QREVIEW

Written by: Annabelle Zha, Ethan Cui,
Keegan Jordan, Tim Chirananthavat

Abstract

Code reviews are an essential aspect of many software development teams. Most tools used by these teams feature a communication method based on comment threads. This is not an ideal method of communication for this purpose since this method is formal and indirect. Our research found improvements in team communication when using platforms that feature instant messaging capabilities. We devised a solution to this problem by implementing a plugin for the Slack messaging platform that integrates GitHub code reviews in instant messaging channels between the author and reviewers. Surveys of developers in our personal network showed optimism for this solution to improve their code review experience. In this paper we discuss some of the design and implementation decisions we made while developing this project as well as results we gathered in some pre-release tests. This project is still being developed and roadmaps and future plans are listed throughout.

Our Goal

Code reviews in software development is an important process used to ensure the quality of code meets a particular standard. 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. Our research and experience has led us to identify an inefficiency in the process due to communication friction. Our goal is to reduce communication friction in the code review process by creating a Slack plugin to enable the use of the messaging platform for rapid and informal conversation among multiple reviewers in a code review.

A code review can be as simple as two people discussing code changes with each other on a single screen. Often times at larger organizations code reviews are done through tools that engage multiple people across multiple teams. The process becomes more complicated and inefficient for reasons such as reviewers not being familiar with the code, time wasted on communication friction, and time spent waiting on scheduling and other dependencies [4]. Current review processes such as comment threads and email are considered asynchronous procedures which tend to hinder the development of a full conversation as concluded in a Microsoft research article suggesting synchronous communication to improve communication and code review efficiency [1]. It is also mentioned in the Microsoft research that collaboration is negatively affected by the lack of instant communication and ultimately slows down the process of developing software [1]. Additionally, the research points out that the biggest challenge in code review is understanding the rationale of the code changes which can be communicated simply through conversation. In conclusion, our research points to problems in collaboration and communication to be a major cost of code reviews and can be improved by instant, conversational communication.

Current Practices

In current practice there a multiple ways to complete a code review. Some ways include in-person communication while others are fully remote. The methods most similar to Q Review are done with tools such as GitHub Code Review that give the user the ability to add comments and request or approve changes. In this method, when new code is requesting review, the reviewers will be notified through email with a link to the change. When a reviewer leaves comments on the code or approves or requests changes, the involved parties are notified through email or alerts on the platform. Emails are then sent back to the reviewers when the author changes the code or responds to comments. For this method, our research showed that "for users who receive a large number of emails, important emails that need immediate attention can be overlooked"

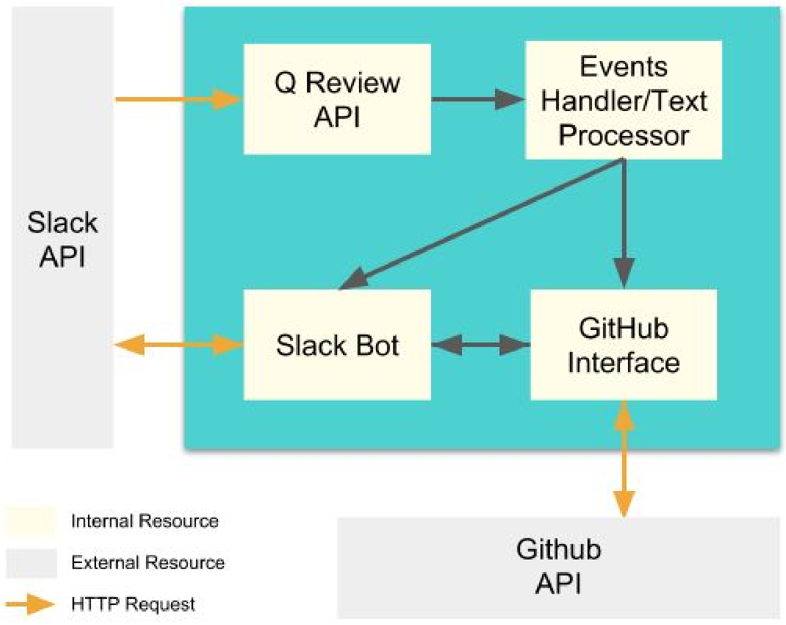
[2]. Q Review is intended to replace this method with communication for these actions done through instant messaging.

