to request a room.
         3. First check if sender selected a valid user to receive message, then emit a ‘private message’ event channel call to see if message sends.
         4. Emit a ‘join room’ event channel and the user will switch to that room (if exists) or an error message will pop up.
         5. There will be at least three active sessions. One user will send a message to the second user and the second user should verify that the message was received while the third user should verify that they did not receive any messages.
      2. 5 workload tests
         1. Test the server by using the command ‘free -m’, ‘df –block-size=1MB’, and ‘nproc –all’.
         2. There will be two active sessions, where one user will send a message, and another user will receive a message in real-time.
         3. Use a stop watch and see if the user can connect within 10 seconds.
         4. Use a stop watch and try to reconnect close to 10 seconds.
         5. There will be more than two active sessions, where one user will send a message, and all users will receive that message in real-time.
      3. 5 tests to generate faults or to force failure
         1. Try to reconnect close to 10 seconds, determined by using a timestamp, after losing a connection.
         2. Try to reboot close to 10 seconds, determined by using a timestamp, after server crashes.
         3. Try sending an empty string to another user.
         4. Try creating a room and then try joining another room without leaving the created room.
         5. Have one active session and try to send a message, an error should occur. Have more than one active sessions and try sending a message.
3. Documenting your design
   1. Define and refine the product design
      1. 
      2. Reading the above block diagram from bottom to top, the client must connect to the server via either the command line or the web browser. (We require that there must always be a server running). Additionally, the server interfaces to the command line and the web browser. Once the user is connected, they can create chat rooms. The user can create multiple chat rooms on one server.
4. Restructuring
   1. Define and refine the code
      1. Please refer to our repository on GitHub. We have improved our code by modularizing the client code, included multiple error handling cases, solved multiple raised issues, implemented more detailed comments, and worked on improving the general quality of our code. For more details, all of our changes can be seen on GitHub.
      2. All of our tests can also be found on our repository on GitHub.

Roles and Responsibilities:

1. Joshua Van Deren
   1. Created test cases for client
   2. Added travis
   3. Added coveralls
   4. Updated client architecture
   5. Implemented Google style guide
   6. Raised issues, fixed issues
2. Joy Tan
   1. SE420 assignment #5 write-up
   2. Improved commenting
   3. Implemented Google style guide
   4. Addressed feedback from SE420 walkthrough
   5. Raised issues, fixed issues
3. Grant Savage
   1. Updated UML Class diagram
   2. Added more test cases
   3. Improved code readability
   4. Fixed issues
4. Jacob Lai
   1. SE310 assignment #5 write-up
   2. Addressed feedback from SE310 walkthrough
   3. Updated Use Case Diagram