Managing software development

Project Report

PRATTLE CHAT SERVER

Sourabh, Rakesh, Aswath, Shweta

Project Overview

The goal of the project was to develop a robust messaging system from a given legacy codebase. The existing codebase will be reused whenever possible to maintain timelines but functionality is planned to be greatly extended. In addition to extending functionality, considerations of scalability will be considered. Our focus is on the server side application, which poses interesting challenges with respect to demonstrating progression to the client and robustness of testing.

The base code handed over to us was a template of a barebones messaging system. This system had one type of message - broadcast message. This broadcast message could be sent by any user and was received by all online users. It had no notion of message or user persistence. There were no security features or authentication in place. Given this system, we worked towards adding core functionality expected from a messaging application.

The first immediate goal that came to mind was to add user interaction to the system in a meaningful way. To facilitate this, we needed to persist users and provide authentication to maintain a profile between sessions. This feature needed to be implemented in an extendable system to allow growth of features like friends lists and special privileges. It also seems necessary to be able to generate groups of users in a dynamic fashion; as in, groups should be easy to join and leave. The control should be with the user and if a group isn't needed, it should be able to be removed.

With the idea of users and groups, messages that are purely server wide do not make a lot of sense. In effect, the usefulness of users and groups are lost if the chat server is a wall of spam mostly mostly irrelevant information. Thus, our next goal was to allow for messaging between users in the form of direct user to user messaging and group messaging. We believe this will be the primary form of communication to be used on our application, and eventually the original broadcast messaging will become functionality restricted to server administrators.

Once core functionality with messaging is achieved, it is critical to consider features for compliance. Compliance to our users and compliance to legal authorities. Another objective of this project is to begin compliance towards the CALEA standard in effect in the US. Our goals are to allow for law enforcement to have their necessary access to our system while maximizing privacy of the general user. With regards to compliance to our users, we expect this system to be a full solution for workplaces, academic institutions and personal usage. With all this in mind, we hope to provide functionality to filter message content and add flagging so the clients that interact with our system can take appropriate action.

We aim to achieve these outlined core functions while maintaining high quality and timely delivery. This project will consist of a four person team executing three SPRINTs lasting 14 days each. Reviews and retrospectives are planned to ensure that the project is on track and that the needs of all stakeholders are met. In the end, we hope to deliver an excellent, robust server side application that is fully tested and meets high quality standards.

Result Overview

We first created a notion of Users and User groups. Users can be instantiated using username and user groups can be instantiated using group names. We persisted these users using LDAP (Lightweight Directory Access Protocol). Each user has a password stored in LDAP using the CRYPT protocol. However, the chosen platform can easily be swapped for other styles of encryption (i.e SHA-256). This enables us to add user authentication and add security to the system. A first time user can register with a username and password and returning users can login with their credentials. Users and user groups can be created, updated/ attributes can be modified, and can be deleted from a system. Groups can have multiple users and they have an administrator with special privileges to add or remove users in the group.

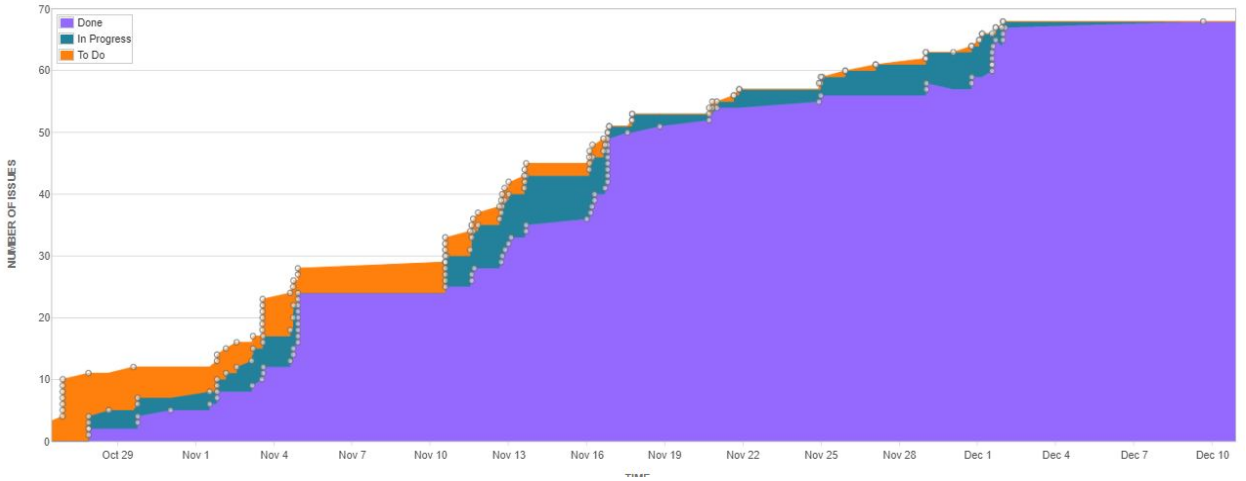
As there exists a notion of individual users and user groups, we added the functionality of direct messaging and group messaging. A direct message can be sent from one user to a specified receiver. When the message is received, the receiver is able to see who has sent the message. In group messages, a message sent by a user in a group to the group is sent to all users in the group.