While there exists a GitHub app for Slack, it does not provide the functionality to interact with code reviews and merely sends notifications of review events that take place on GitHub. We believe this to be an insubstantial solution to our problem since it does not encourage the use of the Slack platform as the location of conversation but rather as a notification center only. Users are still directed to GitHub for the conversation and actions in the review. There is also an instant messaging platform for code reviews called Atlassian Crucible but we found its limited support and marketing focus on enterprise development teams to be too restrictive for solving this problem for all variations of software engineering teams.

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



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.

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.

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.

Commands:

Inline comment

/qreview comment line 76

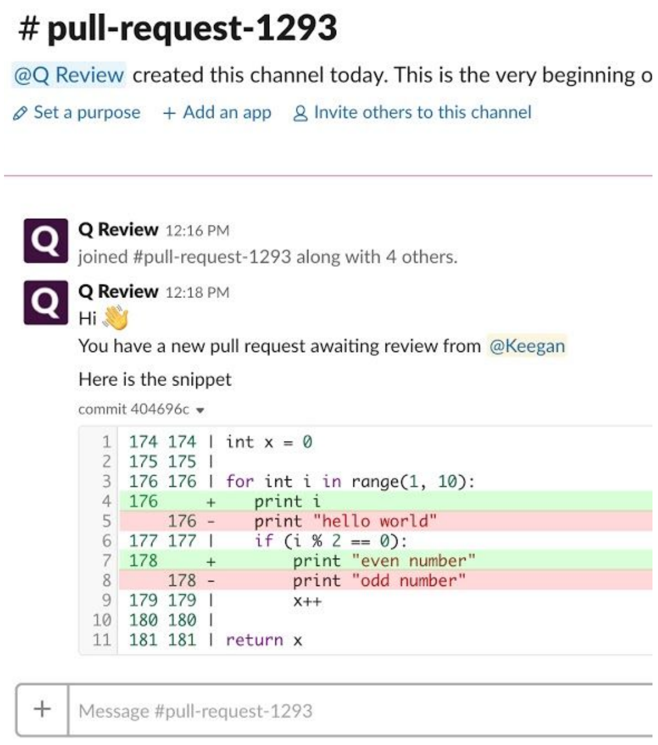
General comment

/qreview comment

Request changes

/qreview request changes

Approve



pull-request-1293

@Q Review created this channel today. This is the very beginning o

Set a purpose   + Add an app   Invite others to this channel

Q Review 12:16 PM
joined #pull-request-1293 along with 4 others.

Q Review 12:18 PM
Hi 👋
You have a new pull request awaiting review from @Keegan
Here is the snippet
commit 404696c

