KarmaTracker: Slack Reward Bot

Final Report

Morgan Pham
morganpham3300@vt.edu

Aiden Barker
aidenbarker@vt.edu

Eunice Hong
eunice2713@vt.edu

Vatsa Bhatt
vb24@vt.edu

ABSTRACT

While software engineering has grown leaps and bounds since its beginnings, there are many problems that plague the field as described by the Software Crisis. A significant piece to the problem is the development of low-quality code. The issue described in the Software Crisis is that machines have become so powerful that we no longer understand how to keep up with it. To attempt to address this growing problem, the proposed solution is to recognize and reward employees for performing actions that work to solve the low code quality problem. For teams that use Slack, this Slack bot named "KarmaTracker" would keep track of each employee's good karma points which are appointed by other employees recognizing these good deeds. A dedicated Slack channel would be created to name the top "Karma Point" climbers each week, giving the space to acknowledge and celebrate these high performers. Management should be encouraged to connect employees' KarmaTracker points to their performance metrics and performance reviews to get a clearer picture of who is working to improve the work of others and therefore, the overall company's.

INTRODUCTION

Whether this low quality is deemed through inefficient code, code failing to meet requirements, or code that is poorly maintained, this problem of quality deterioration must be addressed for the field to continue growing in an upwards direction. Many of these causes can be ameliorated with simple actions such as setting up quick calls to pair program or perform debugging sessions rather than communicating back and forth over Slack, completing a code review for a peer without being instructed to, answering questions about your team's code outside of your own team channel, etc. A strong lack of communication between one's own team as well as cross-teams has contributed to the problem of low-quality code, as some are concerned that spending time helping others won't be acknowledged and will only look like less time spent completing their own work. This kind of collaboration needs to be encouraged, which can be done by having a system to recognize and reward those that take the time to help their peers. Because KarmaTracker stores each employee's points, employees can have a specific place to point to when discussing performance with their managers or whoever else may be concerned, to show that they are strong team players willing to help the overall success of the company.

EXAMPLE OF USEFULNESS

Engineers may not wish to help their teammates complete their tasks if it means that they will have less time to complete their own assigned tasks. This is because there is not always enough value placed upon teamwork and collaboration efforts. With KarmaTracker, managers will acquire a better understanding of how engineers spend time assisting their teammates in their tasks. An engineer that operates on their own and does not work towards the success of the team is not a superstar; an engineer that helps around the team to achieve their mission is.

Many companies have their managers complete evaluations on their engineers to rate their performance. Many different metrics are used, with some companies relying on numerical metrics that don't tell a complete story, such as number of Git commits and lines of code written. Adding in metrics from KarmaTracker will assist in providing a more holistic understanding of an engineer's performance and will allow them to feel more secure in spending time helping their teammates.

It's possible that an engineer may be blocked on their own task because they're waiting on a teammate to complete their part first. This engineer could work on a different assigned ticket, or they could assist their teammate in a debugging session instead. The first option would benefit solely the engineer, but the second option would benefit the team considering it would allow two tickets to get completed rather than only one. KarmaTracker would show this thoughtfulness and would motivate teams to assist each other more often without worrying that others think they are slacking.

BACKGROUND

The idea of karma is oftentimes explained as, "what goes around comes around." It is essentially a cause-and-effect principle that can be applied to all actions of life. Here, karma can be seen as engineers assisting others (a task that isn't necessarily required) can and should lead to positive outcomes.

Another point to be clarified is the idea of tickets, used in the above section. Tickets are tasks that are assigned within project-management systems, such as Agile within sprints. Each individual normally gets assigned their own tickets and are responsible for their progress and completion. A popular metric in work evaluation is viewing the number of tickets completed within a sprint, as well as how many points these tickets were assigned. Points represent the difficulty or effort associated with this particular ticket.

