TaskMaster Final Report

Final Evaluation of the Software that Keeps Workers on Task

Maya Makked
Computer Science
Virginia Tech
Blacksburg VA
U.S.
mayam22@vt.edu

Ethan Bourne
Computer Science
Virginia Tech
Blacksburg VA
U.S.
ethanbourne25@vt.edu

Connor Graves
Computer Science
Virginia Tech
Blacksburg VA
U.S.
cwgraves@vt.edu

Brandon Hoang
Computer Science
Virginia Tech
Blacksburg VA
U.S.
brandonh03@vt.edu

ABSTRACT

Software engineers, engineers, and workers in general often have a difficult time staying on task. Whether it is due to distractions, disorganization, or apathy, many project hours are not being spent working on project milestones.

Our team plans to design software called TaskMaster that will help keep software engineers on task. There will be displayed tasks that guide software engineers as well as other helpful features that champion efficiency.

INTRODUCTION

Distractions, which are especially easy for software engineers to encounter because they are on computers for most of their job, can lead to significant inefficiency. Also, software engineers are assigned several tasks and can become distracted by their own multitasking management.

According to Harvard Business Review, the average employee gets distracted about 50 to 60 times a day. Consequently, employees spend little time in "the flow state" of mind, which makes people up to five times more productive[1]. Trying to multitask breaks the flow because the brain must keep shifting mental gears between different tasks. The active decision to switch tasks itself even contributes to inefficiency[2].

To regulate this issue of task distraction, something must help guide software engineers with their tasks. TaskMaster is our software solution to keep software engineers on task, as it will reduce the distraction of task management by taking the responsibility away from the software engineer and promoting "the flow state."

TaskMaster will track task history and assigned tasks on both the individual and team levels. A Pomodoro timer will be integrated into the application, as small breaks promote efficiency. A website blocker will work alongside the Pomodoro timer. TaskMaster will also provide reminders for tasks and deadlines. For workers with higher authority, there will be a feature included that allows them to easily detect if potentially distracted workers are on task. As a special feature, there will be a motivational support button that the worker can click for a motivational quote.

The user interface will consist of sticky notes containing detailed task information that will persist on screen. There will also be a small digital clock icon serving as the Pomodoro timer that times working and breaks. For higher authority workers who suspect that workers are off task, there will be a button on screen that can be clicked. This will pull up a screen with toggle switches for the higher authority worker's assigned workers. When a switch is on, periodic updates will be sent to the higher authority worker on the lower authority worker's computer activity. The higher authority worker can also view the employee's screen and send warnings on red sticky notes.

The non-functional requirements for TaskMaster will be listed. The usability requirement is that text must be visible a meter away from a typical monitor screen. Any colors used must be high contrast and labels must use intuitive icons. For reliability, the software should have at most one failure every 14 days. Functionality and data must be able to be restored within 8 hours. In terms of performance, the program should not take more than 2GB of RAM, and 2GB of storage at any given time, thus not causing a noticeable performance reduction on modern machines. The supportability requirement for TaskMaster is that it should easily be configured and customizable for different project types and teams. For the implementation/constraint requirement, TaskMaster must be able to run on Windows 10/11, MacOS 12 Monterey/13 Ventura, and Linux.

MOTIVATING EXAMPLE

Imagine you have a team of four developers and one manager working for a software consulting company. The scenario involving this team below will illustrate the need for TaskMaster.

The developer Sophia really likes to make sure she is doing her tasks properly, so she spends a lot of time flipping between her work and her task descriptions in another window. Unfortunately, her well-intended effort causes her to waste a lot of time.

The developer Stella, who tends to be assigned a lot of tasks, tries to multitask thinking she will get more done. The problem is her brain does not get much of a chance to stay streamlined, so her multitasking backfires.

The developer Brendan thinks taking breaks is for the weak and rarely takes them unless he absolutely needs to. He does not realize how tired he makes himself. In fact, his lack of breaks takes a toll on his mental health, which causes him to lose motivation sometimes.

