The ACM Group – CoNetPad
Team Project - Deliverable 1
CSCI 413

1. **Project Title**:

   CoNetPad – The name is derived from the fact that the project software is meant to be a **co**llaborative, **net**work-based, editing tool/**pad**.

2. **Project Description:**

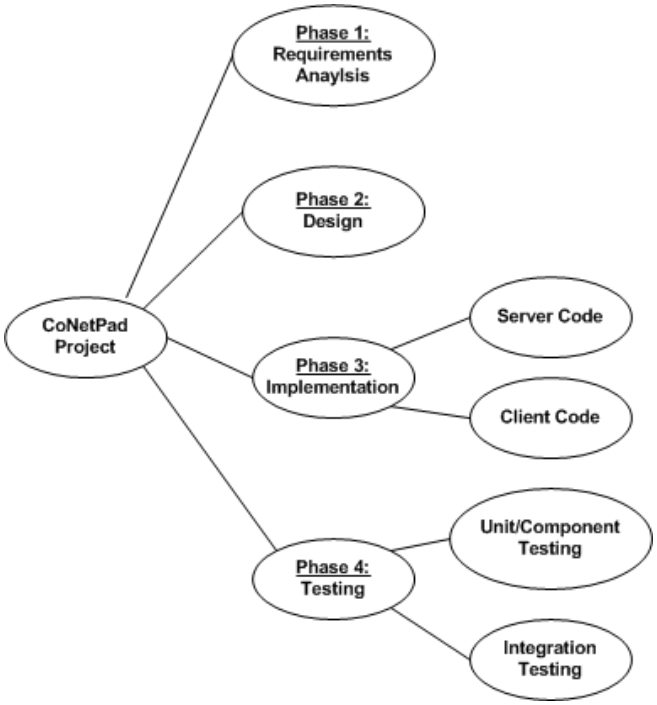
   The objective of CoNetPad is to offer a simple solution for real-time collaboration in small software projects. It is especially targeted for use by small groups or for educational purposes in a mentor-mentee environment. Based on our research of the existing network-based collaboration tools, we have found that most of them fail to offer real-time collaboration. In the case that they do, the tools tend to require a long deployment process or are non-free services offered by a company. With this project, we are trying to offer a program with simple deployment and ease of use. A motivation we had for developing this software was the lack of a simple collaboration tool that could be used for different ACM programming and AI competitions, where traditionally only a single person was able to code at any given time. Another motivation for this project is the functionality that this kind of application could have as an educational tool. This application could allow kids to collaborate together in a simple programming project. It would also allow a professor or instructor to interact with students by providing real-time feedback within the editor itself. To increase the audience of our application, we are going

to be using Java as our programming language, mainly due to the popularity of this language and its compatibility with different operating systems.

3. **Project Plan:**

Our project can be broken down at a high level to consist of the following four phases: requirements analysis, design (system and program), code implementation, and testing. Each of these phases will be broken down further as the course progresses and the techniques for doing so are learned. At this point, we are certain that the implementation phase will be further broken down into two major steps – the implementation of the server code and the implementation of the client code. Additionally, the testing phase is planned to be divided into unit and component testing, as well as testing for component integration. This work breakdown is summarized by the following diagram:

Figure 1: Work Breakdown Diagram



The major activities for our project will thus consists of analyzing and rigorously defining the project requirements, designing the project (system and programs), implementing the server and client code, testing both components, and then testing again once the components are integrated.

The implementation of the server and client code does not need to happen sequentially. Instead, by clearly defining the interface to be used between the server and client (in the design phase), these two steps can be done simultaneously. Each component can also be tested independently. After the components are integrated into the overall system, another round of testing will ensure that synthesis problems have not arisen. The parallelization of the activities in our project is demonstrated in Figure 2.

