CS673S16 Software Engineering
Team 2 - Chat Enhancements
Project Proposal and Planning

Team Member	Role(s)	Signature	Date
Niall Kavanagh	Project Lead	*NK*	2/16/2017
Betsy Berkey	Requirements & Implementation Leader	*BB*	2/16/2017
Raziel Melchor	QA & Design Leader	*RM*	2/16/2017
Albert Lewis	Security & Configuration	*AL*	2/16/2017

Revision history

Version	**Author**	**Date**	**Change**
1.0	**Albert Lewis, Betsy Berkey, Niall Kavanagh, Raziel Melchor**	**2/16/2017**	**Initial draft**

1. Overview

As work becomes less centralized and more distributed, open conversation becomes simultaneously easier, but also more problematic. It's easy to send team members an email or a text message, but difficult to organize these many streams of conversation into a single, shared narrative across a team. Without open access to what is being said, not everyone on a team will have the same understanding of the team's objectives and goals. Tools like email, IRC, and instant messenger can alleviate these problems, but have shortcomings that require a significant effort to mitigate. Email over-communication can result in inboxes filled with messages that cannot possibly all be read, with team members unable to find the messages that are relevant to them. IRC is excellent for ad-hoc conversations, but lacks any historical record of what was said. Instant messaging platforms are ideal for topical conversations with specific people, but make it difficult to have open conversations that everyone on a team has access to.

A solution to this problem is a group messaging platform, a networked, browser-based application that lets people communicate openly with their team members in an open manner. Such tools facilitate open conversation, just like in a shared office space, and are recorded and accessible for reference later on. Conversations may also be private, shared only amongst a specific group of individuals. Many tools offer additional features such as file storage and sharing, and integration with third party services such as Trello, Github, Pivotal Tracker, etc.

Many of these tools are available commercially, with limited feature sets available at no cost. For example, Slack is available to anyone to use for free, but limits the usefulness of some of its features unless a subscription is paid for. An installable, self-hosted platform that provides these features would be most attractive to teams that can't afford the cost or drawbacks of commercial offerings.

During the Spring semester of 2015, a team of students began work on an installable, self-hosted chat application that is part of a project management platform, delivering an application that enables its users to chat in open channels and share files—the beginning of a conversation platform that could someday offer feature parity with commercial offerings. It is the intent of team 2 to build upon this application to make it easier to install, work on, and use by improving its documentation, configuration handling, and adding to its feature set.

2. Related Work

As identified by the team that originally built the chat application, several commercial messaging platforms are available, with the most popular being Skype, Google Hangouts, and Slack[1]. Of the three, the communications tool platform is most like Slack. Unlike Skype and Google Hangouts, Slack offers numerous third-party integrations permitting users to leverage other services to handle things like issue tracking, source control, and more.

The communications tool endeavours to also be a smaller part of a greater whole, tying together a group of tools in the spirit of the Unix philosophy[2].

The major differentiating factor between the communications tool and Slack is that Slack is a commercial, cloud-based service. As of May 2016, Slack had more than three millions daily active users, though only 930,000 of them, less than ⅓, were paying customers[3]. Clearly there is a need for similar software that is not feature-limited merely because its users do not wish to pay the per-user cost of Slack.

3. Proposed High level Requirements

a. Functional Requirements

[1]http://www.geekwire.com/2017/study-microsoft-teams-pass-slack-google-hangouts-second-used-business-chat-app-2018/
[2] https://en.wikipedia.org/wiki/Unix_philosophy
[3] http://fortune.com/2016/05/25/slack-3-million-daily-active-users/

i. Essential Features
   1. The communication tool will allow the user to see other recent or online (currently active) users.
   2. The communication tool will allow the user to participate in private and public chats.

ii. Desirable Features
   1. The communication tool will allow the user to switch channels by typing a command.
   2. The communication tool will allow users to delete/edit messages after sending them.

iii. Optional Features
   1. The communication tool will allow users to set reminders by typing a command in text box.
   2. The communication tool will allow users to receive notifications in chat whenever specific actions on github occur.
   3. The communication tool will push desktop notifications when a user's name or keywords are used.
   4. The communication tool will display a gif or image similar to Slack's giphy feature.
   5. The communication tool will allow users to setup an avatar in their account profile.

b. Nonfunctional Requirements

i. Error-handling **-** handle the following errors from user interactions:
   1. Error: User enters a wrong password multiple times.
   2. Error: User forgot password.

ii. Constraints **-** the limitations that may be encountered are:
   1. Documentation
   2. Logging
   3. Configuration

iii. User Interface - the communication tool is being developed for Boston University students with a high level of computer literacy, a high level of experience with similar applications, and will be expected to use the application frequently.

c. Implemented Features (to be completed at the end of project)

4. Management Plan

a. Process Model

- Iterative Agile development : Due to the nature of our project we'll work on adding features to an existing application. Our project will follow the feature-driven development methodology. By gaining continuous feedback from stakeholders and users this will assist us in improving our deliverable. Some of the benefits we gain from this methodology are, frequent team meetings, tight and frequent deadline with clear and manageable milestones.

In this first phase, each team member has been assigned a leading role in various aspect of the development. The features that are being considered have been weight by the team as to what we believe should be built and implemented.

Work Activities

- Develop the selected features and implement
- Integration, testing and QA will be performed
- Documentation

b. Objectives and Priorities

Objective will focus on meeting critical requirements and delivering the product on time. Due to our time constraint late delivery is not an option therefore our focus is less on quality and more on a functioning software. If time should permit after delivering a working software with features implemented, we will focus on quality and reusability.

c. Risk Management (need update constantly)

To ensure that risks are identified, discussed, and mitigated as soon as possible, the team will formally set aside time during each twice-weekly meeting to discuss new risks and blocking items.

