MILESTONE: DESIGN

Xiaoting Fu, Vinay Gupta, Nasif Imtiaz, Yu-Ching Hu, Guanxu Yu

Problem Statement

Efficient task management for a team, or even for a person, is one of the major challenges in software engineering. There are many platforms (e.g. trello, github) which help in this cause by allowing users to maintain lists of their “To do” tasks alongside with necessary information. But sometimes an engaging interaction may be helpful for the user to really help him/her follow up their work. For example, a person may write down his “To do” list but in real can have no impetus to further checking it in due time. But if there were an entity like a real life person who would constantly and efficiently track the progress and notify the user working as a smart reminder, more can be gained.

Also from a team point of view, often an independent entity is required to continuously evaluate each member’s performance and intelligently assign new tasks to the suitable person based on his/her skill set and previous workload. Many artificial solutions are already there, but making them as interactive and engaging as possible still remains a problem in the domain.

Bot Description

Although efficiently performing these jobs can directly contribute to the eventual performance gain, the uninteresting nature of them makes it difficult to attract people to perform them. Moreover, the repetitive and rule-based logic of such tasks makes it possible for automation, thus saving a lot of resources in the process. Thus a bot could be the best solution to address jobs mentioned in the problem statement having a constant presence in those task managing platforms just like a real-life person.

Our bot does not have a conversation in the typical meaning with the user, but it would ask the users for different inputs from time to time, store them in its memory, and send reminders/suggestions/motivations in a message-like manner. The best fit for the bot could be the “Space Responder” category.

Use Cases

[1] Send Smart Reminder

1. **Precondition:** User must give the due date and optionally expected hours for completion when adding a new task to the list. The user optionally is expected to update the progress of the task also.
2. **Main Flow:** Bot will track the timeline of the task and progress [S1]. If it's necessary, it will send reminders [S2]. It can continuously send reminders until the user takes action [S3].
3. **Subflows:**
   S1: Bot would have a logic system to calculate checkpoints. (e.g., 2 days before the due date if the task has 6 hours of equivalent work remained incomplete)
   S2: It will send notifications to the user. (e.g. send an email to user)
   S3: It will wait for a certain amount of time for the user's response. In case of no response, it will keep sending reminders in a loop until the due date is reached.
4. **Alternative flow:**
   A1: If the user hasn't added a due date and other information for a task in the list, it will continue asking those information (via email) at certain intervals.

[2] Evaluate Team members' Performance

1. **Precondition:** Users must add their tasks in the list with its due date, completion date, prediction time and required number of hours for completion of the task in the platform.
2. **Main Flow:** At certain intervals[S1], the bot will evaluate each member's' performance based on a rule-based logic system[S2], and send appropriate messages from a list hard coded in its memory.
3. **Subflows:**
   S1: It will keep track of time and will get activated after certain intervals.
   S2: It will evaluated all members' performance based on current information available according to its logic system and select appropriate message for all of them.
   S3: It will send (via mail) them to the members.
4. **Alternative flow:**
   A1: In absence of adequate information, it will select generic motivating messages for the users.

[3] Intelligently Assign Tasks

1. **Precondition:** All members will always have their updated skillsets in the platform. When adding a task, the manager must give input the required skillsets for the task.
2. **Main Flow:** The bot will read skillsets[S1] and current workloads of the members[S2] from the platform and its own memory, match them based on a rule-based logic system[S3] and assign the task to one/more members.[S4]
3. **Subflows:**
   S1: The bot reads skillsets from the platform.
   S2: The bot predict free work hours available for each member based on the current information.
   S3: Run these data into a simple logic based function and outputs results assigning one/more members to the task specifying the no. of hours assigned to them.
   S4: Send them mail notifying the assignment. (sending them repeatedly at certain periods unless a response is received)
4. **Alternative flow:**
   A1: If too many information is missing, It doesn't produce results, and notify project manager to handle the job himself. (deflecting duty)

