Communications Tool Enhancements

CS673 Spring 2017 — Team 2

Hi, we're team 2, and–alongside team 3–we're working on enhancements to the communications tool that was developed as part of the project management platform built by previous CS673 classes.

Introduction — The Problem Being Solved

- Communication becomes more difficult as teams scale
- Channels = N(N-1)/2; 2 people: 1 channel, 4 people: 6 channels, **10 people: 45 channels**
- Managing communication is critical to any social endeavor's success
- **Transparency** means everyone has access to the information they need
- **Organization** means the information is categorized and archived
- **Consistency** means information is not replicated

The project management platform is comprised of an issue tracker, a requirements tracker, and the communication tool, which is the module we've chosen to work on.

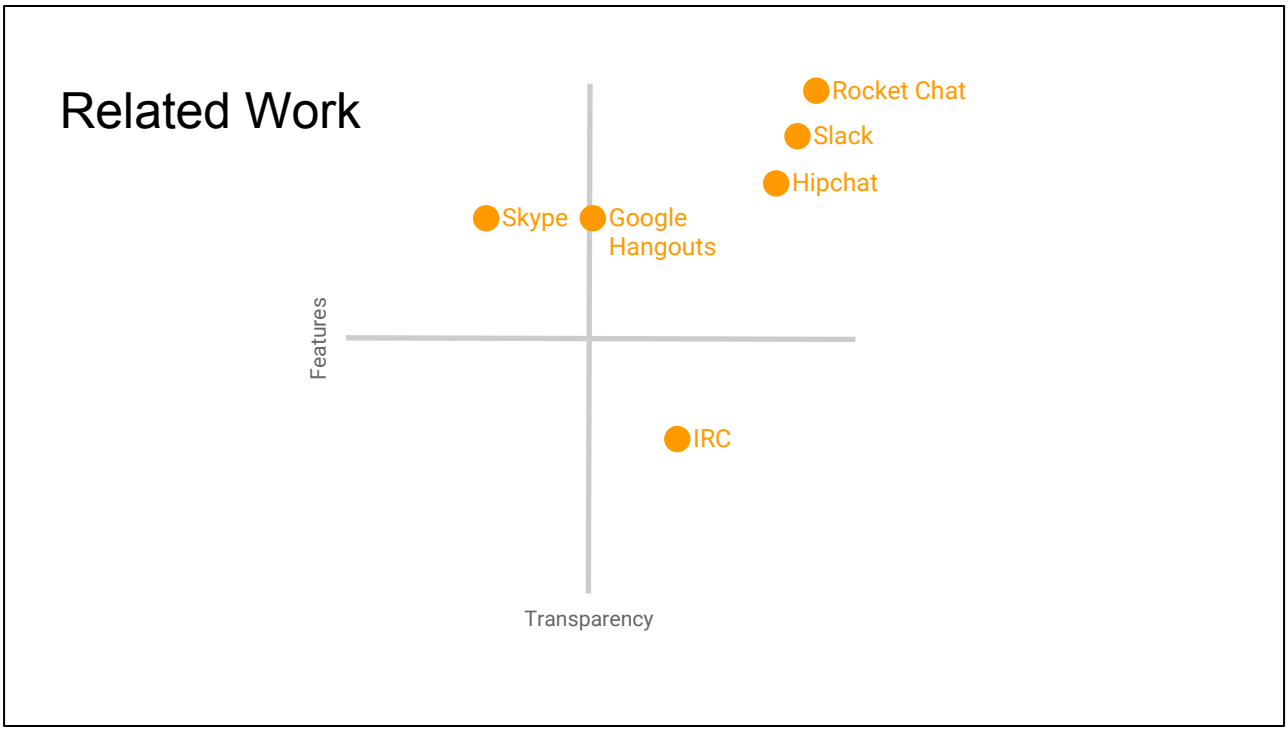
Communication is challenging problem, because it becomes more difficult as the number of people involved increases. Two people have only a single channel of communication—between each other. Four people, which is the size of our team, have six possible channels of one-to-one communication. Ten people, which is the size of teams two and three, have 45 possible channels of one-to-one communication. If you consider that teams don't just have one-to-one conversations, but could have groups of two, three, four, etc. people in a conversation…. well, you get the picture.

