CSC 510 Final Report on Task Tracker BOT

Problem Solved by Task Tracker

Nowadays, building a massive project requires daily task tracking in case of missing the deadline for the project. Therefore, our group came up with an idea of task manager BOT to help deal with the problem described above. In software engineering field, projects are always divided into multiple parts for each developer. Each developer is responsible for tasks assigned by a project manager. In a regular company, there is a daily Scrum meeting for members to set a goal, talk about what they already finished and what they are going to complete later on. However, we only have a weekly meeting because we are students and have other course loads to handle. Thus, task manager BOT helps us to solve the time issue (meeting) and make task tracking easier.

Sometimes, the part of work you are in charge of might depend on others work (concurrent issue), you will have to busy waiting until your colleagues completed their job and then you can start working. With our task manager BOT, it is unnecessary to keep asking the progress of your colleagues, since the BOT will notify every member in the same channel that who completed what task. In this way, we improve the communication time usually wastes on sending personal message or emails. Hence significantly cut down the overhead when developing the code.

On the other hand, from the lecture of Productivity, we know that the average time that a programmer can concentrate on the work is 2 hours per day if he/she is lucky (No other disturbing tasks such as meeting, business trip and so on). For this scenario, we came up with the idea that we transform a daily routine into reward points and make a leaderboard to reflect every member's performance (productivity). The more contribution you commit, the higher rank you will get. The BOT will calculate reward points for developers that finish the assigned tasks according to the different difficulty level. It will show the leaderboard and the target scores for you to achieve in the next week. To whom contributed a lot, the BOT will post congratulation message on a public channel. As for those who did not participate enough, the BOT will send reminder message in public channel as well to alert everyone that you did not make a certain progress and have to work harder. This can be viewed as the peer pressure that urges you to code more and make more contribution. This is why we think our BOT is helpful to increase member's productivity.

Besides, the basic function of a task tracker is to track unfinished task and send out the notification. The BOT will send the user a reminder email for pending task, resembling an annoying alarm clock did not go off until you complete your task. One other aspect of this is that the BOT acts like an active teammate to remind you of things. We can always use hard/soft copy of notes to put down “things to do”, but it’s human nature that we work better when someone actively pushes us forward continuously. Our BOT is the closest human imitation in that regard.

To sum up, our BOT automatically tracks everyone’s work, notify member immediately once a task has been completed and evaluates the performance of members. The BOT leverages the load of wasting time on waiting for others and allow a project to make progress smoothly.

Our three use cases focus on problems we just addressed above:

(1) Send Nagging Reminder
(2) Performance Evaluation
(3) Reminder Body

We will introduce these features in detail in the following section.
Tools we have utilized for our project including the followings:

Primary Features (Xiaoting)

Nowadays, since collaboration between programmers become more and more common. Online development platform like Github, group discussion forum like Slack and task management platform like Trello emerged to improve the collaboration and ease the burden of the team leader. However, none of these tools can simply promote group work. Therefore, the lack of task management tool for software developers and engineers motivate us to start our project. We design a task management bot that resides in Slack (running as a server daemon) that can new changes from Trello team board and send the message or request message to the Slack team.

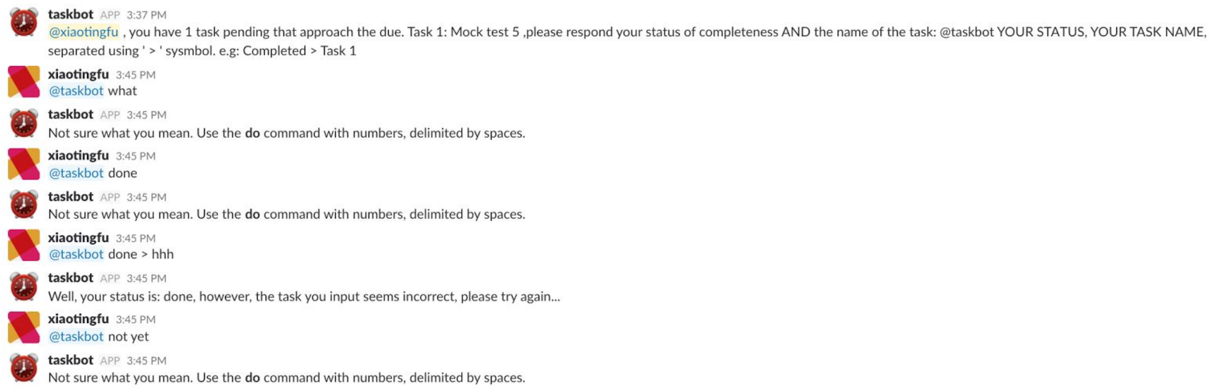
Team member tends to forget their task assigned to them and the deadline of the task, and with time passes by they may only remember until it is too late. Team leader, who is responsible for tracking the task can send the message to the team member to remind him. However, what if a group have no leader and everybody only want to do their part? Now an automatic task tracking bot will be beneficial. At this stage, we manually assign tasks to each team member, but in the future, we would like our bot to be able to act as a team leader also.

