**SmartSDLC – AI-Enhanced Software Development Lifecycle**

***Project* *Documentation***

****

**College: Government Arts College Trichy-22**

**Team Leader:N.Anandhi**

**Email:** [**anandhirajaneelasri@gmail.com**](mailto:anandhiraja28122005@gmail.com)

**Team Member 1:S.Kaviya**

**Email:** [**yavika907@gmail.com**](mailto:yavika907@gmail.com)

**Team Member 2:D.Dharshini**

**Email:** [**dharshinitrichy90@gmail.com**](mailto:dharshinitrichy19@gmail.com)

**Team Member 3:S.Swetha**

**Email:** [**kavithasangaramoorthy@gmail.com**](mailto:kavithasangaramoorthy@gmail.com)

## **1. Introduction**

SmartSDLC – AI-Enhanced Software Development Lifecycle is an intelligent approach to software engineering that integrates Artificial Intelligence into every stage of the traditional SDLC.

Unlike conventional models, SmartSDLC leverages AI for requirement analysis, code generation, automated testing, anomaly detection, and predictive project forecasting.

This results in faster development, reduced errors, continuous documentation, and smarter decision-making.

By combining human expertise with AI-driven automation, SmartSDLC ensures higher efficiency, adaptability, and quality in modern software projects.

**2. Project Overview**

**Purpose:** SmartSDLC leverages AI to optimize the entire Software Development Lifecycle (SDLC). By embedding intelligent automation, predictive analytics, and generative AI capabilities, it enhances productivity, reduces human error, accelerates delivery, and ensures higher quality outcomes.

**Features:**

* **AI-Assisted Requirements Analysis**
  + Converts stakeholder inputs into clear, structured requirements.
* **Code Generation & Review**
  + AI generates boilerplate code and assists in detecting bugs or security vulnerabilities.
* **Automated Test Case Generation**
  + Generates unit, integration, and regression test cases from requirements.
* **Project Timeline Forecasting**
  + Predicts delivery timelines using historical project data.
* **Anomaly Detection in Development Process**
  + Identifies unusual patterns in commits, builds, or deployments.
* **Continuous Documentation**
  + Automatically generates design documents, changelogs, and API docs.
* **Feedback & Retrospective Analysis**
  + Summarizes team retrospectives and suggests process improvements.

## **3. Architecture**

* **Frontend (Streamlit/Gradio):** Interactive dashboards for project status, test reports, documentation, and chat-based project assistance.
* **Backend (FastAPI):** Provides APIs for requirement analysis, code review, test generation, and documentation management.
* **AI Integration (LLMs & ML Models):**
  + Requirement summarization (LLMs)
  + Code generation & review (LLMs + static analysis tools)
  + Test case generation (LLMs + ML models)
  + Project forecasting (time-series ML models)
* **Vector Database (e.g., Pinecone/FAISS):** Stores historical project data, requirements, and code snippets for semantic search.
* **CI/CD Integration:** AI-enhanced pipelines for automated testing, deployment, and anomaly monitoring.

## **4. Setup Instructions**

**Prerequisites:**

* Python 3.9+
* FastAPI, Streamlit/Gradio
* Access to LLM APIs (IBM Watsonx / OpenAI / HuggingFace)
* Vector DB (Pinecone/FAISS)
* GitHub or GitLab integration for CI/CD

**Installation Process:**

1. Clone repository
2. Install dependencies (pip install -r requirements.txt)
3. Configure .env for API keys & DB credentials
4. Start backend (FastAPI)
5. Launch frontend dashboard (Streamlit)

## **5. Folder Structure**

app/ # Backend logic

app/api/ # API routes for AI features

ui/ # Frontend components

models/ # ML/LLM integration modules

utils/ # Helper scripts

sdlc\_dashboard.py # Entry point for Streamlit dashboard

code\_reviewer.py # AI-assisted code review

test\_generator.py # Automated test generation

forecasting.py # Project timeline prediction

doc\_generator.py # AI documentation module

## **6. Running the Application**

1. Run FastAPI server → exposes AI endpoints.
2. Run Streamlit dashboard → interactive interface.
3. Upload requirements, code, or test files.
4. Interact with modules (analysis, review, forecast, docs).

## **7. API Documentation**

* **POST /analyze-requirements** → Structured requirement analysis
* **POST /generate-code** → AI code suggestions
* **POST /review-code** → Detects bugs & vulnerabilities
* **POST /generate-tests** → Automated test cases
* **GET /forecast-timeline** → Predicts project completion date
* **GET /generate-docs** → Returns AI-generated documentation

## **8. Authentication**

* JWT-based token authentication
* Role-based access: Developer, Tester, Manager
* Optional OAuth2 for enterprise integration

## **9. User Interface**

* Sidebar navigation (requirements, code, tests, docs, forecasts)
* AI chat assistant for project Q&A
* KPI dashboard (velocity, bug trends, test coverage)
* Real-time report generation (downloadable as PDF)

## **10. Testing**

* **Unit Testing** → AI modules
* **Integration Testing** → API & CI/CD pipelines
* **Performance Testing** → Scalability of ML/LLM models
* **Manual Testing** → UI interactions

**11. Known Issues**

* LLM may generate verbose or ambiguous requirements.
* Test generation may miss edge cases.

**12. Future Enhancements**

* Multi-agent AI collaboration for DevOps.
* Integration with Jira, GitHub Actions, Jenkins.
* Predictive defect analysis across sprints.
* AI-driven resource allocation suggestions.

