

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 inefficient: large team require more effort since not all members are at the same level of familiarity of the code, they might waste time on other development work, and scheduling becomes more challenging [3]. Additionally, current review process such as comment threads and email tend to hinder the development of a full conversation as they are asynchronous procedures, which counters the conclusion from a Microsoft's research which suggests synchronous communication can better facilitate communication and thus increase code review efficiency[1]. In both the case of one reviewer and multiple reviewers, 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. This claim is supported by the same research [1]. It points out the biggest challenge in code review is to understand the rationale of the code and one of the recommendations to tackle this challenge is direct and instant communication.

Our goal is to improve on collaboration in 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

Some existing practices of code review including GitHub code review feature, GitHub for Slack App, and Crucible code review tool.

As for the GitHub code review feature, 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 requester will then click through email and go into github interface to make the change, which will again be sent out through email to the reviewers. The conversation is driven by email alerts that are much slower and can easily get overlooked [2]. This back and forth process can will slow the review process down because of the lack of real-time communication.

While there are existing Github Slack integrations, the functionality they provide is limited to pull request reminder: wnce users have downloaded the app to their Slack workplace, they can receive messages from the GitHub Slack bot with a link to check the updates of their Git repository when new pull requests or code changes occur. This does not solve the root problem of indirect communication because users still need to jump to different places to take actions.

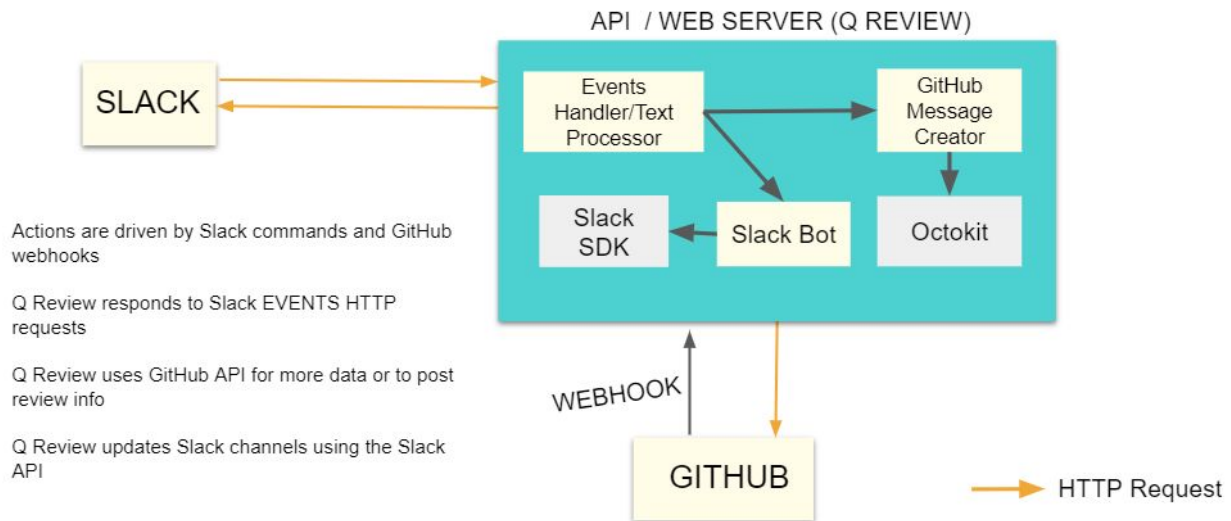
Another existing tool is Crucible - a code review software by Atlassian. However, it only supports BitBucket as the Git Client and is not inexpensive, especially for larger teams. Thus it only provides solutions for teams who are financially abundant and use BitBucket.

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 by providing instant communication through which reviewer can better understand the code[1].

Architecture Overview

Below is a diagram and explanation of the high-level architecture design of our application.



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 improve 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 area 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. Our plan to tackle this risk is to start early in the development of the application components that interact with the external resources. This way we can find out early on if a redesign is needed due to limitations in these interactions. 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
<i>Unknown and Unpredictable</i>	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.

Technologies

The main language will be JavaScript using the Node.JS runtime. We chose Node.JS because this is the most popular language in the slack integration community. The majority of example projects are implemented in Node.JS so we will have many good resources to look for help.

We will use the Github Octokit API to get update on the code review. Specifically, we want to create some webhooks that listen to changes on any code review status and send such update into our server. Also, we will be using some POST API to post comments/changes/approval from our users.

For the Slack bot, we will use the Slack JavaScript SDK to develop it. We chose this tool because using native SDK is the de facto way to develop integration. We chose the JavaScript version such that it fits with the rest of the services. Also, the Slack API website provides step-to-step tutorials and many relevant resources to help us get started on building the bot.

Our internal event handler will be using Express.JS framework. We chose Express because it is one of the most popular Node frameworks to handle routing and it provides a simple yet powerful middleware layer to process raw input, which is a perfect fit for our text-processor use case. In addition, our team member who is responsible for this part has previous experience with Express, which would lower the learning curve and fasten the development process.