The developer Audrey spends a lot of time on distracting websites, which the manager Carlo notices. He tries to keep her on task, but he cannot always monitor her unless he foregoes his own work.

Carlo finds out about TaskMaster and decides to use this for his development team. He quickly notices an increase in the team's productivity because everyone's issues are mitigated by the software. Since each member performs more productively on the individual level, the impact of multiple contributed efficiencies makes itself clear on the team level. The developers on the team can also help each other more because their time is not wasted on distractions.

Sophia no longer finds herself flipping between tabs, as the task she must focus on persists on her screen. Instead of trying to handle task management on her own, Stella follows TaskMaster's guidance on what tasks she should be working on at certain times. Brendan is forced to take breaks because of the Pomodoro timer, and contrary to his expectations, finds the breaks beneficial to his productivity. When he gets unmotivated, he generates a supportive motivational quote. Audrey has a harder time accessing external sites due to the website blocker, and if she is suspected to be off task, Carlo can easily track her activity. The fact that he can view her screen dissuades Audrey from carrying out distracting actions.

BACKGROUND

The concept of a Pomodoro timer may be unfamiliar to some. This type of timer makes use of the Pomodoro time management technique, invented by Francesco Cirillo. How it works is that a worker works in 25-30 minute intervals with two-to-three minute breaks in between. After a certain number of sessions, the worker takes a longer break[4].

The Pomodoro technique is helpful for coding tasks, which can be tiresome. The 25-30 minute intervals are long enough for someone to get substantial work done without leading to exhaustion. If need be, the times of the intervals can be adjusted as long as the Pomodoro concept of doing work in short bursts remains[4]. TaskMaster will allow for this flexibility. Overall, the Pomodoro Technique helps workers feel less overwhelmed by larger tasks.

RELATED WORK

Kanban is an Agile task management concept utilized in task management software. There are three key pillars of Kanban. The first one is visualizing workflow, which is done on Kanban boards. The second pillar is prioritizing work, which is done by limiting work in progress and measuring/managing workflow. The different sections of a Kanban board help with prioritization. The third pillar is using cards that describe tasks to be assigned and completed during an iteration of work.

Scrum is another Agile task management concept. There is a product backlog that contains product features prioritized by the user. The sprint backlog contains features that are to be assigned to a sprint, which is a short period of time dedicated to completing certain tasks. Daily standup meetings are held in which developers share what they did, what obstacles are blocking them, and what they will do.

A lot of task management software exists. Jira, for example, is widely used by software engineers. Jira's main function is to track tickets for tasks. A ticket indicates the status of a task, whether it needs to be done, is in progress, is in testing, or another phase. Jira also makes it easy to divide tasks among a team. Story points are assigned to tickets to indicate their importance and/or how much work they will take to complete. Prioritization of tickets can also easily be done.

GitHub Issues is another mechanism to track tasks. It aims for simplicity. A user can create an issue from a repository, a task list item, a project note, a comment in an issue or a pull request, a particular line of code, or a URL query. A user can also create an issue on several platforms, such as GitHub Desktop and GitHub CLI. Task lists allow for the organization and prioritization of several tasks related to one project[3].

With so much task management software already existing, one might wonder what sets TaskMaster apart. TaskMaster's novel feature is that it is highly active during the user's working activity. Rather than being something to passively reference, TaskMaster actively engages the software engineer. Additionally, TaskMaster has a lot of features that directly serve to focus and motivate the user.

SOFTWARE ENGINEERING PROCESS

For our software engineering process, we are using prototyping because TaskMaster will experience frequent interactions with users, as is the case with TaskMaster's sticky note type interface. We can quickly refine the system to best suit the user without overinvesting our limited resources into each development iteration. Additionally, while we have implemented several set features in TaskMaster, we want the flexibility to change them and add more.

DESIGN

The high-level architectural design pattern for TaskMaster is event-based because such pattern is reactive to real-time events. When an event is triggered by either an input or a change in state, TaskMaster acts. A lot of the system's time is spent idle, waiting for users to interact or for internal timers to trigger. TaskMaster only provides prompts when an event occurs.