We persist messages using a Relational Database Management System, MySQL. Messages sent to a user who is offline are enqueued and when this user comes back online, messages are received. Messages sent to offline receivers can be recalled by the sender. If a user wants to see a message sent in the past, the message can be searched based on the sender, the receiver or the timestamp at which the message was sent.

We implemented IP wrapping and wiretapping with CALEA in mind. The wiretap functionality, where a law enforcement agency can tap into a conversation(user to user or group). This read only channel can only be created by system wide administrators and delivers compliance while maximizing general user privacy. Additionally, we implemented a tree based message content filter, which provides fast detection and support for censoring and/or flagging messages.

For the project we have used Jenkins for Continuous Integration and Sonarqube for build quality. We have added functional and structural tests for all the functionalities implemented in each sprint. We used Mockito framework and JUnit 5 for writing the tests. For each sprint we have been able to achieve a test coverage of over 89% without any bugs or vulnerabilities. Figure 1,2 and 3 (*in Appendix*) shows graph of coverage in each of the sprint. We also made sure no bugs, vulnerabilities, security concerns and critical code smells are present every sprint as shown in figure 4,5 and 6 (*in Appendix*).

We were also successful in completing all the base requirements provided in each sprint and additionally were able to accomplish most of the stretch goals. We have been constantly adding jira tickets from the backlog in each sprint and been completing the tasks in timely manner. The cumulative flow diagram shown below shows that the tickets have been constantly added and completed in every sprint.



From where we started, we expanded a basic chat server to deliver on core functionality expected from a robust messaging server. The features were all designed with future scalability and product success in mind while maintaining all necessary quality requirements.

Development Process

During the course of the project we have been adding and refactoring functionality at a regular basis. Looking into our code frequency in *Figure 7*, we can see that updates in the code have been made at regular intervals. We also have been improving our branching strategy such that we have linked each feature with branch and at the completion of the feature being merged to master. *Figure 8* (*in Appendix*) shows we have two main branches master and Pseudo master. Each feature is developed in a separate branch and when completed, merged to pseudo master. At the end of each sprint we made sure to merge pseudo master to master. We also have been constantly adding commits for the features developed. We can see in *Figure 9* (*in Appendix*) the number of commits have been regularly done every week.