Responsibilities

We divided our responsibilities as follow:

Ethan: Event Handler / Text Processor

Tim: Github Message Creator / Octokit

Annabelle: Slack Bot / Slack SDK

Keegan: Evaluation and architecture design

Interface Design

When a new pull request is made on Github, Q Review fetches the basic information of it, such as the code and requested reviewers. From here, our bot creates a new channel (in this example called “pull-request-1293”), adds the contributor and requested reviewers to the channel, and sends a snippet of the code to the chat. In this example, Annabelle and Ethan are the reviewers and Keegan is the contributor.



The reviewers can type “/qreview” to call our bot, followed by the following commands to make inline comments/general comment, request changes, and approve the pull request:

Inline comment:

- /qreview comment line+ “...”
- /qreview comment line- “...”

General comment:

- /qreview comment “...”



Request changes:



- /qreview request changes “...”


Approve:



- /qreview approve “...”

Then Q Review would send a message to Github API to synchronize the action, and the bot would inform the users after the actions have been made.

 **Ethan Cui** 1:21 PM
 /qreview request changes "please remove the debug code in our production build"


 **Q Review** 1:22 PM
 Changes requested for commit `404696c`  [@Keegan](#)

 **Keegan** 1:23 PM
 Yeah I suppose we don't need it in there. I'll submit a change

 **Ethan Cui** 1:23 PM
 Sounds good, thanks!



When changes are made to the pull request, the bot sends the updated code snippet to the chat. The contributor can respond to the inline comments and general comments using the same commands:


- /qreview comment line+ "..."
- /qreview comment line- "..."
- /qreview comment "..."



 **Q Review** 1:24 PM
 Changes were added to this pull request by [@Keegan](#)
 commit 4046821 ▼


```



1 174 174 | int x = 0
2 175 175 |
3 176 176 | for int i in range(1, 10):
4 176 -   print "hello world"
5 177 177 |   if (i % 2 == 0):
6 178 +   print "even number"
7 178 -   print "odd number"
8 179 179 |       x++
9 180 180 |
10 181 181 | return x
  
```

 **Ethan Cui** 1:26 PM
 Looks good to me!

 **Annabelle Zha** 1:33 PM
 /qreview comment 178+ "Is this line also for debugging?"

 **Q Review** 2:02 PM
 I've added a comment to line 178+ 

 **Keegan** 2:03 PM
 /qreview comment 178+ "It was initially to change the previous print to the right number classification, but I suppose we don't need it" (edited)

 **Q Review** 2:22 PM
 I've added a comment to line 178+ 

The users can easily communicate, adding comments, and requesting changes without leaving the Slack interface. Note: the comments/change requests will be added by QReview github account, not the user's github account.



Annabelle Zha 2:26 PM

/qreview request changes "Please remove line 178+ as it isn't needed in our production build"



Q Review 2:27 PM

Changes requested for commit `4046821` ✨ @Keegan



Keegan 2:27 PM

Okay I'll take that out

When they finally reach an agreement on the code, one of the reviewers can call the bot to approve the changes.



Keegan 2:27 PM

Okay I'll take that out



Q Review 2:30 PM

Changes were made to this pull request by @Keegan

commit 404823c ▼

```
1 174 174 | int x = 0
2 175 175 |
3 176 176 | for int i in range(1, 10):
4 176 -   print "hello world"
5 177 177 |     if (i % 2 == 0):
6 178 -   print "odd number"
7 178 178 |         x++
8 179 179 |
9 180 180 | return x
```



Annabelle Zha 2:44 PM

/qreview approve "Good Job! Thanks for getting this done!"



Q Review 2:44 PM

Great! I'm approving these changes 🚀

Project Setup Manual

Create a Slack app

- Create an app at api.slack.com/apps
- Navigate to the OAuth & Permissions page and add the following scopes:
 - users:read
 - chat:write:bot

Run locally

1. Get the code
 - a. Clone this repo and run

```
$ npm install
```

2. Set the following environment variables to .env: (see .env-sample)
 - a. SLACK_TOKEN: Your app's xoxp- token (available on the Install App page)
 - b. PORT: The port that you want to run the web server on
 - c. SLACK_WEBHOOK: The webhook URL that you copied off the Incoming Webhook page
 - d. SLACK_VERIFICATION_TOKEN: Your app's Verification Token (available on the Basic Information page)
3. Start the app:
 - a. run: `$ npm start`
 - b. In another window, start ngrok on the same port as your webserver: `$ ngrok http $PORT`

Enable Q Review

1. Go back to the app settings and click on Q Review.
2. Set the Request URL to your ngrok URL + /test

Say Hello

1. Add Q Review to the channel you are working in
2. Type the command /qreview

CI/CD

The pipeline for our project is first triggered by a commit to the master branch of the repository. We use Travis CI to build and run tests before deploying to Heroku.



GitHub: <https://github.com/KeegJordan/CSE403>
Travis CI: <https://travis-ci.com/KeegJordan/CSE403>
Heroku: <https://qreview-cse403.herokuapp.com/>

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 complete 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.

Feedback

We have addressed all feedback.

Citation

[1] Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013.

[2] Devasthali, Snehal S., Jayant S. Koppikar, and Prasad P. Purandare. "Preventing a user from missing unread documents." U.S. Patent No. 9,438,551. 6 Sep. 2016.

[3]Porter, Adam, Harvey Siy, and Lawrence Votta. "A review of software inspections." *Advances in Computers*. Vol. 42. Elsevier, 1996. 39-76.