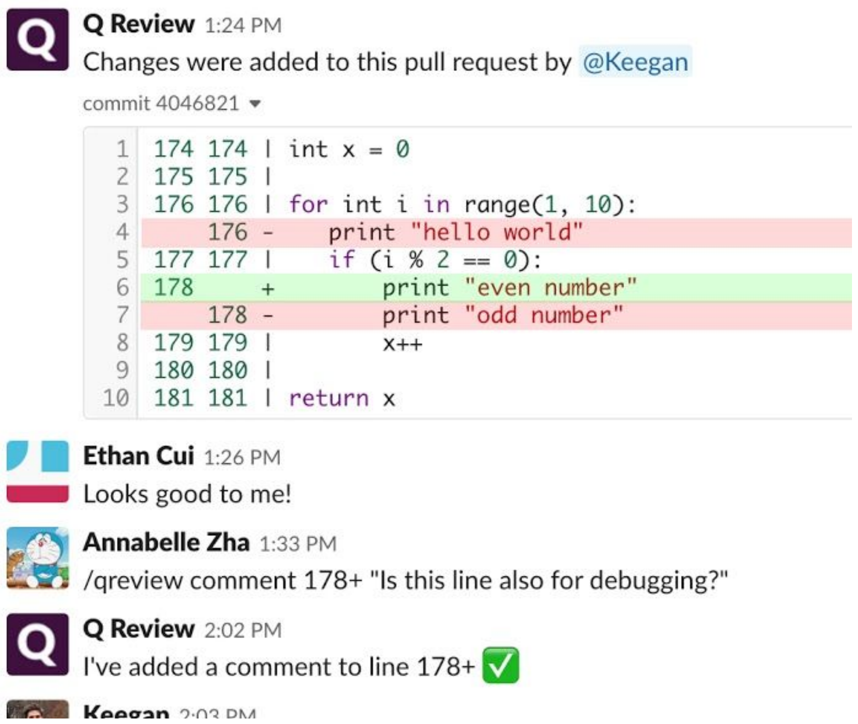
```
174 174 | int x = 0
175 175 |
176 176 | for int i in range(1, 10):
176     +    print i
    176 -    print "hello world"
177 177 |    if (i % 2 == 0):
178     +        print "even number"
    178 -        print "odd number"
179 179 |        x++
180 180 |
181 181 | return x
```

Message #pull-request-1293

/qreview approve

Interface Design

When changes are made to the commit, a new diff is sent to the channel for review



Q Review 1:24 PM
Changes were added to this pull request by @Keegan
commit 4046821 ▾

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

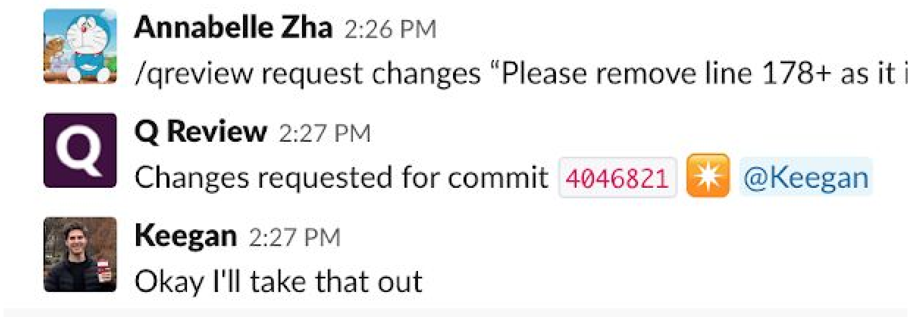
Ethan Cui 1:26 PM
Looks good to me!

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

The users can easily communicate, adding comments, and requesting changes without leaving the Slack interface.


Annabelle Zha 2:26 PM
/qreview request changes "Please remove line 178+ as it i

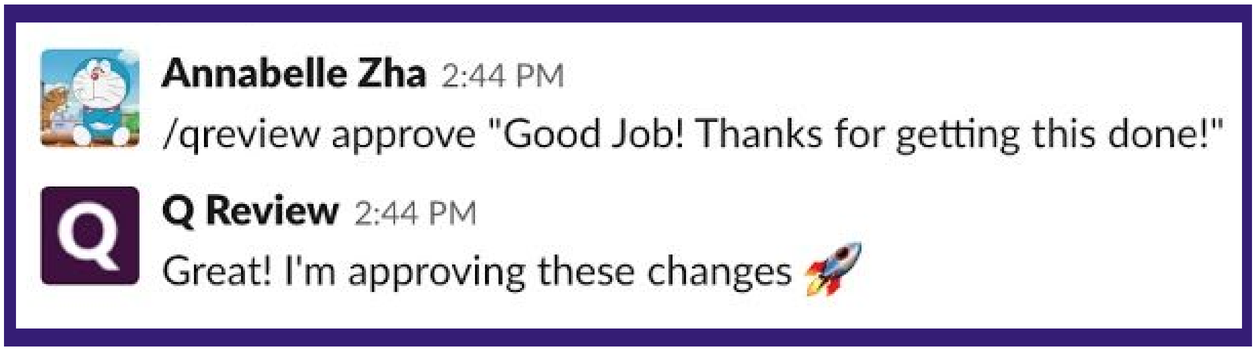
Q Review 2:27 PM
Changes requested for commit `4046821` ✴️ @Keegan

