Deliverable II

for

CoNetPad

TheACMGroup
Justin Anderson
Cesar Ramirez
Josh Tan

CSCI 413
Fall 2012

December 10, 2012

Table of Contents

I. Software Requirements Specification

1. Introduction

1.1 Purpose

This document specifies the software requirements for CoNetPad Version 0.1. It covers the requirements for the entire CoNetPad system, including the client, server, communication, security, and file input/output components.

1.2 Document Conventions

References (in Section 1.5) are provided for any underlined text within a section.

1.3 Intended Audience and Reading Suggestions

This document is intended for customers and developers, though users of the system have played a significant role in determining the specific content that follows. This is because the requirements that we have specified are intended to fulfill the perceived needs of users.

Customers and program managers should understand all aspects of this document. Customers must do their best to ensure that all known requirements are described in this document. It is expected that requirements may be added or removed as the project progresses, but these should be minimal in order to ensure minimal project cost and development time. At the very least, all participants of the project should read Section 1 ("Introduction") and Section 2 ("Overall Description"). Testers should focus on all requirements of the system, including Section 4 ("External Interface Requirements") and Section 5 ("Other Nonfunctional Requirements").

1.4 Project Scope

The aim of this project is to a produce a simple solution for real-time collaboration in small software projects. It is not to replace the fully-featured solutions that already exist in the market. Existing software products like the Eclipse Integrated Development Environment (IDE) or the Netbeans IDE already provide users with a free and comprehensive development environment. These tools can be combined with Version Control Systems (VCS) such as Subversion or Git to provide multiple developers an efficient way to collaborate on projects. In addition, the Eclipse Communication Framework (ECF) versions 2.0.0 and up contain the DocShare plug-in, which implements real-time shared editing. The issue with these solutions is that they require the user to first download and install large software (possibly including additional plug-ins) and then configure their system to correctly use these solutions. Often times, there is also a steep learning curve in actually using the solution.

Some solutions that are similar to ours do exist. These include Gobby, SubEthaEdit, collabedit, and Cloud9. However, none of these solutions contains all the features that our solution intends to provide. Gobby lacks integrated support for Git and remote compilation. SubEthaEdit also lacks integrated Git support and is available only on the Mac OS X platform. Collebedit does not allow for multiple files in a single view and does not utilize the auto-complete feature. Cloud9 is an

excellent solution except for that fact that unlocking all of its features requires the user to be a paid subscriber.

Our goal is to create a free, accessible solution with a functionality that falls somewhere in-between the large, fully-featured IDEs and the bare-boned, real-time collaboration tools available online.

1.5 References

The following products are referenced by this document. Web addresses have been included for those interested in obtaining more information on these products.

Related Products:
Eclipse - *http://www.eclipse.org/*
ECF/DocShare Plugin - *http://wiki.eclipse.org/DocShare_Plugin*
Netbeans - *http://netbeans.org/*
Gobby - *http://gobby.0x539.de/trac/*
SubEthaEdit - *http://www.codingmonkeys.de/subethaedit/index.html*
collabedit - *http://collabedit.com/*
Cloud9 - *https://c9.io/*

Java Tools/Libraries:
JGit - *http://www.jgit.org/*
Xerial JDBC Driver - *http://www.xerial.org/trac/Xerial/wiki/SQLiteJDBC*
Martyr - *http://martyr.sourceforge.net/*
JerkLib - *http://jerklib.sourceforge.net/*

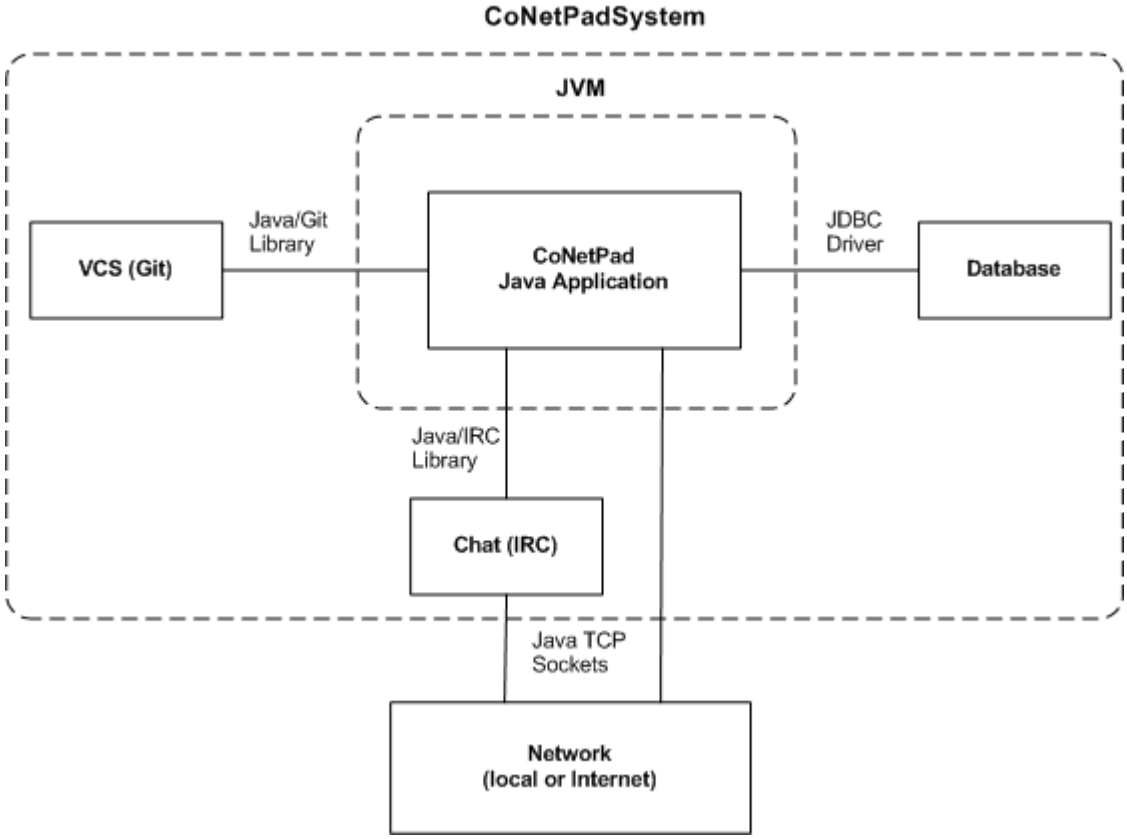
2. Overall Description

2.1 Product Perspective

The CoNetPad system can be viewed as consisting of five discrete subsystems: a Git repository, an SQLite database, the CoNetPad Java application, the Java Virtual Machine upon which the CoNetPad Java application runs, and an IRC server. Each of these subsystems must function correctly in order for the overall CoNetPad system to be fully functional. The CoNetPad system communicates through a network system. This network can be either local or Internet-based. *Figure 1* illustrates this high-level system view.

Figure 1 shows the specific CoNetPad subsystems that directly interact with the network, namely the IRC server and the CoNetPad Java application (running on top of a Java Virtual Machine). The other two subsystems, the Git repository and the SQLite database, do not directly interact with the network. Both of these subsystems are meant to be used locally only by the CoNetPad Java application.

Figure 1:



The interface between the systems and subsystems is also shown in *Figure 1*. The CoNetPad Java application will communicate with the Git repository using a Java Git Library, such as JGit. It will communicate with the SQLite database using an SQLite JDBC driver, such as the one provided by Xerial. Various Java IRC libraries exist that can be used for communication between the CoNetPad Java application and the IRC server, such as Martyr and JerkLib. Both the CoNetPad Java application and the IRC server will communicate over the network using TCP sockets.

2.2 Product Features

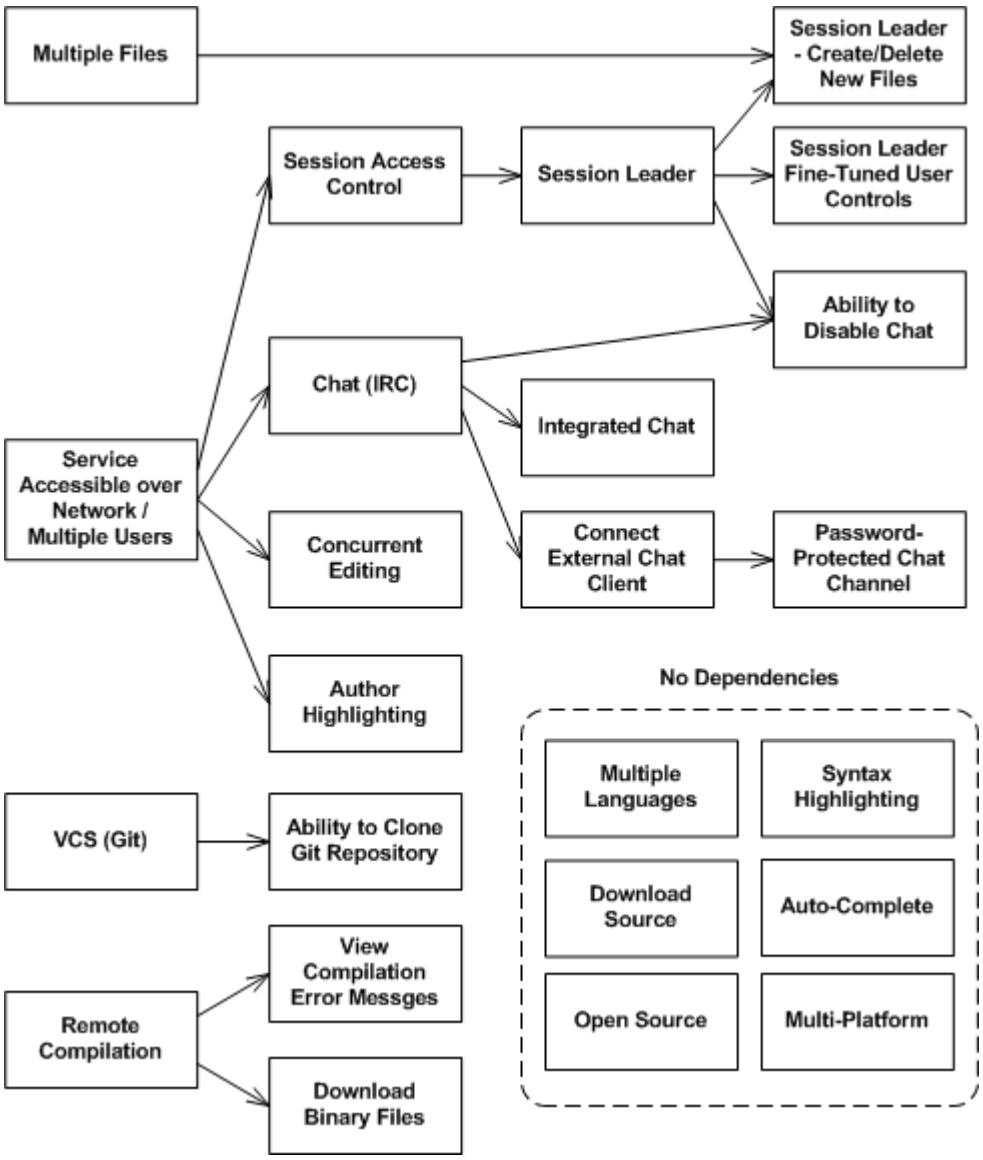
The features that are planned to be included in CoNetPad are as follows:

- Service accessible over a network
    - Concurrent editing between multiple users
        - Color highlighting used to differentiate work of different authors
    - Controlled access to a session (both read and write access)
        - Ability to create a session leader
            - Controlled visibility for specific files in a session
            - Ability for session leader to create/delete files
    - Ability for users to communicate via “chat” interface separate from the files
        - Integrated chat service in client
        - Ability to connect external chat clients to a session chat instance
            - Password protected channel
        - Option for session leader to disable chat

- Ability to edit multiple files in the same session
- Interface to Git for versioning control of source code
    - Ability to clone the Git repository associated with a session
- Ability to download the source code for specific files
- Auto-Complete
- Syntax highlighting
- Option for remote compilation of source code
    - Ability to download resulting binary files
    - Display any compiler error or warning messages
- Support for multiple languages
- Availability on multiple platforms
- Open source project

These features are not equally important. Even if some of these features are left out of the final product, CoNetPad will still be useful as a collaboration tool. Others must be included for CoNetPad to have any significant value. The features that are essential are prioritized over those that are not. The priorities for each of these features are elaborated on in Section 3.

