Elaboration Phase Status Assessment

Confide Instant Messaging Application

Table of Contents

[1 Definitions and Acronyms 4](#_Toc485014150)

[2 Critical and Significant Risks 4](#_Toc485014151)

[2.1 Choosing a protocol, server, and client side API 4](#_Toc485014152)

[2.1.1 Risk – technologies chosen not suitable for CCRD 4](#_Toc485014153)

[2.1.2 Outcome - Successful 4](#_Toc485014154)

[2.2 Learning and successfully using the client side API. 5](#_Toc485014155)

[2.2.1 Risk – Learning curve required to implement API 5](#_Toc485014156)

[2.2.2 Outcome - Successful 5](#_Toc485014157)

[2.3 Implementing and understanding MVC architecture 5](#_Toc485014158)

[2.3.1 Risk – Codebase needs good structure 5](#_Toc485014159)

[2.3.2 Outcome - Successful 5](#_Toc485014160)

[3 Issues Encountered 6](#_Toc485014161)

[3.1 Contact add bug 6](#_Toc485014162)

[3.1.1 Diagnosis and Testing 6](#_Toc485014163)

[3.1.2 Resolution 6](#_Toc485014164)

[4 Executable Architecture 6](#_Toc485014165)

[4.1 CCRD Use Case 6](#_Toc485014166)

[4.2 Remaining Use Cases 6](#_Toc485014167)

[4.3 Implementation 7](#_Toc485014168)

[4.4 Deployment 7](#_Toc485014169)

[4.4.1 Server 7](#_Toc485014170)

[4.4.2 Client 7](#_Toc485014171)

[4.4.3 Testing 7](#_Toc485014172)

[5 Final Project Plan 7](#_Toc485014173)

[6 Testing 8](#_Toc485014174)

[6.1 Master Test Plan 8](#_Toc485014175)

[6.2 Test Plan 1 8](#_Toc485014176)

[6.3 Test Plan 2 8](#_Toc485014177)

[6.4 Test Plan 3 8](#_Toc485014178)

[6.5 Test reporting 8](#_Toc485014179)

[6.6 Acceptance Testing 8](#_Toc485014180)

# Definitions and Acronyms

|  |  |
| --- | --- |
| Definition/Acronym | Description |
| chat system/system | Confide Instant Messaging Application (CIMA) |
| API | Application Programming Interface |
| user | Employee or operator granted permission to use chat application for Pinnacle Corporation business |
| XMPP | Extensible Messaging and Presence Protocol |
| account | User chat account |
| chatroom | A container for multi-user chat sessions |
| CCRD use case | Critical Core Risky Difficult Use case |
| thread | Sequence or transcript of chat messages between two users or a group of users |

# Critical and Significant Risks

## Choosing a protocol, server, and client side API

### Risk – technologies chosen not suitable for CCRD

We explored several protocols, and quickly decided on the XMPP protocol. This narrowed down the possible servers and client side APIs. Our strategy was to get the server and client side API up and running as fast as possible, to avoid wasting time and to mitigate the CCRD use case as quickly as possible.

We found a good server called Openfire, and set that up. There were 3 appealing client side APIs, so we each implemented a basic program in each to both test the 3 APIs, and at the same time the Openfire server.

### Outcome - Successful

The outcome is that we chose to stay with the XMPP protocol, and the Openfire server. And although we thought we were going to go with the Smack API, we decided to go with the Babbler API. If we had not performed these tests, we would have gone with the Smack API; and we believe that this would have hurt our project.

This risk is now resolved, as we have successfully chosen a protocol, server, and client side API.

## Learning and successfully using the client side API.

### Risk – Learning curve required to implement API

We all had to read the documentation for the Babbler API, to initially become familiar with it. To gain further understanding of the API, we each took some functionality each iteration that required connection to the server via the API, requiring us all to become more familiar with it.

Implementing real functionality was a great way to understand the API more deeply than we could have from just reading about it.

### Outcome - Successful

The application is progressing well in terms of functionality so the use of this API has proven to be successful both in terms of relative ease of use and effectiveness in implementing new features for the application.

This risk is now resolved, although (as expected) it is still an ongoing challenge to grow our understanding of the Babbler API.

## Implementing and understanding MVC architecture

### Risk – Codebase needs good structure

To avoid a huge mess of code as our codebase grew, we needed to implement an architecture with some structure and rules. We decided to go with the MVC architecture. It took some effort to initially get up and running, and to get everyone to understand how it was working. To mitigate this risk of not everyone understanding this architecture, we have all attempted to implement some of the functionality that uses this architecture.