As for now, our bot's functions are closer to task reminder, with each existing task cards in Trello our bot detected, it will send an email and direct message to remind the bot and also ask for the progress of the task. We summarize the following functions of our bot:

1. Real-time Trello and Slack Integration
As we know, Slack and Trello are two different platforms, while Slack is more focus on communication between team members, Trello doesn't support communication channel and is more reliable on team member's collaboration in editing the cardboard and share task information. We find the integration of this two platform more interesting than focus on only one of them. By integrating these two platform, our bot is able to fetch and update data from Trello and post a summary of the information in Slack.

2. Multi-threading for handling multiple chats with different users
Our taskbot can respond to multiple users at the same time in the private channel. This is one part we achieve with multi-threading programming in Python. We found this problem that when three of our testers interacted with BOT at the same time. With an older version of our bot, it will generate an error message of multi-threading problem and crash the program. However, the latest version of a program already fix this problem and become more user-friendly in this sense.

3. User Exception Handling
Without a task name, our bot will prompt the user to give a proper command. We will give the user a list of commands that are available in our command set, like Linux man. A user can get the help of improving their input. We are able to handle unexpected commands, for example in Figure 1, if user input "@taskbot done > ", he will get prompt "Well, you finish the task, however, you didn't specify your task, please fill in the task name after '>' ".



taskbot APP 3:37 PM
@xiaotingfu , you have 1 task pending that approach the due. Task 1: Mock test 5 ,please respond your status of completeness AND the name of the task: @taskbot YOUR STATUS, YOUR TASK NAME, separated using ' > ' sysmbol. e.g: Completed > Task 1

xiaotingfu 3:45 PM
@taskbot what

taskbot APP 3:45 PM
Not sure what you mean. Use the **do** command with numbers, delimited by spaces.

xiaotingfu 3:45 PM
@taskbot done

taskbot APP 3:45 PM
Not sure what you mean. Use the **do** command with numbers, delimited by spaces.

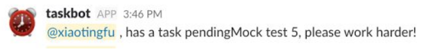
xiaotingfu 3:45 PM
@taskbot done > hhh

taskbot APP 3:45 PM
Well, your status is: done, however, the task you input seems incorrect, please try again...

xiaotingfu 3:45 PM
@taskbot not yet

taskbot APP 3:45 PM
Not sure what you mean. Use the **do** command with numbers, delimited by spaces.

Figure 1. “Taskbot” handling user unrecognized input

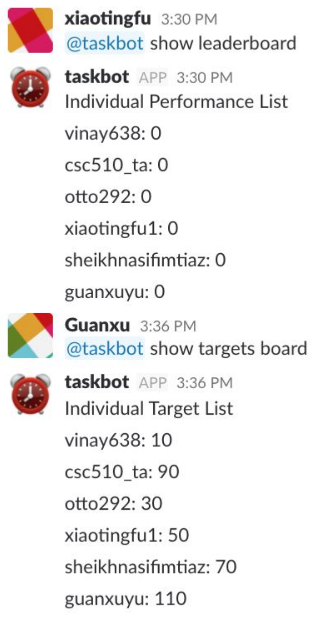


taskbot APP 3:46 PM
@xiaotingfu , has a task pendingMock test 5, please work harder!

Figure 2. “Taskbot” respond to user input

4. Leaderboard Performance Measurement

We design a performance calculation mechanism, based on the level of difficulty of the task and the time interval between due date time and the task completion time. The leaderboard will be refreshed on a weekly basis, and the past score will be archived and saved into our database. The idea behind this design is to give an overview of each member’s performance in each week and to balance their workload accordingly. However, one fascinating future investigation is to see how the leaderboard will affect each team member’s performance, will the outperformer continue beating his teammate, or will they began a competition instead of constructing a balanced workload.



xiaotingfu 3:30 PM
@taskbot show leaderboard

taskbot APP 3:30 PM
Individual Performance List
vinay638: 0
csc510_ta: 0
otto292: 0
xiaotingfu1: 0
sheikhnasifimtiaz: 0
guanxuyu: 0