Figure 2 is an activity graph that shows the milestones corresponding to each activity in our project. It includes the estimated number of (business) days that each activity will require to be completed. For example, we estimated that the server implementation will take approximately 25 days, whereas the client implementation will be less difficult and only require 20 days. Using these estimates, the project schedule can be further analyzed using the Critical Path Method. Using this method, we find that the minimum amount of time required to complete the project is 62 (business) days.

Figure 2: Activity Graph

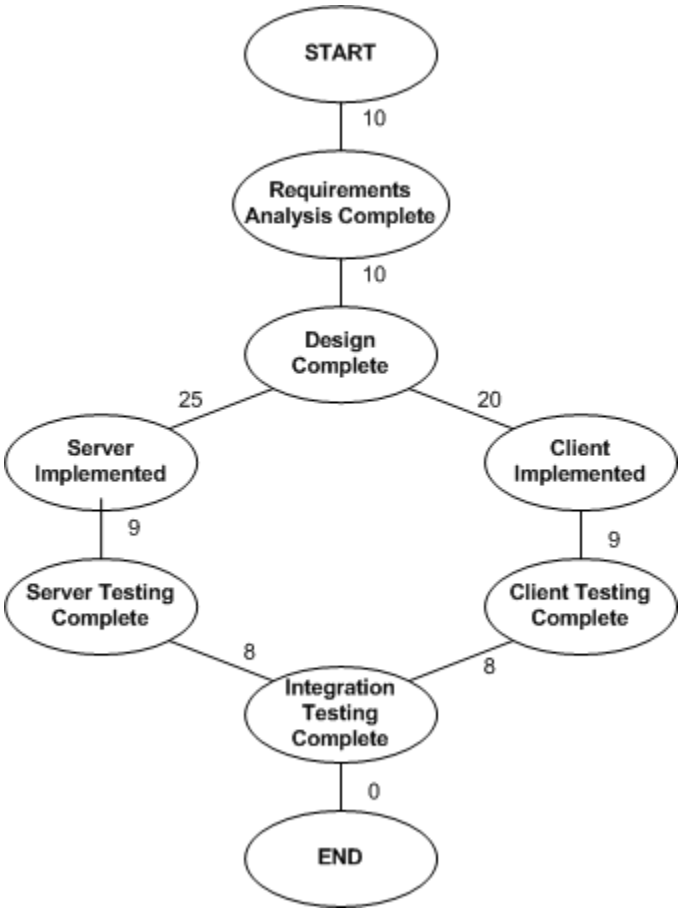


Figure 3: Task List

Task Name	Time Estimate	Earliest Start Time	Latest Start Time	Slack
Requirements Analysis	10 days	1	1	0
Design	10 days	11	11	0
Server Implementation	25 days	21	21	0
Server Testing	9 days	46	46	0
Client Implementation	20 days	21	26	5
Client Testing	9 days	41	46	5
Integration Testing	8 days	55	55	0

The various activities that comprise our project are shown in Figure 3 in a task

list. This list shows the time estimates for each activity, along with the earliest and latest times (in days) that each activity can be started. Each task that falls along a critical path is associated with 0 slack days. Due to our estimation of the client implementation taking less time the server implementation, we can see that both the client implementation and client testing have 5 associated slack days.

A Gantt chart (Figure 4) and a PERT chart (Figure 5) provide alternative ways to visualize the activities that comprise our schedule and their relation to one another. In both charts, the critical path is colored red, while the non-critical paths are colored blue. It is assumed that the first activity will begin on September 3 and the last activity will end on September 27 (coinciding with the due date for the Deliverable 3 code).

