Q Review

By Annabelle Zha, Ethan Cui, Keegan Jordan, Tim Chirananthavat

Our Goal

Code review (also known as peer review) is an important process used to ensure code quality meets a particular standard in the software development industry. It refers to the activity of having one or more people assess changes in the source code of a program and identify areas of improvement. When a code review involves only one reviewer, the task of reading the code and giving feedback is quite simple. When there is more than one reviewer involved, the process becomes more complicated and often results in one reviewer finishing the entirety of the review, multiple reviewers overlapping their efforts, or no reviewers stepping up to complete the review. In both cases, collaboration is negatively affected by the lack of instant communication and causes the efficiency of the review to fall and ultimately slows down the process of developing software.

Our goal is to bring collaboration to the code review process. Specifically, we want to create a plug-in for Slack, the most popular communication platform in workspace, to enable instant multi-direction discussion among many reviewers and authors in the code review process.

Current Practices

In one traditional approach, when new code is requesting review, the project maintainers will be notified through email with a link to the change. When a reviewer leaves comments on the source code or requests additional changes in the code, the involved parties are notified through email. The conversation is driven by email alerts that are much slower and get lost in inboxes.

For example, a contributor submits a pull request that will change the behavior of a module on which multiple teams depend. Each team will need to review this code change to make sure it will not break any existing code. Team 1 and team 2 think the change is acceptable whereas team 3 thinks this change will break their pipeline. Team 3 leaves some comments and proposed alternatives. Team 1 and team 2 will now need to look at the proposed alternatives again to make sure this suggested changes fit into their code. This back and forth process can take multiple days to complete and is

primarily affected by the lack of real-time communication in many code review platforms.

Our Approach

Our approach to the code review process looks like this: when a pull request is submitted, our Slack plug-in will automatically create a new slack channel involving the contributor who submitted the request and the reviewers. They can discuss the code change in question and use Slack commands to comment or request changes in real-time. Once the reviewers give feedback, the author of the code can change it accordingly and discuss the changes until they reach an agreement on a final version. Finally, the involved parties can ship the code right from the conversation without missing a beat. This approach is expected to increase the efficiency of the review process keeping the development cycle short and sweet.

Stakeholders

As our approach aims at improving the productivity of collaborative code review process, projects involving more than one reviewers will most likely to benefit from our approach. Main stakeholders for our project would be programmers who work in a collaborative environment. Our project will directly change the way they perform collaborative code review so they would care.

Besides direct stakeholder, other indirect stakeholders who will be affected by our tool include project managers who care about improving productivity of their team, managers and company liaisons who make final decisions on the timeline, budget, and scope of the project, partner companies who supports a project with their third-party tools and therefore wants to ensure the compatibility, and the Slack platform whose functionality in collaborative code review will be expanded.

If our project is successful, it can reduce the complexity in collaborative code review. This way the feedback loop will be significantly shortened. Frequent and in-time feedbacks will likely to avoid the accumulation of mistakes in code. Ultimately, if being used wisely, our project can probably accelerate the software development process.

Risks and Payoffs

One of the largest risks we are taking in this project is the completion of this project in the given time frame. Because of the lack of information we have now about the specifics of the implementation of this tool, it is very difficult to predict which areas will take the most time. Additionally, there may exist roadblocks along the way that slow down the development process or halt it completely. Our plan to mitigate these risks is

to first complete a minimum viable product that succeeds in implementing a code review interface in Slack. The payoff of this approach will be us knowing early on whether a crucial aspect of our implementation plan requires a redesign or if we are able to achieve the core functionality of our program. We then have the flexibility to add features with excess time and improve on the first version. Another risk we face when using third party applications such as Slack, GitHub, Phabricator, or other tools, is the uncertainty in interacting with these applications. For instance, we may find that the functionality we need is unavailable for public use. The payoff for succeeding in these integrations is having completed an application that will fit in to many developer workflows across a wide variety of teams and organizations.

Cost and Development Time
We do not anticipate having any costs associated with this project. We plan to use free public APIs and frameworks to develop the tool with. We also do not expect to use cloud services to a degree where a free tier is not sufficient. In order to finish this project in the time frame allotted to us, we will follow a strict schedule and parallelize the development of multiple components in order to progressively test the flow of the tool. A rough schedule is listed below and includes the currently known elements required for the completion of this project. Completion time refers to the time a single developer will take to complete the component. Components of the same priority will be completed in parallel and before the items of the following priority.

Component	Time (hrs.)	Priority
Events API (Slack Bot)	10 hrs	1
Text Processing & Response (UI)	10 hrs	1
GitHub/Phabricator Code Review Integration	30 hrs	1
Experimentation	5-20 hrs	2
Slack Channel Creation	10 hrs	2
Slack Text Help and Error Response	10 hrs	2
Slack Directory Registration	2 hrs	3
Art, Design, Documentation, etc.	3 hrs	3
Unknown and Unpredictable	10 hrs	N/A
Total Time	90-105 hrs	

Schedule *

Week 1

- A Slack application is registered and ready for development
- A Text Interface for the users is defined and implemented
- Data is sent manually to GitHub / Phabricator for testing

Week 2

- Text interface triggers events in Slack app
- Events trigger interaction with GitHub / Phabricator API

Week 3

- Version 1 complete: User is able to complete a code review within Slack
- Experiments run on code review productivity difference

Week 4

- Slack user experience is expanded: New reviews create new channels
- Completed reviews archive old channels
- Application assists users in usage and configuration for application

Week 5

- Application is deemed 'complete'
- Application is registered to Slack App Directory
- Custom art is created and emojis are integrated into Text Interface

* assuming team member productive hours is averaged at 5 per week

Midterm and Final Checkpoints
Our midterm checkpoint will be to see if we can successfully create a flow of commands from Slack that result in changes to GitHub. A baseline for this would be the implementation of a Slack events processor and a module that sends data to the GitHub API.

The final checkpoint will be an end to end test of the user experience in completing the code review once as an author and once as a reviewer. Additionally, it will be an evaluation of whether or not we are able to reduce friction in multi-directional communication in the code review process.

Feedback Improvement
The feedback we received contained concerns of the initial project pitch being too "idealized". Without a definition of a problem we were trying to solve, we could not assume our tool would fit a need. Based on this feedback, we shifted our focus into a tool specific for collaborative code review. While there are some tools for general code review, the friction brought up by collaborative code review is not addressed in Slack.
Also, to address our idea being too "idealized", we proposed an experiment methodology to show whether our proposed tool can improve productivity in collaborative code review:
1. We pick an open pull request that needs to be reviewed.
2. We then try to compete the code review using the traditional method explained in the current practices section
3. We then use our application in using our code review method.
4. In both processes, we will measure the time spent in answering questions, time to make changes, and total time spent to complete the review.

We hope to see a reduction in time spent across all areas of the code review process.