Code Quality Intelligence Agent

Context

Modern software teams struggle to maintain code quality at scale. Between code reviews, testing, security, and managing technical debt, developers spend significant time on quality assurance tasks that could be automated or augmented with intelligent tooling.

Your challenge is to design and build an **AI-powered Code Quality Intelligence Agent** that analyzes code repositories and generates actionable, developer-friendly reports. The agent should go beyond simple linting — understanding structure, detecting real issues, and providing practical insights to help developers improve their codebases.

Problem Statement

Build an interactive **Code Quality Intelligence Agent** that can:

- **Analyze Code Repositories**
  - Accept local files, folders, or entire codebases as input.
  - Support multiple programming languages (**at least 2 required**).
  - Understand relationships between different parts of the code.

- **Identify Quality Issues -** Detect at least **3 categories** of concerns, such as:
  - Security vulnerabilities
  - Performance bottlenecks
  - Code duplication
  - Complexity issues
  - Testing gaps
  - Documentation issues
  - Any other

- **Generate Quality Reports**
  - Produce a detailed report about the codebase.
  - Explain *why* something is an issue.
  - Suggest *how* to fix it.
  - Prioritize issues based on severity and impact.

- **Provide Interactive Q&A**
  - Let developers ask natural-language questions about the codebase.
  - Provide conversational, clear answers.
  - Support follow-up questions and clarifications.

Bonus Layers

- **Web Deployment**: Accept a GitHub repo URL and run analysis directly on the web.
- **Richer Insights**: Visualize dependencies, hotspots, or quality trends.
- **Integrations**: Connect with GitHub/GitLab APIs for pull request reviews or CI/CD checks.

Super Stretch (For 120% Candidates)

If you really want to challenge yourself and show engineering depth, try:

- **RAG (Retrieval-Augmented Generation)** for handling very large codebases.
- **AST Parsing** for precise structural analysis.
- **Agentic Patterns** to manage reasoning and multi-step workflows.
- **Automated Severity Scoring** to prioritize issues intelligently.
- **Developer-Friendly Visualizations** for reports (e.g., dashboards, charts).

Design & Implementation Requirements

Engineering Depth

- Prefer popular frameworks like Langchain, Langgraph, Autogen, CrewAI, etc for building agents
- Clean, modular architecture with reusable components.
- Explore techniques like AST parsing, agentic design patterns, and RAG.
- Support multiple languages (at least 2).

Provide a simple CLI entry point:

```
<your-agent-command> analyze <path-to-code>
```

User Experience

- Reports should be **comprehensive but easy to read**.
- Interactive Q&A should feel natural and conversational.
- Any deployment (if attempted) should be simple to use (e.g., lightweight web UI or API).

Creativity

- Extra credit for unique features that real developers would value, such as severity scoring, dependency graphs, or trend tracking.

Deliverables

1. **Working Agent**
    - Runnable locally with setup instructions.

- CLI interface with analysis command.
- Must support at least 2 programming languages.
- Extra credit for deployed agent (web or API).
- **Code**: Upload your project to a GitHub repository and share the link. Important - Use a non-identifiable / obfuscated project name to avoid plagiarism.

2. **Documentation**
   - README.md: Setup and usage examples.
   - Architecture Notes: High-level design decisions (include diagrams if helpful).
   - Major design decisions and trade-offs.
   - Technical Notes: Any external tools, APIs, or libraries used. Any creative features or integrations you added.

3. **Video Demo (5–7 minutes)**
   - How to run the agent on a repo.
   - Example quality report(s).
   - Demo of interactive Q&A with the agent.
   - Use your own repo or a sample repo for demonstration.
   - Key engineering and design decisions.
   - Challenges you faced and how you solved them.

Evaluation Criteria

1. **Feature Depth Across Levels**
   - **Core Features**: Basic repository analysis, multi-language support, and issue detection.
   - **Bonus Layers**: Did you extend the agent with deployment, richer insights, or integrations?
   - **Super Stretch**: Did you explore advanced techniques like RAG, AST parsing, or severity scoring?
   - Your evaluation will consider **how far you went across these levels**.

2. **Problem-Solving & Approach**
   - **Usefulness -** Reports and suggestions should be practical, actionable, and developer-friendly.
   - **Interactivity -** The Q&A interface should be conversational, clear, and support follow-ups.
   - **Creativity & Innovation -** Unique insights, integrations, or developer-focused features.

3. **Maintainability & Engineering Decisions**
   - Clean, well-structured, documented, and testable code.
   - Sound architectural choices and thoughtful engineering trade-offs.

4. **Communication -** Clarity of explanation in documentation and the demo video.

The best solutions are those that developers would actually want to use every day. Code Quality Intelligence is your chance to build something practical, intelligent, and fun — we're excited to see your creativity shine! ✨