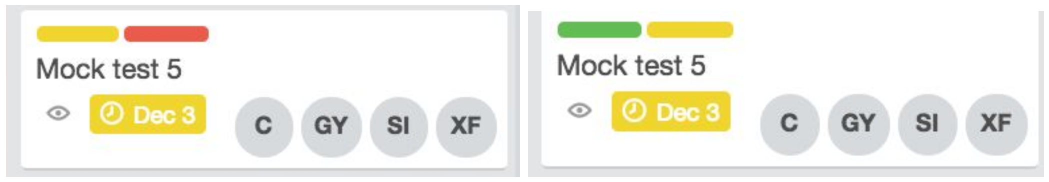
Guanxu 3:36 PM
@taskbot show targets board

taskbot APP 3:36 PM
Individual Target List
vinay638: 10
csc510_ta: 90
otto292: 30
xiaotingfu1: 50
sheikhnasifimtiaz: 70
guanxuyu: 110

Figure 3. Users ask “taskbot” to display the target and actual leaderboard

5. Separate Private Message and Public Message

Our bot is able to collect user progress feedback via private direct message channel since we thought it would be too much a mess to put everything in the general channel. However, it is explicitly designed to post the congratulation and encouragement message in the public channel, so that every other team member can get the latest information of this user's progress and will be motivated to work hard to catch up with him/her.



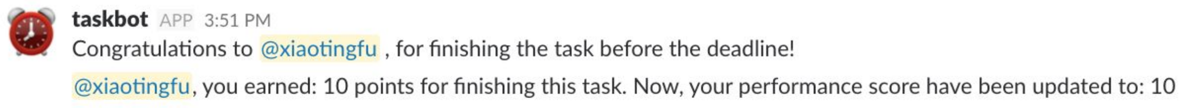
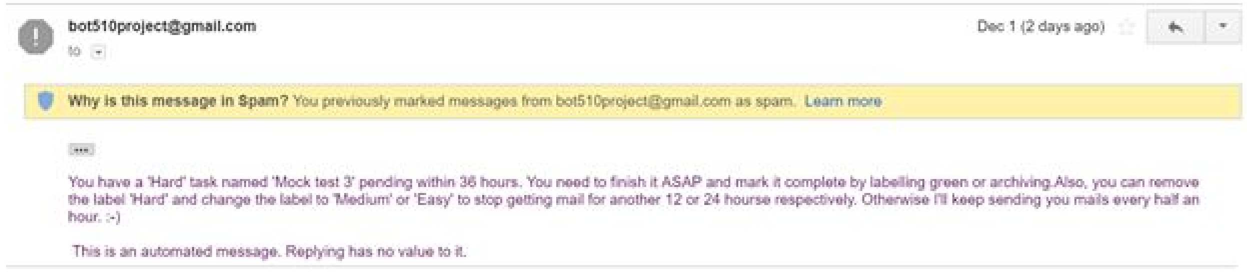

taskbot APP 3:51 PM
Congratulations to @xiaotingfu , for finishing the task before the deadline!
@xiaotingfu, you earned: 10 points for finishing this task. Now, your performance score have been updated to: 10

Figure 4. Left image: Trello card before user input; Right image: Trello card after user input Bottom image: User responds "taskbot" in direct channel "@taskbot completed > Mock test 5" in public channel and "taskbot" post a congratulation message in general public channel

Now, we will explain in brief the primary features of the BOT with the help of screenshots. We have three use cases.

In the use case 1 - we are sending nagging reminders to the team members who have tasks pending in the near future. The task whose deadline is, say 36 hours, from now is picked from Trello and the BOT keeps on sending nagging reminders to the member till he updates on the task.

Below, you will see the BOT sent an email to the team member for the task - Mock test 3 because it was due within 36 hours.


bot510project@gmail.com
to

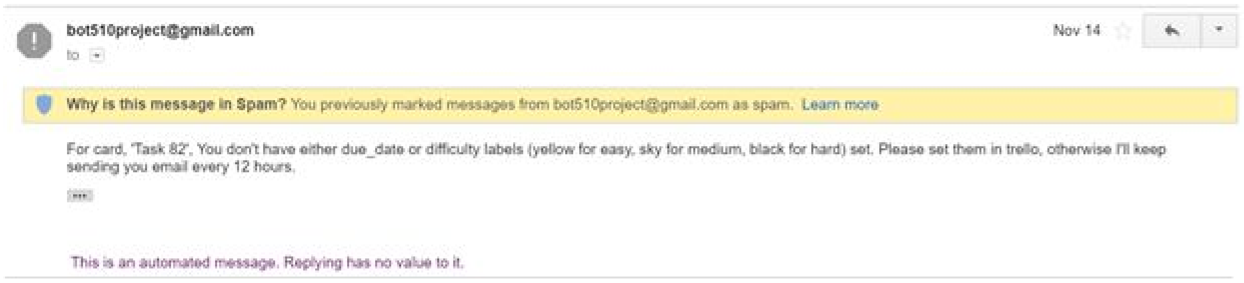
Dec 1 (2 days ago)

Why is this message in Spam? You previously marked messages from bot510project@gmail.com as spam. Learn more

You have a 'Hard' task named 'Mock test 3' pending within 36 hours. You need to finish it ASAP and mark it complete by labelling green or archiving.Also, you can remove the label 'Hard' and change the label to 'Medium' or 'Easy' to stop getting mail for another 12 or 24 hourse respectively. Otherwise I'll keep sending you mails every half an hour. :-)

This is an automated message. Replying has no value to it.