Figure 2:



In addition, certain features require the implementation of other features in order to exist. *Figure 2* shows these prerequisites for those dependent features. The dependent features are shown to the right of those features that they depend on.

2.3 User Classes and Characteristics

The primary user classes that we anticipate will use this product are small project developers and instructors of an introductory programming language course.

Developers who use CoNetPad will likely do so for small projects. These projects will likely consist of less than 1,000 lines of code and span no more than a few files. The frequency with which these developers use our product will depend on the frequency with which they must create small-scale programs in a group setting. We anticipate that this use class will use all of the features planned for CoNetPad, except for perhaps visibility control of files. In addition, developers who begin a project with CoNetPad, but later realize that they underestimated the size or complexity of the project, will be able to clone the Git repository associated with the session. From there, they will have the option of using this repository with a larger, fully-featured IDE.

The other primary users that we envision using our product are instructors who are trying to teach a programming language, along with their students. CoNetPad would provide an environment in which an instructor could assign a small programming assignment to students and then watch in real-time as they attempt to solve it. Instructors would have the ability to provide hints through the chat interface and even edit the file the student is working on to provide corrections or additional help. While we don't expect members of this use class will make use of all CoNetPad features, the ability to do so will remain. A feature that we believe will be especially utilized by this user class is the ability to disable chat between students, as well as disable the ability for students to view all files being worked on in the session. This will prevent cheating and other behaviors that are non-conducive to learning.

Secondary user classes include members of groups competing in a programming competition. One example is the programming competition performed at the University of Illinois at Urbana-Champaign called MechMania, a competition that the NDSU chapter of the ACM is involved with. CoNetPad would provide an easy way for all members to work collaboratively and in real-time. Another user class might be students who wish to collaborate with each other on the task of note-taking during a class lecture. Although these students would not likely require all the available features of CoNetPad, they could simply use a subset to accomplish their goal.

2.4 Operating Environment

The CoNetPad application will be written in Java and will run within a Java Virtual Machine. Therefore, it will be available to the Windows, Mac OS X, and Linux operating systems, so long as a Java Virtual Machine is supported on these operating systems – something that we expect will continue for many years.

In addition, the CoNetPad system requires the existence of Git software, an IRC server, and an SQLite database. The relationships between these components are illustrated in Section 2.1.

2.5 Design and Implementation Constraints

The constraint that we perceive will be the most limiting is the requirement that the user experience not be damaged by network performance issues. Since people will only use CoNetPad if it is mostly lag-free, this constraint in non-negotiable. Developers of this software will need to ensure that

CoNetPad can function seamlessly with multiple users on projects containing multiple files. If there is a large amount of lag between the time users type something and the time they see it on the screen, it will be very difficult to use CoNetPad as a serious collaboration tool. Although developers will be able to specify a maximum number of simultaneous users, this number should not be less than ten. In addition, although the minimal set of user requirements can include the existence of a broadband connection, developers will need to ensure that CoNetPad works properly even on slow DSL network connections.

The hardware requirements for this application will be minimal – all that is required is a small amount of disk space for the software installation. Therefore, the options available to developers should not be limited. A set of minimal system requirements will be published with this software that will ensure developers access to the required user resources.

Software constraints include those constraints that apply to subsystems used by the CoNetPad application. Therefore, developers must operate within the constraints of the Git software, the SQLite database software, and the IRC server software. More specifically, they will be limited by the APIs and libraries that create the interface between the CoNetPad Java application and these components. Also, since the CoNetPad application is meant to take advantage of the existing Java Virtual Machine network, it must be programmed using the Java programming language. Developers can use either the Java SE 6 or Java SE 7 version.

Networks connections should be made using SSL for integrity protection, authentication, and confidentiality. In addition, users should be required to provide a password to enter a private session. The password for the session will be created by the session leader and should only be stored in a salted and hashed state.

2.6 User Documentation

Although this product should be simple enough that most people will not require additional documentation, a user manual should be provided with it. Additionally, explanatory tooltips should be used with the user interface.

Since this project is meant to be open-sourced, there is an increased need for developers to fully document their code. This will allow others to understand and modify the source code in an efficient manner.

2.7 Assumptions and Dependencies

It is assumed that a Java Runtime Environment will continue to be supported on the major operating systems (Windows, Mac OS X, and Linux). If this is not true, then our potential audience will be significantly reduced.

In addition, it is assumed that access to APIs and libraries needed for communication with a Git repository, SQLite database, and IRC server will remain available. Since this is currently the case and nothing suggests this will change, we don't believe this is a high-risk assumption.

Software dependencies are described in detail in Section 2.1.

3. System Features

3.1 Service Accessible Over a Network

3.1.1 Description and Priority

The CoNetPad application service should be available over a network. This network could be either a local network or the Internet. Without the ability for users to access CoNetPad over a network, the usefulness of the application is severely restricted. Many other features require the implementation of this feature; therefore, the completion of this feature is a very high priority.

Priority: High

3.1.2 Stimulus/Response Sequences

By specifying the location of an existing CoNetPad server, users of the application should be able to connect to that server through the CoNetPad software.

3.1.3 Functional Requirements

This feature depends on clients having access to a network on which the CoNetPad server application is installed. If clients specify an incorrect server address or are not on a network that has access to such a server, than they will not be able to utilize the networked features of the application.

REQ-1: Existing server that has the CoNetPad server application installed.
REQ-2: A network to which the CoNetPad server and clients are connected.

3.2 Ability for Users to Concurrently Edit a File

3.2.1 Description and Priority

Users should be able to concurrently edit files belonging to a project. Any action, even the entering of a single letter, should be reflected in the files of the other group members in real-time. Since real-time collaboration is a major goal of this software, this feature is very important to complete.

Priority: High

3.2.2 Stimulus/Response Sequences

Whenever a user makes a change to a file, this change should be propagated to all others with minimal delay. This includes adding, modifying, or removing content.

See Section 5.1 for more specific time requirements.

3.2.3 Functional Requirements

The ability for multiple users to edit a file simultaneously (using separate clients) depends on the ability for the application to utilize a network connecting these users. If a user gets disconnected from the network, then the ability for that user to make further changes should be restricted. When the user reconnects, the user's client should re-sync with the server before allowing changes to be made. This will avoid a user getting disconnected, making many changes while offline, and then submitting all those changes at once when the connection is reestablished – possibly at time when those changes are no longer valid or needed.

REQ-1: Network that connects users and on which the CoNetPad server is installed.
REQ-2: Ability for clients to communicate through CoNetPad on a network.

3.3 Use of Color to Differentiate User Changes

3.3.1 Description and Priority

The application should make use of color to highlight changes made by specific users. An unobtrusive light color should be assigned to each user. The specific colors assigned to each user do not need to be the same, given different user perspectives. The important thing is that the colors provide the ability to differentiate changes made by one person from those made by another. A legend should be provided that allows a user to cross-reference a particular color with a particular user.

Priority: Medium

3.3.2 Stimulus/Response Sequences

If User A adds a block of code, then that code should be highlighted with a certain color, say light green. Suppose User B, whose associated color might be light blue, then later changes a single variable name within that block. The overall appearance of that block of code might then be light green, with a single variable name within highlighted in light blue.

3.3.3 Functional Requirements

Although this feature is most useful when multiple users are editing a file simultaneously, it does not require the concurrent editing feature of this software; only a functional network on which the client and server communicate is required.

REQ-1: Ability for clients to communicate with a CoNetPad server located on their network.

3.4 Session Access Control

3.4.1 Description and Priority

An option should be available for a session to have controlled access. The main types of session access can be categorized as restricted, read-only, and read/write access. The type of overall access should be determined by how the session is initially configured and whether or not the user attempting to connect to a session supplies the correct session password (if one exists).

A session may be designated as public or private. Private sessions may be further configured to specify whether users who fail to authenticate successfully (by providing a pre-shared key) have only read access or whether they have no access at all.

User-level access controls may be overridden by a session leader, if that feature is available (see Section 3.6).

Priority: Medium

3.4.2 Stimulus/Response Sequences

<u>For a public session:</u>

Anyone with the session identifier can connect to the session and both read and write to the files.

For a private session with restricted secondary access:

Anyone with the session identifier and correct password can connect to the session and read/write files. Those without the correct password cannot access the session at all.

For a private session with read secondary access:

Anyone with the session identifier and correct password can connect to the session and read/write files. Those without the correct password can still connect to the session and watch as changes are made, but they cannot make any changes themselves.

3.4.3 Functional Requirements

The prerequisites for this feature include the existence of both a password and an access control setting for a particular session, both of which should be determined at the time of session initialization.

The ability to connect to any session will also be dependent upon how many users are already connected. In order to minimize latency, an upper limit to the number of users connected to a session will be hard-coded into the application. Any attempts to join a session with the maximum number of allowed users will be rejected.

REQ-1: Ability to set password for a particular session.
REQ-2: Ability to configure a session's settings to correspond to a specific access control mode.

3.5 Session Leader

3.5.1 Description and Priority

A specific user in a session will be designated as the session leader. The particular user chosen to be the session leader will be the user that initializes that session. Session leaders should correspond to group leaders for a project, since they will be given more power than other users.

Priority: High

3.5.2 Stimulus/Response Sequences

By creating a new session, that user automatically is labeled the session leader.

3.5.3 Functional Requirements

Each session will have at most one session leader. Once a session is created, the specific user designated as the session leader cannot be changed.

3.6 Session Leader Fine-Tuned User Controls

3.6.1 Description and Priority

Session leaders will have the ability to override access control settings on a per-user basis. They may designate a session as being read-only by default, even if the user has

the correct password. The session leader might then allow write capabilities to users manually.

Additionally, a session leader may hide files altogether from a user. This would be useful for when an instructor of a class (the session leader) does not want his students (the other session members) to view each other's work, in order to prevent cheating and promote learning through trial and error.