Keegan 2:27 PM
Okay I'll take that out

Note: the comments/change requests will be added by QReview github account, not the user's github account.

Interface Design

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


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

View the GitHub Repository Here

Create a Slack app

1. Create an app at api.slack.com/apps
2. Navigate to the OAuth & Permissions page and add the following scopes:
   * `users:read`
   * `chat:write:bot`
3. Click on "Install App to Workspace"
4. Navigate to the Slash Commands page and add a command (e.g., `/qreview`). Put anything for the request URL and short description for now.

Run locally

1. Get the code *Clone the repo and run `npm install`
2. Start the dev environment by running `npm run dev`

Enable Q Review

1. Go back to the app settings and click on Q Review.
2. Navigate to the Slash Commands page and set the Request URL to your ngrok URL (printed by `npm run dev`) + /test

* For example, `https://ade1f065.ngrok.io/test`

3. You should now be able to try out the command in Slack. The format is

        /qreview [command] [message]

Currently, the supported commands are:

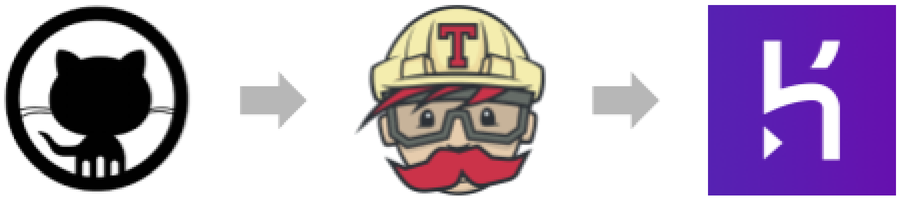
        /qreview approve “message”
        /qreview request changes “message”
        /qreview comment “message”
        /qreview help

Alternative Approach

If you are having trouble setting up the project, please contact our team. We can simply invite you to our workplace and let you try out Qreview there!

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.



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

Performance and Quality Measurement

We will use time between messages within a pull request as our main measurement. Specifically, we will measure the difference between the current practice of completing the review on GitHub and our solution to illustrate how our method can reduce the total time spent on a code review. To obtain our data, we plan to set up some event listeners for pull request changes. Whenever there is a new change to the pull request, the event listener will store the request time in a database. Within the database, we will use one table to store the response time of conventional methods and another will store that of our solution. We can then compare the data in theses two tables to output a meaningful metric. In addition, since the data will be stored in the database, we can easily automate the metric output process by writing a script for it.

Secondly, we will measure request response time. When a user comments or makes a change we want the Slack to update as soon as possible. For this we write the command used and the response time to a persistent file and monitor the changes in time. While we are expecting time to increase as logic is implemented, we hope to be able to reduce that time through performance-focused design and reduction of access to large data storages.