RELATED WORK

GitHub Stars and Contributions is relevant to KarmaTracker as they demonstrate the effectiveness of recognition and reward systems in promoting collaboration and code quality improvement within the software engineering community. These GitHub features serve as a great example for promoting good behaviors and interactions among team members in the context of KarmaTracker. Similarly, to how GitHub users may "star" repositories and "follow" developers to recognize and monitor their contributions, KarmaTracker allows users to give points to peers who do actions that improve software quality, inspire cooperation, or facilitate cross-team communication. This recognition system matches with the purpose of your project, which is to promote collaboration and communication among software developers.

Furthermore, KarmaTracker's user recognition and leaderboard functions are inspired by GitHub's contribution history, which showcases a developer's involvement and skill through their pull requests, problems, and commits. KarmaTracker encourages good contributions by recording and presenting users' points and achievements. It also allows others to evaluate their colleagues' engagement and talents. Furthermore, GitHub's involvement in supporting code reviews and issue tracking demonstrates how better communication and cooperation may result in higher-quality code—a premise that is closely aligned with the project's goals.

As this project progresses, we can draw on GitHub's success in promoting collaboration, code quality, and recognition to inform the design and implementation of KarmaTracker. By adapting and integrating these concepts, KarmaTracker can effectively encourage better communication and teamwork among software engineers, thereby enhancing the efficiency and quality of software development processes.

SOFTWARE ENGINEERING PROCESS

Our team chose an Agile software engineering approach, namely Scrum, for the KarmaTracker project. This decision is motivated by the project's aims of encouraging cooperation and communication among software developers since Scrum's iterative and adaptive approach fits in perfectly with these goals. We want to build KarmaTracker in stages, starting with essential capabilities like point tracking and user recognition. This tiered approach allows us to present consumers with a working bot sooner, allowing them to reap its primary benefits. Furthermore, it allows us to implement additional features in later sprints based on user input and altering priorities, in accordance with Agile principles that stress delivering incremental value and reacting to changing requirements. Flexible conditions like this allow us to iterate back to previous stages and continue improving features as needed.
Following development, we will emphasize deployment and monitoring, ensuring that KarmaTracker is easily accessible from our Slack workplace while also providing monitoring and error tracking systems to ensure seamless functioning. By doing sprint retrospectives at the conclusion of each development cycle, our team will be able to reflect on what went well and what may be improved in our development process, allowing for ongoing progress. Finally, we will maintain an active user feedback loop throughout development, aggressively seeking and incorporating user insights and data-driven choices to align the bot with user expectations and improve its performance. We want to provide a robust, user centric KarmaTracker by adhering to our Agile process, which successfully handles the difficulties of low-quality code and fosters improved communication between our software engineering teams.

HIGH-LEVEL DESIGN DECISIONS

The decision that our team made regarding high-level design, was to follow the Layered Architecture pattern. Due to being able to easily separate the functionalities of each aspect of KarmaTracker, this pattern was a clear choice. In the current conceived state, the layers would consist of the user interface, business logic, and the integration layer. The user interface layer for example, manages the interactions and processes the commands. The business logic layer would keep track of points, and the integration layer would manage external services for the system. Following this modular design that is inherent to the Layered Architecture pattern. This would lead to a system that has good scalability, which enables new features to be implemented without disturbing the rest of the system. This would make KarmaTracker easy to maintain and flexible to adapt to changing needs in the future.

IMPLEMENTATION DESIGN PROCESS

By opting for a continuous integration (CI) design process, our team aims to streamline the implementation of KarmaTracker. This approach involves integrating code changes into the project repository frequently and automating the build and testing processes. The CI pipeline, combined with the Layered Architecture pattern, ensures that each incremental change is quickly incorporated into the system. As our team makes changes, automated tests are executed to promptly identify and address potential issues, maintaining code quality and functionality. The synergy between CI and the Layered Architecture pattern enhances development efficiency, allowing for a faster and more reliable implementation of KarmaTracker. The frequent integration of code changes ensures that the evolving system remains stable and ready for further enhancements, aligning with the project's modular and scalable design principles.