Priority: Low

3.6.2 Stimulus/Response Sequences

A session leader may configure a session to be read-only by default, including for those users that know the correct password associated with the session. Users who connect must then obtain explicit permission from the leader before they can write to any files.

A session leader may also configure a session to hide all files from all other users, by default. Users would only be able to read and write to files once they are granted explicit permission by the session leader.

3.6.3 Functional Requirements

REQ-1: Designation of a particular user as session leader.

3.7 Multiple File per Session

3.7.1 Description and Priority

Sessions should be able to handle multiple files at once. These files should be able to be switched between by users, provided they are not made invisible by the session leader. Multiple files are common, even in small projects. For example, a small project written in Java will likely contain at least two different classes, which are normally separated into separate files.

Only the session leader has the ability to create or delete files within a session.

Priority: High

3.7.2 Stimulus/Response Sequences

Once a session leader has a created or deleted a file from a session, that file will be added or removed from each user's view. Whenever a user switches to a new file, the current state of the file should be synchronized.

3.7.3 Functional Requirements

In order to easily manage session files in a group setting, only the session leader will be able to create or delete files. Additionally, there may be an upper limit on the number of files a session may hold, if such a limit is needed to ensure an acceptable low level of latency.

REQ-1: Designation of a particular user as session leader.

3.8 Session Chat

3.8.1 Description and Priority

Session users should have the ability to chat with one another using an interface that is separate from the files being edited. This is important because coding projects require communication, even for relatively simple ones. Our group has decided to impose the design requirement that the chat be implemented using an IRC system, although the specific way that it is implemented is not restricted. This is because prior research has been done that shows that existing Java libraries exist for interfacing with an IRC server. In addition, it is desired that chat feature have the IRC "style".

The IRC server that is connected to may either be hosted on the same machine as the CoNetPad server application is on, or it may be hosted on a freely available IRC server, such as Freenode.

Priority: High

3.8.2 Stimulus/Response Sequences

Users will be able to chat through a separate section of the client GUI. Communication through chat will be in real-time, since this feature is inherent in the IRC protocol.

3.8.3 Functional Requirements

REQ-1: Existing IRC server to connect to.
REQ-2: Java API for IRC that can be used.

3.9 Ability to Connect External IRC Client

3.9.1 Description and Priority

Users may desire the ability to connect an IRC client separate from the CoNetPad application to the same IRC channel that CoNetPad uses. This might be because they are already familiar with or want the features of another IRC client. Users of CoNetPad should be allowed to do this by being provided the IRC server URL and channel name.

Since users of CoNetPad that are using an external IRC client do not also need to see the same information in a chat section of CoNetPad, they should also be able to hide the CoNetPad chat window if desired.

Priority: Low

3.9.2 Stimulus/Response Sequences

Users should be able to retrieve the IRC server URL and channel name, in order to connect externally. The chat window within the client should be removed from the CoNetPad GUI for a particular user if that user chooses a specific setting.

3.9.3 Functional Requirements

REQ-1: User-provided external IRC client.

3.10 Ability to Disable Chat Session-wide

3.10.1 Description and Priority

The ability to disable chat session-wide should be available for session leaders only. This feature will be useful for the same reasons that restricted visibility of files among

users will be – in a classroom instruction setting. Although at times an instructor may wish for his students to be able to communicate amongst each other about a project, there may be times when he or she wishes to evaluate students on an individual basis. In these cases, it will be useful to prevent cross-student communication by disabling the chat feature.

Priority: Low

3.10.2 Stimulus/Response Sequences

The session leader should have access to a setting that disables user-to-user chat. Session-leader-to-user chat should not be affected by this setting.

3.10.3 Functional Requirements

REQ-1: A specific user designated as session leader.

3.11 Ability to password protect IRC channel

3.11.1 Description and Priority

The IRC channel that corresponds to a CoNetPad session should be able to be protected with a password. This password should be the same password that is used for the CoNetPad session. If the user is using the CoNetPad integrated chat client, then the password will be automatically entered for the user. No further authentication besides authenticating with the CoNetPad server is required. However, if users decide to use an external IRC client, they will need to enter the correct password in that client as well.

Priority: Low

3.11.2 Stimulus/Response Sequences

If the CoNetPad session requires a password, then the corresponding IRC channel will as well. For external IRC clients, this password will need to be entered correctly before access to the channel is provided.

3.11.3 Functional Requirements

REQ-1: IRC channel corresponding to password-protected CoNetPad session.

3.12 Version control system

3.12.1 Description and Priority

A version control system (VCS) should be used in conjunction with the CoNetPad application to track changes made at discrete intervals. It has been determined that CoNetPad should use the Git version control software as its VCS, although not every feature of Git will be integrated into CoNetPad.

A Git repository should be associated with each CoNetPad session. It is to be used for persistent storage of session files and version control. The specific points in time when the set of files within a CoNetPad session are committed and pushed to the underlying repository are decided by the session leader, through some control only accessible to the leader. In addition, if the files in a session have been modified and have not been manually committed and pushed for a period 30 minutes, the CoNetPad system will automatically perform a Git commit and push to the associated repository.

The reason that the design constraint of using Git for version control has been imposed is because our group members have the most experience with this system. In addition, we wanted to use a distributed VCS, rather than a centralized one, for reasons of flexibility in future upgrades.

The ability to branch from and merge to the CoNetPad session repository will not be supported in this version of the software.

Priority: Medium

3.12.2 Stimulus/Response Sequences

The session leader will be able to manually commit session files and push them to the associated session repository.

Every 30 minutes that pass in a session with modified files, in which session files have not been committed/pushed, the system will automatically perform these activities.

3.12.3 Functional Requirements

This feature requires an API for the Java-based CoNetPad application to interface with the Git software.

REQ-1: Java Git API that is available for use.

3.13 Ability to clone the session repository

3.13.1 Description and Priority

It may be useful for users to clone the Git repository associated with a CoNetPad session in order to work on it outside the application. Although there are no current plans to allow the merging of these repositories back into the user session, the cloned Git repository could be used with conventional development tools and services outside of CoNetPad.

Priority: Low

3.13.2 Stimulus/Response Sequences

Users that specify the desire to clone the Git repository will be able to download a copy of the repository to their local machine.

3.13.3 Functional Requirements

REQ-1: A Git repository that is associated with the CoNetPad session.

3.14 Ability to download source code for a file

3.14.1 Description and Priority

If users do not care about the repository underlying the CoNetPad session (if it exists), they can also download only the source files. A single source file can be downloaded, or all (visible) source files associated with a session can be downloaded.

Priority: Medium

3.14.2 Stimulus/Response Sequences

By triggering some form of control in the application, users can download individual or all source files associated with the session, provided they are visible.

3.14.3 Functional Requirements

N/A

3.15 Auto-complete

3.15.1 Description and Priority

The CoNetPad application should suggest word completions for users as they type source code. This will allow the users to be more productive, without having to memorize function names, class names, and other programming language-specific identifiers.

Priority: Medium

3.15.2 Stimulus/Response Sequences

When the user begins typing a word that must end in a certain way, e.g. "<ClassName>.", then the application should suggest a list of possible endings.

3.15.3 Functional Requirements

Due to the limited time that can be spent on this project, it is required that an existing Java library or tool be used to implement this feature.

REQ-1: Existing library or tool that implements auto-complete.

3.16 Syntax Highlighting

3.16.1 Description and Priority

As users enter source code, the appearance of the code should be modified to allow for easy human understanding of the source code structure. A common tool for this is syntax highlighting, where specific types of words are highlighted in a distinct way to differentiate them from other types of words. These types include keywords, class names, primitive types, and selection or control structures, among others.

Priority: Medium

3.16.2 Stimulus/Response Sequences

Source code should be automatically highlighted (using colors) to make various types of words appear different from other words not of the same type.

3.16.3 Functional Requirements

Due to the limited time that can be spent on this project, it is required that an existing Java library or tool be used to implement this feature.

REQ-1: Existing library or tool that implements syntax highlighting.

3.17 Remote Compilation

3.17.1 Description and Priority

Users should be able to remotely compile source code for certain types of programming languages (see Section 3.19). The application should display any compiler messages to the users, especially compiler error messages due to incorrect syntax.

Priority: Medium

3.17.2 Stimulus/Response Sequences

When the user presses the control to compile the current source code, the compiler output should be displayed to all users.

3.17.3 Functional Requirements

This feature will require that the appropriate compiler for whatever programming language is being developed is installed on the server machine. This will not be feasible for all possible programming languages.

REQ-1: Compiler for particular programming language installed on server.

3.18 Download binary files resulting from compilation

3.18.1 Description and Priority

In addition to seeing the compiler output messages for a remote compilation, users should have the ability to download the binary files corresponding to a compilation directly to their local machine.

Priority: Medium

3.18.2 Stimulus/Response Sequences

When the user presses the control to compile the current source code, the compiler output should be displayed to all users, along with a link to download the binary code associated with compilation. This link will be replaced with a new one whenever a new compilation is performed.

3.18.3 Functional Requirements

This feature will require that the appropriate compiler for whatever programming language is being developed is installed on the server machine. This will not be feasible for all possible programming languages.

REQ-1: Compiler for particular programming language installed on server.

3.19 Support for Multiple Languages

3.19.1 Description and Priority

Support should be provided for various languages, with a focus on the more popularly used language in modern software. We will focus on the Java programming language to begin with. However, in time, various other languages including C# and C++ should be included.

The specific support provided depends on the set of other features that are provided. For instance, syntax support and auto-complete features may be provided for certain languages. Additionally, remote compilation may be offered for a subset of programming languages.

Priority: Low

3.19.2 Stimulus/Response Sequences

By specifying the language for a particular file at the time of its creation, various features specific to that language should be available to users, e.g. syntax highlighting or auto-complete.

3.19.3 Functional Requirements

The functional requirements for this feature depend on the level of support that will be provided. In addition, the availability of free and easily obtainable syntax-highlighting/auto-complete libraries or compilers for specific programming languages will affect whether or not extra support is provided for them or not.

3.20 Open Source Project

3.20.1 Description and Priority

At the end of the development, this project should be released as open source software. The software will be free and other developers will be able to use or modify the software at their own will.

Priority: Low

3.20.2 Stimulus/Response Sequences

N/A

3.20.3 Functional Requirements

TheACMGroup developers should strive to make all code as well-documented as possible, to allow for others to easily understand and modify the CoNetPad code.

3.21 Multi-platform

3.21.1 Description and Priority

It is important that CoNetPad be available on multiple platforms. In this way, the software's potential audience is not needlessly restricted. Given the decision to use Java as the implementation programming language, CoNetPad should be inherently cross-platform, due to the fact that a version of the Java Virtual Machine is available multiple operating systems.

