**Software Requirements**

**Specification**

**For**

**AI Code Reviewer**

**Version 1.0 approved**

**Prepared by**

**TIRTHORAJ BHATTACHARYA**

**TANMESH SINGH**

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY ALLAHABAD**

**04/04/2025**

**Table of Contents**

**Table of Contents ............................................................................................ ii**

1. **Introduction .......................................................................................... 3**
   1. Purpose ............................................................................................. 3
   2. Document Conventions .................................................................... 3
   3. Intended Audience and Reading Suggestion .................................... 3
   4. Product Scope ................................................................................... 4
   5. References ......................................................................................... 4
2. **Overall Description …………………………………………………… 4**

2.1 Product Perspective ………………………………………………… 4

2.2 Product Functions …………………………………………………... 5

2.3 User Classes and Characteristics …………………………………… 5

2.4 Operating Environment ………………………………………...…... 8

2.5 Design and Implementation ………………………………………… 8

2.6 User Documentation ………………………………………………… 8

2.7 Assumptions and Dependencies ……………………………………... 9

1. **External Interface Requirements ………………………………….... 10**
   1. User Interfaces ………………………………………………….…. 10
   2. Hardware Interfaces …………………………………………….…. 10
   3. Software Interfaces ………………………………………………… 11
   4. Communication Interfaces …………………………………………. 11

**1. Introduction**

**1.1 Purpose**

This Software Requirements Specification (SRS) document describes the functional and non-functional requirements for the AI-Powered Code Reviewer system, version 1.0. This document covers the Minimum Viable Product (MVP) phase of the system, which will integrate with GitHub to analyze JavaScript/TypeScript code changes against company standards using AI to detect issues that traditional static analyzers might miss.

**1.2 Document Convention**

The following conventions have been used in this document:

* Requirements are organized by functional areas and are numbered hierarchically.
* "Must" indicates a mandatory requirement.
* "Should" indicates a desirable requirement.
* "May" indicates an optional requirement.
* TBD (To Be Determined) is used for details that will be defined in future versions of this document.

**1.3 Intended Audience and Reading Suggestions**

This document is intended for:

* **Development Team:** To understand what needs to be built
* **QA Team:** To develop test plans and test cases
* **Project Manager:** To plan the implementation and allocate resources
* **Stakeholder:** To ensure the system meets business requirements

**1.4 Product Scope**

The AI-Powered Code Reviewer system is designed to serve as a preliminary checkpoint before developers submit code for formal peer review. It aims to:

* Automate initial code quality checks to ease senior developers' workload.
* Help junior developers receive immediate feedback on their code changes.
* Improve overall code quality and consistency.
* Reduce the number of review iterations required before code acceptance.
* Enforce company coding standards consistently.

The system will integrate with GitHub to analyze pull requests, focusing initially on JavaScript/Type Script code analysis, and will provide feedback directly within GitHub's interface.

**1.5 Reference**

* GitHub API Product Perspective Documentation: <https://docs.github.com/en/rest>
* ESLint Documentation: <https://eslint.org/docs/latest/>
* Hugging Face Models Documentation: <https://huggingface.co/docs/transformers/index>
* JavaScript/TypeScript Code Style Guide: [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*]
* Node.js Documentation: https://nodejs.org/en/docs/

**2. Overall Description**

**2.1 Product Perspective**

The AI-Powered Code Reviewer is a new, standalone system that integrates with existing GitHub workflows and tooling. It operates as a service that:

1. Listens for GitHub webhook events when pull requests are created or updated
2. Fetches the code changes from GitHub
3. Processes the code through static analysis and AI-based pattern recognition
4. Generates feedback based on the analysis
5. Posts the feedback as comments in the pull request

**2.2 Product Functions**

The major functions of the AI-Powered Code Reviewer include:

* GitHub webhook integration for pull request analysis
* Static code analysis using ESLint with custom rules
* Basic pattern detection against reference code examples
* AI-assisted code quality classification using pre-trained models
* Pull request commenting with inline suggestions and feedback
* Simple web dashboard for configuration and visualization of analysis metrics
* User authentication and authorization for dashboard access

