William Trefethen

# Talk

Test Plan Document

## Table of Contents

# 1. Introduction

The purpose of this document will be to explicate the test plan for the Talk java application currently under development. It will outline the scope of testing and the methods being utilized. It will also examine both unit-wise testing needs and integration-wise testing needs.

## 1.1. Purpose and Scope

This document will serve as a guiding and comprehensive document for the testing process of the initial development of the Talk application. Its scope will be all necessary testing arrangements for the application.

## 1.2. Target Audience

The target audience is both developers/testers and clients. For developers and testers this document will server as an authoritative source from which to guide processes. For clients it will serve as an informal contract and scope-of-work document with which to hold developers accountable.

## 1.3. Terms and Definitions

**Unit**: a somewhat independent system within the larger software product which can be individually examined for feature-fullness and error-proneness
**Integration**: this term refers to places where units meet and interact - a classically error prone area which together with units makes up the majority of testable sub-systems.

# 2. Test Plan Description

This section will review the scope, schedule and criteria of the test plan.

## 2.1. Scope of Testing

The scope of testing will be comprehensive with respect to the functional requirements of the product. Using this method, all possibly use cases that users can enact on the system must be tested - i.e. if there's not a button for it there's not a test for it. In addition to this functional driven testing, individual units will be tested feature wise, meaning that all exposed function they offer will be tested with possible inputs (normal and edge cases) for the enforcement of reusable, stable code. This unit-wise testing will cover more unforeseen issues as well as prepare the code to be maintained over the evolution of the product.

## 2.2. Testing Schedule

Unit-wise testing will be performed as the units are developed. Due to high interactivity between some of the modules, and the inevitable stack-up effects any lower level issues could cause, each new unit will be unit tested upon completion and before its formal integration. The general schedule is that networking/packets will be first, followed by the user system, the chat system, and finally the GUI system. Upon each of these sections being developed, unit tested, and integrated, integration testing will be performed between the newly introduced module and the currently in place ones. This test-as-you-go method will serve to minimize largely impactful low level errors later on in development or maintenance, greatly reducing possible unforeseen costs.

## 2.3. Release Criteria

Obviously, the system must be error free in all functional requirements it serves to fulfill. This means the entire use-case set must be error-free in all possible inputs. In addition to this it must be free of errors in any possible user flow the GUI allows. The product will be allowed faults in relation to transport-layer issues involving packet dropping, connection loss etc (although the system is utilizing TCP protocol which should ensure these not to be issues).

# 3. Unit Testing

This section will outline the individual units in the product which must be tested and the use cases by which they will be tested

## 3.1. Packet System

This unit is responsible for delivering data asynchronously back and forth between servers and clients. Its basic job is to take data, package it up, send it across the network, unpack it, and give it back to the other units. It is designed to receive and send non-block and asynchronously. The only publicly exposed function is to add a packet to the socket. In some circumstances, a connection may be lost - these types of transport and network layer issues will not be tested for. However, to ensure networking is not faulty over an average connection, the system will be tested on separate machines (server on one, client on the other).

### 3.1.1. Add a Packet

Add a generic packet to a socket, the expected result is for the packet to be successfully unwrapped and received on the other machine's socket. This must be tested server to client as well as client to server. Edge cases include: multiple clients connected, single client connected - each including two sub-cases (server->client and client->server).

### 3.1.2. Client Disconnect

When a client socket disconnects the server needs to properly shut down its socket for that client (not loop, keep the thread running, etc). The test will be to connect a client, then disconnect it, then reconnect. If the server successfully shuts off the socket, then re-opens one upon the next connection, it has passed. Edge cases include multiple and single client connections as well.

## 3.2. User System

This system is responsible for saving and loading user data, registering new users, and handling login/logout of users. These responsibilities will be the guide for the tests required.

### 3.2.1. Load Users

When the load users function is called, the unit must (given a correctly formatted JSON file) populate its users structure. Edge case: zero users in the JSON file. Invalid JSON files will not be tested and will be considered out of scope.

### 3.2.2. Save Users

When the save users function is called, the unit must create a correctly formatted JSON file for the users structure. File IO errors resulting from permissions will not be tested for and are considered out of scope. Edge case: 0 users in structure.

### 3.2.3. Login

Given a username and password attempt, return either the user id if the login was valid, or one of 2 error codes for either (-1) incorrect password or (-2) username not found - additionally add the user to the online users structure. This function will be tested with no users in the structure.

## 3.3. Chat System

This system is responsible for saving and loading chat data, the creation of new chats, and the processing of new messages in chats. Thus, these are the functional use-cases it must be tested for

### 3.3.1. Load Chats

When the load chats function is called, the system must populate the chats structure using a properly formatted JSON file. This will be tested with 0 chats as well as a single and multiple. This will not be tested for supporting incorrectly formatted JSON files.

#### 3.3.2. Save Chats

When the save chats function is called, the system must correctly populate a JSON file with the chats structure. This will be tested with 0, 1 and multiple chats in the structure.

#### 3.3.3. New Chat

When the a new chat is requested, the system must correctly add the new chat object to the structure and link the users to the chat. This will detested with 0, 1, and multiple chats in the structure.

#### 3.3.4. Enter Chat

When the a new user is created, they must be added to the global chat, so the feature of adding to a user to any chat will be tested for unit-level stability. This will be tested on a user with no chats, as well as a user with multiple chats.

#### 3.3.5. Chat Message

Given a new message from a user, the system must add that message to the correct chat object and broadcast the message to other online and listening users. This will be tested when no other users are online, 1 other user is online and listening, and multiple are online and listening. It will also be tested with a chat containing only 2 and a chat containing many users.

### 3.4. GUI System

This system is responsible for correctly displaying data and allowing user input. It will be developed and tested last to ensure that the data it is being given is correct and the functions it implements work as planned. From this basis, anything that gores wrong must be within the GUI.

#### 3.4.1. Screen Push/Pop

Push a screen to the screen stack. If the screen displays, then pop it, if it reveals the original page, the screen stack is correctly working. This will be tried with a 1 deep, as well as a many deep screen stack.

### 3.4.2. Display Alert

Given a string, display it as a pop-up message that can be closed out. This will be tested both with and without a display already being present to be comprehensive with respect to overriding notifications.

# 4. Integration Testing

Once each of the units are developed and unit tested, each one can be insured to work in and of itself. From here, the unit is worked into the full system along with all previously integrated units - this is the process of integration. The first step of this process is integrating users and chat with packets. In fact, users and chat need to be developed using packets, so their integration happens simultaneous to their development (although as much separation and independence as possible is maintained to reduce error-proneness). The second step of this process, once the entire system is working with a command line interface, is to develop and integrate the GUI system.

## 4.1. Packets with User Logins/Logouts

The packet system (3.1) will be used very closely with the user system - namely in the login/online features. Sometimes, packets may behave differently in their handlers once a socket is connected to a legitimately logged-in user. Because of this, once the user system is in, all packets will be tested with a logged in and a non-logged in client to ensure access restricted packets are behaving correctly.

## 4.2. Message Notifications While Listening to a Chat

Once chat is in, it needs to be confirmed that the chats are only notifying clients which are online and listening (watching in the UI) to that chat. This system can only be tested after full integration of Chat with the user and packet system.

## 4.3. Packet blasting from UI

After the GUI system is in, there is a number of situations in which buttons will cause packets to send. It needs to be ensured that if a connection is slow and the user spams any of these buttons, that the behavior of the system is stable and correct (i.e. not resending the packet over and over, or supporting packet spamming and un-expected packet repeats).