PM3 - Code Analyzer

Group: Control+Alt+Defeat

Process Deliverable

The SE process that we chose to use to complete the group project, as outlined in our project proposal and in PM2, is Agile. Because the process we are using is Agile, for the process deliverable we will submit a summary from our end of sprint retrospective, and our prioritized tasks.

What went well:. We specified requirements for our problem to give us a clearer framework to work.
What didn't go well: Throughout the sprint, communication was not the best. There was a significant amount of days and weeks where there was an imbalance of communication with each other which led to some confusion.
What can be improved: Communication can be improved, specifically the communication around the plan for our project milestones. Proper planning will lead to a smoother process for our team and communication is the main thing we can improve to optimize team performance.

Prioritized tasks for the next sprint:

- Complete PM4 by the due date
    - Process Deliverable III
    - Black Box Test Plan
- PM5
    - Final Presentation
    - Final Report
    - Retrospective
    - Presentation Preparation
- Communication throughout our team, specifically around uncertainty within project milestones.

High-level Design

For our high level design we have chosen to implement a Model-View-Controller (MVC) Architecture. The MVC architecture has pros and cons, and we believe that our tool will be optimized best through the MVC architecture.

The MVC architecture breaks the application or tool into three distinct parts: the model, which handles the data, the view, which is what the user interacts with, and the controller, which connects the previous two together. The model component is what will contain the logic and meat of our application. This holds the algorithms that will analyze

our code and optimize team forming. The view, as stated previously, is what the user will interact with. This is where the user will input their own code and personal information, which will then be sent to the model for analysis. The controller is what communicates between the view and model, and updates the model and view accordingly.

There are a couple of reasons that we chose to use the MVC architecture. The first is that, because of the complexity of our tool, MVC makes it much easier to reuse components and have multiple levels of abstraction. This will make it much easier to implement in our design phase because we have to implement less code and we can reuse components. Because of the nature of MVC, it will be much easier for us to add new features and change components without affecting the whole application as well. If we want to change the user interface or the optimization algorithm, we can do that without having to change the whole system. Although there are both advantages and disadvantages for using MVC, we believe that this architecture will be best for our system and would allow for the easiest and most effective implementation.

Low-level Design

We would use a behavioral pattern because we need to make sure that the program responds based on the different behaviors of people. It will need to use the algorithms to assign responsibilities.

Specific Subtask: Code Issue Resolution Workflow

The Advantage of Using the Behavioral Design Pattern:

Sequential and Flexible Issue Handling:

- Issues in the code can be processed in a defined sequence where each handler (e.g., syntax checker, optimizer) is responsible for resolving a specific type of issue.
- If a handler cannot process the issue, it passes the responsibility to the next handler in the chain.

Extensibility:

- Adding new handlers (e.g., for a new type of issue) does not require changes to existing handlers, adhering to the **Open/Closed Principle**.

Dynamic Configuration:

- The chain of handlers can be dynamically configured at runtime based on project requirements.

Improved Modularity:

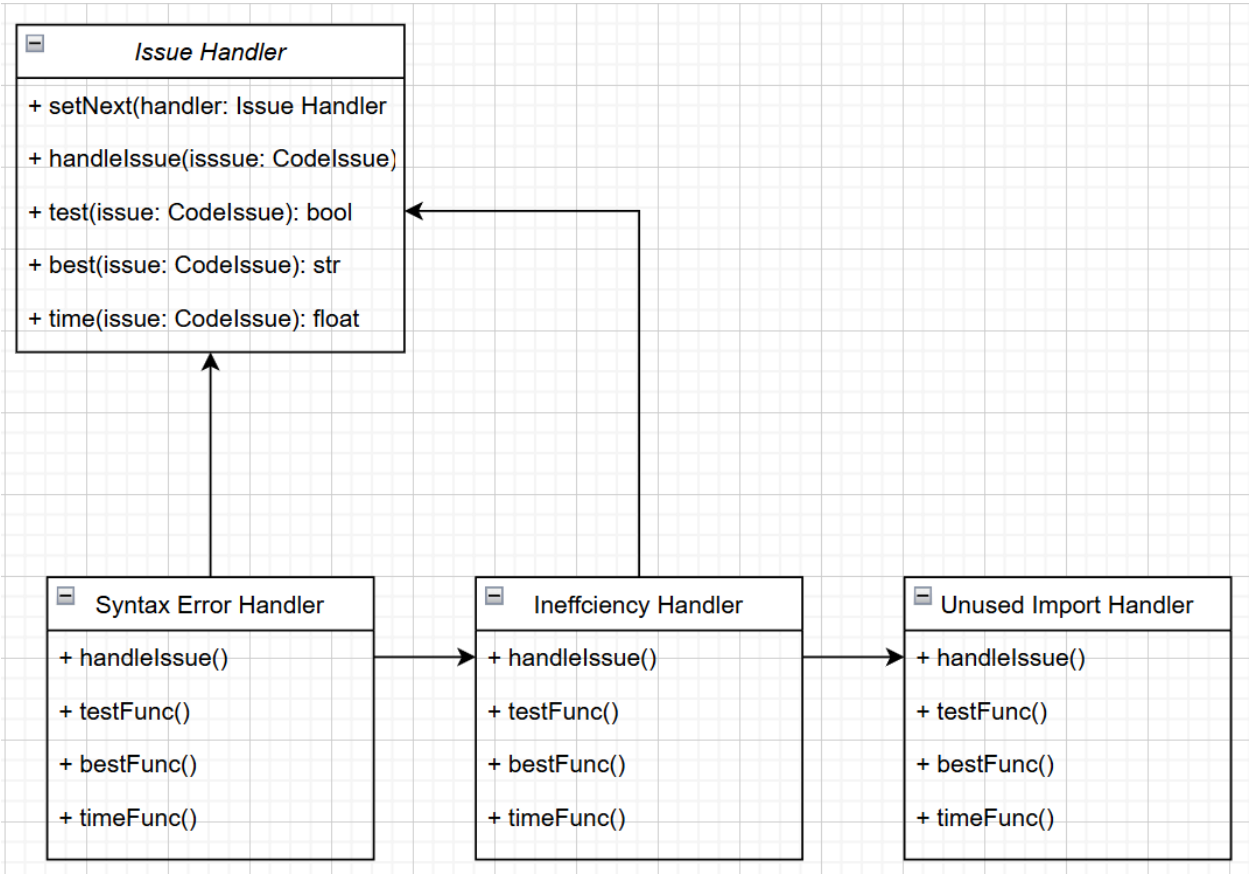
- Each handler focuses on a single responsibility, promoting high cohesion and low coupling.

Enhancements:

1. **Test Function:**
    - Each handler tests if it can process the given issue
    - If it cannot handle the issue, it passes it to the next handler in the chain
2. **Best Function:**
    - Determines the “best” resolution for the issue, comparing different handlers or applying additional logic to prioritize solutions.
3. **Time Tracking:**
    - Measure the time each handler takes to process an issue using Python’s time module

This is an example of how the code would work. It would compare your code to the best code known and then give you suggestions based off of it.