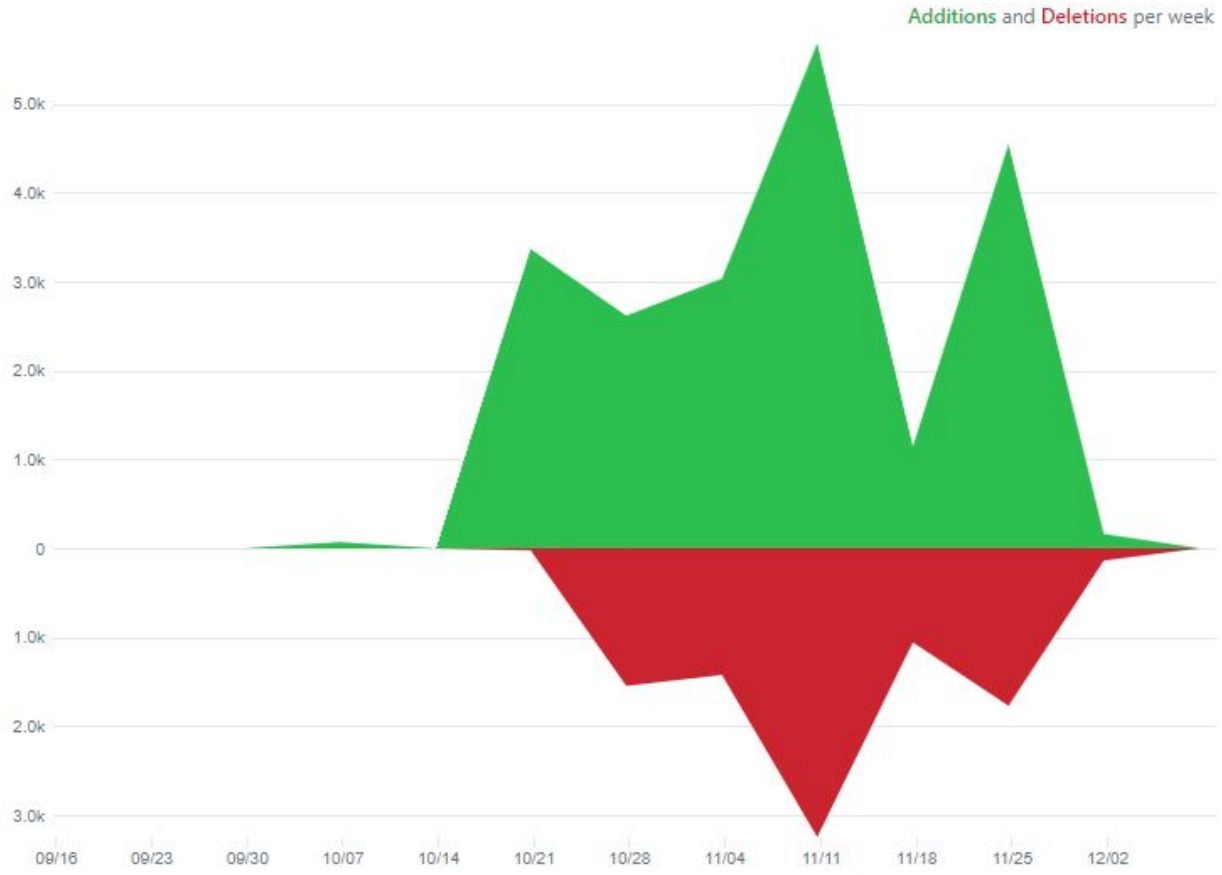


Figure 7: Code Frequency

Agile software development process model was followed for the project. Development took place in 3 Sprints with new requirements being shared towards the end of every sprint. We had sprint planning done before the beginning of sprint to discuss the functionalities to be implemented in the sprint based on the requirements given to us, we discussed and divided the tasks amongst team members. We created sprint in Jira and created tickets in Jira for all the requirements in scope for the sprint and added them to sprint backlog. We kicked off the sprint by creating pseudo master branch for the sprint and we created

branch for each Jira ticket and started developing the functionality. We had daily Slackups (an alternative for daily scrum) which helped us track the development progress on a daily basis. We had sprint reviews with TA at the end of every sprint and we got constructive feedback from the TA.

For deployment we had Jenkins automation server for continuous integration and continuous delivery. SonarQube was used for continuous inspection of code quality which helped us detect bugs, code smells and security vulnerabilities. The existing code and new code being added to codebase were forced to meet 85% test coverage by Jenkins build which made sure our deliverables are of high quality.

We started the project by understanding legacy code and then we added functionalities as per the sprint requirements. We made sure we complete all the required features before attacking the stretch goals in every sprint. At the end of every sprint when we merged the code from pseudo master to master branch we had automated the deployment to AWS, so we had the latest prattle server up and running on AWS at the end of every sprint.

What worked for us:

Using the right tools as part of the development process:-

Jira helped us organize sprints and tasks efficiently making it easy for us to use agile development process in the project. Slack helped us communicate with team members and instructors. Git for version control and peer review. Jenkins helped us maintain build, eliminate manual deployment and maintain high quality on deliverables.

Technical design for requirements:-

During sprint planning the technical design of major features of the application was discussed as a team, so this design clarity helped in smooth development.

What didn't:

Scheduling:-

Scheduling team meetings was an issue as there were time conflicts and availability issues amongst team members. We did come up with frequent online check-in to mitigate lack of in person meeting time. We could have made the division of work to be more granular to mitigate any one task requiring lots of in person collaboration to complete.

Changing requirements:-

Even though requirements were given to us, understanding the details of the requirements needed constant communication with the teaching staff. This lead to some changes in design and implementation towards the end of sprints which was difficult to cater. From our end we could have been more proactive in clarifying and justifying the decisions we made.