Managing this communication is key to the success of any team effort, and managing communication means optimizing it for specific things.

If communication is transparent and open to everyone, then everyone on the team can be assured that they will have access to the information they need to do their work.

How the information is organized gains importance as the amount of information increases. We only have a dozen or so slides in this deck, but if we shuffled them around and moved bullet points from one slide to another, this presentation wouldn't make very much sense. Of course, we hope that it makes sense the way we have it organized now.

Consistency means the information is not replicated. If you have the same information in two places, then you must maintain it in two places. Chat rooms are great, but they're not the best place to store very structured information, which is why we use separate tools for things like requirement tracking, issue tracking, and source control. If your communication tool can encourage you to move structured information to structured content management systems, you decrease redundancy and decrease the amount of time you have to spend keeping things up to date.



There are quite a few tools available to facilitate group communication. Let's take a look at a few of them across two dimensions: features and transparency.

By features, I mean things like group chat, direct messaging, voice, video, third-party integrations.

By transparency, I mean how easy is it for people in the group to get to the information they need.

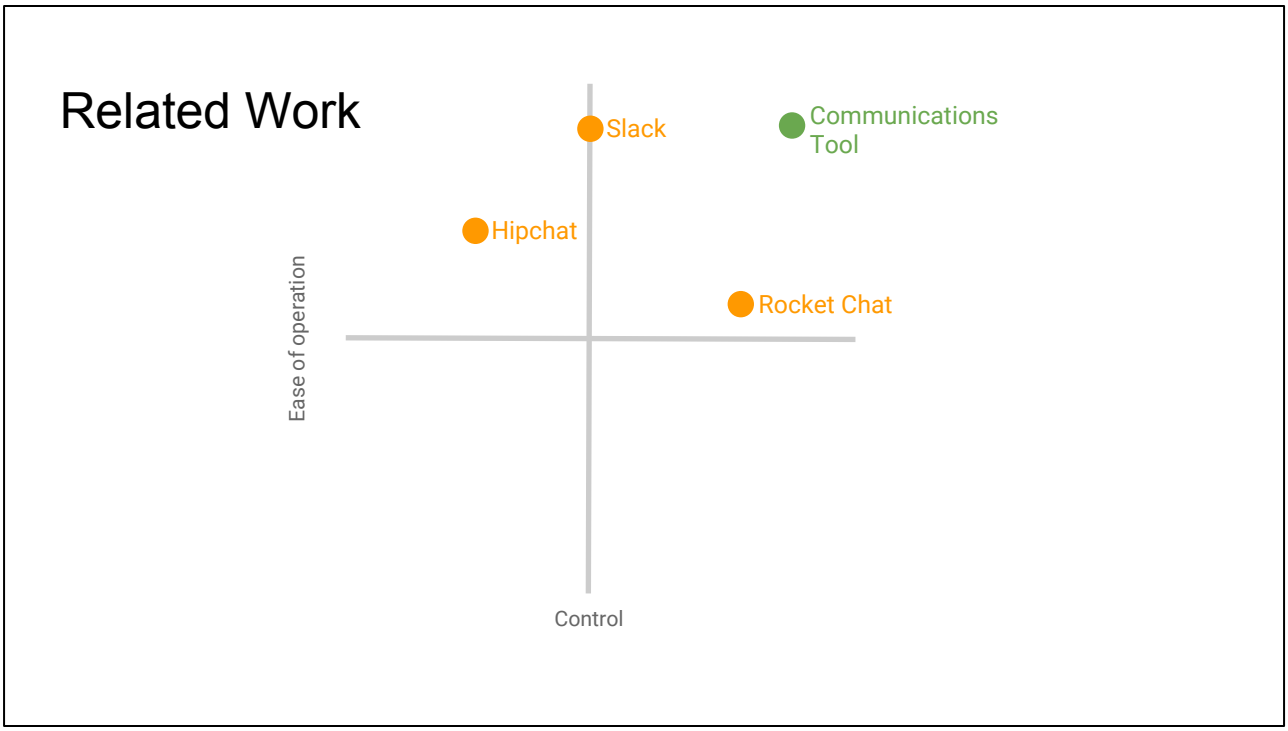
Skype and Google Hangouts have impressive feature sets, letting you chat directly with individuals or groups and offering features like voice and video calling.

