Project Number. GFP0807

Internet Based Suicide Counseling System

A Major Qualifying Project Report:

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

Gregory Sheaffer

Date: February 1, 2010

Approved:

Professor Gary F. Pollice, Major Advisor

1. Suicide Prevention
2. Samaritans
3. Instant Messaging

Abstract

The Samaritans of Boston is an organization which provides anonymous phone counseling for depressed and suicidal individuals. The organization is currently seeking a way to expand their services using the internet to provide teenagers with a more familiar environment for interaction.

This project worked to create a tool to provide online counseling services based on Instant Messaging. The project was generally a success, implementing a solid beta version of the software which includes a server that handles IM distribution and various administrative tasks and a fully featured IM client.

Future work to complete all of the desired features and finalize the application is still required.

Acknowledgments

Our Sponsors - The Samaritans of Boston

Our Adviser - Gary Pollice

Other Team Members (Separate MQPs) – Elliot Pennington & Mathew Madden

Table of Contents

List of Illustrations

1. Introduction

The Samaritans of Boston, founded in 1974, is an organization that provides anonymous telephone counseling services. Most of their work involves handling suicidal or depressed individuals, and their services include a special hotline for teenage callers known as the 'Samariteen' help line that is staffed by teenage volunteers. The Samaritans approached us seeking a way to expand this hotline and make it more accessible to its target demographic.

A previous IQP's research concluded that using new technology would yield these results, and that an online instant messaging or texting service would be the most effective solution. Teens, the research discovered, were less likely to use the phone hotline and were more comfortable with newer text-based technology. Similar solutions, revolving around anonymised email and cell phone text messaging, had been successfully implemented by a Samaritans group in the UK in previous years.

Our team endeavored to create a set of software applications to support such an instant messaging system for the group. The product produced, while not entirely complete, is a functional proof of concept that, with some additional work, could be implemented with the group and possibly help to save the lives of a multitude of teenagers.

The remaining sections of the paper are as follows:

1. *Methodology: A discussion of our approach to the problem, our language and library decisions, requirements for the project, and our design.*
2. *Results and Analysis: A discussion of the final project iteration, its features, and the testing done.*
3. Future Work and Conclusions: A discussion of possible future improvements that could be made to the project, and conclusions based upon it.

2. Methodology

Our methodology focused on an agile development approach. That is, we worked hard to ensure that our application could easily accept new requirements or accommodate changes to the old requirements.

The steps involved in our work can be broken down into three general categories:

Requirements

We talked with the Samaritans on several occasions, both in person and over the phone, to develop requirements.

After we decided upon an Instant Messaging solution, the basic elements needed came together quickly. Multiple Samaritan counselors would need to be able to log in securely and have multiple independent conversations with outside clients from behind a central screen name. All conversations needed to be completely anonymous, with neither the outside user or the counselor having any access to any screen-names or other personal information. History logs of all conversations would be kept for review by administrative staff to monitor performance and handle complaints.

Throughout the later steps, we returned to this stage repeatedly to evaluate old requirements or add new ones. We presented our design to the Samaritans at various stages to get feedback and ensure we were working towards a product that they would want to use.

Design

Once we had a clearer idea of the requirements the Samaritans were after, the client/server distribution design we settled upon seemed like an obvious choice. Illustration 1 shows a basic model for the overall system design. A central server, running an automated Instant Messaging client under a single screen name, accepts any incoming instant messages, processes them (removes screen-names, records history, etc.), and then distributes them to client applications run by internally connected counselor clients.

The server would establish two-way conversations between teens (clients) and Samaritan counselors. It would use these conversations to ensure that all incoming and outgoing messages involved were all properly routed to the correct recipient. Our design work included detailed use cases and storyboards, which we then reviewed and used to refine requirements and create lists of features necessary for each part of the system.