Retrospective

"Good code is necessary but not sufficient for project success."

Recalling this from the first Managing Software Development class, we now realize the truth in the statement. This project helped us gain experience about how teams actually work in a professional environment and how live software projects are managed. This course and especially this project has made us feel ready to work on software and taught us through experience how important it is to go about development of software in a systematic way. Through this project we learned to work along the SCRUM methodology, which is widely used at workplaces. This course is truly valuable for our future endeavours, as it gave insight into how teams at companies work.

We learnt that communication is the key for a team to work cohesively. One of the core things we stressed on during the execution of the project was to communicate the design changes, any additions or decisions made to all the team mates. We also made sure we were all on the same page. We had slack-ups where we discussed progress we made during the day, any issues we had faced or new ideas we thought were good to look into. All of us were aware on what each one was working on.

One of the challenges we faced was that at the beginning, when we were handed over the legacy code to work on and test, we were not very well versed with the client server architecture and how it worked. Due to this, we found testing the code during the first sprint difficult. However, we all sat together and did a code walk of the legacy code, to make sure all of us understood how the system worked. This was important and it helped us develop additional features more easily after understanding the base code.

The feedback we received from the TA was vital to our success in this project and one of the greatest aspects of this experience. She was very helpful and initiated a direct chat channel with our team. We could reach out to her immediately with any questions regarding implementation and requirements and know the feedback was more specific and guided. She was fantastic about responding and very clear about what she needed from us. We believe that having a channel with TA and the team is vital for this project.

Specifically, Srinithi helped guide us and give us insights into what we were doing right, and how could we possibly improve our application. She gave us tips on good practices like merging branches to a dummy master until this dummy master was ready to be merged with the master branch, how to keep the network of branches clean, and many other insightful tips. We realized through her help that this project had more emphasis on process than others we have worked on at Northeastern. No other class we have taken at Northeastern focuses on the process through this lens and it was a very unique insight on how code is only one aspect of Software Development.

Despite these great aspects, we think a couple changes could really enable the experience to be more smooth. The workload for the class weighted towards second half of the semester. This made it difficult to keep up with the course load at the end while being fairly relaxed at the start. One especially big challenge for us was the SPRINT1 beginning during midterms week. It could just be bad luck this semester, but starting the first SPRINT earlier might have been helpful. Perhaps not the actual requirements, but an overview of the project and goals with the source code. Just enough to start to familiarize with the problem and meet our groups.

Another possible aspect for improvement for these SPRINTS would be a bigger emphasis on the toolchain prior to the start of SPRINT. Everyone had a good handle on GIT thanks to the helpful assignment. If we had assignments, or mini-quizzes to emphasize Alex's presentation on JIRA and Jenkins we feel we would have been more prepared. This project for us ended up being a bigger challenge with respect to following process and meeting quality successfully. If these lecture items were emphasized in the first half, it would have been helpful. Without having perspective on the coding ability of the class at large, for our group, we could have seen the design patterns lectures pushed to later in the semester to make room for the lectures on the tools we needed for SPRINT.

Additionally, we feel we fell short with the usage of the TEAMMATES. We had some issues with distribution of work and these issues persisted throughout the project. In the future, it might have been helpful for each of us to hear where we contributing positively to the project and areas that we should improve. We understand this tool is primarily for instructor use, but if the form had more mandatory elements that highlight strengths and weaknesses, the individual feedback could also be routed to us for improvements over time. The main concern here is that the individual grade did not provide insights on personal improvements through each of the SPRINTS.

At the end of this project, we learned to work well as a team, and most importantly got a preview of how working at a company, developing software in future is going to be. We will take lessons learned during the project and the course in general, forward, out of the classroom and implement while doing projects in future, as we have learned how to manage processes and coordinate and collaborate to work towards making a successful project.

In conclusion, we would like to thank Prof. Mike Weintraub for constant guidance and teaching us about the various principles of managing development of software, our TA Srinithi Ramesh for mentoring us during the project sprints and Alex Grob for helping us tirelessly with Jenkins issues and the github server.