Specific risks that have already been identified for monitoring and discussion:

- Overcommitment: taking on too much work. The team will attempt to mitigate this by ensuring work is prioritized in the backlog such that essential features and deliverables are completed first.
- Communication: The team is using technology to support most interactive meetings. Team will constantly evaluate if this needs to change, i.e. if in-person meetings are required.
- Inter-team integration: Two teams will be working on the same application. It will be critical to communicate with the other team and coordinate work and changes so that integration of our work with their work is possible.

d. Monitoring and Controlling Mechanism

Project monitoring will be discussed weekly, with special attention given to it at the beginning of each phase. Additional meetings will be scheduled whenever the team believes it is necessary.

Meetings will occur every Saturday at 11:00 a.m., and every Tuesday at 7:00 p.m. On Sunday night, it is encouraged that team members give an update of their progress.

e. Schedule and deadlines (need update constantly)

Milestone	Deadline
Project Proposal & Planning Document	2/16/2017
Iteration 0 Presentation	2/16/2017
Iteration 1 Presentation	3/16/2017
Iteration 2 Presentation	4/6/2017
Final Presentation	4/27/2017

5. Quality Assurance Plan

Product Metrics

We will use metrics for the analysis, design, implementation stages. Additionally, we will use metrics for testing and maintenance [4].

Size is an indicator for the amount of effort involved with coding, testing, and integration. Metrics that depend on LOC will be used, such as errors/KLOC , defects/KLOC, cost/KLOC.

Additionally, we want metrics that measure mean time to failure, user problems, and user satisfaction.

Regarding metrics that evaluate the design of our project, will mainly be concerned with measuring complexity [5]. The metrics we might use are (a) number of entries and exits per module (package)

[4] http://ecomputernotes.com/software-engineering/classification-of-software-metrics
[5] *Software Engineering: Modern Approaches pg. 526*

(b) Graph-theoretic complexity for architecture : (# of modules in the architecture) - (# of modules having at least one connection between them) + 1

We will use radon in order to measure Halstead complexity metrics of the source code [6]. For testing metrics, the Software Test Documents and The Software Verification and Validation Plan will specify the tests to be executed. For maintenance metrics, we will use the Software Maturity Index to evaluate the stability of the program over the course of the semester.

Process Metrics

The metrics we will use to evaluate our processes will be (a) number of errors found before software release (b) defects detected and defects reported by user after release (c) Time spent fixing errors (d) conformity to schedule (e) estimated cost vs actual cost

Coding Standard

All coding standards will be validated using automated tools. For any python code we will follow the PEP 8 style guidelines, and we can use the python pep8 package to validate guidelines automatically [7]. For javascript, we can use JSLint in order to validate style guidelines [8].

Documentation Standard

Project Artifacts

The following documents will be produced: (a) SPPP (b) SDD (c) STD (d) SQAP

Code Documentation

We will use an automated tool like Doxygen to generate documentation [9]. Additionally, we will use it to diagram code structure.

Tools

We are using Pycharm with Django support as our IDE. We are using standard requirements for Django v1.7.4 [10] . For project management, we are using Pivotal Tracker. The entire team is using macOS for development.

Inspection/Review Process

[6] https://pypi.python.org/pypi/radon
[7] https://pypi.python.org/pypi/pep8
[8] http://www.jslint.com/
[9] http://www.stack.nl/~dimitri/doxygen/index.html
[10] https://github.com/CS673S17-Team-2/Final_Project/blob/develop/requirements.txt

Code Reviews

We will have a different branch for each feature. We will then issue pull requests to integrate the feature branches into the develop branch. We will use github to do code reviews. Any commits will be reviewed by at least another team member.

Inspection Process

The goal is to identify issues as early as possible. The author of the code will be responsible for fixing the issue.

Well will continuously review requirement specifications, project management, and configuration plans. While all the group members will be responsible for generating feedback on all of parts of the project, the leaders for the different roles will be responsible for requesting feedback and making sure that the task is given enough attention.

Testing

We will follow the guidelines in the Testing document for unit, system, and acceptance tests. Testing will occur at the end of each development cycle to verify that the software meets the requirements.

Defect Management

We will use Github Issues or Bugzilla in order to track defects. Each defect should be classified by (a) severity (b) priority (c) type and (d) source . In this stage we will discuss repairing defects and possible enhancements.

6. Configuration Management Plan

a. Configuration items and tools

The two areas of this project that requires configuration management are the documentation stored on team 2 Google drive and our Github repositories. Github is used for our version control tool. We have one repository and ten branches. We use the developer branch for development purposes.

b. Change management and branch management

Team documents will be hosted and edited in Google drive via Google doc. The configuration lead will track these documents and will be notified of newer versions of document. The configuration lead will save and push all documents to the official documentation repository. Our Git repository will be managed by our (configuration

or environment and implementation lead), he/she will be responsible for ensuring code was tested and reviewed before commit.
Our team and team 3 are currently working on the same project and off the same development branch. We will collaborate with team 3 to discuss a proper approach for merging as we approach time for a build.

c. Code commit guidelines

We encourage small commits checker code by utilizing PEP 8, online or install python module.
Code commit will be done using the below workflow.

1. After code modification
2. get status of file with "git status" command to show difference in file
3. add a file "git add -A or git add fileName" to add new file
4. commit the file with comment "git commit –m 'comment' " .
5. Push to github.com with "git push

7. References

(For more detail, please refer to encounter example in the book or the software version of the documents posted on blackboard.)

Braude, Eric J.; Bernstein, Michael E. (2016-02-15). Software Engineering: Modern Approaches (Page 233). Waveland Press, Inc.. Kindle Edition.

8. Glossary