A low-level design pattern used in TaskMaster is observer, which is in the behavioral design pattern family. When an employee takes an action such as finishing a task or going on improper websites, the manager will ideally be notified and prompted to interact with the system.

The design process we used for TaskMaster's user interface is storyboarding. We chose a design that users would easily understand. Icons are intuitive, such as the sticky notes for tasks, clock icon for the Pomodoro timer, and toggles for switches that the manager uses for monitoring employees. Colors for certain messages on the sticky notes are intuitive to the types of messages. For example, yellow or light blue indicate a to-do item while red indicates a warning. The labels in TaskMaster, such as "Employee Monitoring" for the monitoring screen and the names of the employees next to the toggles, provide straightforward guidance.

TaskMaster portrays consistency through the sticky note display style of different features. The task sticky notes stay prolonged on the screen so that they can be emphasized to the user for extended periods of time. As a result, the user does not need to click between screens to view the sticky notes, which can be distracting. The manager can easily click a small, constantly-on-screen button to expand the employee monitoring screen, allowing them quicker access to the monitoring toggles. This places more of a feeling of control in the manager. Overall, TaskMaster provides a user-friendly experience.

IMPLEMENTATION AND TESTING

For implementation we chose an object-oriented programming language because the entities of TaskMaster can be represented by objects that simulate the real world. For example, a sticky note is a real-life object. We coded in the Eclipse IDE. For version control, we used GitHub and manually inspected changes through code reviews. To track our issues, we used a Kanban board. We created sturdy documentation so we could easily reference our work and allow users a deeper understanding of TaskMaster.

We carried out feature-driven development, as there are several defining features that we wanted to integrate into the system. These features include the sticky note task management, the employee monitoring, the Pomodoro timer, and the motivational quote generator. We planned and designed around these features and built them with code. We made sure to continuously integrate changes to make the implementation process run smoothly.

For static code analysis, which probes source code without program execution, we used the Lines of Code, Cyclomatic Complexity, and inheritance tree depth code metrics. We also used code clone detection. For dynamic code analysis, we logged our code through method entry instrumentation and used the Eclipse debugger.

For testing we went with Black Box testing. The primary concern for testing TaskMaster is that the important features work properly without the system failing. The exact inner workings of the source code should not be an issue if TaskMaster works as intended. With Black Box testing we were able to focus the testing on those features. This also works well with creating prototypes as the later prototypes were useful for testing.

DEPLOYMENT PLAN

The initial release of TaskMaster would be a basic deployment. With our approaches to the design of TaskMaster we hope to have the core functionality implemented without any notable bugs. There is risk associated with a basic deployment, but TaskMaster should not have any problems with outages and the prototyping will ensure that there does not need to be rollbacks. From there we will take some time to offer support and listen to feedback from the initial release.

The initial focus of maintenance upon deployment will be corrective. If there were any problems with the base features, the main priority will be correcting those. From there the primary focus of maintenance will be perfective to see if there are new features we want to add or improve on existing features. Any adaptive or preventative maintenance would be during this period as well, but most likely further down the line.

From there we will use continuous integration and continuous delivery for faster rollout. For any big releases of future versions, we will use canary deployment. Any big changes to existing features or new features being added can then be received by a portion of the users. This will allow for feedback and an easy reversal if the feedback is negative. Maintenance will work in the same way, with the initial priority corrective fixes, and other types of maintenance following subsequently.

DISCUSSION

A limitation to TaskMaster is that some users may not like that items persist on the screen because they may find it distracting. While this may be the case, users will still save a lot of time by not having to repeatedly switch tabs just to manage tasks. Additionally, when such users may be distracted, they will be distracted by tasks that they need to do instead of irrelevant external sources. As a result, their tasks will be drilled into them more. Other features such as the employee monitoring screen are expandable by clicking on small buttons, so those should not be distracting on-screen.