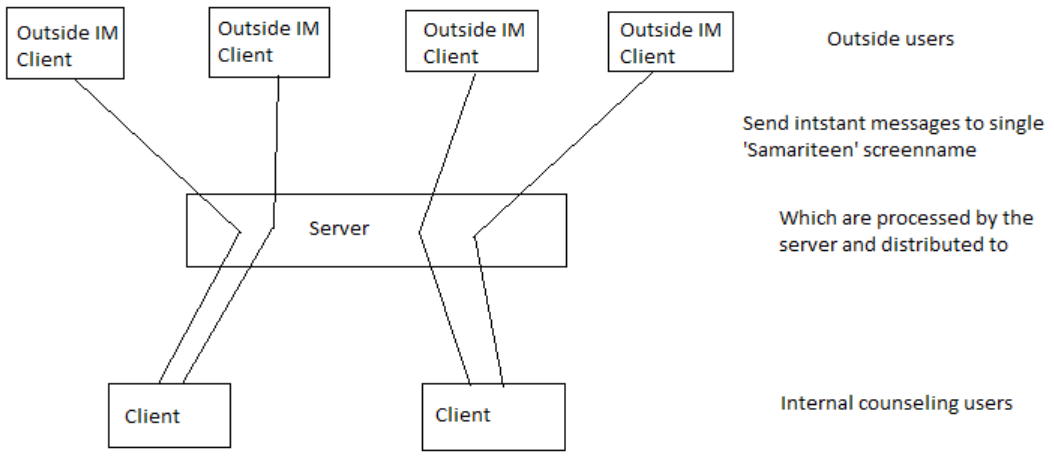

Illustration 1: Design Overview

Implementation

We selected the Python® programming language for building the application, due to the fact that we found several simple libraries for handling instant messaging functionality in an application. In addition, the Pydev plugin made working in the language easy on the Eclipse™ development platform. Although several members had limited experience with the language, we believed this was a surmountable barrier and would provide valuable experience.

For our client GUI implementation we selected the TKinter library. Our initial research turned up several other options, notably PyQT, but we wanted to move forward quickly and TK seemed to be the best choice from that perspective. The examples we found of TKinter applications showed promise, the learning curve appeared quite manageable, and the fact that it was already included in most major Python distributions made it easily accessible.

Unfortunately we ran into several problems with the library as the project progressed. The first, and most major, problem was with the way TK windows and frames are managed in the Microsoft Windows® OS... our initial architecture depended on the fact that conversation windows could be entirely separate, but in Windows a single application cannot run multiple root TK windows even if they are in separate threads. This issue was eventually remedied by changing the design to use the current tabbed frame setup, tying all of the windows to a single root, but still delayed the overall project by several weeks while we tried to track down the issue. The second issue had to do with cross-platform compatibility problems causing errors and missing features when the client, which was initially

written and tested in a Macintosh® OS, was ported to a PC. We solved this by changing the project package to include a full set of standardized Python libraries. The final issue was with the simplicity of the library itself. While simple GUI windows are easy to get working, more complex windows presented difficulty. Advanced features, such as the tabbed frame redesign, took more work than expected and some tricks that were found online. The tabs are actually simulated using standard window frames and radio buttons that change the content in the viewed frame to that of another.

In the end these were overcome, but using a more powerful and up to date tool like PyQT initially would probably have avoided the issues and saved us time in the long run, despite the longer setup time and learning curve.

The initial client interface design was based upon several independent windows for each new conversation or selection, with a main window that handled functions like logging in/out and various general options. An early example from an initial concept storyboard can be seen in Illustration 2 .