APPENDIX

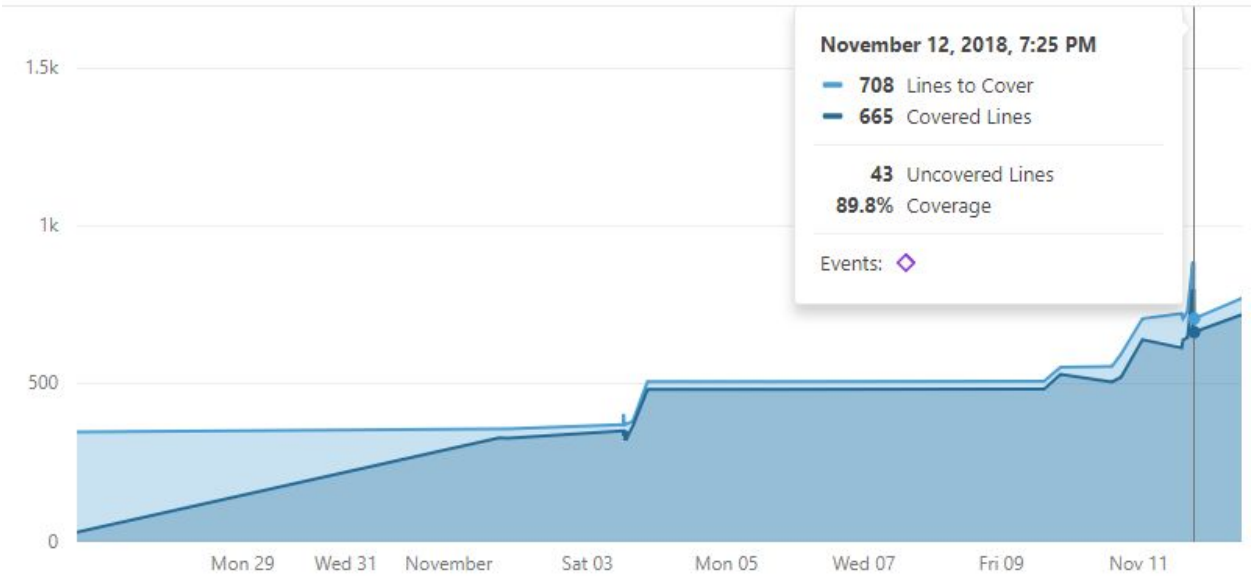


Figure 1

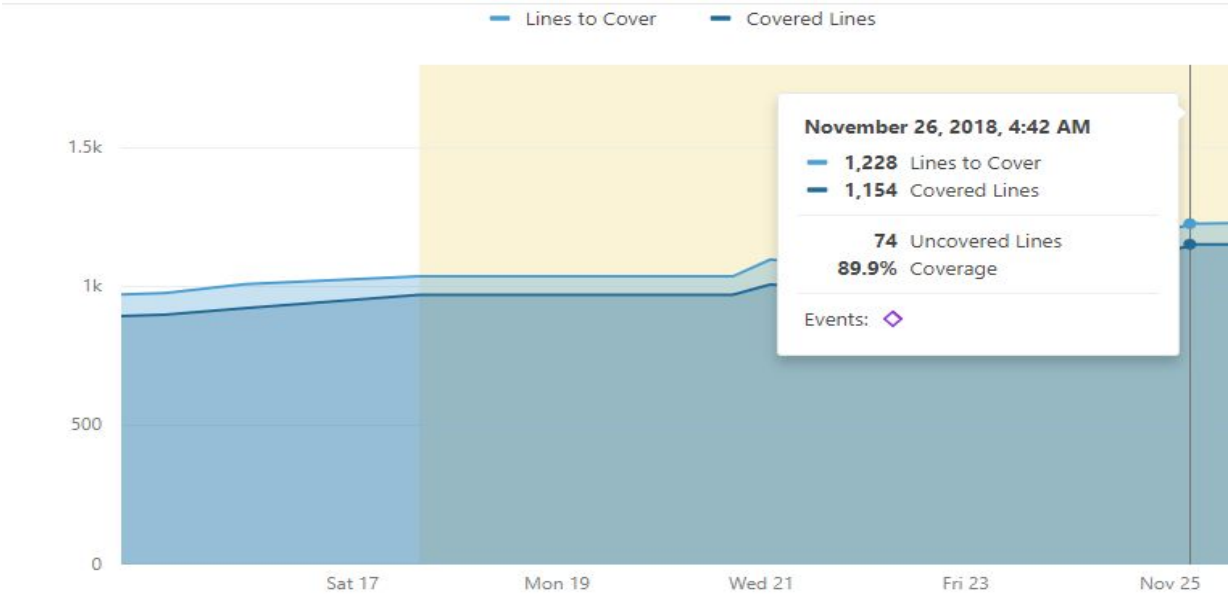


Figure 2

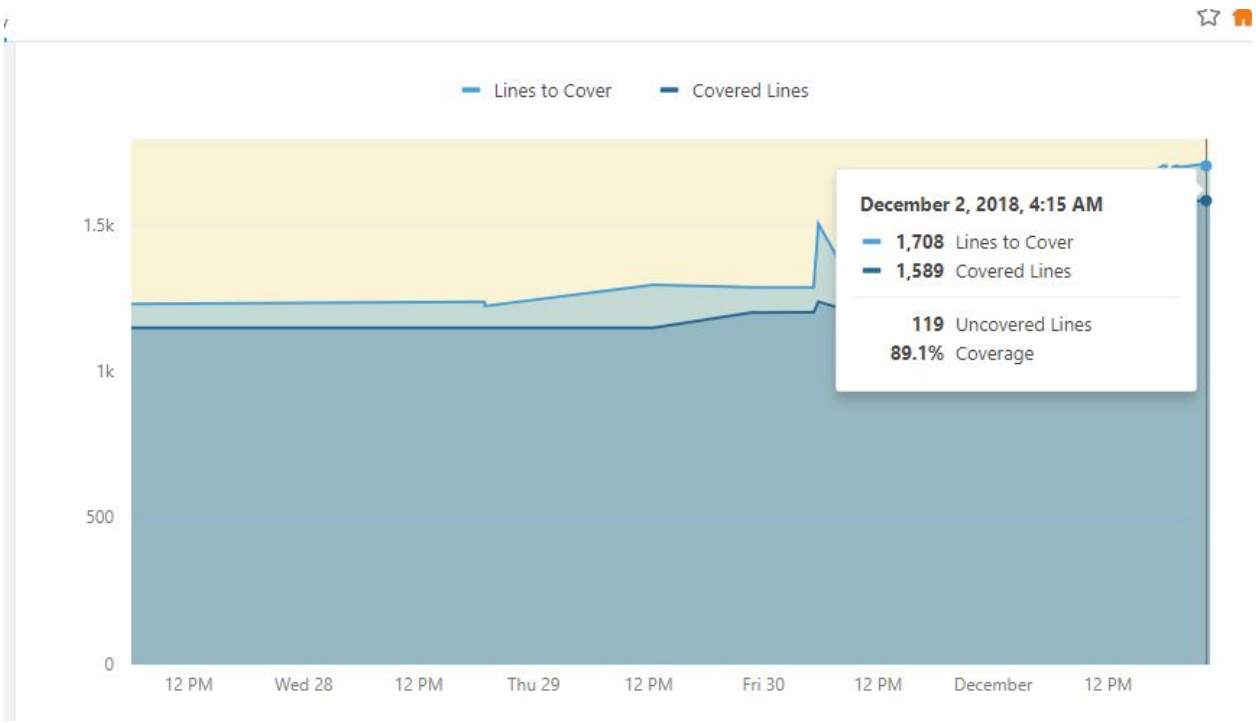

Lines to Cover
Covered Lines
1.5k
1k
500
0
12 PM
Wed 28
12 PM
Thu 29
12 PM
Fri 30
12 PM
December
12 PM
December 2, 2018, 4:15 AM
1,708 Lines to Cover
1,589 Covered Lines
119 Uncovered Lines
89.1% Coverage