Design Sketches

Wireframe

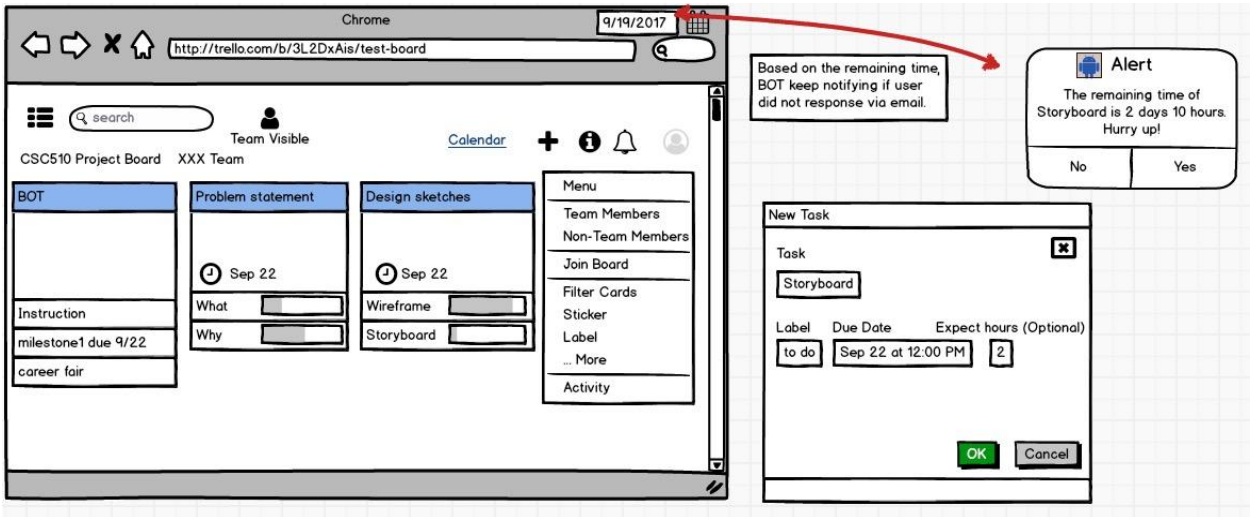

Figure 1. Send Smart Reminder

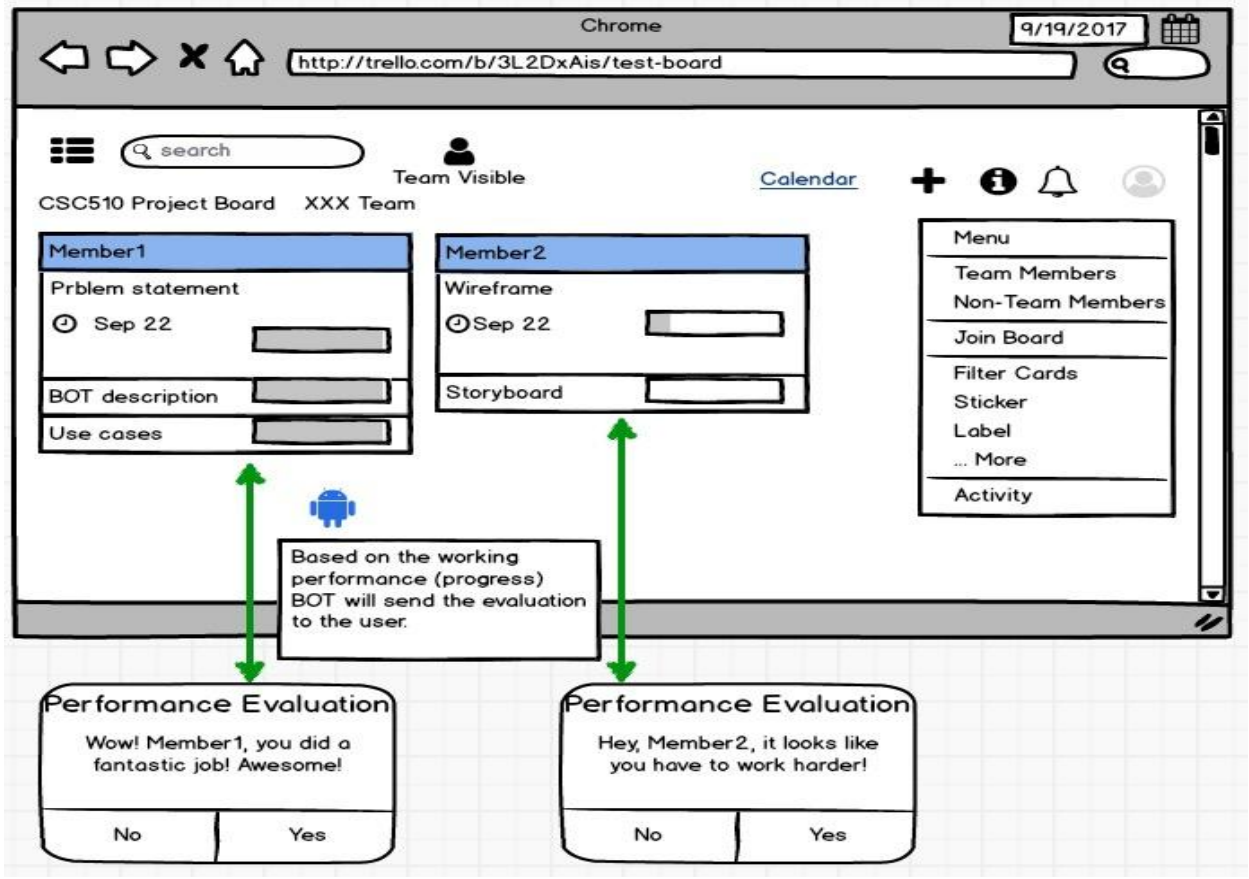

Figure 2. Evaluate Team members' Performance

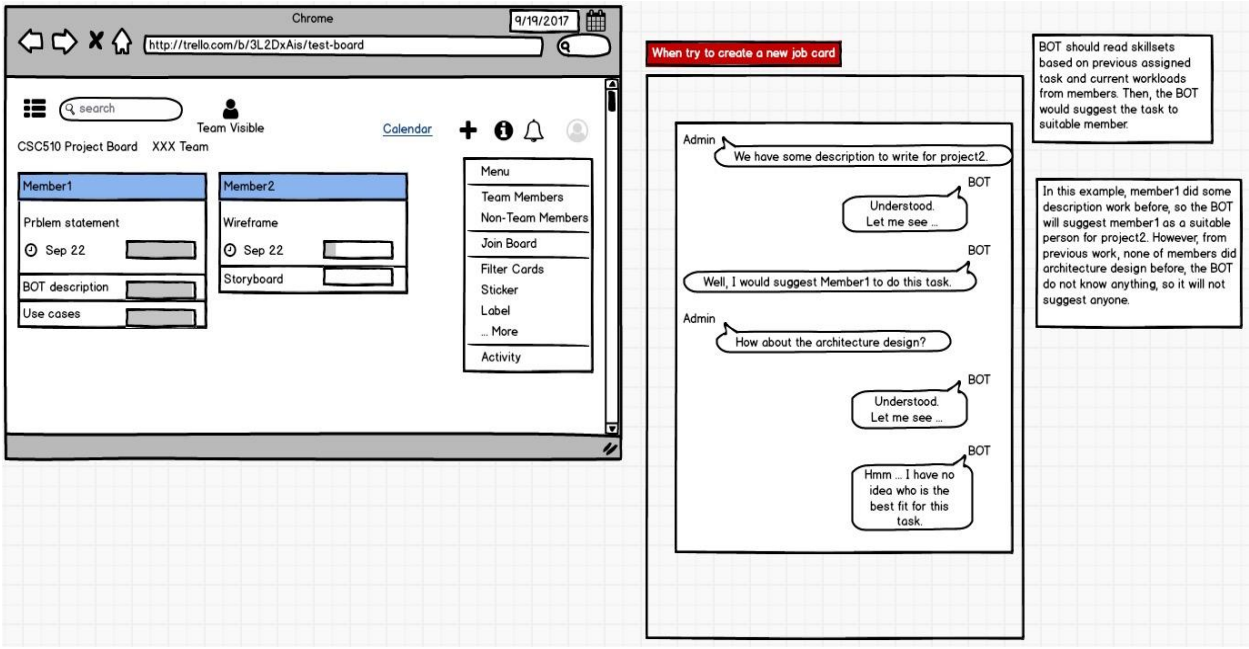

Figure 3. Intelligently Assign Tasks

Storyboard


Requirements submitted.
1
Tasks assigned
Jasmine : Code
Perer : Test
Neil : Deploy
2
BOT is telling me that I am running late on my test.
You better start fast. The PM will get the notification in the afternoon.
3
You have cleared two checkpoints. Three checkpoints remaining and 2 days left.
4
Project completed. Performance report available.
Perer - Late
Jasmine - Code broke
5
I want better performance next time.
6

The Project Manager will input all the required information about the project and the team into our BOT. Then, the BOT will intelligently find out the best match for skills of the team members and requirements of the project. Once the PM has the matched result, (s)he can assign the tasks to the team members. Then, the BOT will keep a track of the progress and anticipated soft deadlines. It will also send notifications to the team members in case there is a delay in work. At last, when the project is completed, the BOT will intelligently analyze the performance of the members and report it to the PM. The Project Manager will input all the required information about the project and the team into our BOT. Then, the BOT will intelligently find out the best match for skills of the team members and requirements of the project. Once the PM has the matched result, (s)he can assign the tasks to the team members. Then, the BOT will keep a track of the progress and anticipated soft deadlines. It will also send notifications to the team members in case there is a delay in work. At last, when the project is completed, the BOT will intelligently analyze the performance of the members and report it to the PM.