**2.3 User classes and characteristics**

1. **Junior Developers (Primary Users)**

* Frequent use; will receive feedback on their code submissions
* May have limited experience with code reviews
* Need clear, actionable feedback with examples
* Technical proficiency: Moderate to high

**2.** **Senior Developers/Code Reviewers (Secondary Users)**

* Less frequent use; will configure rules and review system effectiveness
* Experienced in code review processes
* Need ability to customize rules and provide feedback on AI suggestions
* Technical proficiency: High

**3. System Administrators (Tertiary Users)**

* Infrequent use; will manage system configuration and monitor health
* Need access to logs, performance metrics, and configuration settings
* Technical proficiency: High



Fig-1: Use Case Diagram for AI Code Reviewer

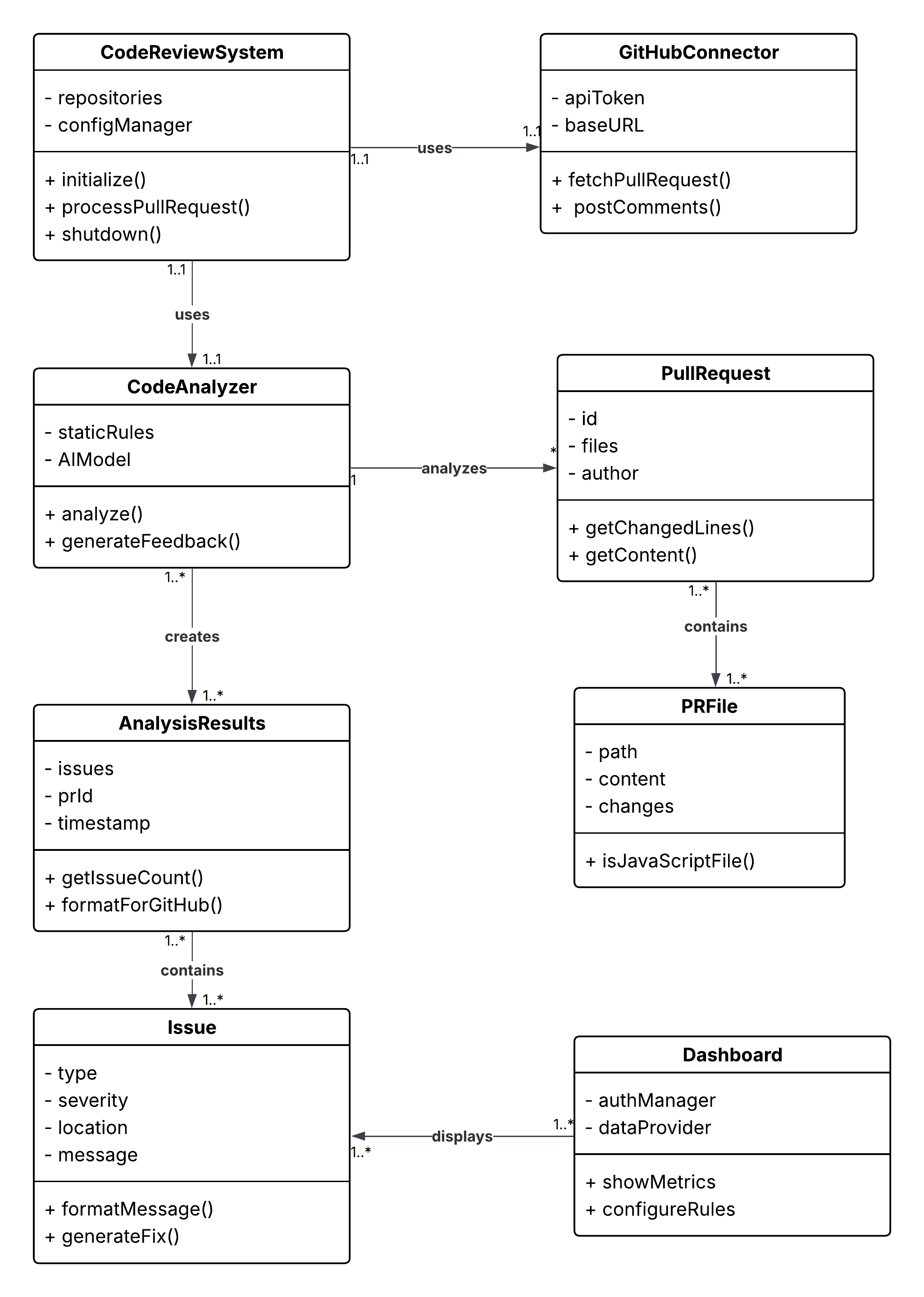


Fig-2: Class Diagram for AI Code Reviewer

**2.4 Operating Environment**

The AI-Powered Code Reviewer will operate in the following environment:

* Deployed as Docker containers on Linux-based servers.
* Backend implemented as Node.js tech-stack.
* Frontend dashboard implemented with React.
* Integrated with GitHub via webhooks and API.
* Minimum Node.js version 18.x required.
* Support for modern web browsers (Chrome, Firefox, Safari, Edge).
* Internet connectivity required for GitHub integration.

**2.5 Design and Implementation Constraints**

The system is subject to the following constraints:

* Must use free and open-source tools to minimize costs.
* Initial focus on JavaScript / TypeScript analysis only.
* Will use pre-trained models rather than custom ML training for the MVP.
* Must comply with GitHub API rate limits.
* Must complete analysis within 5 minutes of PR submission to provide timely feedback.
* Must be containerized for easy deployment.
* Must work within GitHub Actions environment constraints.

**2.6 User Documentation**

The following user documentation will be provided:

* Installation and setup guide for system administrators.
* User manual for developers explaining the feedback system.
* Configuration guide for senior developers to customize rules.
* Dashboard usage guide for visualizing and monitoring metrics.
* API documentation for potential integration with other systems.

**2.7 Assumptions and Dependencies**

The system makes the following assumptions and has these dependencies:

* GitHub repositories are accessible to the system via API tokens
* Users have GitHub accounts and proper permissions
* Repositories have properly structured JavaScript/TypeScript code
* Node.js and necessary dependencies are available in the deployment environment
* Pre-trained models for code analysis are available and suitable for the task
* ESLint and relevant plugins can be used for static analysis
* Internet connectivity is available for API calls and model access

**3. External Interface Requirements**

**3.1 User Interfaces**

**3.1.1 GitHub Pull Request Interface**

* The system will post comments directly to GitHub pull requests.
* Comments will be formatted with Markdown and include:
  + - * + Issue description
        + Code snippet highlighting the problem
        + Suggested fix (when possible)
        + Severity level (Critical, Major, Minor, Suggestion)
        + Reference to relevant coding standards

**3.1.2 Admin Dashboard**

* Web-based interface accessible via modern browsers
* Responsive design supporting desktop and tablet resolutions (min width: 768px)
* User authentication required
* Main sections:
  + - * + Overview/metrics dashboard
        + Configuration settings
        + Repository management
        + Rule customization
        + Analysis history

**3.2 Hardware Interfaces**

The system does not directly interface with hardware. It will run on standard server infrastructure with:

* Minimum 4GB RAM
* 2 CPU cores
* 20GB storage
* Network connectivity

**3.3 Software Interfaces**

**3.3.1 GitHub API**

* REST API v3
* Authentication via OAuth tokens
* Methods used:
  + - * + Pull Request events webhook.
        + Get PR content
        + Post PR comments

**3.3.2 ESLint**

* Version 8.x or higher
* Custom rule configurations
* JavaScript and TypeScript parsers

**3.3.3 Hugging Face Models API**

* Pre-trained code understanding models
* REST API for model inference
* Authentication via API tokens

**3.3.4 Database**

* MongoDB for configuration and historical data storage
* Connection via standard MongoDB driver
* Minimum version 4.4

**3.4 Communications Interface**

* HTTPS for all external communications
* WebSocket for real-time dashboard updates.
* JSON for data interchange format.
* GitHub webhooks for event notifications.
* RESTful API endpoints for system integration.

