PM2 - Code Analyzer

Group: Control+Alt+Defeat

Process Deliverable 1

The SE process that we chose to use to complete the group project, as outlines in our project proposal, 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: For the most part we stayed on top of our work and didn't wait until the last minute to get it done. We specified requirements for our problem to give us a clearer framework to work under.
What didn't go well: Throughout the sprint, communication was not ideal. There were multiple days and weeks where there was not a lot of communication with each other, which meant some team members were not on the same page.
What can be improved: When team members have questions or are unclear about something, they can take the initiative to reach out and ask questions.

Prioritized tasks for the next sprint:

- Complete PM3 by the due date
  - Project Milestone 2
  - High-Level Design
  - Low-Level Design
  - Design Sketch
  - Project Check-In Survey
- Stay in touch with the team, ask questions if we are unsure.

Requirements Analysis

Non-functional Requirements

Non-functional Requirement 1: Performance

1. **Preconditions**
   - The code analyzer tool must be fully configured and integrated with the project's code repository, such as GitHub or GitLab.
   - Code repository should be accessible, and network conditions should support data transfer within defined performance thresholds.

2. **Main Flow**
    - The code analyzer performs code parsing to understand structure and syntax [P1].
    - The analyzer completes static analysis and reports potential issues within a predefined time limit [P2].
    - The tool processes code efficiently, managing resources to handle large-scale repositories [P3].
3. **Subflows**
    - [P1] The analyzer parses code within 30 seconds per 1,000 lines to avoid delays.
    - [P2] Static analysis completes within 5 minutes for repositories up to 10,000 lines.
    - [P3] The system dynamically allocates resources based on the repository size to maintain performance.
4. **Alternative Flows**
    - [E1] If analysis time exceeds the limit, the tool provides feedback and a partial report.
    - [E2] Network interruptions during analysis trigger a retry mechanism to resume from the last step completed.

Requirement: The code analyzer completes parsing and report generation within 5 minutes for codebases up to 10,000 lines.

Category: Performance – This specifies a time-bound performance requirement for processing a codebase, defining expected processing efficiency.

Non-Functional Requirement 2: Usability

1. **Preconditions**
    - Users must have appropriate access to the tool and be familiar with basic configuration steps.
    - Interface guidelines should be pre-set to ensure a standardized layout and interaction flow.
2. **Main Flow**
    - Users can initiate code analysis with minimal clicks, using a clean, intuitive interface [U1].
    - Visual feedback on analysis status and completion time estimates keep users informed throughout the process [U2].
    - Clear documentation is provided, outlining each step from configuration to analysis report generation [U3].
3. **Subflows**

- [U1] Interface prompts are minimal but guide users effectively, reducing setup time by 20%.
- [U2] A progress bar updates users on each stage of analysis.
- [U3] Comprehensive user guides and tooltips are accessible within the interface.

4. **Alternative Flows**
    - [E1] If users encounter errors, the system directs them to troubleshooting resources.
    - [E2] The interface offers quick actions for repeated tasks, such as setting a default analysis type.

Requirement: The tool's interface provides visual feedback for each step, including analysis progress and completion status.

Category: Usability – This requirement focuses on the user interface and interaction, emphasizing ease of use and feedback clarity for users.

Non-functional Requirement 3: Reliability

1. **Preconditions**
    - The system must be connected to stable network services and have backup capabilities for critical operations.
    - Version control is enabled for the codebase and analysis configurations.
2. **Main Flow**
    - The tool performs code analysis with a reliability rate of 99.9% uptime during peak usage times [R1].
    - Error logs are generated automatically and stored securely for each analysis session [R2].
    - Scheduled backups of analysis reports ensure no data is lost [R3].
3. **Subflows**
    - [R1] Uptime is actively monitored and automatically adjusted for high-volume usage.
    - [R2] Error logs capture all system faults and are accessible to authorized admins only.
    - [R3] Backup schedules align with each analysis run and store copies for 90 days.
4. **Alternative Flows**
    - [E1] If the system experiences downtime, the analysis queue pauses and resumes once service is restored.
    - [E2] Missing logs trigger a system alert for immediate investigation.

Requirement: The code analyzer maintains a 99.9% uptime to support CI pipelines, especially during peak development hours.

Category: Reliability – This ensures system availability, specifically uptime, to support critical operations, which directly impacts user trust in the system's dependability.

Non-functional Requirement 4: Security

1. **Preconditions**
    - Users must be authenticated with appropriate permissions to access and run the code analyzer.
    - Secure access protocols (e.g., HTTPS) are in place for data transmission between the code analyzer and the repository.
2. **Main Flow**
    - The code analyzer encrypts all data in transit and at rest to ensure data security [S1].
    - Access to analysis reports is restricted based on user roles, allowing only authorized users to view or download sensitive information [S2].
    - Authentication and authorization checks are performed for each session, logging user access details [S3].
3. **Subflows**
    - [S1] Data encryption is enabled for all communications, following AES-256 standards.
    - [S2] Role-based access control (RBAC) limits report visibility to specific user groups, such as admins and security analysts.
    - [S3] Login sessions are logged, and access history is available for auditing.
4. **Alternative Flows**
    - [E1] If unauthorized access is detected, the system automatically triggers an alert and blocks further access until verified.
    - [E2] Failed authentication attempts lock the user out after three retries, prompting a security review.