Although the architecture was initially difficult to get working, now we have a template that we can follow when creating new functionality, which makes it very simple to extend.

### Outcome - Successful

The outcome is that the MVC architecture is working well, and we are still using it. All team members are growing their understanding of it by continually spending time in the code base.

# Issues Encountered

## Contact add bug

We encountered an issue where we were adding contacts successfully, and they were staying in our contacts as we were logging in and out, but when that user would log in, then the next time we log in, that contact is gone. This was very hard to debug as it was hard to get in and see what was going on in the server. It was a very significant risk as it related to the CCRD use case and therefore we could not do without this feature.

### Diagnosis and Testing

To resolve this risk, we set up other XMPP servers to test our code against them and determine if the problem was in our code or in the server, set up our current server (Openfire) locally to see if that would make any difference, and continued to explore our code to look for a solution.

### Resolution

In the end another issue was found to do with requesting a presence subscription, this functionality was added at the end of the last iteration in the merge; and when solving this issue, the original contact-add issue was also resolved.

The outcome is that we successfully resolved this risk, and were able to move on with the project.

For more information see [this document](https://github.com/csuproject/instantmessenger/blob/master/docs/Elaboration/ContactProblemFindings.docx).

# Executable Architecture

## CCRD Use Case

We do have an [executable architecture](https://github.com/csuproject/instantmessenger/blob/master/docs/Elaboration/ScreenShots.docx) which that covers all of our CCRD use cases.

Our CCRD use cases are:

* Create an account
* Add a user to contacts
* View contacts
* Chat
* View chats

We chose these CCRD use cases because the main CCRD functionality for our app is the ability to engage in a chat. However, to do this you need to be able to create an account, add a contact to chat with, view your contacts to choose a specific contact to chat with, and view the chat with that contact. So, all of these CCRD use cases support the Chat use case.

## Remaining Use Cases

The aspects of the architecture that were left out are:

* Group chat
* Manage contacts

This is because of limited time. We already have chat, so that just needs to be extended to group chat, and we already have some ability to manage contacts, so that also just needs to be extended.

## Implementation

All flows for the CCRD use cases were implemented.

End-to-end production level support for the CCRD use cases was reached to the extent that each component of the CCRD Use Cases can be smoothly and easily done with a decent user interface, the reason it is not more polished is simply because of time constraints.

## Deployment

### Server

The Openfire server is currently deployed and can be accessed at [teamorange.space:9090](http://teamorange.space:9090/login.jsp?username=jim). Password is “jim”.

### Client

The client side desktop app we have been developing is available as an executable jar file at <https://github.com/csuproject/instantmessenger/tree/master/ExecutableJar>.

Download the zip file, extract it, and run the file TeamOrangeInstantMessenger.jar.

### Testing

To test the app without two computers, two instances of the app can be run side by side on one computer, log in as two different users, and chat between them.

# Final Project Plan

We have got a final project plan. It has not really been rescoped, but more so items have been moved around as we have learned how to split the development up into iterations and work items.

The project plan adds up to us reaching our standard target from our initial plan, which we are happy with and see as a good goal.

# Testing

## Master Test Plan

We have a master test plan, which we used to help us plan out our iterations, particularly the earlier iterations; as the later iterations were more influenced by what did and didn't go to plan in previous iterations.

## Test Plan 1

The first test plan was about the functionality of creating an account. We had a series of steps with expected outcomes; and these were all successfully completed.

## Test Plan 2

This test plan was about the functionality of contact management. We successfully completed all steps, including adding, retrieving, and creating contacts.

## Test Plan 3

Test Plan 3 tested the functionality of engaging in a chat. We successfully completed all steps, including sending messages, receiving messages, and notifying users.

## Test reporting

Test scripts with a series of steps and expected results were created for all CCRD functionality.

All test scripts were successfully completed without complications.

The documents and recorded outcomes can be seen here:

[Test Script 1 - Create Account](https://github.com/csuproject/instantmessenger/blob/master/docs/Elaboration/TestScripts1-CreateAccount.docx)

[Test Script 2 - Login](https://github.com/csuproject/instantmessenger/blob/master/docs/Elaboration/TestScripts2-Login.docx)

[Test Script 3 - Contacts](https://github.com/csuproject/instantmessenger/blob/master/docs/Elaboration/TestScripts3-Contacts.docx)

[Test Script 4 - Chat](https://github.com/csuproject/instantmessenger/blob/master/docs/Elaboration/TestScripts4-Chat.docx)

## Acceptance Testing

We did not perform formal alpha level user acceptance testing, however functionally our chat client has advanced to the point where we would be likely to get good results from such testing.