1. **System Features**
   1. **GitHub Integration**

* + 1. **Description and Priority**

The system must integrate with GitHub to receive notifications about new or updated pull requests and to post analysis results. This feature has HIGH priority as it is fundamental to the operation of the system.

* + 1. **Stimulus/Response Sequences**
* **Stimulus:** GitHub sends a webhook event when a pull request is created or updated.
* **Response:** System acknowledges receipt and queues the PR for analysis.
* **Stimulus:** Analysis is completed.
* **Response:** System posts comments to the PR with findings.
  + 1. **Functional Requirements**

**REQ-1.1:** The system must register webhooks with GitHub repositories to receive PR events.

**REQ-1.2:** The system must authenticate with GitHub using OAuth tokens.

**REQ-1.3:** The system must retrieve the content of pull requests, include file diffs.

**REQ-1.4:** The system must post comments to pull requests, both at the PR level and as inline comments.

**REQ-1.5:** The system must update comments when reanalysing a PR that has been updated.

**REQ-1.6:** The system should track PR status changes (open, closed, merged).

* 1. **Static Code Analysis**
     1. **Description and Priority**

The system will perform static code analysis on JavaScript/TypeScript files to identify syntax errors, code style violations, and potential bugs. This feature has HIGH priority as it provides the foundational level of code quality checks.

* + 1. **Stimulus/Response Sequences**
* **Stimulus:** System receives PR content for analysis.
* **Response:** System runs ESLint with custom rules on the code.
* **Stimulus:** ESLint completes analysis.
* **Response:** System captures and formats the results for feedback.
  + 1. **Functional Requirements**

**REQ-2.1:** The system must use ESLint to analyse JavaScript/TypeScript code.

**REQ-2.2:** The system must support custom ESLint rule configurations.

**REQ-2.3:** The system must check for syntax errors and report them with line numbers.

**REQ-2.4:** The system must check code against company style guidelines.

**REQ-2.5:** The system must identify common code issues such as unused variables, missing error handling, and potential memory leaks.

**REQ-2.6:** The system should provide severity levels for each identified issue.

**REQ-2.7:** The system should suggest fixes for common issues when possible.

* 1. **AI Pattern Recognition**
     1. **Description and Priority**

The system will use pre-trained AI models to detect patterns and issues that go beyond what static analysis can identify. This feature has MEDIUM priority as it enhances the basic static analysis but is more complex to implement.

* + 1. **Stimulus/Response Sequences**
* **Stimulus:** Static analysis is completed.
* **Response:** System prepares code for AI model analysis.
* **Stimulus:** AI model processing is completed.
* **Response:** System captures advanced pattern findings.
  + 1. **Functional Requirements**

**REQ-3.1:** The system must use pre-trained Hugging Face models for code understanding.

**REQ-3.2:** The system must identify patterns related to code maintainability.

**REQ-3.3:** The system must compare code against reference examples to identify deviation from best practices.

**REQ-3.4:** The system should identify potential logic errors that static analysis might miss.

**REQ-3.5:** The system should evaluate code complexity and suggest simplifications.

**REQ-3.6:** The system may identify security vulnerabilities in the code.

* 1. **Feedback Generation**
     1. **Description and Priority**

The system will generate clear, actionable feedback based on the results of static analysis and AI pattern recognition. This feature has HIGH priority as it directly impacts the usefulness of the system to developers.

* + 1. **Stimulus/Response Sequences**
* **Stimulus:** Analysis results are available.
* **Response:** System generates formatted feedback with appropriate detail level.
* **Stimulus:** Feedback is ready.
* **Response:** System posts feedback to GitHub PR.
  + 1. **Functional Requirements**

**REQ-4.1:** The system must generate feedback that includes issue description, location, and severity.

**REQ-4.2:** The system must format feedback using GitHub-compatible Markdown.

**REQ-4.3:** The system must provide inline comments for specific code lines when applicable.

**REQ-4.4:** The system must provide a summary comment with overall assessment.

**REQ-4.5:** The system should include suggestions for fixing identified issues.

**REQ-4.6:** The system should include references to relevant coding standards or documentation.

**REQ-4.7:** The system should use a constructive tone in feedback to encourage improvement.

* 1. **Admin Dashboard**
     1. **Description and Priority**

The system will provide a web-based dashboard for configuration, monitoring, and visualization of analysis metrics. This feature has MEDIUM priority as it enhances usability but is not critical for the core functionality.