IRC has fewer features–no video or voice–but is more open; chat rooms persist and, if they are public, open to anyone who wants to join.

Hipchat and Slack are commercial applications with similar feature sets and the ability to integrate with a variety of third-party services. They offer persistent chat rooms with archived content—for a fee.

Rocket Chat is an open-source alternative to Slack; it even has a few features Slack doesn't. You can host it yourself.

So with so many options, why bother developing our own?



Well, we need the grade.

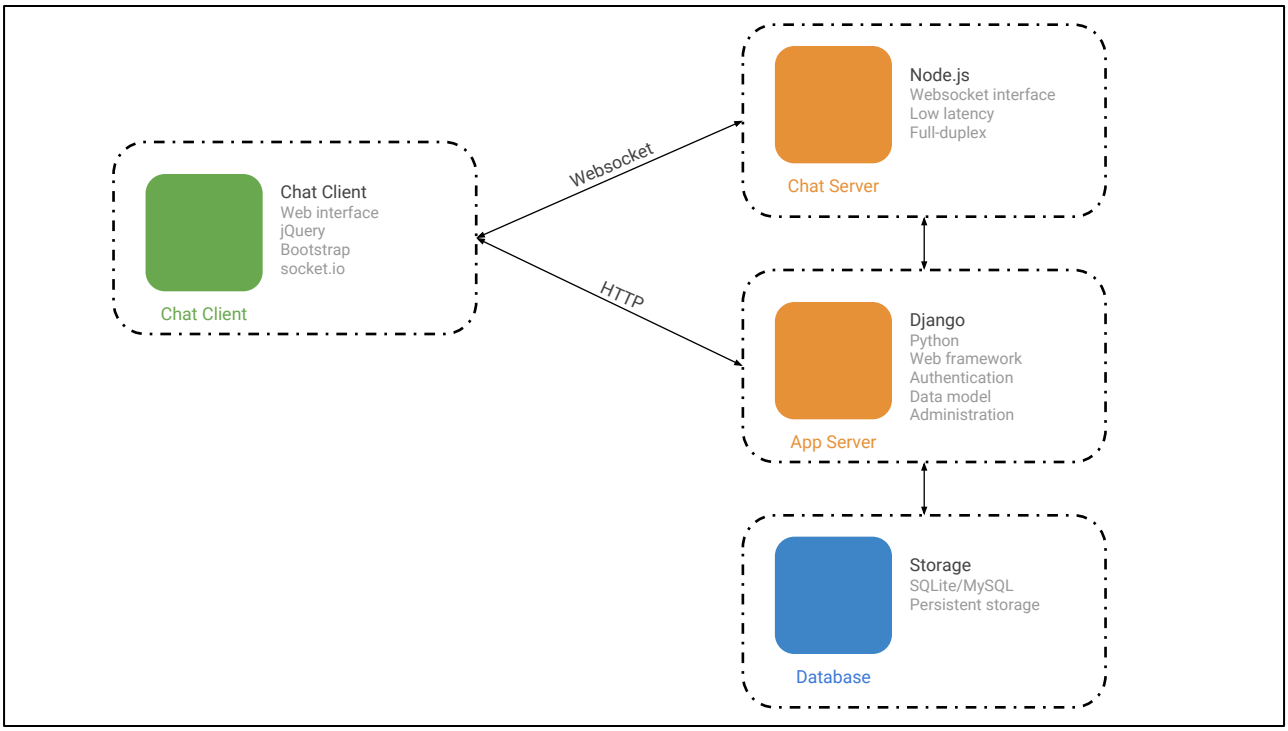
Really, only three of these applications meet the criteria for transparency, organization, and consistency that we set on the introduction slide. So let's examine them again on two more dimensions: ease of operation, or how difficult it will be to get them running and use them, and control, how free we are to configure how they work.

Rocket Chat, which as we said has more features than any of the others, comes at a bit of a cost when we consider ease of use. You need to install and configure it, which can be onerous given it's impressive feature set.

Hipchat lags on both ease of use and control, relative to the other offerings.