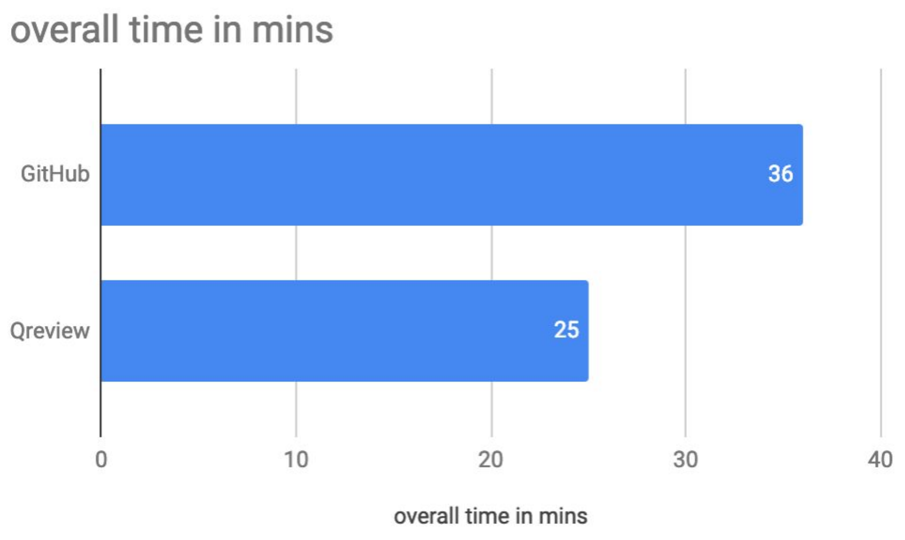
Lastly, we will also collect some qualitative data from our target users. For this study we will apply convenience sampling, which means we will recruit most accessible subjects as our participant. It is least costly in terms of time, money, and energy. Besides, as the appropriate sample size of a qualitative research is “one that adequately answers the research question”[3], we think it will be adequate to have an approximation of 10 people to respond to our survey. The majority of our participants are UW Computer Science/Engineering students and some are junior software engineers of the surrounding technology companies. Most of them have had some experiences with code review at some point of their lives. We will send out a Google form survey to them and here is a link of the survey: Link. In the form we ask the users some general questions on their previous code review experiences and the inconveniences they encountered, then we introduce them Q Review and record their responses to it. We hope to gain some insight of the need of our target users as well as their attitudes towards Q Review.

Comparative Study
In order to evaluate the effectiveness of our tool on solving our stated problem, we designed a comparative study. We designed the study as following:
There will be three participants in the study. One “contributor”, one “reviewer” using GitHub pull request interface, and another “reviewer” using Qreview. The contributor in this study is “contributing” a function to a Java project. He will submit his code through pull request. Two reviewers will be given identical “standard answer” of the code”. The reviewers’ goal is to make the submitted answer similar to the “standard answer”. By “similar” we mean they are algorithmically same. I.e. same data structure, same algorithm, identical asymptotic runtime complexity. However, the reviewers are not allowed to directly disclose answers to the contributor. They are required to first read through and understand their code and then make suggestions to the contributors as in normal code review.

In the first part of the study, the contributor will finish this process through GitHub pull request interface. In the second part, through Qreview. In the process, we ask the contributor to only record the time spent in communicating with reviewers and leave out code implementation time as our focus is to reduce communication cost. We then gather both the overall time used in communication and the average individual commit time to compare the two approaches. A more detailed test protocol is available in our repository.

Per our test result, Using GitHub pull request interface, it took about **36 minutes** to complete the whole process. Using Qreview, the time is about **25 mins** (there is only one code change in both tests). That is, Qreview is 25% faster than traditional approach. While this is only one case study without more data because of time limit, we think this is a preliminary proof of the effectivity of our project in reducing time between messages and facilitating the development of conversation.