Priority: High

3.21.2 Stimulus/Response Sequences

N/A

3.21.3 Functional Requirements

N/A

4. External Interface Requirements

4.1 User Interfaces

Figure 3:

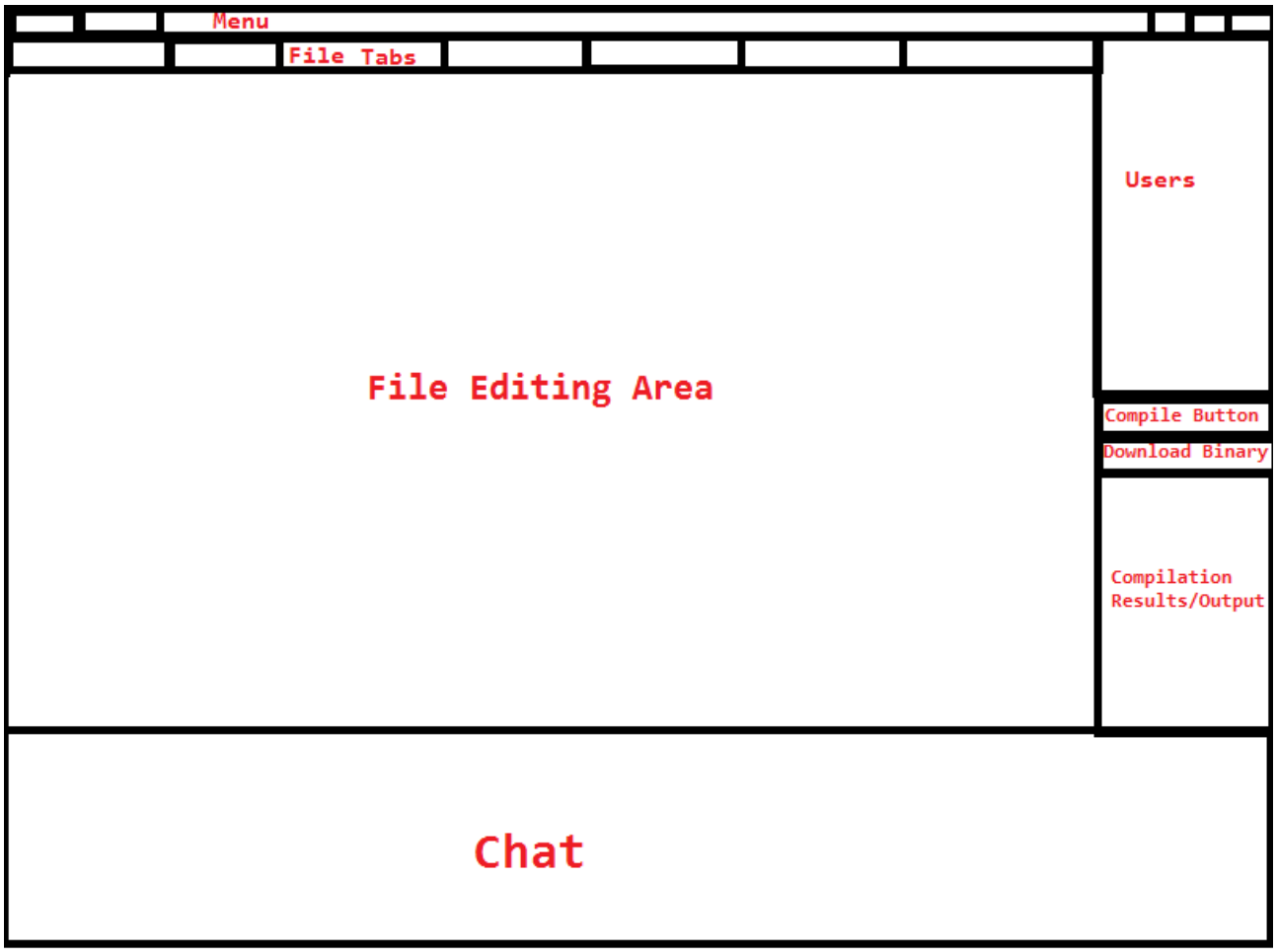


Figure 3 shows the general layout plan for the client user interface. Each source code file will appear in a tab and the source code will be visible in the file editing area. The chat window will allow the user to interact with other group member via IRC chat. Each user should be listed in the "Users" window, along with their associated color (to be used in chat and source code highlighting). The compile button will allow any user to compile the current code. When this is done, the output will be displayed to all users, along with a link to download the binary. All other features will be accessible through a menu system.

CoNetPad tries to be as user friendly as possible for both use and deploy. The client will be developed using java swing and the server will read configuration from a file and command line options.

The interaction with the client will require a keyboard, mouse, and a graphical user interface. The first interaction will require the user to input the address of the server. Once the connection is established the user will be prompted for their username and password. If the user is not registered,

they will be allowed to create an account, using their email address as a unique identifier. Once a user is logged in they will be presented with a view of the current session, the documents opened (if any). The user will be able to write using the keyboard in both the chat box and the document area.

The server itself requires little interaction from the user and only requires a keyboard and a basic command line interface. Configuration files and command line options will be read when the server starts and no other interaction will occur until the server stops.

4.2 Hardware Interfaces

CoNetPad is a standard desktop application and requires a keyboard (either physical or virtual), a pointing device (mouse or touch input), and a display.

4.3 Software Interfaces

CoNetPad will be written in Java and will run on any OS that supports a Java Virtual Machine. For user and session management, the server will interact with a local SQLite database. In order to provide communication between the users, the client will include a chat box that will connect to an IRC server.

To communicate between the server and the client, each client will establish a TCP socket connection for general-purpose communication. One more extra socket will be required for communicating between the client and the IRC server.

4.4 Communications Interfaces

The two communication interfaces are going to use SSL sockets between the server and the client and using the standard IRC protocol between the client and the IRC server.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

In order to achieve a seamless experience for the user, our system has to provide quick responses in the order of magnitude of hundreds of milliseconds or less. To achieve this, we will measure the total time that it takes from the time a command is sent from one client until this command reaches all other clients - we call this interval the transaction time.

If the server is deployed in a local area network, the transaction time should be less than 100ms. In the case the server is deployed over the Internet, the transaction time should be less than 1 second.

We are considering these requirements under reasonable circumstances, like no external traffic blocking the communication or some other kind of technical difficulties. Reasonable circumstances imply an allocation of bandwidth of at least 50 kB/s.

5.2 Safety Requirements

Our product does not require any kind of special safety training or certification.

5.3 Security Requirements

CoNetPad protects its users and their data through the use of two security layers. The first layer is a user management layer that provides the creator of a session the ability to choose which users get access to their sessions and their read/write permissions. Each user needs to follow a simple sign-in process where they define their email address, username, and password.

The second layer is implemented in the sockets that will be used. Rather than using normal sockets, we are using SSL Sockets, which provide encryption and protection against eavesdropping of user information and session data.

5.4 Software Quality Attributes

TheACMGroup has a strong focus in creating the best quality software. CoNetPad is an example of such commitment.

CoNetPad is a service with high availability across both LAN and the Internet. CoNetPad uses the consumer-producer pattern to simplify the concurrency between server and clients. This makes the server highly reliable, even under high stress.

Flexibility is another benefit of CoNetPad. The goal of CoNetPad is to provide a collaboration tool for small teams. Some of the use cases we proposed previously included education and software development. CoNetPad it not limited to this kind of work, as it can be used as a text editor in any area that requires real-time collaboration. The simplicity of deployment and use make it ideal for both technical and non-technical users.

TheACMGroup decided to use Java for various reasons, one of the most important ones being portability.

Appendix A: Glossary

IDE: Integrated Development Environment. A complete software development environment that includes various tools.

VCS: Version Control Systems. Use to track the history of software changes and versions.

Auto-complete: Involves the program predicting a word or phrase that the user wants to type in without the user actually typing it in completely.

IRC: Internet Relay Chat. A protocol for real-time Internet text. Uses “channels” for group communication.

API: Application Programming Interface. Specifies the interfaces that allows software components to communicate with each other.

DSL: Digital Subscriber Line. Technology in which internet access is provided by transmitting data over a local telephone network.

Tooltip: When the user hovers the pointer over an item, without clicking it, a tooltip may appear and provide additional information about the item being hovered over.

II. System Architecture

Figure 4:

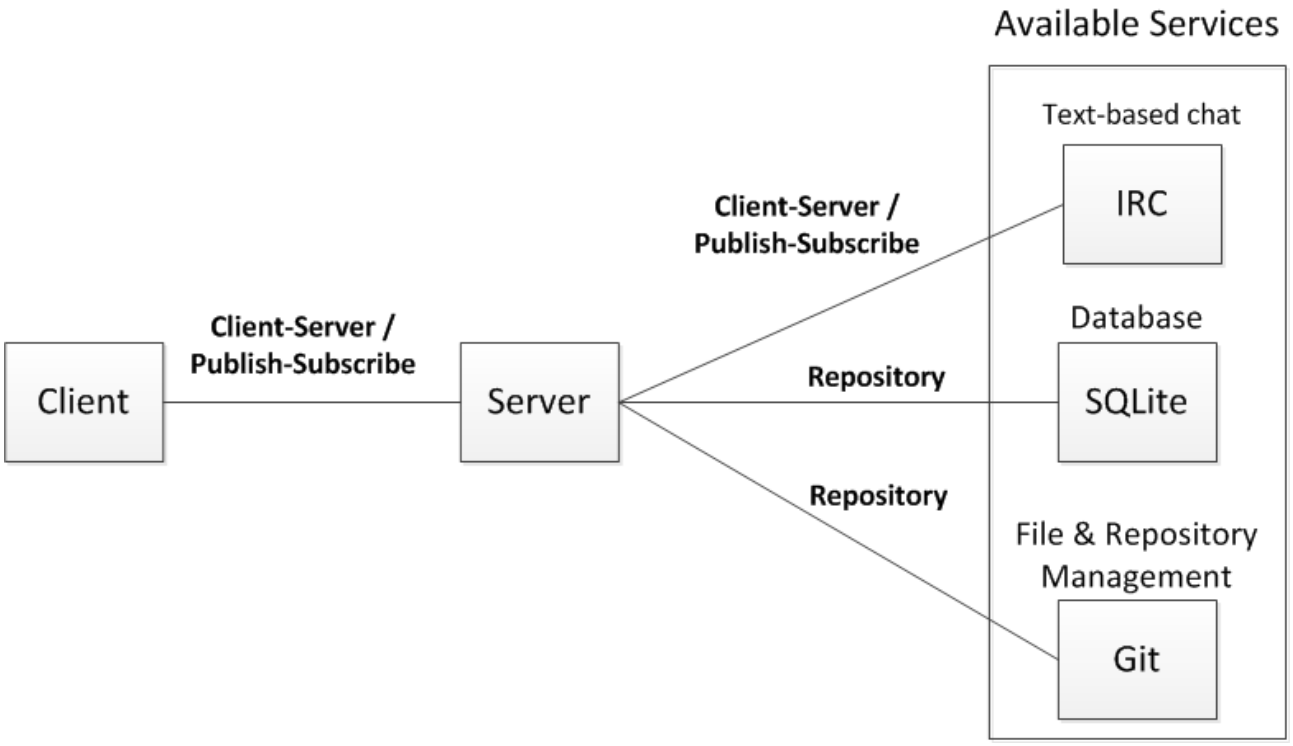


Figure 4 shows the overall system architecture to be used with CoNetPad. The architecture is a hybrid of common system architectures. The architecture used between the client and server components, as well as between the application server and IRC server, is a mix of the Client-Server and Publish-Subscribe architectures. Clients send requests to the server, e.g. to edit their local files. The server then responds with a confirmation – it is only at this time that the client-side files are updated. This is to maintain data integrity. This setup corresponds to a Client-Server architecture. However, since a client must also receive events from other clients (relayed through the server), the architecture also contains a Publish-Subscribe component.