Architecture Design

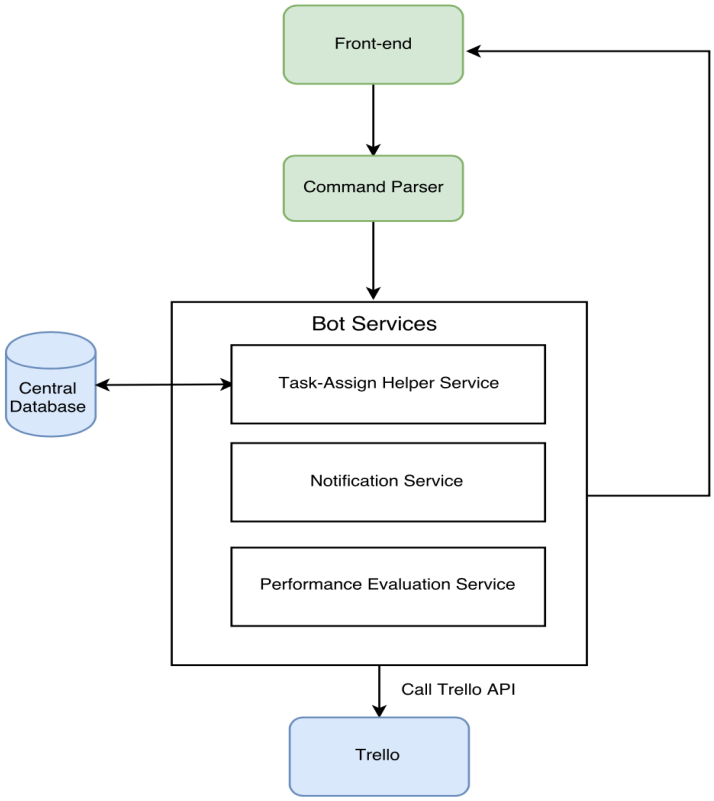

Figure 4. Architecture Design Diagram

Trello, the most popular task management platform, allow team members to cooperate and manage their tasks. The goal of our application is to generate a Bot that resides in Trello, which can help programmers collaborate and also ease the burden of project manager on assigning tasks to team members. Figure 1 depicts the general architecture of our Bot. Since the front-end design is already achieved by Trello, we will mainly focus on the data and control flow. The data flow begins with a user's interaction with the front-end, with user's input detected by the bot, the input is parsed to be a command which will trigger one of the bot services. When a bot service is triggered, it calls Trello API to retrieve data from Trello and also the Central Database.

Command Parser

This component handles the input statements, recognizes what the user want to do. Then pass the user's intention to the bot. So the bot can understand what the user wants to do and then call the certain version.

***Notification Servic*e**

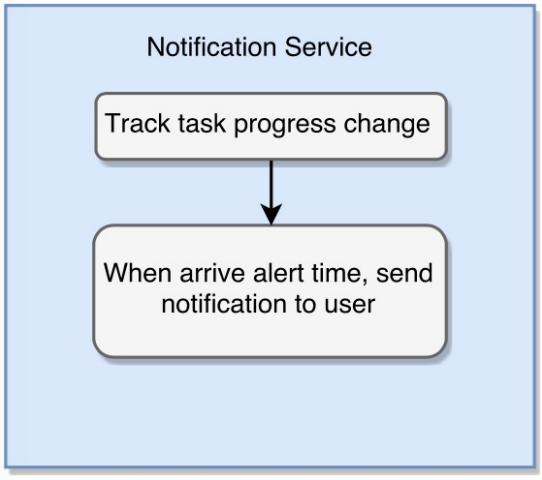

Figure 5. User-Interaction Diagram for Notification Service

The Notification Service is corresponding to Use Case 1. In our design, we create independent service for these three use cases. The notification service will be running all the time, tracking user's task progress change. When the alert time is reached but the user still not finish the task, we will send a notification to the user, inform him to start working with the task.

