Code Analyzer

Control+Alt+Defeat

Trevor Jarzynka
Computer Science
Virginia Tech
Blacksburg, VA, USA
trevorj04@vt.edu

Zekun Li
Computer Science
Virginia Tech
Blacksburg, VA, USA
zekunli@vt.edu

Cooper Robson
Computer Science
Virginia Tech
Blacksburg, VA, USA
cwrobson10@vt.edu

Ben Fielder
Mechanical Engineering
Virginia Tech
Blacksburg, VA, USA
benf02@vt.edu

ABSTRACT

In software engineering there are two major problems holding developers back, time spent understanding code and adding onto existing code. Companies currently try to mitigate these problems by holding team meetings, code reviews, and constant communication. Our solution is an algorithm that finds similar coding styles amongst company teams. It will provide a visual representation within a website that shows who codes similarly and what style techniques they most commonly use. In addition, our program will allow for database searches within a company's code to allow software developers to search for solutions to their problems before going to the internet or other sources. We believe that this program will optimize software developers' time and more innovation will come from it.

INTRODUCTION

In software development a major problem lies in when teams expand and it is in collaborative coding. This issue is critical for companies to address, as team development often leads to larger-scale, higher-quality innovation compared to individual efforts. To mitigate this, organizations employ techniques such as team meetings, code reviews, and continuous communication. While these methods are effective to an extent, significant time is still lost in understanding other’s code which can be inherently complex. Our program will provide an online dashboard showing team members coding styles. This tool will enable project managers to gain deeper insights into their developers approaches, facilitating the creation of more efficient and cohesive teams. Additionally, it can be integrated into the hiring process, assisting organizations in identifying candidates who best fit their technical needs and team dynamics.

The way in which we will accomplish this is by tapping into a company's code database like git or similar technologies to analyze all code and who commits the code. After this process, our natural language processing algorithm will upload profiles for each employee to the online dashboard. The profiles will include styles for major coding techniques which are customized by the employer. For example, commenting style, how they utilize if statements, for loops and what algorithms are commonly used like Dijkstra's or Bellman-Ford. In addition, we will implement a code database that stores small snippets of code written by people within the company to mitigate copying online code which can lead to cyber security issues.

With all of these features we believe that companies will want to use this tool as it benefits code development without requiring drastic change to existing code development. The cyber security aspect is another part of this program that companies are in need for solutions as cyber security is one of the largest issues within software engineering. With that said our program will help mitigate two major problems within software engineering and companies that use our product will produce innovative products quicker and more secure.

RELATED WORK

Research Studies:

Besides conducting our own research we found a research paper titled *Today was a Good Day: The Daily Life of Software Developers* to further highlight the issue our program will solve. In this study about 32% of a software engineer's day at Microsoft was spent actually coding with a sample size of 5,000 employees. With the same sample size it was also found that 35% of their time was spent reading, reviewing and understanding code written by other team members. This highlights the importance of this issue since more time is spent understanding code then writing it at one of the largest software companies in the world Microsoft.

Existing Programs:

Current programs don't currently exist that do what our program will do. Similar software programs are Git, SonarQube and JetBrains Qodana. Our program has a unique focus that SonarQube and JetBrains Qodana programs don't currently have which is company software development and Git does not currently have code analysis for large companies as well. We believe this highlights an opportunity for a program like ours to fill a need for companies. Additionally, our secure code snippet database designed to prevent the use of unauthorized external code further differentiates our solution. This cybersecurity aspect is another pressing concern that existing tools do not adequately address.

The combination of this and the gap in current solutions highlights an opportunity for our program to fill an unmet need in the industry. By addressing the inefficiencies in team coding workflows and incorporating cybersecurity safeguards, our tool represents a necessary innovation for modern software development.

RELEVANT EXAMPLE

Scenario:
A mid sized company called CodeSolutions creates unique and innovative solutions for their customers. They worked on many different customer products at the same time by having 5 development teams in their company. They are finding that the product development process is taking longer than expected for many of their teams and are looking to improve their development process. This is where our code analyser comes in. Our code analyser is linked to their code database like GitHub and GitLab and reviews the code and commits. It runs for a week to gather current team data and builds a dashboard for the company. In this dashboard the company finds many of their team members code very differently then their team members. They decide to change some junior engineers around and swap two senior engineers and find their development process is significantly faster.

Relevance to Software Engineers:
Our code analyzer helps software engineers by identifying and showcasing similar coding styles among team members. This allows for quicker comprehension of colleagues code, and enables developers to focus more on creating innovative solutions rather than deciphering existing code. Additionally, the integrated code search database accelerates onboarding for junior engineers, guiding them to adopt the company's established coding practices and reducing reliance on external, potentially dangerous solutions from the internet. This ensures consistent, secure, and efficient development across the entire team.

IMPLEMENTATION

High Level Design Decisions:
For our high level design we chose to implement a Model-View-Controller (MVC) architecture, as outlined in PM3. The MVC architecture has both pros and cons, and we believe that our product will be best optimized through the MVC architecture.

Before detailing exactly why we chose to use MVC, it is important to have an understanding of what it is. The MVC architecture divides the application into three components: the Model, View, and Controller. The model is responsible for managing the data and contains the core logic of the application. This is where the algorithms reside that will analyze code and optimize team formation. The view represents the user interface, allowing users to interact with the application by entering their code and information.

