SMARTSDLC – AI-ENHANCED SOFTWARE DEVELOPMENT LIFECYCLE

1. Introduction

Project Title: SmartSDLC – AI-Enhanced Software Development Lifecycle

Team Members:

- Team leader : Karthikeyan .M
- Team member : K.S.Balamurugan
- Team member : P.dhanush
- Team member : S.mohamed ismail

SmartSDLC is designed to modernize the conventional Software Development Lifecycle (SDLC) by embedding Artificial Intelligence and automation into every phase. It empowers developers, testers, and project managers with intelligent insights, reduces manual effort, and ensures faster, more reliable, and cost-effective software delivery.

2. Project Overview

Purpose

The purpose of SmartSDLC is to optimize the end-to-end software development process. Traditional SDLC models often face challenges like delays in requirement gathering, inaccurate project estimations, manual bottlenecks in testing, and inefficient bug detection. SmartSDLC integrates AI-powered modules for:

- Requirement analysis
- Project estimation
- Automated code reviews
- Intelligent test case generation
- Real-time defect prediction and monitoring

By doing so, SmartSDLC reduces risks, improves code quality, and enhances collaboration among stakeholders.

Key Features

- **AI-Powered Requirement Analysis**
  Extracts structured requirements from unstructured text documents using NLP.
- **Intelligent Project Planning**
  Predicts timelines, budgets, and risks based on historical data.

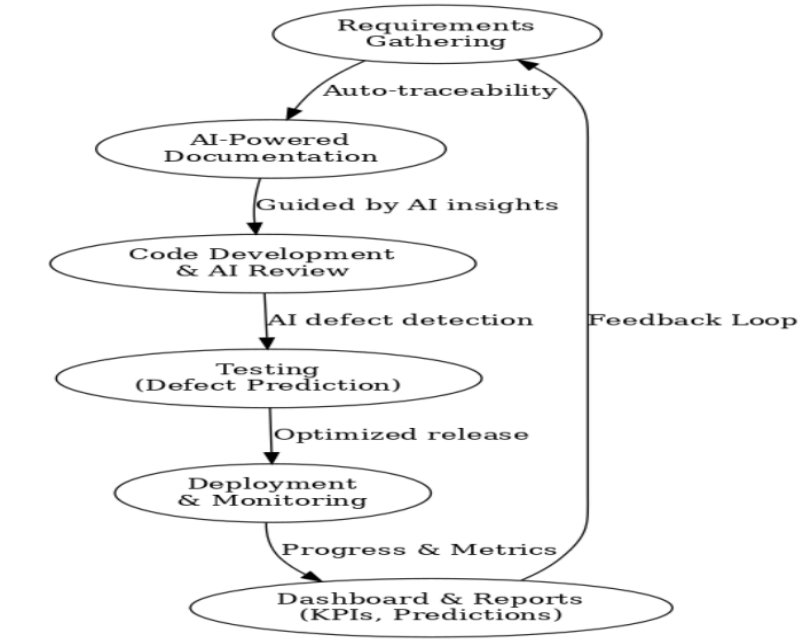
- **Automated Code Review**
  AI engine checks for vulnerabilities, best practices, and optimizations.
- **Test Case Generation**
  Generates unit and integration tests directly from code and requirements.
- **Defect Prediction**
  Identifies likely defect areas before deployment to minimize failures.
- **Continuous Monitoring & Anomaly Detection**
  Tracks application performance, security, and stability in real time.
- **AI Knowledge Assistant**
  A conversational chatbot for developers and managers to query project insights.
- **Dashboard & Reports**
  Interactive UI with visual KPIs, progress tracking, and downloadable reports.

3. Architecture

- **Frontend (React/Streamlit):**
  User-friendly dashboards for requirements input, progress tracking, and AI suggestions.
- **Backend (FastAPI/Django):**
  Exposes REST APIs for requirement analysis, code reviews, testing, and monitoring.
- **AI/ML Modules:**
  - NLP models for requirements analysis
  - ML models for defect prediction
  - Deep learning models for automated code review
- **Vector Database (FAISS/Pinecone):**
  Stores embeddings of project documents for semantic search.
- **CI/CD Integration:**
  Automated pipelines for testing, security scans, and deployment.
- **Monitoring System:**
  Real-time anomaly detection with logs, metrics, and alerting systems.

SmartSDLC – Architecture Flow

Architecture Flow Diagram



The flow diagram illustrates how SmartSDLC enhances each stage of the Software Development Lifecycle (SDLC): 1. Requirements are gathered and auto-documented by AI. 2. AI ensures requirement traceability and guides code development. 3. AI models analyze and predict defects during testing. 4. Deployment is optimized with AI-driven monitoring. 5. Dashboards provide KPIs, predictions, and reports. 6. Feedback from dashboards loops back to improve requirement gathering.

4. Setup Instructions

Prerequisites

- Python 3.9+
- Node.js (for frontend)
- Docker (optional, for deployment)
- Database access (PostgreSQL/MySQL)
- API keys for ML services

Installation Process

1. Clone the repository.
2. Install backend dependencies (pip install -r requirements.txt).
3. Install frontend dependencies (npm install or pip install streamlit).
4. Configure environment variables in .env file.
5. Start backend server with FastAPI/Django.
6. Run frontend dashboard with React/Streamlit.
7. Connect to CI/CD pipeline.

5. Folder Structure

app/ # Backend APIs and ML models

frontend/ # React/Streamlit UI

models/ # Pre-trained AI/ML models

tests/ # Unit and integration tests

utils/ # Helper scripts for logging, preprocessing

docs/ # Project documentation and reports

6. Running the Application

- Launch backend API server.
- Start frontend dashboard.
- Upload project documents/code.
- Explore dashboards for requirements, planning, testing, and monitoring.
- Download AI-generated reports.

7. API Documentation

- POST /analyze-requirements → Extracts structured requirements.
- POST /review-code → AI-powered code review.
- POST /generate-tests → Generates automated test cases.
- GET /predict-defects → Predicts potential defect areas.
- GET /project-status → Returns project health metrics.

8. Authentication

- JWT-based authentication for secure API access.
- OAuth2 for enterprise integration.

- Role-based access (Admin, Developer, Project Manager).

9. User Interface

- Sidebar navigation (Requirements, Planning, Testing, Monitoring).
- KPI dashboards with charts and analytics.
- Real-time chat assistant for queries.
- Downloadable AI-generated reports in PDF/CSV.

10. Testing

- **Unit Testing:** ML models and APIs.
- **Integration Testing:** Interaction between frontend, backend, and AI modules.
- **System Testing:** Validating performance and usability.
- **Edge Case Testing:** Large files, malformed inputs, unusual requirements.

12. Known Issues

- Limited dataset may reduce accuracy in defect prediction.
- Code review may differ across programming languages.
- Heavy computation needed for large enterprise-scale projects.

13. Future Enhancements

- In-IDE assistant integration (like GitHub Copilot).
- Expanded training datasets for better accuracy.
- AI-driven project cost estimation.
- Support for additional frameworks and languages.
- Blockchain-based traceability of project changes.