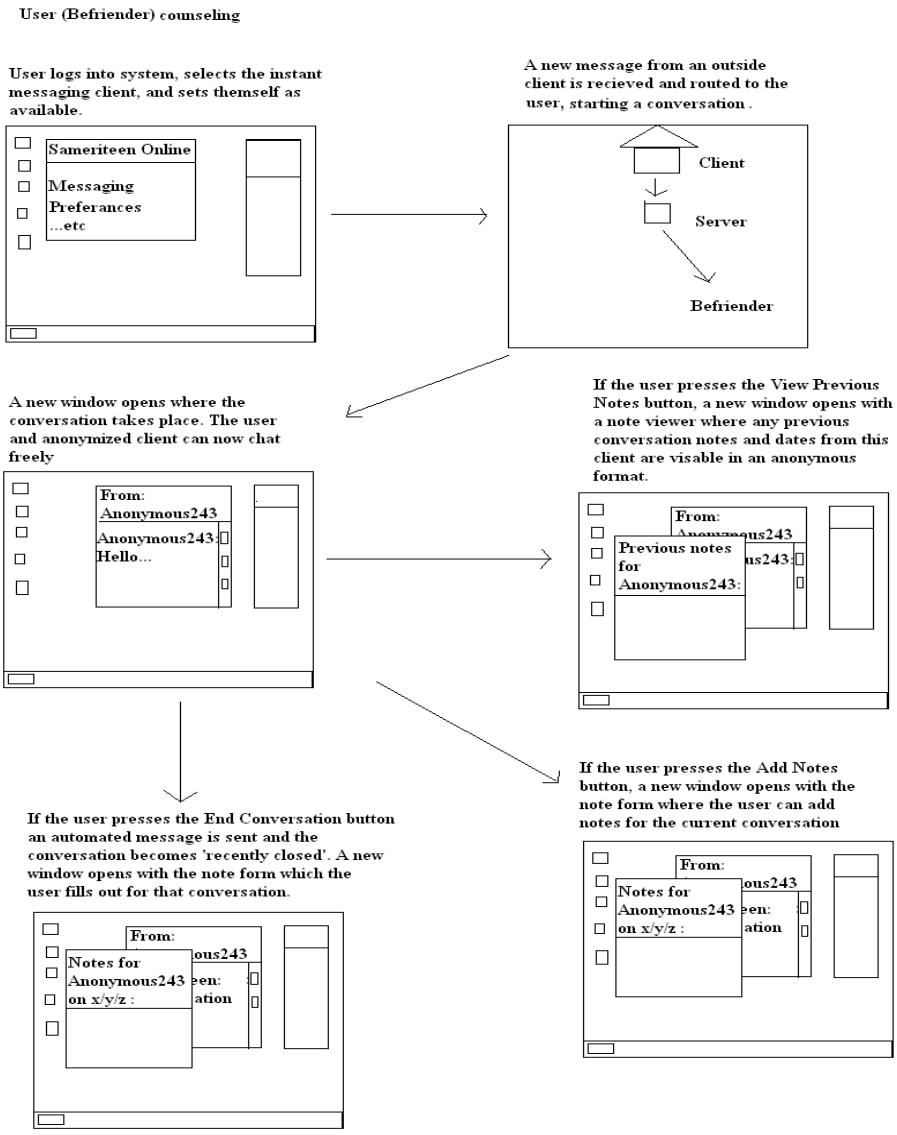

Illustration 2: Concept Storyboard

After the problems mentioned earlier with the TKinter library manifested, the design was shifted to one based upon a single window with functions differentiated using a tab system. The main tab windows are broken up into 3 types: menu (for availability, logout/login, and options), conversation (for instant message communications) and information(history or note page logs). Some

screen-shots of the final implementation (as viewed in the Windows 7 OS) can be seen in illustration 3.