The use case one also takes care of the problem when a task is included into Trello but some informations, like due date or difficulty label, is missed by the user. As you will see :


bot510project@gmail.com
to

Nov 14

Why is this message in Spam? You previously marked messages from bot510project@gmail.com as spam. Learn more

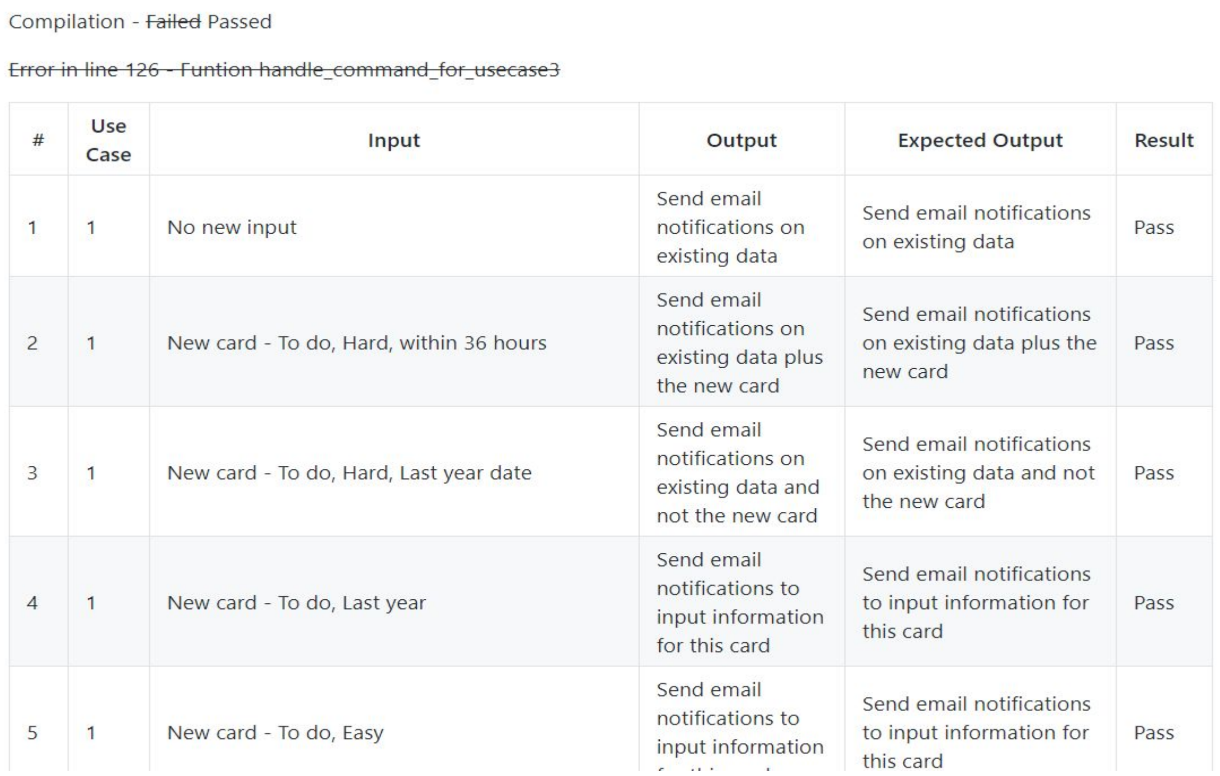
For card, 'Task 82', You don't have either due_date or difficulty labels (yellow for easy, sky for medium, black for hard) set. Please set them in trello, otherwise I'll keep sending you email every 12 hours.

This is an automated message. Replying has no value to it.

For Task 82, the user did not enter the due_date or the difficulty label and hence received an email from the BOT.
Usecase 1 runs on a thread and keeps on checking every time duration about the status and sending nagging reminders till the task is updated.