The CoNetPad application server has a Repository architecture in relation to the database and the Git repository. Data is both read from and written to these components. *Figure 4* shows the architecture relationships by labeling the connecting lines.

III. UML Design Documents

1. Class Diagrams

1.1 Abbreviated Class Diagram

Figure 5 is a UML class diagram that focuses on the major classes that will need to be implemented for the CoNetPad software. In this diagram, classes corresponding to the major components (client, server, database, IRC server, and Git repository) are highlighted using a bold outline. Important relationship cardinalities are also shown.

1.2 Expanded Class Diagram

For those who want more detail than what is provided in *Figure 5*, *Figure 6* has been included. *Figure 6* is an expanded class diagram. This class diagram not only includes the major classes than TheACMGroup developers will need to implement, but also those existing classes in the Java Class Library or from component APIs that the created classes will depend on. Relationships to interfaces are also shown. This allows for more clarity in some situations. For example, the differences between the subclasses of CNPTask may be more readily apparent.

Figure 5:

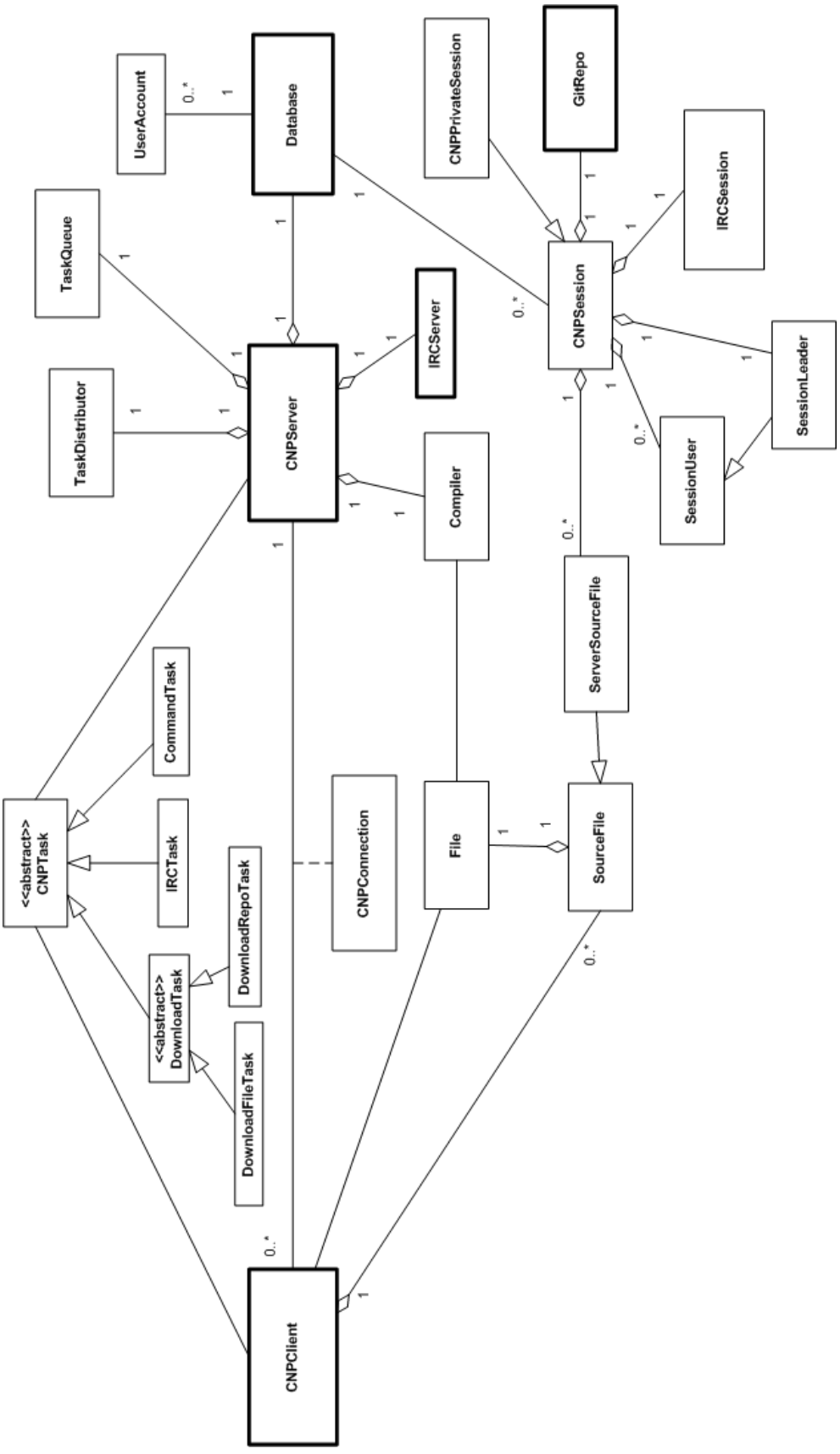
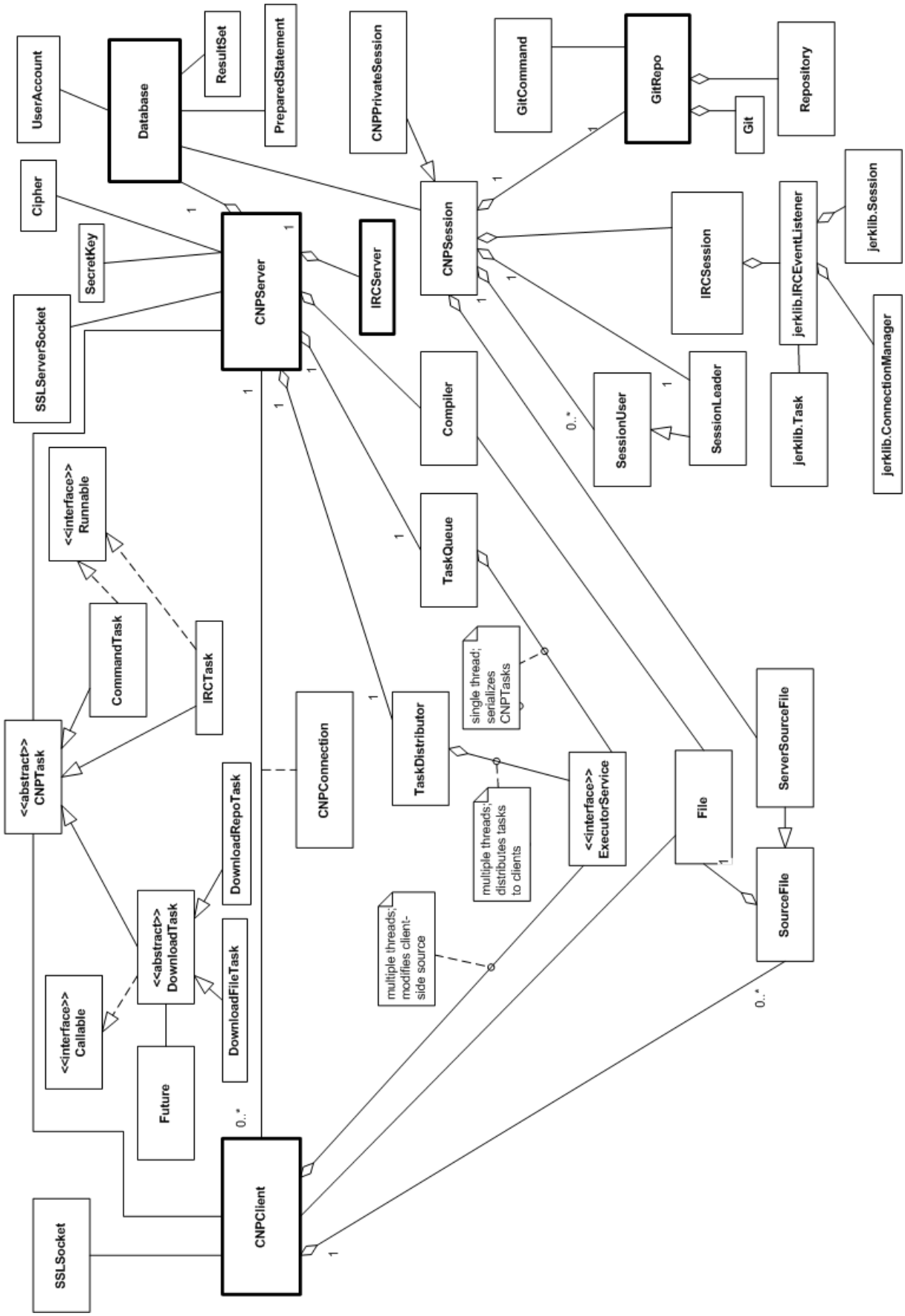

UserAccount
Database
CNPPrivateSession
GitRepo
IRCSession
TaskQueue
CNPSession
IRCServer
SessionLeader
TaskDistributor
CNPServer
Compiler
SessionUser
ServerSourceFile
CommandTask
<<abstract>>
CNPTask
IRCTask
CNPConnection
File
SourceFile
DownloadRepoTask
<<abstract>>
DownloadTask
DownloadFileTask
CNPClient
0..*
1

Figure 6:


UserAccount
Database
ResultSet
PreparedStatement
CNPPrivateSession
GitCommand
GitRepo
Repository
Git
Cipher
SecretKey
SSLServerSocket
CNPServer
IRCServer
CNPSession
IRCSession
jerklib.IRCEventListener
jerklib.Session
Compiler
SessionUser
SessionLeader
jerklib.Task
jerklib.ConnectionManager
<<interface>> Runnable
TaskQueue
CommandTask
IRCTask
<<abstract>> CNPTask
CNPConnection
TaskDistributor
single thread; serializes CNPTasks
multiple threads; distributes tasks to clients
<<interface>> ExecutorService
File
ServerSourceFile
SourceFile
DownloadRepoTask
<<abstract>> DownloadTask
DownloadFileTask
multiple threads; modifies client-side source
<<interface>> Callable
Future
SSLSocket
CNPClient
0..*
0..*
1

1.3 Detailed Class Diagram Tables

In both of the previous two diagrams, class fields and methods have been omitted for the sake of clarity. These components are shown below within their associated class.