Figure 4: Gantt Chart

Task Name	Duration	Start	Finish	Predecessors
Requirements Analysis	10 days	Mon 9/3/12	Fri 9/14/12	
Design	10 days	Mon 9/17/12	Fri 9/28/12	1
Server Implementation	25 days	Mon 10/1/12	Fri 11/2/12	2
Server Testing	9 days	Mon 11/5/12	Thu 11/15/12	3
Client Implementation	20 days	Mon 10/1/12	Fri 10/26/12	2
Client Testing	9 days	Mon 10/29/12	Thu 11/8/12	5
Integration Testing	8 days	Fri 11/16/12	Tue 11/27/12	4,6

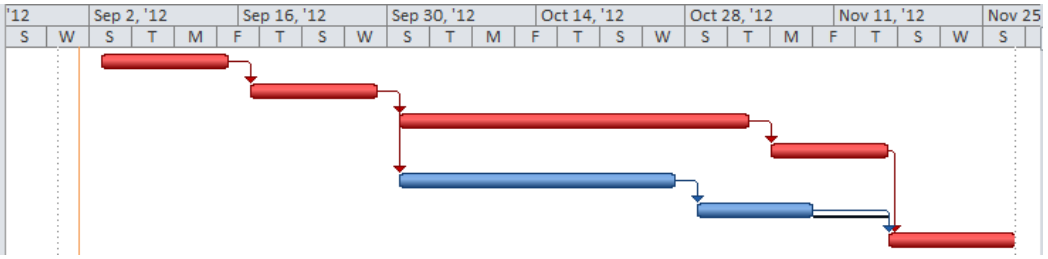
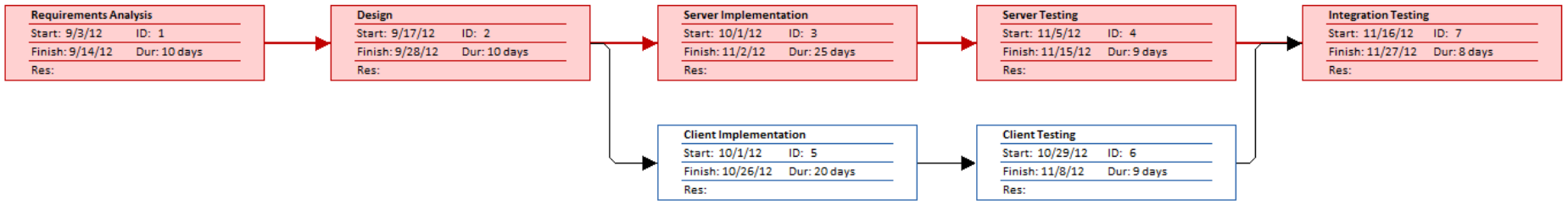


Figure 5: PERT Chart



4. **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.

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.

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.

Difficulties with highlighting and auto-complete:

Highlighting and auto-complete are two features that relieve the necessity of memorization and allows the programmer to focus on the functionality. But the implementation for these features is still unclear for the team. The team will study the process of implementing this feature early on in the design process so that the decision of implementing it or not is taken early on. This way, we can prevent unnecessary lost time in the development process. If the decision is made to

implement the code highlighting and auto-complete features, periodic testing will be done (under various network conditions) to ensure that the feature does not end up detracting from the user experience. In the worst case scenario, one or both features may be removed.

Latency problems with network:

Our application is a collaboration tool that aims to offer real-time collaboration on more 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.

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.

5. **Progress Report:**

   The ACM Group consists of Josh Tan, Justin Anderson and Cesar Ramirez. The specific role for each member has not been defined yet, as each member has a broad experience in design, coding, and testing. As the development cycle progresses and as the overall design is formulated, the specific roles will be defined. Our team will follow a distributed, egoless team structure. Based on our team size and personal experiences, we consider each of us to be capable of taking appropriate decisions. Also, we believe that the teamwork and communication inherent in such a democratic team structure will allow us to produce quality software. We will use Git as our configuration management tool, instead of SVN. The repository update policy will require that each member commit their work at least twice a week - once on Tuesday night and again on Thursday night. With this policy, each member can track the progress of the overall project in a more frequent manner. Frequent updates also allow for a more seamless integration between the different components of the application.