TESTING APPROACH

The testing approach that we thought would be best used alongside KarmaTracker would be Unit Testing. We chose Unit Testing as since it focuses on testing the individual components, unit tests can be easily split up accordingly. Testing small portions of code to ensure they work as intended. This approach will help prevent major bugs from arising as knowing that each smaller function is operating as expected. This would also allow us to detect bugs earlier during development, as they would be easier to find when testing small portions of code. Also, as unit tests are isolated from one another, this ensures that each layer of the system is working as expected, which works well with our decision to follow a Layered Architecture. Ensuring proper functionality at each layer is also crucial to validate that the system is communicating amongst layers properly so that each layer is inputting and outputting the expected outcomes.

DEPLOYMENT PLAN

For the deployment of KarmaTracker, the process involves several key steps to seamlessly integrate the bot into our software development environment. The initial phase focuses on accessing the Slack workspace, where a dedicated channel "#karmatracker" is created for notifications and updates. All team members are invited to this channel, setting the foundation for effective communication. The next step involves configuring KarmaTracker settings, including the installation of the bot and the registration of team members to commence tracking their karma points.

Moving on, a formal announcement is made on a general channel, giving the introduction of KarmaTracker and its purpose. Additionally, a training session or documentation is provided to guide team members on utilizing KarmaTracker effectively. The next step involves monitoring and support, where tools are implemented to track KarmaTracker's performance, and a designated channel is assigned for team members to seek assistance or report issues. We'll also connect KarmaTracker points to performance metrics for team reviews, establishing a loop for ongoing improvement.

Post-deployment, continuous monitoring tools are implemented to track KarmaTracker's performance, user engagement, and potential issues. Regular checks ensure that the system operates smoothly and that users experience minimal disruptions. Additionally, a dedicated support channel is established where team members can seek assistance, ask questions, and report any challenges they encounter. This support mechanism not only provides a solution-oriented approach but also fosters a sense of community engagement, encouraging users to actively participate in the successful implementation of KarmaTracker. This proactive monitoring and support framework aims to address concerns promptly, guaranteeing a positive and effective user experience throughout the deployment and beyond.

Finally, this deployment plan aims to smoothly add KarmaTracker to our software development. It focuses on clear communication, straightforward training, monitoring, and connecting performance metrics. This will create a collaborative atmosphere while addressing issues related to low-quality code. Ongoing feedback will help KarmaTracker continuously improve our software development processes.

DISCUSSION

In conclusion, KarmaTracker offers a proactive solution to the ongoing problems of low-quality code and communication breakdowns within software development teams. The Software Crisis has brought to light the increasing difficulties of sustaining code quality in a fast-evolving technical context. This Slack bot solves this by implementing a system of positive contribution recognition and reward, resulting in a more collaborative and communicative atmosphere. However, it is important to recognize the system's limits.

A few limitations of the KarmaTracker would include subjectivity of recognition as team members may interpret actions differently, potentially leading to disagreements or biases in the points allocation process. There is also the risk of employees engaging in actions solely for the purpose of accumulating points, potentially compromising the authenticity of their contributions. Additionally, the introduction of a leaderboard and frequent notifications may present a risk of distracting team members from their primary tasks, diverting their attention from the actual goals of the software development process.

Future implementations could further refine and enhance KarmaTracker. Integrating machine learning algorithms for smart recognition would reduce subjectivity by analyzing historical data and patterns. This would allow the system to recognize and reward positive behaviors without explicit commands, significantly improving efficiency. Another valuable addition could be the implementation of anonymous recognition, fostering a more inclusive environment where team members feel comfortable acknowledging contributions without concerns about favoritism or biases.

Furthermore, connecting KarmaTracker with project management tools would enable the bot to automatically reward contributions related to task completion and milestones achieved. This integration would provide a more comprehensive view of team members' efforts, aligning recognition with project goals and objectives. In navigating these limitations and embracing future implementations, KarmaTracker stands for continuous improvement, contributing to the creation of a positive, collaborative, and productive software development environment.

REFERENCES

[1] stars.github.com/program/
[2] https://www.webmd.com/balance/what-is-karma