CNPClient
- socket: SSLSocket - connection: CNPConnection - clientExecutorService: ExecutorService - sourceFiles: ConcurrentHashMap<String, SourceFile>
+ connect(serverURL: String, sessionName: String): CNPConnection + compile(filenames: List<String>): void + downloadAllSources(): void + downloadSourceFile(filename: String): void + sendIRCMessage(message: String): void + retrieveIRCMessage(): String + sendEditorCommand(command: CommandTask): void + executeEditorCommand(command: CommandTask): void

CNPServer
- clientConnections: ConcurrentHashMap<Account, CNPConnection> - socket: SSLServerSocket - database: Database - compiler: Compiler - taskSerializer: TaskQueue - taskCourier: TaskDistributor - key: SecretKey - cipher: Cipher - ircManager: jerklib.ConnectionManager
+ compile(filenames: List<String>, session: CNPSession): boolean + addTask(task: CNPTask): void + distributeTask(task: CNPTask): void + createAccount(username: String, password: String): Account + retrieveAccount(username: String, password: String): Account + createCNPSession(sessionURL: String): CNPSession + createCNPSession(sessionURL: String, password: String): CNPSession +retrieveCNPSession(account: Account, sessionURL: String): CNPSession + connect(username: String, sessionURL: String): ClientConnection +createIRCSession(session: CNPSession): boolean

CNPSession
- gitRepo: GitRepo - ircSession: jerklib.Session - leader: SessionLeader - sourceFiles: ConcurrentHashMap<String, ServerSourceFile> - permissions: ConccurentHashMap<Account, Account.AccountPermissionLevel> - sessionURL: String
+ createFile(filename: String): boolean + deleteFile(filename: String): boolean + commitAndPush(message: String): boolean + commitAndPush(): boolean + clone(connection: CNPConnection): boolean + addUser(username: String): boolean + removeUser(username: String): boolean

CNPPrivateSession
- sessionPassword: String
+ passwordIsCorrect(attempt: String): boolean

CNPConnection
- session: CNPSession - client: CNPClient - server: CNPServer

IRCServer
- manager: ConnectionManager
+ createSession(sessionURL: String): jerklib.Session

Database
- dbConnection: Connection - driverClass: String
+ createAccount(username:String, password: String, email: String): boolean + retrieveAccount(username: String, password: String): Account + retrieveSession(sessionURL: String): Session + createSession(sessionLeader: String, sessionPassword: String, isPrivate: boolean): boolean +createSessionAccount(session: session, account: Account, permission: Account.PermissionLevel): boolean + retrieveAccountPermission(session: Session, account: Account): Account.PermissionLevel + sessionIsPrivate(sessionURL: String): boolean

Account
- username: String - email: String

<<enumeration>> **Account.PermissionLevel**
READ READ_WRITE UNRESTRICTED

GitRepo
- repo: Repository - git: Git - localRepoPath: String
+ addFile(file: ServerSourceFile): boolean + commit(message: String): boolean + push(): boolean + clone(connection: CNPConnection): boolean + removeFile(file: ServerSourceFile): boolean

Compiler
- compiler: Compiler - javaCompilerPath: String - cppCompilerPath: String
+ compile(files: List<ServerSourceFiles>, outputURL: String): boolean

SourceFile
- fileAsString: StringBuilder - type: SourceFile.SourceType
+ editFile(keyCode: int, location: int): boolean

ServerSourceFile
- file: File
+ writeToFile(): boolean

<<enumeration>> **SourceFile.SourceType**
JAVA CPP GENERAL

TaskQueue
- serializeService: ExecutorService
+ addTask(task: CommandTask): void + addTask(task: DownloadTask): void

TaskDistributor
- distributeService: ExecutorService
+ distributeTask(task: CommandTask): void + distributeTask(task: DownloadTask): void

<<abstract>> ***CNPTask***
- connection: CNPConnection
+ *execute(): void*

<<abstract>> ***DownloadTask***
+ call(): V *+ execute(): void*

<<enumeration>> **DownloadTask.FileType**
BINARY SOURCE

DownloadFileTask
- filenames: List<String> - type: DownloadTask.FileType
+ call(): V + execute(): void

DownloadRepoTask
+ call(): V + execute(): void

IRCTask
- message: String
+ execute(): void + run(): void

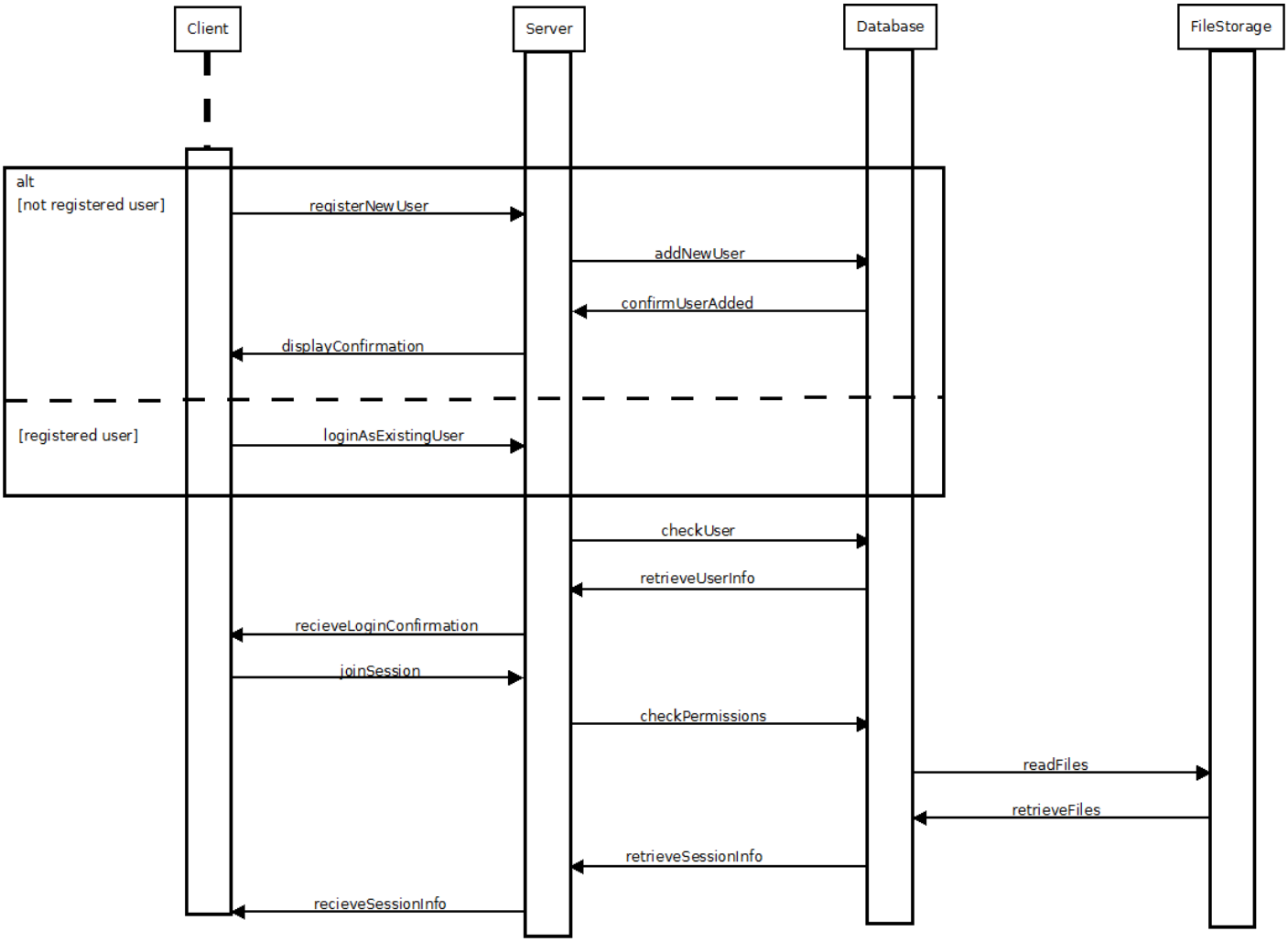
CommandTask
- event: KeyEvent - filename: String - locationInString: int
+ execute(): void + run(): void

2. Sequence Diagrams

2.1 Login Sequence

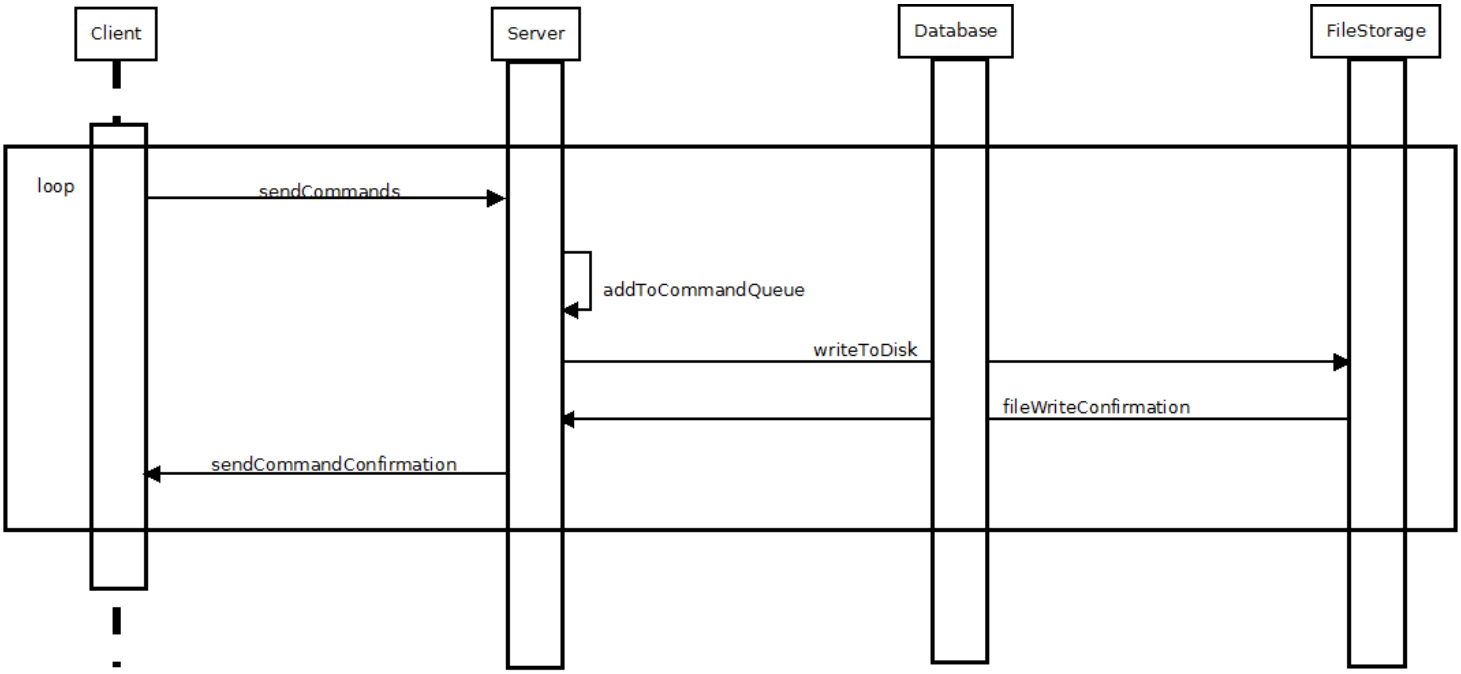
The user will use its credential to identify itself with the server. If the account does not exist, the user will be able to create an account. Once a connection is established the server will gather the user information and present the possible sessions he can join. This is shown in *Figure 7*.