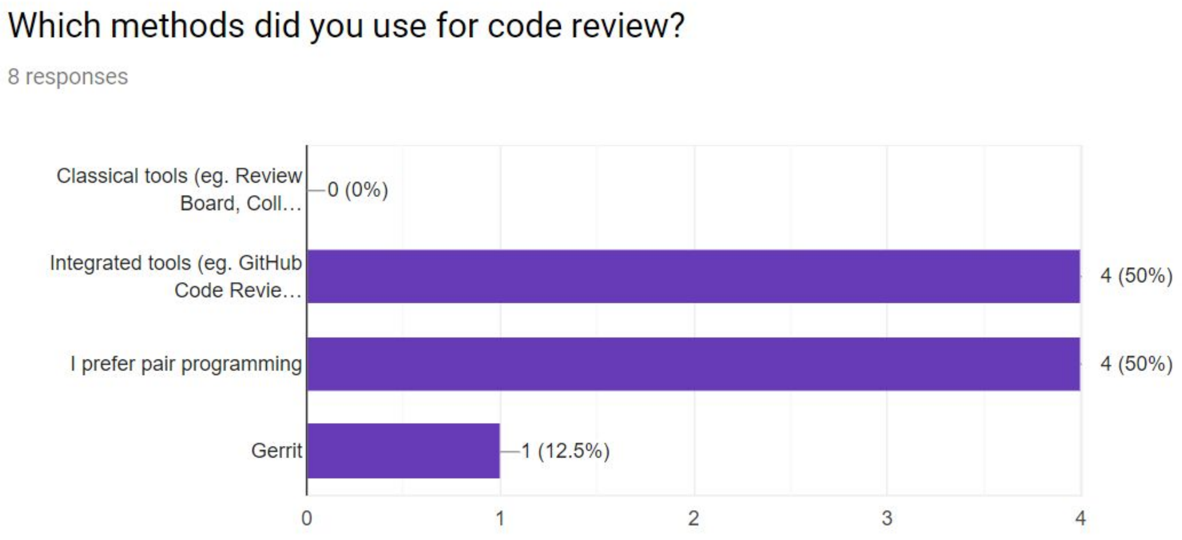
Here is the graph for the test:



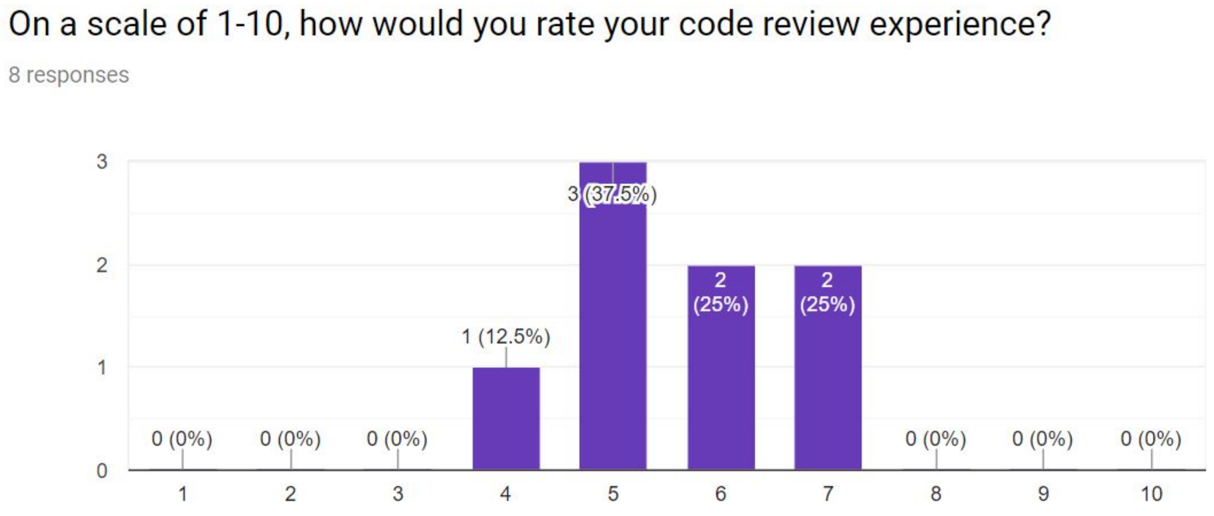
To replicate the test result, you can reproduce this study through our test protocol in the repository.