Performance Evaluation Service

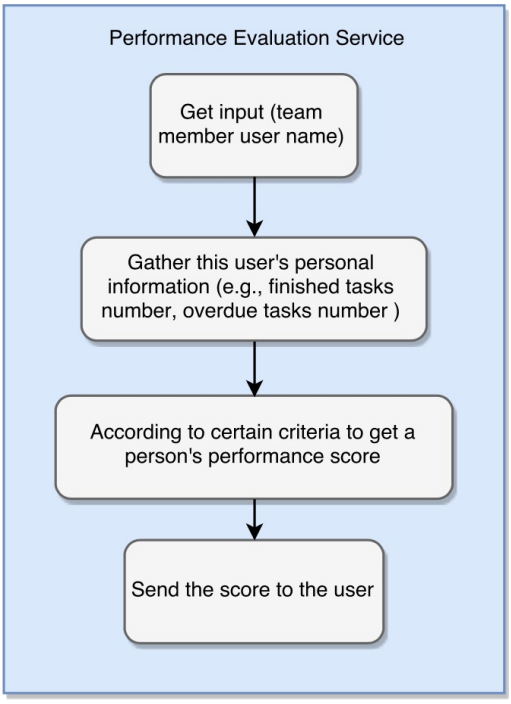

Figure 6. User-Interaction Diagram for Performance Evaluation Service

In this service, first, the bot get the team member's Trello user name from the parser. Then, it will use the Trello API to gather relevant information to generate a score for this person. This score could reflect the person's performance. Relevant personal information may include the total number of finished tasks at each difficulty level (easy, medium, hard) during a certain period, the total number of overdue tasks during a certain period, working efficiency, etc. Then the bot will push this score to the user.

Task Assignment Helper Service

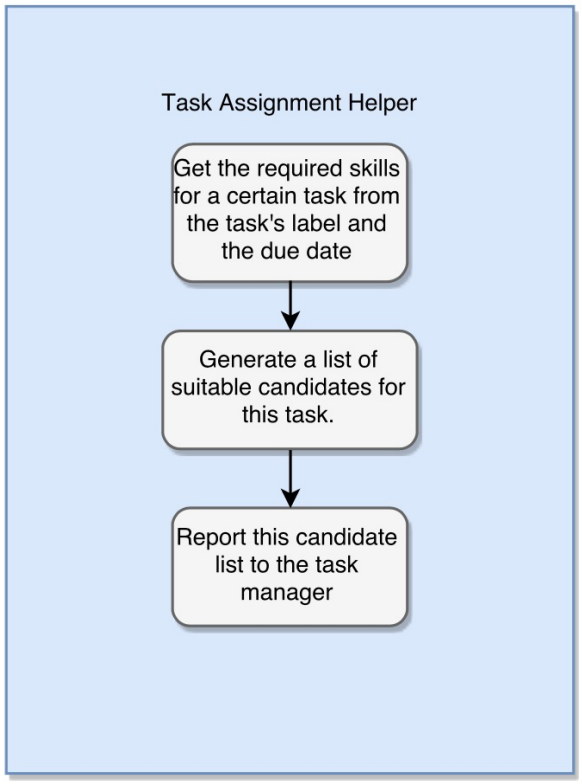

Figure 7. User-Interaction Diagram for Task Assignment Helper Service

In this service, first, the bot gets the task name/number from the parser. Then it will call Trello API to get required skills and difficulty level from the task label. Then the bot looks up this central service database to get team members skills sets and their available time to generate a recommendation list and push this list to the team manager. Then, with the help of this recommendation list and the team manager's personal preference, he/she could easily assign this new task to someone in the team.

Constraints

Only the team manager could use the Task Assignment Helper Service.

Only the team manager can use Performance Evaluation Service to check all team member's performance evaluation score.

A team member(except team manager) has no right to check other's performance evaluation score. He/She can only check own performance evaluation score.

A team member can not receive other people's alert notification. The alert notification can only be sent to the task owner.

Design patterns

Singleton pattern: We use this pattern to create a client. There are two types of clients in our application. One is used to call Trello APIs, and the other is used to connect to our central database. To lower the memory usage, we only create one client for each of these two types.

Builder pattern

For each client, we use builder patterns to build a request. So we can easily to customize every request. Then the client can send the request.