The input is then passed to the model for processing. Finally, the controller serves as the intermediary between the model and the view, communication between the two and updating them as needed.

We chose the MVC architecture for a couple of reasons. First, given the complexity of our tool, MVC provides better reusability of components and supports multiple levels of abstraction. This simplifies the design phase by reducing the amount of code we need to write and allowing us to reuse existing components. Additionally, MVC's structure makes it easier to add new features or modify parts of the application without impacting the entire system. For example, we can update the user interface or optimize the algorithms independently. While MVC has its pros and cons, we believe that it is the best decision for our system, offering a straightforward and effective path to implementation.

Implementation Processes:
There is a lot to consider when thinking about translating our software design into the concrete system. Because of the complex nature of our system, different parts of our system will use different software engineering philosophies for implementation. Specifically, the feature driven and behavior driven development will be used. For our user interface or view, as specified in the previous section, we would use feature driven development. This is because we know exactly what we want the user interface to look like including all of its features. The primary purpose of this part of the development is to gain functionality for our user interface, and we could do this through this iterative and incremental process that will focus on the features of the user interface. The second process that we would use is behavior driven development, and that would be for our algorithm, or the controller. Because our app requires an advanced algorithm with results that are not completely known, the best process would be behavior driven development. We know generally how we want our controller to operate, and therefore we can focus on this behavior as opposed to specific results or tests.

Testing Approach:
It is imperative that we implement and execute our software with the intent of finding errors, and fixing them. To do this, we would use the white box testing approach.

The white box testing approach is a common testing approach that uses code to guide the testing. Essentially, this is used to verify that the expected output matches the actual output. Because we wil likely be using pre existing libraries for our project, we will likely have no need for unit testing. It is more likely that we would use integration testing, where we test groups of subsystems and eventually the whole system. It is possible that whoever the client is would use validation testing to confirm that what we, as the developers, delivered what we said we would

DEPLOYMENT PLAN

Our code analyzer will use Continuous Integration and Canary deployment to deploy our software. Continuous Integration will allow us to automatically build, tst and analyze software in response to every software change committed to the source repository. This will be very good because it will allow us to detect defects and fix them earlier and faster, reduce repetitive processes, and can release deployable software at any time. It also allows a global mechanism for feedback, it increases developers confidence about changes, low risk releases, and also creates happier teams. That is why we will use Continuous Integration.

For testing purposes we will use Fuzz Testing, Mutation Testing and Chaos Engineering.

Canary Deployment is the lowest risk, it also has very fast and cheap rollbacks and it is cheaper than blue and green testing. This will be done first by defining our scope. We will choose a small set of repositories or projects for the initial deployment. Then we will select repositories with representative codebases (common languages, frameworks and team workflows). There will need to be set success metrics and also need to create a rollback plan in case we need to disable the analyzer in case of issues. This type of analyzer will allow us to put out our new updates quickly and also cheaply. We will implement it so we can release the application incrementally to a subset of users in small phases. That is why this is the perfect deployment strategy.

FUTURE OPPORTUNITIES

Future growth opportunities for our product lie in expanding its reach beyond company integration to cater to individual developers. While our initial focus is on providing a robust solution for teams, we envision enhancing the product with features such as AI-powered security detectors and AI-driven style assistants. These tools could help individual developers write secure, consistent, and high-quality code more efficiently in their everyday work.

However, achieving widespread adoption has its limitations. One major obstacle is marketing costs; substantial capital will be required to build brand awareness and reach a broad audience. Additionally, the development and refinement of AI tools raise questions about their potential effectiveness and reliability. Ensuring our AI capabilities meet the high expectations of individual developers will be critical to the success of this expansion.

Despite these challenges, we believe the potential for our product to revolutionize coding practices for both teams and individual developers makes these efforts worthwhile. With the right advancement in AI technology we believe our product will be able to revolutionize the software development industry.

CONCLUSION

Our project addresses the challenges of time spent understanding code and adding onto existing code, which are common bottlenecks in collaborative software development. Our Code Analyzer provides a solution by identifying similar coding styles among team members, visualizing coding practices through a comprehensive dashboard, and incorporating a secure and searchable code snippet database. These features improve the code comprehension, reduce reliance on external resources, and make the team collaboration more efficient and innovative.

While our current focus is on team-level integration within company workflows, future work will explore improvements for individual developers. Features such as AI-powered security detectors and style assistants could future improve code quality, security, and consistency. However, challenges such as marketing costs for product promotions and the need for robust AI refinement remain as limitations for widespread adoption for companies. These will be important for scaling the tool effectively.

Despite these challenges and limitations, the Code Analyzer demonstrates strong potential to make a big change for coding practices. By optimizing developer productivity, improving team dynamics, and enhancing cybersecurity, our solution offers a valuable step forward in modern software development.

References

"Asana." Agile Methodology: What Is It and How It Works,Asana, https://asana.com/resources/agile-methodology. Accessed 27 Sept. 2024.

Meyer, André N., et al. Today Was a Good Day: The Daily Life of Software Developers.Microsoft Research, Apr.2019, https://www.microsoft.com/en-us/research/uploads/prod/2019/04/devtime-preprint-TSE19.pdf. Accessed 27 Sept. 2024.