Survey Results

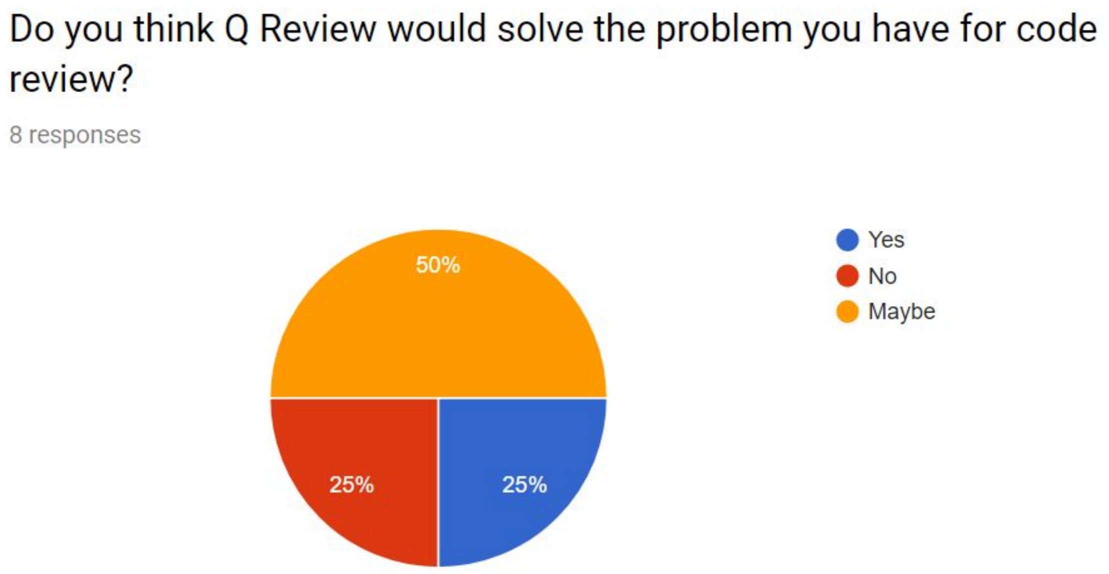
As mentioned above, we created a survey to understand the current practices and needs of developers who currently use code reviews in their projects. The initial results are shown below.



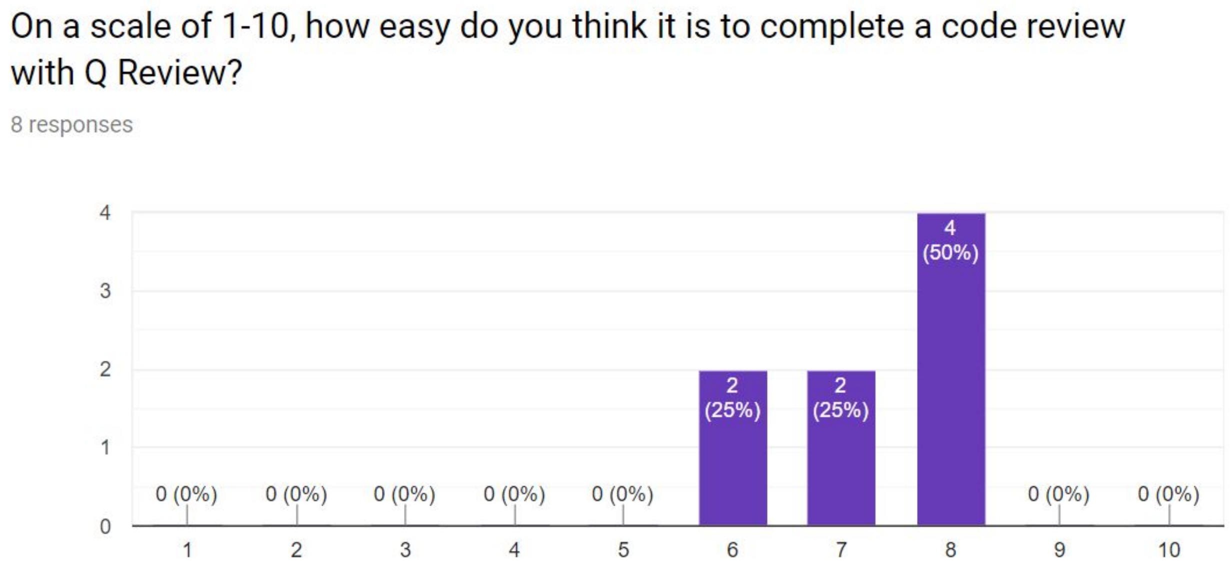
(*Note that for this question our respondent can choose more than one option. Our 4th option for this question is “other”, and someone typed in Gerrit, so Gerrit is shown on the graph)
We hoped our users currently use GitHub as their method of creating code reviews since our application is focused on improving that method. **62.5%** of users currently use GitHub or other integrated tools to complete code reviews. “I use GitHub for code review so I don’t need to switch back and forth between my project repository and the code review tool” said one of the participants.