Acceptance Testing - Exploratory Testing and Code Inspection
As a part of deployment milestone, we were required to pass the acceptance testing and check for all the edge cases that might break the code. Also, we were required to make sure that there is no hard coding as the TAs will perform testing on self-defined inputs. Considering all this in mind, we deployed our complete application in the virtual environment so that we do not need to demo the project to the TAs manually. To avoid, a manual restart of the BOT in case of the crash, we also used the automatic process restart tool such as forever. The BOT is up and running as per the instructions and can be tested any time without any assistance what so ever from the team.
As for the Exploratory Testing and Code Inspection, we made sure that nothing was hard coded. We creatively thought all the test cases where the code can break and actually out of the 22 test cases, the BOT passed the 20 which was very impressive. Though our BPT broke on 2 test cases and crashed in one of them, we fixed the problem promptly on time and corrected the minor compilations errors as well.

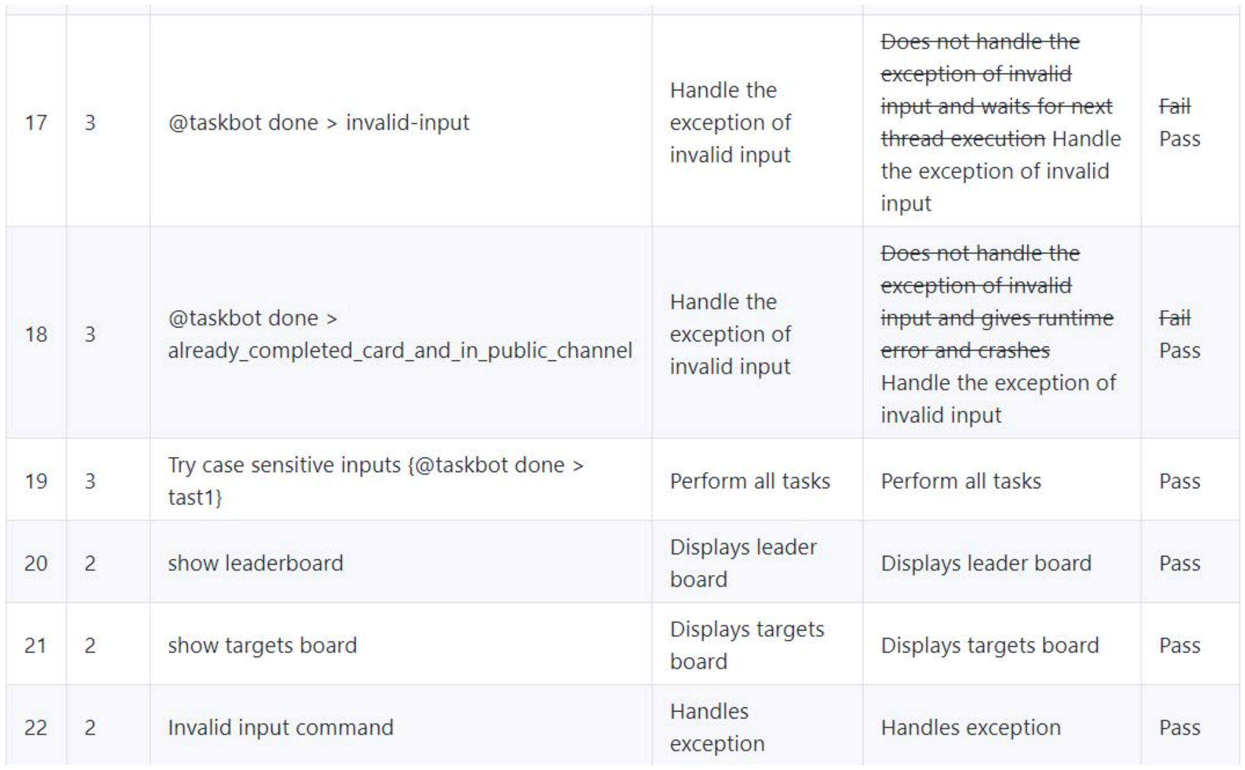
Below, you will find few of the edge cases we tried on our BOT. Test case 17 and 18 failed, other than that everything passed. So, we worked on the two test cases and corrected the logic in time.