Figure 7:



2.2 Write to Server Sequence

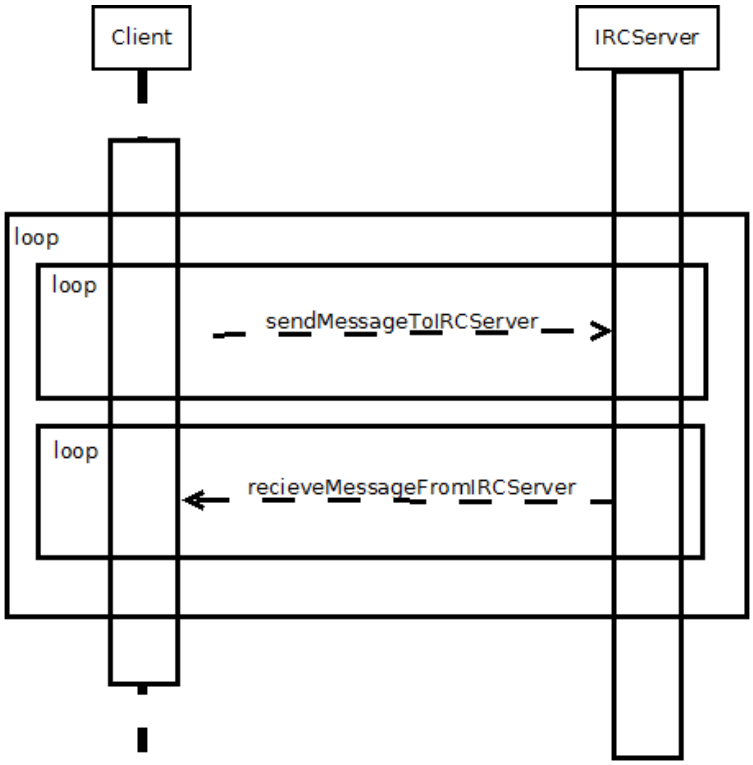
Figure 8:



A message is fired to the server with the action taken by the user. The server will add this command to a command queue and once the action is executed, a message will be sent to the client. (*Figure 8*)

2.3 Chat Sequence

Figure 9:

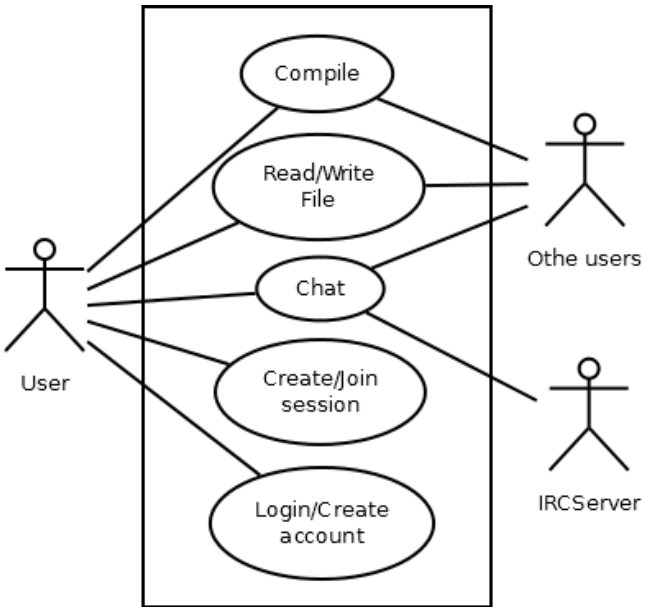


The client and server can send messages asynchronously, as demonstrated by the dashed lines in *Figure 9.*

3. Use Case Diagrams

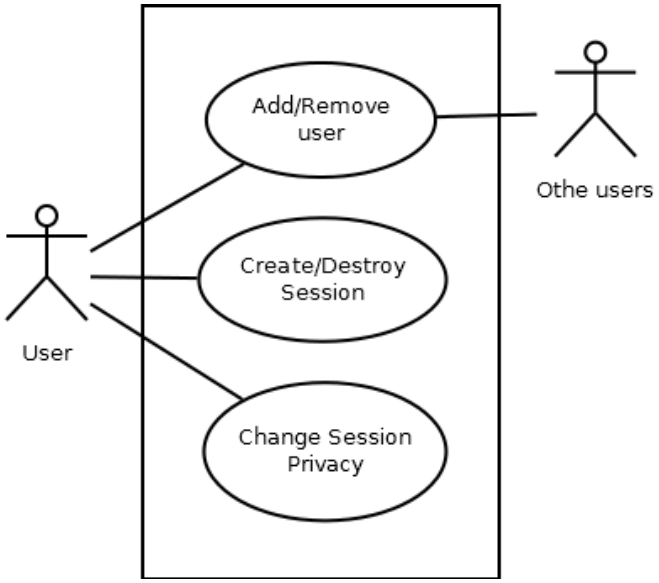
3.1 General Case

Figure 10:



3.2 Exception Case

Figure 11:



3.3 Use Case Text

General Case (*Figure 10)*

Main success scenario

1. User logs in
2. Client joins a session
3. User can modify files
4. User can interact with other users
5. User can compile file
6. Client terminated

Extensions

1a. User is not registered
    1. User creates a new account

2a. Session does not exist
    1. User can create a new session

Exception Case *(Figure 11)*

Main success scenario

1. User logs in
2. Client joins a session
3. User can modify files
4. User can compile file
5. Client terminated

Extensions

3a. User wants to change privacy of session
    1. User can make session private or public

3b. User wants to add or remove users in a private session
    1. User can add or remove users

5a. User wants to remove the session data
    1. User can delete the session if all users have been removed

4. Database ER Diagram

Figure 12:

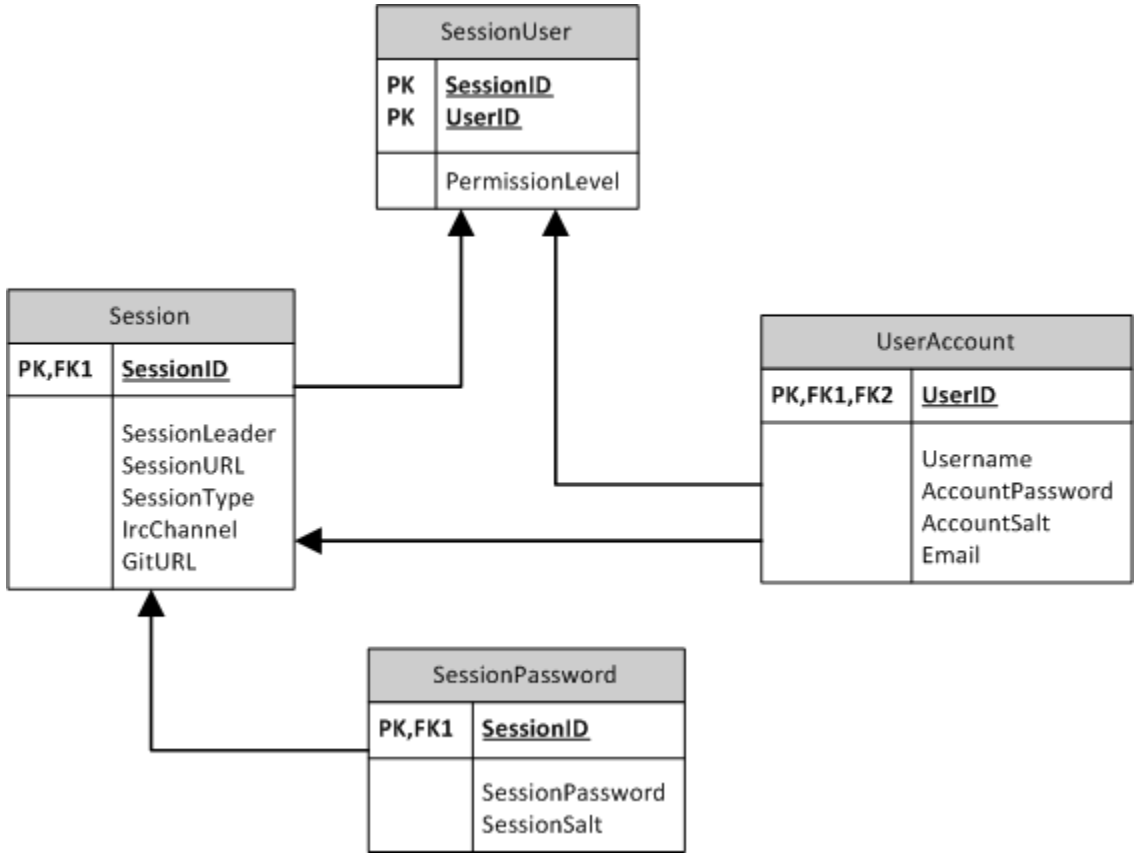


Figure 12 shows the ER diagram for the SQLite database. A SessionUser corresponds to a single relationship between one Session and one UserAccount. A separate SessionPassword table is present in order to avoid NULL values in the Session table – not all session will have a corresponding session password (e.g. public sessions).

IV. Test Plan

1. Performance Requirements

TheACMGroup has developed a test suite that allows for measuring the time it takes a message to go from a client to all the others. This testing tool also supports command line arguments, making automation really simple. We will develop a script for this testing suit so we can run a speed test after every commit. For achieving this, we will configure Git to run the test suit after every successful commit. These automated tests will check the speed of a message under ideal conditions,

but we will also check the speed of the application under real-life circumstances. This test will be done manually twice a week during our meetings.

2. Safety Requirements

Our product does not require any kind of special safety training or certification.

3. Security Requirements

The security of CoNetPad is based on two layers; each one has to be tested in an independent way. The user management is based on a standard UNIX permissions system; but rather than using a password file, we will keep the information in a SQLite database. This information is going to be encrypted as well. Josh Tan has extensive knowledge of software security and encryption. He is going to be in charge of overseeing this security layer. Once the implementation of this layer is completed and tested, it will be frozen - in this way, we will attempt to prevent regressions. The second layer of security within CoNetPad involves the use of SSLSockets. This class is part of the Java API and provides enterprise level of security for communications. We will follow the best practices described by Oracle to make sure our use of SSLSockets is as safe as possible.

4. Software Quality Attributes

To assure the high quality of our software and the satisfaction of our users, TheACMGroup is putting great effort in testing and assuring the correct functioning of CoNetPad. We will use Eclipse to generate unit tests for all classes that we will be using. As we are following an agile approach we will be doing automated tests regularly. All the classes will follow a specified interface and the tests will make sure that no edge case or exception occurs. Once a set of classes are completed they will be integrated and tested. To organize the tests and tasks, each module (set of classes that work together to achieve a task) will be worked on in a separate branch. Before the branch is merged to master it will be tested – if the test is successful it will be merged. Once the first version is completed, the team will start using it regularly during the meetings. Also the application will be shared with members of the ACM to receive feedback from a bigger audience. All these practices will assure CoNetPad will be high quality software. Not only will we find and fix as many bugs as we can, but we will also make sure the interface implementation is as intuitive as possible.

V. Risk Management

Our software project is composed of different elements, the two main ones being the server and client. These components need to interact asynchronously and offer the user an experience of seamless collaboration with their teammates. For achieving this goal, we have defined five major risks that could affect our development cycle.