* + 1. **Stimulus/Response Sequences**
* **Stimulus:** User navigates to dashboard URL.
* **Response:** System presents authentication screen.
* **Stimulus:** User logs in.
* **Response:** System displays dashboard with relevant metrics and options.
  + 1. **Functional Requirements**

**REQ-5.1:** The system must provide a secure login mechanism for authorized users.

**REQ-5.2:** The system must display key metrics such as:

* Number of PRs analysed.
* Common issues found.
* Resolution rates.
* Analysis time.

**REQ-5.3:** The system must allow configuration of analysis rules and thresholds.

**REQ-5.4:** The system must allow management of connected repositories.

**REQ-5.5:** The system should provide filtering and searching of historical analysis results.

**REQ-5.6:** The system should allow export of metrics in common formats (CSV, JSON).

**REQ-5.7:** The system may provide user management for dashboard access control.

**5. Other Nonfunctional Requirements**

**5.1 Performance Requirements**

REQ-P1: The system must complete analysis of a typical PR (up to 10 files, 1000 lines of code) within 5 minutes.

REQ-P2: The system must support concurrent analysis of at least 5 PRs simultaneously.

REQ-P3: The dashboard must load initial data within 3 seconds with standard internet connectivity.

REQ-P4: The system should scale to handle up to 100 PRs per day.

REQ-P5: Database queries must complete within 1 second for dashboard operations.

**5.2 Safety Requirements**

REQ-S1: The system must not modify code in repositories directly.

REQ-S2: The system must maintain separation between analysis environments to prevent cross-contamination.

**5.3 Security Requirements**

REQ-SEC1: All communications must use HTTPS encryption.

REQ-SEC2: GitHub access tokens must be stored using secure credential management.

REQ-SEC3: User authentication must use industry-standard practices (OAuth, JWT).

REQ-SEC4: The system must not expose sensitive code to unauthorized parties.

REQ-SEC5: The system must log access attempts and system operations for audit purposes.

REQ-SEC6: The system should support role-based access control for the dashboard.

**5.4 Software Quality Attributes**

REQ-Q1: The system must achieve 99% uptime during business hours.

REQ-Q2: The system should have a modular architecture to allow easy extension for additional languages.

REQ-Q3: The system must be containerized for easy deployment and scaling.

REQ-Q4: The system should have comprehensive error handling to prevent crashes.

REQ-Q5: The system should include monitoring and alerting for operational issues.

REQ-Q6: The system should have at least 80% test coverage for critical components.

**5.5 Business Rules**

REQ-B1: Only authorized team members may modify rule configurations.

REQ-B2: Analysis results are visible to all contributors on a PR.

**6. Other Requirements**

REQ-O1: The system must maintain a history of analysis results for trend analysis.

REQ-O2: The system should support internationalization for error messages and feedback.

REQ-O3: The system must provide documentation for setup, configuration, and usage.

REQ-O4: The system should be designed to facilitate extension to additional programming languages in future versions.

**Appendix A: Glossary**

1.**PR**: Pull Request, a GitHub feature for proposing code changes

2.**ESLint**: A static code analysis tool for identifying problematic patterns in JavaScript code

3.**Webhook**: A method for an app to provide real-time information to another app

4.**API**: Application Programming Interface

5.**MVP**: Minimum Viable Product

6.**Static Analysis**: Examination of code without executing it

7.**AI Model**: Artificial Intelligence algorithm trained to recognize patterns

8.**Docker**: Platform for developing, shipping, and running applications in containers

**Appendix B: Analysis Models**

[To be added in future versions]

**Appendix C: To be determined list**

1.Specific pre-trained AI models to be used

2. Detailed performance benchmarks for various repository sizes

3. Complete set of custom ESLint rules

4. Dashboard UI mockups

5. API endpoint specifications for external integration