Class diagram

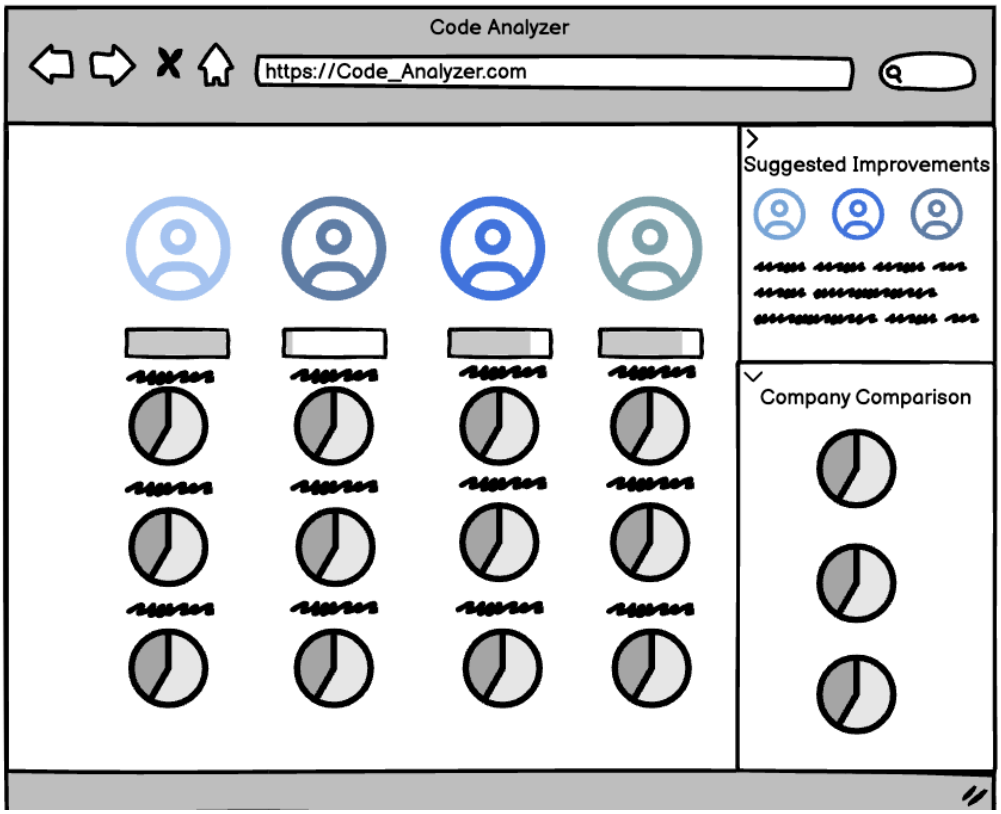


Design Sketch

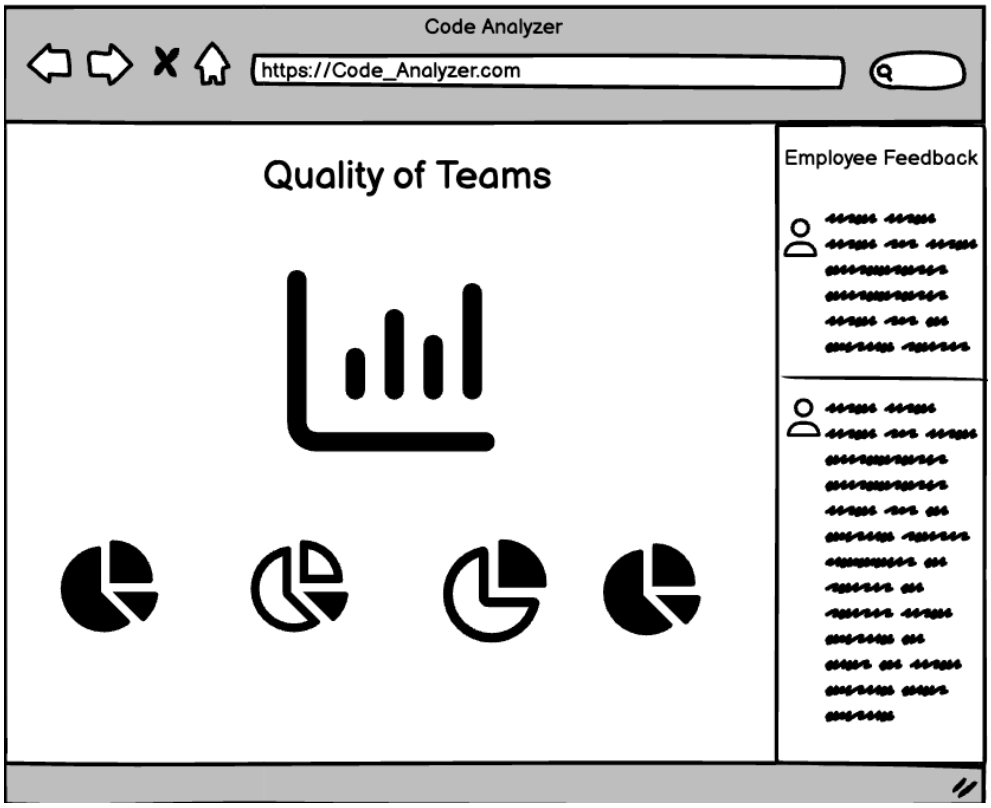
For our UI we chose to do a wireframe. The three main things we focused on for our UI are:

- Consistency
    - This makes the user experience a lot less confusing
- Guidance
    - Guiding the user through our program with labels and simple buttons allows for a easier user experience
- User control and freedom
    - We want our users to have freedom in how they utilize our tool so customization plays a big role in our design

We provided two views that a user would have. This is a wireframe so the final product will be more polished.
This is a view of a specific team



This is a view of the company's teams


Code Analyzer
https://Code_Analyzer.com
Quality of Teams
Employee Feedback