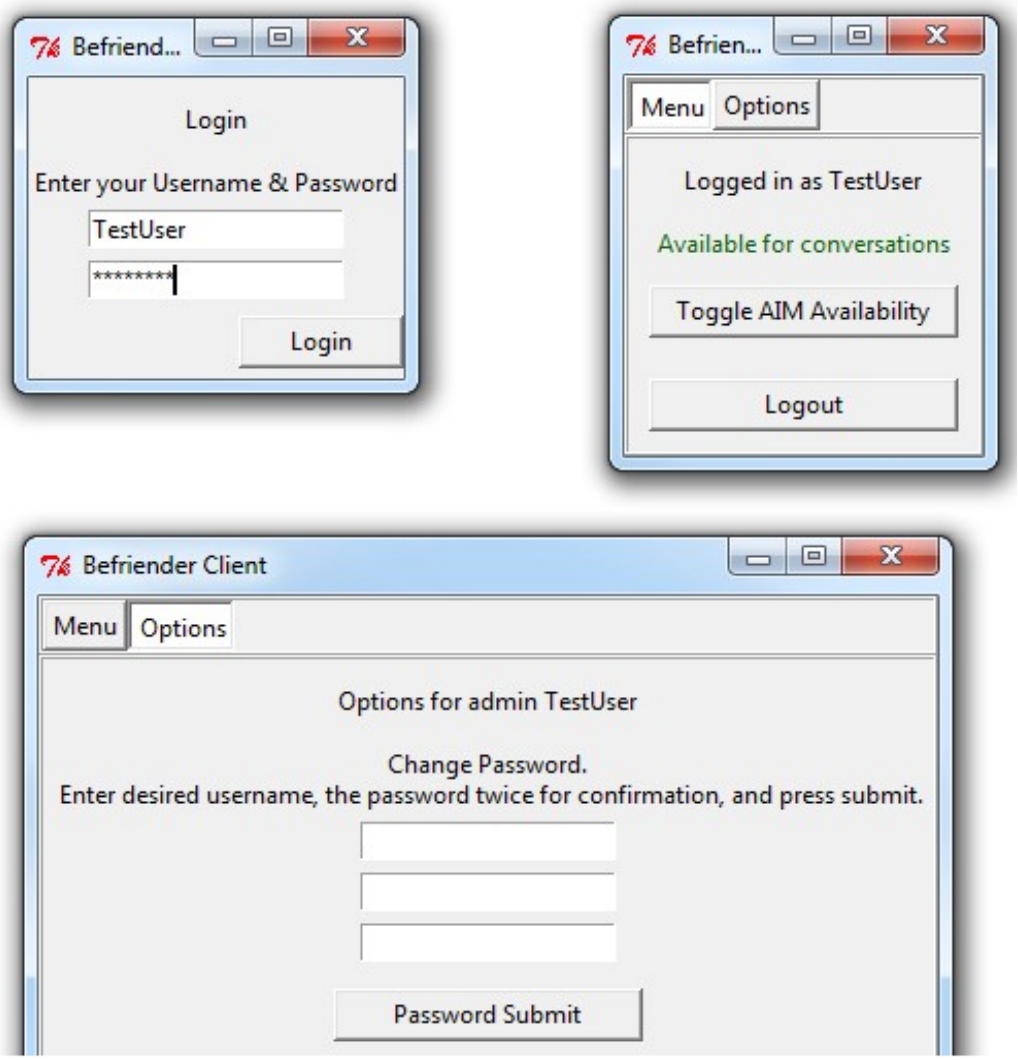

Illustration 3: UI Example

3. Results and Analysis

Setbacks

The project had several setbacks and time sinks that could have been avoided. There were repeated scheduling and time conflicts between the members, combined with a general lack of continual communication, that made our weekly team meetings one of the only times we met as a group. This in turn made testing and dividing work difficult, and progress from the various members hard to track and build on. Some project members were unavailable intermittently, sometimes for entire 7-week quarters, while another left the project before it was entirely complete. Communication with the Samaritans was also intermittent, and we learned only after several months of work that the group had decided that a different approach to the project would have been better.

Choosing the Python language when the majority of the group had little experience with it also slowed the project by the weeks it took for the members to familiarize themselves with it. Software issues also set the work back by several weeks, notably the TKinter and Python compatibility issues in the client. These issues took an exceptionally long time to discover and resolve, as they were based on mostly undocumented issues with the underlying language and libraries and not necessarily problems in our own work as we at first assumed.