Requirement: Code data is encrypted both in transit and at rest to protect against unauthorized access, particularly during analysis and report generation.

Category: Security – This requirement emphasizes data protection through encryption, a key security aspect to safeguard sensitive code information.

Non-functional Requirement 5: Scalability

1. **Preconditions**

    - The system is hosted in an environment that can scale resources based on demand, such as a cloud platform.
    - Storage and compute resources are configured to handle large codebases and concurrent analyses without significant performance degradation.
2. **Main Flow**
    - The code analyzer scales to support multiple users running concurrent analyses without impacting performance [SC1].
    - The system automatically adjusts memory and processing power based on the size of each codebase to optimize processing time [SC2].
    - Load balancing distributes requests to avoid bottlenecks during peak usage times [SC3].
3. **Subflows**
    - [SC1] Each user session is isolated, ensuring concurrent analyses don’t interfere with each other.
    - [SC2] For large codebases, additional resources are provisioned dynamically to reduce analysis time.
    - [SC3] Load balancing across servers distributes incoming requests evenly.
4. **Alternative Flows**
    - [E1] If the system reaches its resource limit, new analysis requests are queued and processed in order.
    - [E2] In cases of high demand, users are notified of estimated wait times before analysis can begin.

Requirement: The code analyzer supports concurrent analyses for multiple users, managing resources dynamically to handle large projects without significant slowdown.
Category: Scalability – This addresses the system's ability to handle increasing load by managing resources effectively for multiple users.

Functional Requirements

Use Case 1: Code Analysis

Primary Actor: Developer

Preconditions:
The developer has installed and configured the code analyzer tool.
The developer has a project with source code.

Postconditions:
The code analyzer generates a report identifying potential issues, such as errors, warnings, and security vulnerabilities.

Basic Flow:

Initiate Analysis:

Developer triggers the code analysis process, either manually or through an automated build process.

Code Parsing:

Code analyzer parses the source code to understand its structure and syntax.

Static Analysis:

Code analyzer performs static analysis to identify potential errors, warnings, and security vulnerabilities without executing the code.

Dynamic Analysis (Optional):

For certain types of analysis, the code analyzer may execute the code to identify runtime errors and performance issues.

Report Generation:

Code analyzer generates a report detailing the identified issues, including their severity, location, and potential solutions.

Review and Remediation:

Developer reviews the report and addresses the identified issues.

Use Case 2: Code Optimization

Primary Actor: Developer

Preconditions:

The developer has installed and configured the code analyzer tool.

The developer has a project with source code.

The code analyzer has identified performance bottlenecks or inefficiencies.

Postconditions:

The code analyzer suggests optimization techniques to improve performance.

Basic Flow:

Performance Analysis:

Code analyzer analyzes the code to identify performance bottlenecks, such as inefficient algorithms, excessive resource usage, or slow database queries.

Optimization Recommendations:

Code analyzer suggests specific optimization techniques, such as using more efficient algorithms, reducing redundant calculations, or optimizing database queries.

Code Refactoring:

Developer applies the recommended optimization techniques to the code.

Performance Re-evaluation:

Developer re-runs the code analyzer to measure the impact of the optimizations.

Use Case 3: Security Vulnerability Scanning

Primary Actor: Security Analyst

Preconditions:

The security analyst has installed and configured the code analyzer tool.

The security analyst has access to the application's source code.

Postconditions:

The code analyzer identifies potential security vulnerabilities in the code.

Basic Flow:

Security Scan:

Security analyst initiates a security scan of the application's source code.

Vulnerability Detection:

Code analyzer scans the code for common vulnerabilities, such as SQL injection, cross-site scripting (XSS), and buffer overflows.

Vulnerability Report Generation:

Code analyzer generates a report detailing the identified vulnerabilities, including their severity, location, and potential exploitation methods.

Vulnerability Remediation:

Security analyst reviews the report and works with developers to address the identified vulnerabilities.

Use Case 4: Code Style Enforcement

Primary Actor: Developer

Preconditions:

The developer has installed and configured the code analyzer tool.

The developer has a project with source code.

A specific coding style guide is defined.

Postconditions:

The code analyzer identifies style violations and suggests corrections.

Basic Flow:

Style Check:

Code analyzer checks the source code against the defined style guide.

Style Violation Detection:

Code analyzer identifies any deviations from the style guide, such as inconsistent indentation, naming conventions, or formatting.

Style Violation Report Generation:

Code analyzer generates a report detailing the identified style violations and their locations.