Compilation - ~~Failed~~ Passed

~~Error in line 126 - Funtion handle_command_for_usecase3~~

#	Use Case	Input	Output	Expected Output	Result
1	1	No new input	Send email notifications on existing data	Send email notifications on existing data	Pass
2	1	New card - To do, Hard, within 36 hours	Send email notifications on existing data plus the new card	Send email notifications on existing data plus the new card	Pass
3	1	New card - To do, Hard, Last year date	Send email notifications on existing data and not the new card	Send email notifications on existing data and not the new card	Pass
4	1	New card - To do, Last year	Send email notifications to input information for this card	Send email notifications to input information for this card	Pass
5	1	New card - To do, Easy	Send email notifications to input information	Send email notifications to input information for this card	Pass



17	3	@taskbot done > invalid-input	Handle the exception of invalid input	~~Does not handle the exception of invalid input and waits for next thread execution~~ Handle the exception of invalid input	~~Fail~~ Pass
18	3	@taskbot done > already_completed_card_and_in_public_channel	Handle the exception of invalid input	~~Does not handle the exception of invalid input and gives runtime error and crashes~~ Handle the exception of invalid input	~~Fail~~ Pass
19	3	Try case sensitive inputs {@taskbot done > tast1}	Perform all tasks	Perform all tasks	Pass
20	2	show leaderboard	Displays leader board	Displays leader board	Pass
21	2	show targets board	Displays targets board	Displays targets board	Pass
22	2	Invalid input command	Handles exception	Handles exception	Pass

For use case 1, we changed the tasks in Trello with different inputs like a user giving multiple labels to one task or when the due date is not given. We even checked by changing the time in a thread. All the test cases passed with the BOT. For use case 2, the user input is required. Which means in case of a wrong input, the BOT should handle the invalid input and not break. Also, in case of correct input, it should behave as expected. Here also, all the test cases were passed by the BOT. For use case 3, all the test cases passed but 2. We were not handling the invalid input by the user and also when the user was giving card which is already_completed_card_and_in_public_channel as input, the BOT crashed. So, we fixed the two cases and now the BOT is handling all the edge cases.

Reflection on Development Process and Project

- **Reflection from the Point of View of a Developer**

Design

We have two phases at this stage. One is project design, which means we need to decide what we want for this bot, what functions and services should our bot have. The other one is how we design our underlying software architecture. For project design, we have some meetings to talk about it. And we really did a good job on this part. We start from a basic idea -- task manager, then we extend its functions by exploring current functions in Trello. Next, we analyze what we can do better by collaborate Trello and Slack. And finally, we decided to enable the primary features above. For the second part, software architecture, we designed essential components of our bot and how they interact with each other. In summary, we truly did an excellent job in this phase.

Bot

At this stage, we implement basic components and set up infrastructures for our bot. But here, at real implementation, there still more things to do. Firstly, we do not clearly decouple modules from every use case. In other words, different use cases may reimplement lots of code. In fact, they could share some common modules to avoid duplicated code. Secondly, we didn't design good interfaces and templates for these 3 use cases. The consequence is that it is hard for the developer of one use case to make the contribution to other use cases. In the future, when we start to implement a new project, it is important to take some time to think about how to keep the code simple and easy to comprehend.

Selenium testing

In this preliminary stage of development, while we haven't collected any data from real-user, mocking strategies are applied to help us test our application beforehand. We created a "Test Board" in Trello as our fake board, creating cards and then import JSON data from the platform to form our mock.json mocking data. We didn't use the mocking data generation API like nock to generate mocking object, since we have an easy access to the mocking data that more resemble the real data from Trello.

We conducted Selenium testing based on Chrome to test our four functionalities and fixed the JUnit testing orders. The four JUnit tests we have conducted are "sendNaggingReminder", "testEmailSent", "performanceEvaluation", and "reminderBuddy".

Test Name	**Expected Results**	**Selenium Case Pass Criteria**
sendNaggingReminder	When user input "@firsttest usecase 1",three new lines of message "sheikhnasifimtiaz is asked to complete example task 4" will be received.	The last line of message contain the follow content "sheikhnasifimtiaz is asked to complete example task 4". We are expecting three new line of message, this message will be sent in order, therefore if we receive the last of these three line of message, it will be true that we receive all three line of message.

testEmailSent	Gmail "sent" tabs have three emails sent to three different users: simtiaz, vgupta8 and xfu7 in the latest time.	Since Slack cannot clear chat history, the previous testing result will affect the current test, therefore, we get the latest message by matching the bot response time and the email sending time.
		Therefore, there are two constraints for this case to pass:
		The sent email will contain the bot response time and also the user name.
		The following are our implementation code:
		assertTrue(simtiaz.contains("simtiaz")
		assertTrue(simtiaz.contains(lastTime.toLowerCase()));
		assertTrue(vgupta8.contains("vgupta8"))
		
		assertTrue(vgupta8.contains(lastTime.toLowerCase()))
		assertTrue(xfu7.contains("xfu7"))

		assertTrue(xfu7.contains(lastTime.toLowerCase()))

performanceEvaluation	Leaderboard message will be displayed when user input “@firsttest usecase 2”, the message should contain the following content:
	vinay638: 0
	sheikhnasifimtiaz: 0
	xiaotingfu1: 0
	otto292: 0
	guanxuyu: 0
		If the last message “guanxuyu: 0” is visible, then pass the test.
		String path = "//div[@id='msgs_div']/div[1]/div[2]/ts-message[last()]//span[@class = 'message_body' and text() = 'guanxuyu: 0']";
		WebElement slackbot = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(path)));
		assertNotNull(slackbot);

reminderBuddy	New message called “what is your progress, mate?” will be received when user input “@firsttest usecase 3”	If the last message “what is your progress, mate?” is visible, then pass the test.
		