Currently their experience for current methods have room for improvement.



25% of participants were certain Qreview would solve the problems they have in code reviews, 50% of participants' responses indicate Qreview may increase their efficiency in code review. Only 25% of participants think Qreview would be unhelpful.



Finally, participants on average marked the use of Q Review over their current methods to be easier to use. One participant said, "I think Qreview can avoid the hustle of switching between different tools to communicate, instead it allows us to finish everything within one tool!" Another participant, who is also a Slack user, said "Qreview will help automate some procedures such as invite related reviewers to a specific channel. Although the idea seems a little raw, I think the overall principles behind this is promising".

Our main conclusions for this preliminary survey results is that users are uncertain that Q Review will be a solution to their current methods but have hopes that it will outperform the existing platforms in place. We will continue to collect more data to ensure better accuracy.

Raw Data and Graph reproduce
Since all of our tests are either survey-based or user-study, our graph is generated based on our gathered data.

Survey Graph
Comparative Study

If you want to reproduce the whole tests, you can use our survey questions and comparative study protocol to conduct further testings yourself. These information are available under '~/evaluation' directory.

Discussions
The most insightful discussions that helped our project is about how to prove the effectivity of our project. There are two sides of it. First, initially we made a claim that by using our tool, the code review efficiency will be improved. When discussing this with course staff members and within our team, we realized we need evidence to back up our claim. We then cited some research paper that supports our claim. Second, it has been pointed out that providing concrete numbers and metrics are necessary in proving the effectivity of the project. Based on this discussion, we designed a set of methodology to measure our effectivity. We also gathered qualitative data through surveys.

Lessons Learned

Process
- Team leader should assign concrete tasks to each team member at every checkpoint to make sure everyone is on track.

- In development stage, it is essential to make sure the project is easy to set up on all members' platform. Otherwise, development hours can be unnecessarily wasted on project setup.
- In order to avoid bottleneck module, there should be concrete deadline for each member's task. This way, if something is not on track, the team can address it early to resolve it instead of getting blocked in the later development process.

Communication
- There should be a file that organizes all the useful sources we found during the development, such as an API manual, helpful stack overflow posts, etc.

Conclusion and Future Work

This project is on its way to a release and a comprehensive test of its effectivity in improving the communication mechanisms in the code review process. The lessons we learned in designing and implementing this application has given the team many avenues to explore in the future of this project. We foresee our application being integrated with the existing GitHub Slack plugin to enable its users to complete code reviews in an instant messaging environment. We will continue to refine our user interface to make a seamless transition from the current approach for our users and we hope to implement an interactive diff viewer within the Slack channels. This project was an exploration into improving developer efficiency and will constantly be an area of focus for our team.

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]Marshall, Martin N. "Sampling for qualitative research." *Family practice* 13.6 (1996): 522-526.

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

Appendix

Responsibilities

- Ethan:	Event Handler / Text Processor
- Tim:	Github Message Creator / Octokit
- Annabelle	Slack Bot / Slack SDK
- Keegan	Evaluation, Ops, and Design

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.

Feedback

We are continuing to improve on the wording used to introduce our research backing the problem space.