Slack is really easy to use, but doesn't offer that much control, and even then only when you pay for it. 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.

What we're looking for is something that is as easy to run and use as Slack, but offers a degree of freedom and control that Slack does not.



Enter the chat client. It's a browser based interface written using Bootstrap and jQuery. It talks to a Django backend over HTTP, but also a Node.js backend using a websocket for passing messages over a low-latency, persistent connection. The Node.js service also talks the the Django backend using a RESTful interface.

The basics of this application are already here; you can log in, chat in public rooms, and even upload files. Our plan is to make this even easier to work on by improving the documentation and configuration, but also to make it more useful by fixing bugs and adding features, which is what Betsy will tell you about.

User - BU student

- Characteristics of the user's tasks and jobs
  - Type of use of this application: ***discretionary***
  - Frequency of use: ***frequent***
  - Turnover rate for students: ***high***
  - Importance of task: ***low***
  - Repetitiveness of task: ***low***
  - Training anticipated: ***none***
- Physical characteristics of the user
  - Age: ***young***
  - Gender: ***male***, ***female***
- Level of knowledge and experience
  - Computer literacy: ***high***
  - System experience: ***high***
  - Experience with similar applications: ***high***
  - Education: ***college***
  - Reading level: ***< 5 years schooling***
  - Typing skill: ***135 wpm***
- Psychological characteristics of the user
  - Probable attitude toward job: ***neutral***
  - Probable motivation: ***moderate***

So to identify the requirements of our chat feature we started by identifying our user - a BU student. This will help when we are developing our graphical user interface and also when assessing the value of each feature to be added.

After going through this checklist, we can describe our user or BU student as having a high level of computer literacy, plenty of experience with similar chat applications and will be expected to use the application frequently.

You can see here all the characteristics of the student. These may not be as important as computer literacy but we still identified them. They are relatively young, using this chat application their own discretion, both male and female and finally have a high level of education.

Requirements

Functional

Essential Features
- See other recent or online (currently active) users.
- Participate in private and public chats.

Desirable Features
- Switch channels by typing a command.
- Delete/edit messages after sending them.

Optional Features - backlog
- Set reminders by typing a command in text box.
- Receive notifications in chat when specific actions on github occur.

Nonfunctional

Error handling
- User enters a wrong password multiple times.
- User forgot password.

Constraints

User interface

First we came up with a list of possible features and assessed their value. We presented these to the client and asked for their perceived value. After getting the client's feedback, we then weighed the perceived value with the risk of each feature and its effort level. From there we determined the order of our backlog and which features we thought were essential, desirable and optional.

Our top, or essential, features that we will be implementing first are the ability for the user to see other users currently online or who were recently online and the ability for the user to participate in both public and private chats.

Other features that we'd like to implement and see just as important are the error handling nonfunctional requirements. When a user enters a wrong password multiple times, we will need a response or warning for the user to understand they have tried too many times. If a user goes to sign into the app and has forgotten their password, we would like to implement the ability to find their password in some way.

The full list of our backlog is here. You can see the other backlog features we have listed under both functional and nonfunctional requirements.

Management Plan

Process Model

- Agile Process
  - Due to time constraint, 3 one-month iterations
  - Feature driven development
  - Components
    - Backend
    - Web
  - Track requirements in Pivotaltracker

Objective and Priorities

- Features
  - High priority features first.
  - Deliver on schedule
  - Consider lower priority features
  - Focus on quality

-For this project our team have decided to use an agile process methodology.
Few Benefits are:

- frequent team meetings
- Tight and frequent deadlines
- Clear and manageable objectives

Risk Management

- Risk management is a standing topic for every status meeting
- **Overcommitment** — Are we taking on work we can realistically complete?
- **Communication** — We meet online. Will this be enough?
- **Team integration** — Coordinating with team three.

Risk management is a standing topic for every status meeting; we've set aside a block of time to review risks we've already identified and talk about things that are worrying us in case we need to start tracking new risks.

We've already identified three things we need to monitor and control:

We're worried about overcommitment because we don't want to commit to completing more work than we can do. We can mitigate this by ensuring we groom our backlog of work such that the most important work, the "essential features", is completed first.