Another limitation to the project is that some users may think it is a violation of privacy for higher ups to be able to track your activity. Different companies can have different contractual agreements that employees adhere to regarding privacy and decide whether use of the monitoring feature is permitted. Thus, employees cannot file complaints if they signed their company's contract that allows the feature. On the other hand, if a higher up tries to use the feature when it is not contractually allowed, they can be reported. The monitoring feature is configurable to work on designated networks, so it can be limited to the company network.

One possible extension that could be made to the project is that it could become a plugin for other issue management software like Jira. This would allow TaskMaster to be more automated, further increasing its convenience. Not only would making TaskMaster a plugin improve the convenience of TaskMaster, but it would also improve the convenience of the other issue management software. Changes in TaskMaster could automatically be reflected in Jira and vice versa as appropriate.

A feature that could be added to TaskMaster is one that automatically arranges the sticky notes in an optimal way based on the user's workspace. This would save the user time because they would not have to rearrange items on the screen when they switch tabs. TaskMaster could also detect mouse placement and move sticky notes accordingly.

In the future, the scope of TaskMaster's use could broaden. The software's primary purpose is for software development, but it could be integrated into other fields. Particularly, TaskMaster could be useful for students pursuing education because students often struggle with time management. TaskMaster could even become a tool for managing general life tasks.

CONCLUSION

Software engineers have a lot of potential to create new platforms, innovative applications, or other major projects with the use of the computer in front of them. This computer can also be used for less industrious tasks like social media platforms, videos of all types, and games like Minesweeper. Distraction is a problem for many software engineers because they are on their computers a lot of the time, and many people are easily distracted throughout a long day. TaskMaster looks to solve this by optimizing the time you work, managing the tasks you need to complete, giving you breaks and motivation when needed, and keeping you productive during work time. TaskMaster can be used across teams to allow the manager of the team to have an oversight of how the rest of the team is doing and assign tasks with TaskMaster.

Throughout this project we have gone through the complete process of software design with TaskMaster. Our software process was prototyping, an iterative model with a focus on creating prototypes and getting customer feedback prior to release. We created use cases and models for those use cases in order to better understand and analyze the requirements for TaskMaster.

For the design phase we found a high level and low level design pattern that would work well with TaskMaster. The high level design pattern was event based, which prioritizes reactions to real-time events. The low level design pattern was an observer, which is part of the behavioral design pattern family. Observers are useful for reacting to what happens within the system and activating the proper response. We also created a design sketch to help with designing TaskMaster.

The final phase of software design was to plan how we would test TaskMaster and plan how it would be deployed and maintained. Testing would be done with black box testing as the focus is on individual features that need to be working, and the inner workings of the code are less crucial as long as the system works as intended. Deployment would be initially basic, but for future iterations we would use canary deployment. Maintenance would have an initial focus on corrective, then afterwards the focus would be on perfective, with adaptive and preventative maintenance done as needed.

Although TaskMaster has not been implemented, it gave us a good idea of the best practices and processes software engineers can use to create software. By walking through the process, we can see the difficulties software engineers face doing their job and how they must adapt to the project they are working on. If we wanted to create TaskMaster, this would be all the groundwork needed to where we can just review our work and begin coding.

REFERENCES

[1] Steve Glaveski. 2019. "10 Quick Tips for Avoiding Distractions at Work." Harvard Business Review, Harvard Business Publishing, Brighton, MA, https://hbr.org/2019/12/10-quick-tips-for-avoiding-distractions-at-work.

[2] Dana Dornsife and David Dornsife. "Are There Benefits of Multitasking?" MAPP Blog, University of Southern California, Los Angeles, CA, https://appliedpsychologydegree.usc.edu/blog/benefits-of-multitasking/.

[3] GitHub. "About Issues." GitHub Docs, GitHub Issues, https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues.

[4] Bryan Collins. 2020. "The Pomodoro Technique Explained." Forbes, https://www.forbes.com/sites/bryancollinseurope/2020/03/03/the-pomodoro-technique/?sh=43fe96ad3985.