		String path = "//div[@id='msgs_div']/div[1]/div[2]/ts-message[last()]//span[@class = 'message_body' and text() = 'what is your progress, mate?']";
		WebElement test = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(path)));
		assertNotNull(test);

Table 1. Selenium Test Cases and Case Pass Criteria

Table 1 shows the our Selenium test measurement criteria. For use case 1, we have two test cases: "sendNaggingReminder" and "testEmailSent. For the "sendNaggingReminder" function, we want to test if a user input "@firsttest use case 1" command to slackbot, will “firsttest” bot responds by sending the names of users who have task that is due within 1 day deadline. Selenium API allows us to automatically login to Slack and send the message to our bot. When the new messages match with our expected message it will be a pass. Secondly, in "testEmailSent” function, we want to test will our bot sends an Email to this user’s email address as a reminder. Our test case will check if the email is sent to these people by checking the sender's Gmail box with the timestamp same as the bot response time. For use case 2, the test case is designed for "performanceEvaluation" function. We want to test whether firsttest bot will respond by sending the names of users as long as their performance scores, when user input "@firsttest use case 2" command to slackbot. For use case 3, the test case is designed for "reminderBuddy" function. We want to test whether our bot will send directed message to a user "what is your progress, mate?" when the he/she input "@firsttest use case 3" command to slackbot.

Task Tracking
For the Task Tracking part, we used Trello to track our progress. We set up list for every week and cards for every team member. Tracking progress is good for improving the performance of the whole team. And it is an easy way to evaluate the performance of every team member.

Service
In this milestone, we polished our code and implemented main flows of our service. And then we incorporated everyone's code. Here we meet lots of issues like conflicts between each use case, concurrency issue, etc. Those issues caused a lot of trouble and affect our efficiency a lot. After that, we need to think the reasons for those problems and consider how to avoid that situation next time. We did not have a design review before the implementation of our service. So, everyone, had own understand how the bot looks like and how to implement the bot. Thus, this will affect the usage of our resources. That's where the conflicts come from. The solution will be using a lock to protect shared resources and reduce race condition. Then we could maintain concurrency for multi-threading. Another main issue is the namespace issue. At first, we just simply collect everyone's code and put it in one file. But we got lots of name conflicts and running time errors. To solve that, we make everyone's code as a module and call the use case service from the corresponding module. And also, we should think about using design patterns to help us organizing code.

Deploy
Automatic deployment is an increasingly popular feature of development where the developer can migrate their system configuration to a different machine or even different operating system. We implement our migratable deployment feature using Ansible, which is a configuration management tool to help automatic deployment. With Ansible and a deployment script, we are able to design a list of tasks to be performed on a set of client machine while we running the script on the server machine. Our deployment scripts can do the following tasks, set environment variables in the target machine, install project-related packages, verify the version of each packages, clone our project github repo (our repo is a private one, which increases the difficulty of deployment, we achieve it by generate a github ssh key and save it in our server, after the server connect with the client, it distributes the key to the client and the client is allowed to authenticate and clone from our private GitHub repository), and finally run our bot continuously.
Our deployment code allows our running bot to have the following features:
Crash recoverable

Whenever there is a system on the client machine, our bot will stop running. However, we set up multiple machines, that if one has gone through a system update that will shut down the machine, the other one will start the bot immediately. Also, we implement “forever” package, wherein a single machine that a fatal bug cause our bot to stop, forever service will start the bot again. Also, since we are remotely controlling our system using SSH, whenever we lose connection to the server, the ansible script will stop running. To solve this problem, we utilize one ubuntu command called “tmux” which allow us to run the deployment script continuously.

Adapt to change in Github repository

Also since we are running the deployment script continuously, our deployment script will always be able to adapt to new changes in our master repository.