Like everyone else here, we're pretty busy. As a team we've decided that we're all comfortable enough using tools like Google Hangouts to facilitate our meetings, giving us greater flexibility in terms of coordinating our individual schedules. This has worked out well when we need a quick ad-hoc meeting; for example, we meet a couple of extra times this week to review and practice our presentation. That being said, we're cognizant of the risks inherent in not having regular, in-person meetings outside of our normal class hours.

We're also keeping an eye on how we integrate our work with team 3. Oliver's team has already opened up their Pivotal Tracker backlog to us, and we've already accepted a pull request from them, so things are working really well from our perspective so far. We'll need to keep an eye on this going forward to make sure we're aligned on things like coding standards and branching strategies.

Configuration Management

- Configuration
  - Utilizing google drive and Github for our repositories.
    - Github for our code base
    - One Repository various branches. (will work with team 3 on merge decisions, etc)
    - Google drive for software development documentation (consider migrating to Github)
- Code commit guidelines
  - Utilize Pep 8 https://www.python.org/dev/peps/pep-0008/
  - Small logical commit
  - Test before commit
  - get status of file with “git status”
  - add a file “git add -A or git add fileName”
  - commit the file with comment “git commit –m ‘comment’ ” .
  - Push to github.com with “git push

Quality Assurance

Metrics

- Product Quality
  - Intrinsic Quality
    - Defect Density Rate
    - Mean Time To Failure
  - Customer Satisfaction
- Process Quality

Standards

- Coding Standard
  - PEP8
  - JSLint
- Documentation Standard
  - (a) SPPP (b) SDD (c) STD (d) SQAP
  - Sphinx

http://www.informit.com/articles/article.aspx?p=30306

1. Product metrics describe the characteristics of the product such as size, complexity, design features, performance, and quality level.
2. Process metrics are used to improve software development and maintenance in order to evaluate process improvements
3. For product quality, size is an important indicator for the amount of effort involved with coding, testing, and integration.
4. Metrics that depend on LOC will be used, such as errors/KLOC , defects/KLOC, cost/KLOC.
5. LOC can be measure in different ways
   a. Count only executable lines.
   b. Count executable lines plus data definitions.
   c. Count executable lines, data definitions, and comments.

5. Since we are working mostly with python and the team members will be the same throughout, the exact definition of LOC may not be as important as keeping a constant definition.
6. The MTTF is often used for safety-critical systems. It measures the time between failures. We probably will not use it.

7. We also have to consider how satisfied the customer is with the product. We are

concerned with knowing how satisfied they are with capability, functionality, usability, performance, reliability, installability, maintainability, documentation/information.

8. 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

Standards

Code

Standard not just for the sake of having standards, but for maintaining quality.

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 . For javascript, we can use JSLint in order to validate style guidelines .

We will use an automated tool like Shipx to generate documentation . Additionally, we will use it to diagram code structure.

Quality Assurance

- Inspection Process
  - Identifying issues early on
- Testing
  - Testing “early” and “often”
  - Types of tests
    - Regression
    - Black Box and White Box
- Defect Management
  - Tracking
  - Classification
  - Solutions and enhancements

The other aspects in Quality Assurance

Inspection Process

The goal is to identify issues as early as possible. The sooner we find out the issues in our code, the easier it will be to detect and fix them.

Here we will inspect

1. For Variables and Functions
   a. Are meaningful names used?
2. Initialization
   a. Are variables initialized before first use?
3. Loops
   a. Do they always terminate sucessfully ?
4. Comments
   a. Is the code properly commented ?
   b. Do the comments describe the corresponding code ?
5. Defensive Programming

Testing
The principles of testing early and often are a major feature agile methodologies

Testing early refers to testing each method before adding additional methods to a class

Testing often refers to testing whenever small modifications are made. Our goal will be to making sure previous tests pass before testing new fnctionality.

The types of tests will include:

a. Regression testing : referes to making sure previous functinoality was not broken when adding new functionality.
b. Black box texting refers to testing functionalitty without having to know how it was implemented.
    i. Using selenium in order to test
    ii. Ad hoc user testing
c. White box testing
    1. Unit tests
    2. Interface testing validates functions exposed by modules
    3. Integration testing validates combinations of modules
    4.

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.

Questions