Figure 3

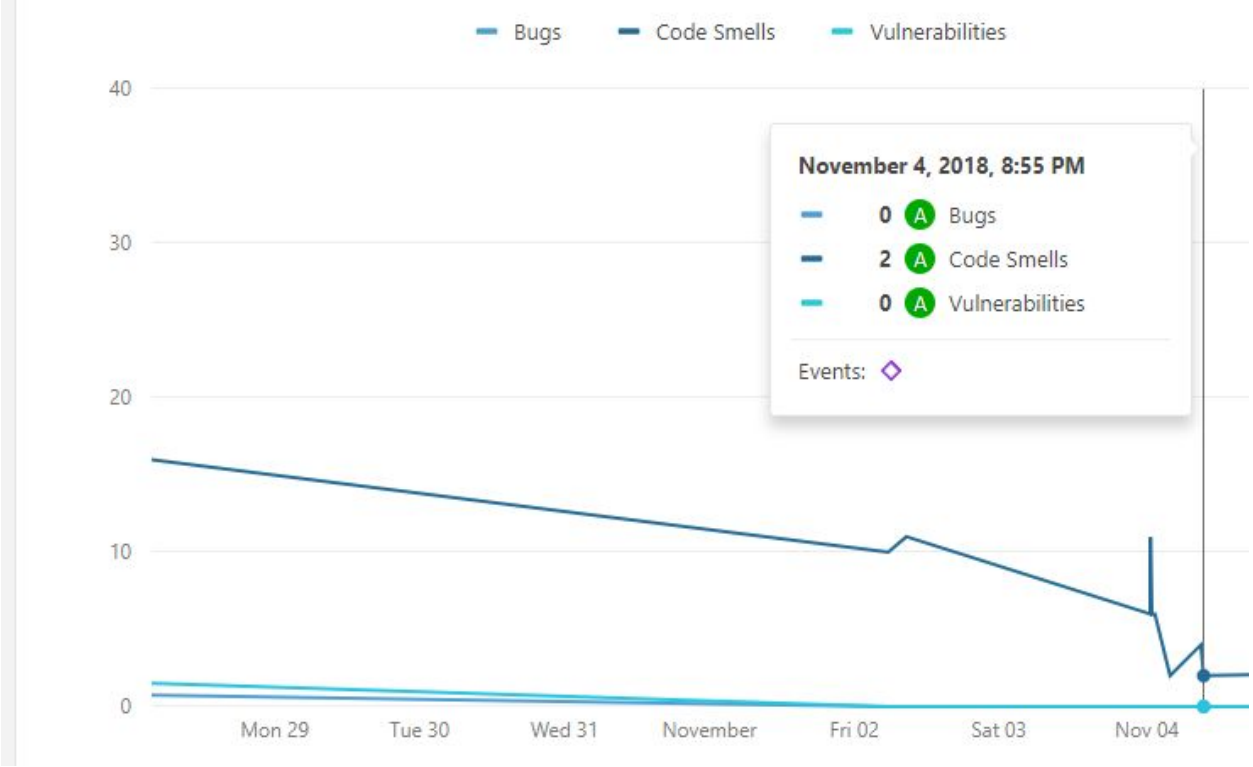


Figure 4

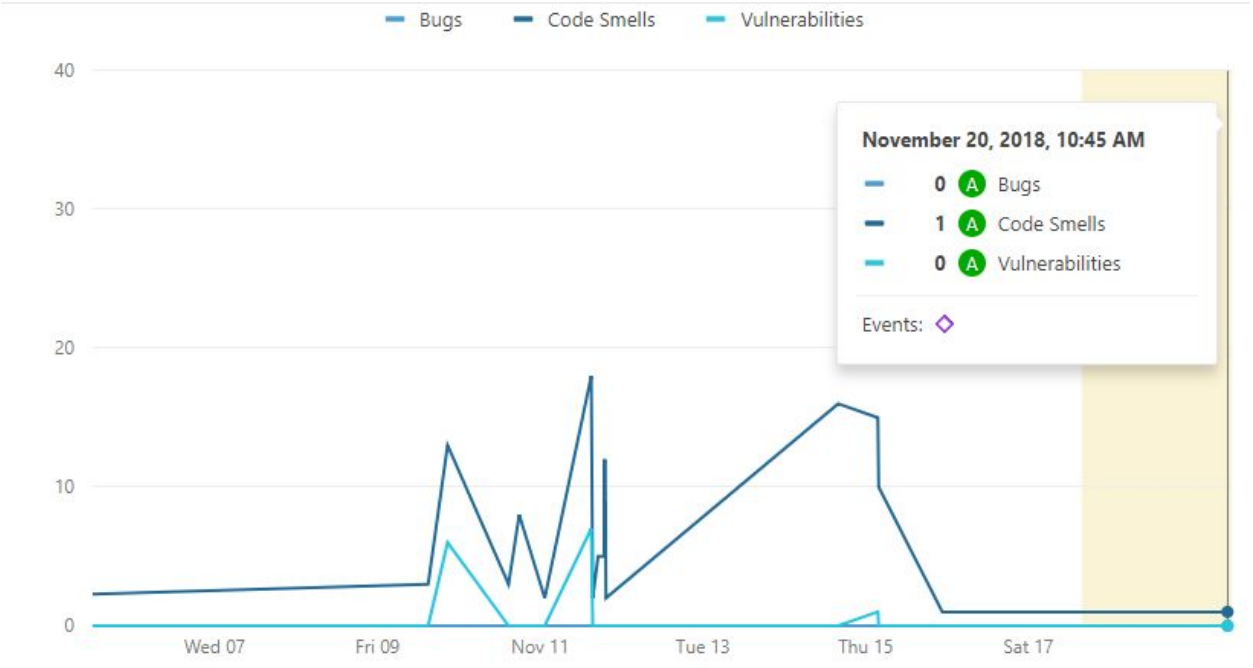


Figure 5