Reflection from the Point of View of a Tester

As a developer of the bot and also the first-time user, I experience the mental and physical effect that our bot has on me. For example, one morning, when our team leader in Trello assigned a new task for me, I start receives email “Xiaoting, you have one Hard Task: Use case 3 Debug due within 2 days”, even if I try to ignore it, it turns out to be impossible since if I take no action it will keep sending me email every other hour. Then I opened my Slack Channel to talk to my teammate, but our bot is also in our channel, it knows that I have a task unfinished and send me a direct message “@Xiaoting, you have 1 task: Use case 3 Debug pending, please respond with YOUR STATUS and TASK NAME, …” Since I haven’t finished it, I respond, “not yet”, then taskbot send a message to our general channel saying that “..., please work harder”. My team mate saw it and he asked me if I have any difficulty in finishing this task, I even not start it! Since I just simply forget it. Our bot is very helpful in reminding me of my task and when is the due date, since the deadline is always the first productivity force.

Limitations and Future Work

In this project, we implement a task reminder bot with its functionalities tested with the developer team. Although we have completed what we have proposed to complete in our design use cases, there are still limitations and a lot to improve in the future.

Since this is a semester-long project, the project is still limited to a development stage, it requires more users to test and report their experience. Also, there are many functionalities we want to achieve. The following parts are functional we want to improve in the future:

1) **Assumptions made on the use pattern of “Trello” :** As we built our “task tracker” bot on top of “Trello”, we had to deal with conflicts between what features an ideal task manager should have and what “Trello” offers. For example, Trello does not have any explicit way to input difficulty label for any card(task). Also, there isn’t any progress bar

for each card which we could have used for calculating points on the leaderboard. Although “Trello” has a list of “Labels” available in different colors for each card which users can use in their own customized way. We made use of this feature and assumed a meaning for each color of label. However, if any team wants to use our bot, they would have to adhere to our defined use pattern after reading the documentation, which is certainly not the ideal case.
We need to address this issue if want our bot to be used by general public. We shouldn’t enforce any change in the general use pattern that “Trello” offers and add our features in a way that it would be intuitive to the user, and they will not be required to go through a written documentation.

2) **Name matching between Slack and Trello :** To match a member’s profile both in ‘Slack’ and “Trello’, we used the name matching method with their “full name” in both Slack and Trello. The full name of a member need not be “exactly” same on both the platforms, but should be acceptable “closer”. (e.g. the bot successfully matches “Sheikh Nasif Imtiaz” in slack and “Nasif Imtiaz Ohi” in Trello).

   However, people can have same names and it presents a big problem. Although Slack members can change their mail id at any time, the ideal solution would have been to add an additional “email id checking” to remove any confusion. But the Trello token we had does not give us the permission to see email id of all members. We need to contact “Trello” authority to solve this issue in future.

3) **Message Interface in Slack:** Slack offers us an API to post a message in more fashionable ways than just raw texts. We can use those features to make our bot posts messages in more exciting ways. Also, we can use dropdowns and other features to make users’ life easier when they inform the bot about their progress.

4) **Trello Task Completion Judgement:** Currently, we are implementing task completion status using Trello label. However, it is not how Trello designed to judge whether a task is finished. Each Trello card with due time has a checkbox on its left to determine whether the task is finished on time. We want to remove the task status label and use Trello built-in status checkbox to indicate the status of a task.

5) **Actual Users Testing:** Due to time limitation, we could not test our bot with actual users. If time allowed, we plan to involve users in our development process, constantly ask their opinion and make improvement based on that. An actual user testing will elicit more bugs that are beyond developer’s knowledge, which will help us build a more robust system.

6) **Comprehensive Command Sets and User Help Manual:** We can make the bot more conversational using the popular APIs and can include a better list of commands for the bot. We also need to write down a "user help manual" as the users at least need to know the basic commands and constraints that our bot has. For example, if a user inputs a command not listed in our command set, we will respond with a help usage page, indicating all the available commands and the example of how to use them.
7) **Formatted Response:** Currently, our bot is only in charge of sending and responding to a certain message. However, it will be more interesting if our board has a memory, and can treat each team member differently based on their past behavior, for example, if someone did a bad job our bot will send pacify and also motivational message to encourage this member, or if someone did a good job last week, our bot will remind him of his good history and motivate him to keep working hard this week.

8) **Intelligent task assignment (where the bot take the role of a manager):** This is something we plan to do in the **"Future"** where the bot can evolve as a project manager from only a "tracker". If we can ensure the users give more fine-grained information about a task and our bot will have known the skillset of each member in the team, when a new task comes on the board, our bot can automatically decide which member is best fit to do this job based on his/her own skill set, free time and the task requirements. We can also add artificial intelligence to the add so that our bot grows to understand about the team over time, and gets better at managing it.

9) **Use Sentiment Analysis to read user input -** We can add the functionality of analyzing the update of the team member on the status of his/her task. We can use Naive-Bayes algorithm, Logistic regression and doc2vec with Apache Kafka and Spark. This way we can read any type of inputs the user might have and this will scale the BOT exponentially.