Code Formatting:

Code analyzer can automatically format the code to adhere to the style guide, if supported.

Use Case 5: Technical Debt Analysis

Primary Actor: Technical Lead

Preconditions:

The developer has installed and configured the code analyzer tool.

The developer has a project with source code.

Postconditions:

The code analyzer identifies technical debt and provides recommendations for remediation.

Basic Flow:

Code Analysis:

Code analyzer analyzes the source code to identify potential technical debt, such as outdated code, unused code, or poor design patterns.

Technical Debt Report Generation:

Code analyzer generates a report detailing the identified technical debt, including its severity, location, and potential impact.

Prioritization:

Technical lead prioritizes the identified technical debt based on its severity and impact.

Remediation Planning:

Technical lead plans a strategy for addressing the technical debt, considering factors such as development resources, project deadlines, and risk tolerance.

Requirements Specification

User Story 1:
As a product manager I find one of the main things that holds back the project is developer flow. We currently utilize Git to manage our code base but we still struggle to hit our goals in time. I would love a tool that helps me visualize who works well together on my team of developers. This will increase our workflow and allow for a more streamlined development process

Acceptance Criteria:

- I can see a visual representation that shows the relationships and similarities in coding styles among team members
    - 85 Function points
        - This will require database to frontend skills and may include a lot of tweaking to get correct but the most technically challenging part is the algorithm that creates the relationship which is why this task is quite difficult.
- Each team member has a personal view that highlights strengths and weaknesses
    - 45 Function points
        - This is a subset of the original view so the code should be pretty similar to what the program already has. Given this I felt the level of difficulty is not very large as most should be copy and paste.
- I can designate groups and see the analysis for each group
    - 25 Function points
        - This is a smaller task that requires some technical knowledge but given the size of the task it should not take very long
- I can click on a team member or group to see common techniques they use
    - 25 Function points
        - This is a smaller task that requires some technical knowledge but given the size of the task it should not take very long

User Story 2:
As a college student I have to work with other people on class projects. I always get paired with someone who does not work well with me. We use Git to help contain our code but I wish I could be better paired with someone who codes like me. If I could upload my previous projects and my professor could pick similar people I would save so much time and headache.

Acceptance Criteria:

- I can connect my GitHub account to a website instead of uploading code
    - 20 Function Points
        - This would require some understanding of GitHub's API to interact with and pull code from. The level of difficulty does depend on the API but assuming it's relatively simple to use then this would not require much effort to implement.

- I can get paired with someone and contact them easily
  - 60 Function Points
    - This would require a specific UI to be built for users to interact with. It would need a professor and student view which adds some difficulty. The technical knowledge for this is not drastic but this is a large task to complete and that is the reason for the difficulty rating.
- I get accurately paired with someone who is similar to me
  - 100 Function Points
    - This is the development of the main algorithm. This is the hardest task the website would require as it includes natural language processing and a lot of fine tuning in order to produce an acceptable output.

User Story 3:
As a CEO of my company I want to improve our employee's experience and productivity. Through our HR department I have found we can improve our team construction for optimal productivity and employee satisfaction. We utilize Git and other tools to maintain and store our code and are open to implementing new tools that can improve our workflow

Acceptance Criteria:
- Easy and seamless integration within the company
  - 30 Function Points
    - This is an important aspect of the development of the program and is split amongst all tasks. The reason I gave it a relatively easy rating is because all other tasks have this in mind so the final integration should not require much change to be implemented seamlessly. With that said the difficulty can go up if the product is not developed with this in mind.
- Hierarchy overview allowing for managers to work together for optimal teams
  - 40 Function Points
    - For the most part this is a UI integration which should be pretty simple. The level of difficulty is increased as hierarchical profiles can be complex but overall this should be an average task to complete.
- Works with Git and our current tools
  - 20 Function Points
    - This is reliant on the Git API but should be relatively simple to implement. Other tools should not be necessary to implement within our software but to implement them it would depend on their API's .

User Story 4:
I am a software engineer and I want to spend more time coding. Before getting into the job my understanding of life as a software engineer included a lot more coding what I found in reality. On a good day about 40% of my day is actually coding and the rest is reading code, meetings,

and communication. I would love a tool that helps mitigate the time it takes to understand and approve others' code as I would be able to focus on what I love most, which is coding.

Acceptance Criteria:

- Provides accurate analysis of code that matches people with the optimal team members for code development
    - 100 Function Points
        - This is the development of the main algorithm. This is the hardest task the product would require as it includes natural language processing and a lot of fine tuning in order to produce an acceptable output.
- Provides company database search to increase security
    - 70 Function Points
        - This would require the development of a search algorithm and a user interface to interact with. This is a large task to complete and that is the main reason for its difficulty.
- Seamless integration within the company's current tech stack
    - 20 Function Points
        - This is reliant on the Git API but should be relatively simple to implement. Other tools should not be necessary to implement within our software but to implement them it would depend on their API's .