Final product and testing

However, despite these obstacles, the final product is a working early beta version of the application. The client side is for all intents fully functioning. It

provides all of the necessary features, and many others: multiple simultaneous instant message conversations, various options, note entry, note and history viewing, etc. It also includes a simple graphical user interface for ease of use. The final client consisted of 12 classes, 70 functions, and approximately 2000 lines of code. As the full finalized list of requirements was completed, the client application was a decidedly successful part of the project.

Once the framework for sending/receiving messages was solid, the client testing was done incrementally through the UI, usually with the full server running and outside instant messaging test messages. Lower-level mechanical testing was done through the Eclipse debugger or with simple print statements at various steps to ensure everything was working.

The other half of the application, the server side, is stable and handles basic IM distribution. However, this half requires a significant amount of additional work and the creation of a database before it can completely meet the initial requirements (this work is detailed in the next section).

4. Future Work and Conclusions

The project as it stands is essentially two thirds complete.

The most obvious missing part is a database component. Without one set up and integrated into the server, many of the initial requirements are unattainable (most notably secure log-in and note/history recording and retrieval). A database that can handle these tasks securely needs to be implemented, and then calls for accessing or modifying it need to be added to the necessary parts of the server (As documented in the appropriate classes).

The server, while functional on a basic level, is also missing many major components. It still does not properly distribute conversations in complex situations (multiple outside clients conversing with multiple inside clients who log in and out, for example), which should be a high priority fix. It also does not properly handle the availability setting for clients. All of the client sent commands and responses are unimplemented aside from basic test values (the server allows any log-in attempt to succeed, for example), which should also be a high priority once the database is implemented. The last major missing feature is name encoding to protect the anonymity of outside users, which was an important requirement for the Samaritans final implementation.

The client is, for all intents, fully functional. Any future work should focus on re-factoring the code to clean it up and fixing some of the lingering UI issues. Most of those issues have to do with the advanced elements of the tab setup, such as removing tabs or having them flash when activity occurs within one that

is not being viewed, but there are some basic issues such as better input validation in some options (this is detailed in the comments for the appropriate classes). Rebuilding the UI on a better base, such as PyQT, is also an option worth considering if time permits.

The project as a whole also needs work to package the software for distribution and use, along with more extensive testing in larger-scale situations closer to those that it would be expected to work under. The Samaritans expressed a distinct interest in a similar system for use on their website, perhaps based off of other embedded instant messaging systems. Such a system would be an interesting followup project.

Overall, as previously stated, the project is about two thirds complete. Despite missing features (almost entirely due to the missing database), it is a working prototype for the intended system that is ready to be expanded upon and tested more thoroughly. Most of the setbacks and delays suffered were issues with the underlying tools themselves rather than our design, and our robust framework allowed us to quickly implement solutions once the problems were located.

Glossary

Instant Message: Instant messaging (IM) is a form of real-time direct text-based communication between two or more people using shared clients. The text is conveyed via devices connected over a network such as the internet. *IM* falls under the umbrella of 'chat', as it is a real-time text-based networked communication system, but is distinct in that it is based on clients that facilitate connections between specified known users (often using "Buddy List", "Friend List" or "Contact List"), whereas Chat includes web-based applications that allow communication between (often anonymous) users in a multi-user environment. An analogy would be comparing a telephone and a bar-room. With the telephone, you have to know the contact information to reach the other person, whereas you just show up at the bar and see who is there to chat with. (Taken from http://en.wikipedia.org/wiki/Instant_Message)

References, Other Projects

*Nguyen, Tuong-Vi (2007). Samaritans' Teen Line. Unpublished Interactive Qualify*ing Project. Worcester Polytechnic Institute.

*Pennington, Elliot (2009). Samariteen AIM Hotline. Unpublished Interactive Qualify*ing Project. Worcester Polytechnic Institute.

The Samaritans of Boston website (http://www.samaritansofboston.org/)