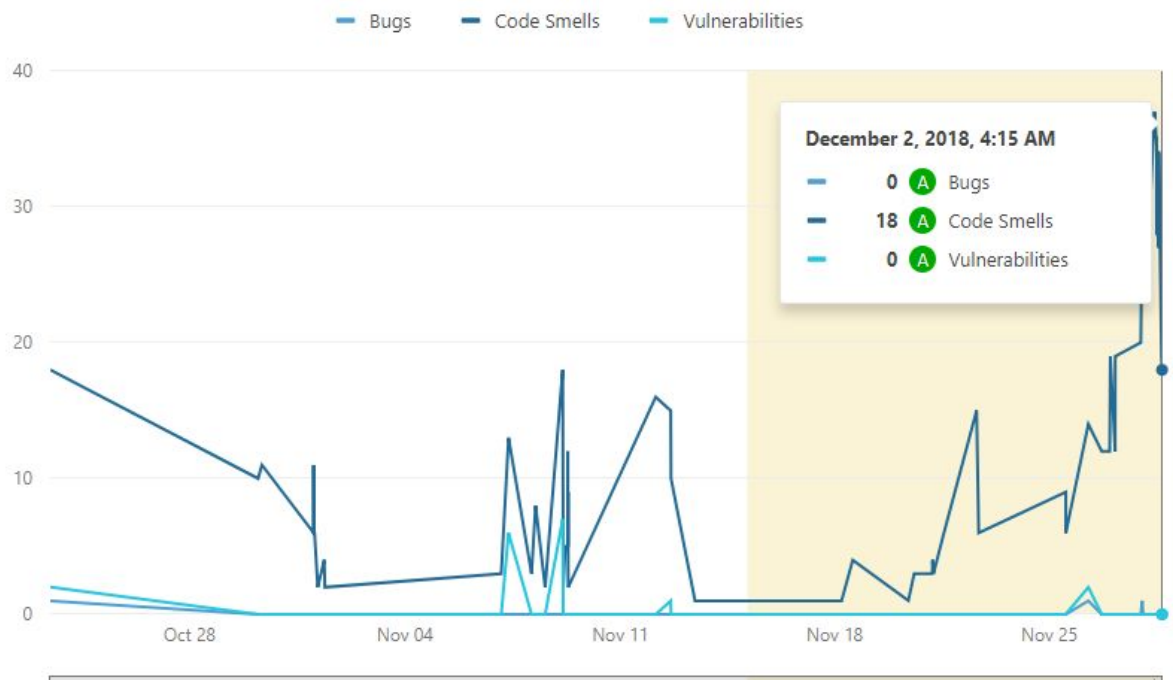


Figure 6

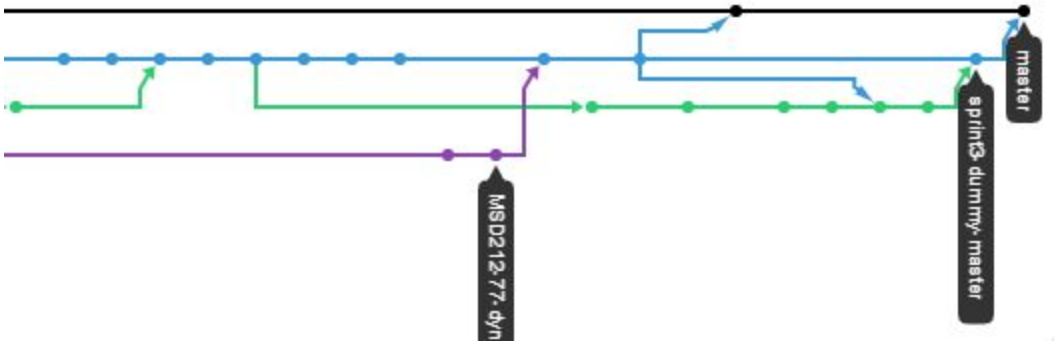


Figure 8

Figure 9