1. Potential errors related to sockets and network

The number of connections and sockets open at one time can cause unexpected problems, so we therefore have to make sure that our server will be able to catch and handle any exception. Also, both the server and client need to work even in adverse conditions, such as low bandwidth, high latency, or temporary disconnections. For testing these aspects, we will need to test our server and client under different conditions in order to make sure that they will be able to handle any connection problems. These tests will also aid in our understanding of the scalability of our application. We will develop a testing tool for quick and automated testing so that testing can be both frequent and efficient. In the case that we find that our network architecture does not work appropriately, we will simplify it by removing the number of sockets used or reducing the data sent and received. We will then accommodate to our limitations. All of our members have worked with sockets and networking so we are confident that the probability of this risk is relatively low.

2. Problems with database concurrency and data integrity

Having multiple sources edit a single file at once is a common source of concurrency problems and data loss. For overcoming this problem, our team is thinking of a queue-based program design that will prevent the problem of multiple writes to a file while maintaining the sensation of concurrent editing for the users. To make sure this system works, we will need to develop a prototype early on in our development cycle and test it via stress tests so that we can be confident about its reliability. This is an important requirement as this is going to be a main component of our server. Also, regular tests are going to be made to make sure that the reliability is maintained throughout the entire development process. In the case that the database has problems with concurrency which we cannot fix (due to our approach), then we will need to move to a more traditional approach like using a database lock. In the worst case scenario, we would need to remove the real-time editing feature.

3. Latency problems with network

Our application is a collaboration tool that aims to offer real-time collaboration on one or more files. This requires low latency between the time an action is triggered by a user and the time this action is received by all users. For achieving this goal, we will test the client and server under different circumstances, making sure that each message takes less than 1 second between the time it is sent and the time it is received. This testing will be done periodically and after any significant changes in the code to ensure that latency problems do not arise. If latency problems do arise, we may modify the frequency and/or size of the data exchanges on the network.

4. Time constraints due to personal circumstances

There is always the possibility that personal circumstances occur which may conflict with this project. Also, all the team members are active members of the Association for Computing Machinery, which may cause some time conflicts, due to the fact that there are a large number of competitions, conferences and corporate events scheduled this semester. We can combat this risk

using strict schedules and by continuously checking that we are on time with our progress. If any major deadlines happen to occur during an ACM event, we will make sure to schedule our project activities in such a way that we finish the work required for the deadline at a time before the ACM event occurs. If we fail to progress at the expected rate, we may be forced to cut from our feature set. However, we will always emphasize providing the best user experience, even in the case that we must limit our feature set. The availability of one remaining slack day will also be useful.

VI. Project Plan

Figure 13:

Task Name	Duration	Start	Finish	Predecessor
Requirements Analysis	10 days	Mon 9/3/12	Fri 9/14/12	
Design	18 days	Mon 9/17/12	Wed 10/10/12	1
Define Client-Server Interface	3 days	Thu 10/11/12	Mon 10/15/12	2
Server - Session Management	8 days	Tue 10/16/12	Thu 10/25/12	3
Server - Network Component	6 days	Tue 10/16/12	Tue 10/23/12	3
Component Tests + Integration Tests	3 days	Fri 10/26/12	Tue 10/30/12	4,5
Server - Database	3 days	Wed 10/31/12	Fri 11/2/12	6
Server - Git	4 days	Wed 10/31/12	Mon 11/5/12	6
Component Tests + Integration Tests	3 days	Tue 11/6/12	Thu 11/8/12	7,8
Server - IRC	4 days	Fri 11/9/12	Wed 11/14/12	9
Server - Security	3 days	Fri 11/9/12	Tue 11/13/12	9
Component Tests + Integration Tests	3 days	Thu 11/15/12	Mon 11/19/12	10,11
Client - Network Component	6 days	Tue 10/16/12	Tue 10/23/12	3
Client - User Interface	7 days	Tue 10/16/12	Wed 10/24/12	3
Component Tests + Integration Tests	3 days	Thu 10/25/12	Mon 10/29/12	13,14
Client - Command Tasks	3 days	Tue 10/30/12	Thu 11/1/12	15
Client - Download Tasks	4 days	Tue 10/30/12	Fri 11/2/12	15
Component Tests + Integration Tests	3 days	Mon 11/5/12	Wed 11/7/12	16,17
Client - Security	3 days	Thu 11/8/12	Mon 11/12/12	18
Component Tests + Integration Tests	3 days	Tue 11/13/12	Thu 11/15/12	19
Client-Server Integration Tests	5 days	Tue 11/20/12	Mon 11/26/12	12,20

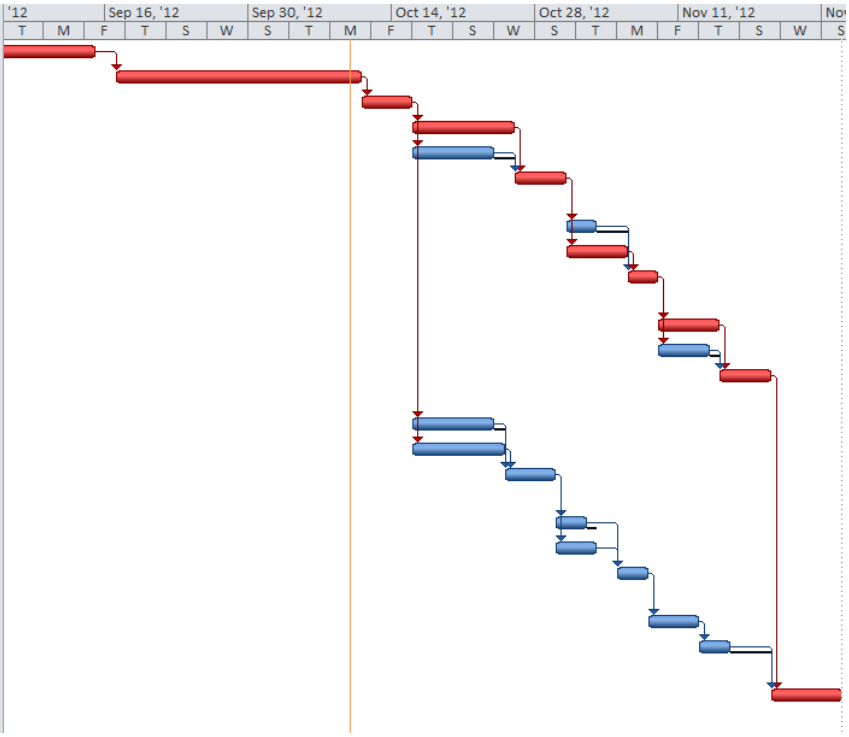


Figure 13 shows the updated project plan for CoNetPad. Knowledge obtained from the design phase has allowed the overall project schedule to be broken down into discrete subtasks. The main division between these tasks is the division between client and server. Our estimations still suggest that completion of the server will take longer than the client, since the critical path falls along the server side.

VII. Meeting Minutes

Meeting Minutes for August 30, 2012

1) Determined which features are required, and the features that were additional.
2) We finished deciding on the project details.
3) We came up with a project title.

4) We broke the project into parts and came up with a tentative plan for the project.
5) We came up with a list of possible risks.
6) We came up with a tentative idea/plan for the presentation.

Meeting Minutes for September 6, 2012

1) Began working on formally defining requirements
2) Looked over SRS template
3) Discussed different possible approaches to system design

Meeting Minutes for September 13, 2012

1) Discussed each section of the SRS template
2) Decided on way in which Git will be used with CoNetPad

Meeting Minutes for September 20, 2012

1) Talked and made decisions on our database
2) Talked and made decisions on design documents
3) Discussed and worked on deliverable 2

Meeting Minutes for September 27, 2012

1) Discussed the remaining tasks that need to be done for Deliverable II
  a) explanation of the architecture of the system
  b) UML design documents
    i) class diagram
    ii) sequence diagram
    iii) use case diagram(s)
  c) ER diagram / relational schema
  d) test plan
  e) updated risk management
  f) updated project plan

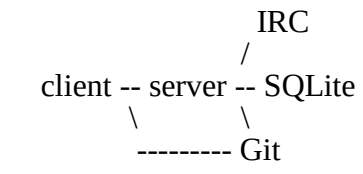
2) use case diagram specifics
  * actors: user (self), users (other than self), IRC server (optional; only if feature allowing connecting to external IRC server is implemented)
  * use cases:
  ** download source/binary
  ** login/create account
  ** create/delete/modify files
  ** edit user settings
  ** chat
  ** clone Git repo
  ** compile source

3) specific classes that will likely be used
  * for the server:
  ** Repository
  ** Account
  ** Session
  ** Compiler
  ** IRCServer
  ** Server

** CommandQueue
* for the client:
** UI
** Connection
*** ServerConnection
*** IRCConnection
** User // may or may not have an associated Account
** IRCCommand
* shared classes:
** SSLSocket
** File
*** ServerFile
*** ClientFile
// Note: these are tentative - will likely change as we progress in the design process

4) specific information that the SQLite database will likely need to handle
* user information
** userid
** username
** password // hashed
** user-specific salt

5) specifics on architecture:



* 3 tier system
* client-server relationship between all directly connected components

6) Plan/Assignments:
* we plan to meet at 9am on Tuesday, for whoever can make it (Cesar and Josh confirmed)
* individual goals for Tuesday:
** Josh - class diagram, system architecture, and updated project plan (also will try to finalize the SRS)
** Cesar - sequence diagram, use case diagrams, updated risk management, and test plan
** Justin - ER diagram, relational schema, user interface layout/design
* the overall goal is to get the majority of Deliverable II done by Thursday

Meeting Minutes for October 2, 2012

1) Discussed what each member was able to accomplish.
2) Listed the components that still needed to be finished and divided the work among members
3) Agreed upon the overall, high-level design for the classes.

Meeting Minutes for October 9, 2012

1) Discussed the ER Diagram for the database. Decided on the various tables that the database would hold.
2) Discussed the system architecture and checked with Dr. Do to ensure it was appropriate.

3) Discussed changes that needed to be made to the use case diagram - needed to add a few use cases.
4) Divided the remainder of the project tasks into subtasks - to be used in the updated project plan.
5) Assigned work to be done that day for the tasks still unfinished:
   Cesar: use case diagram/text fixes, sequence diagram fixes
   Justin: ER Diagram, System Architecture
   Josh: UML Class Diagram, Project Plan, meeting minutes

VIII. Progress Report

Our group has successfully completed the software requirements analysis and definition stage of our project. The majority of the software design has also been decided upon. This includes detailed class diagrams that specify much of the lower-level software design. These two stages took longer than we initially expected, so the project plan has been modified to reflect this.

Although the software design has been mostly completed, it will likely be changed as the project progresses. This is particular true of the design for the source file classes. Currently, each file is represented as a string and this string is periodically written to a file (which is periodically pushed to a Git repository). Our group is considering using some form of a random-access file instead of the string design, in order to improve efficiency.

The next step will be to clearly define the interface between the server and client components. With this in place, we will be able to work on these components separately and in a modular way. After this is done, the actual implementation phase can begin.