SmartSDLC – AI-Enhanced Software Development Lifecycle

Team ID: LTVIP2025TMID29288

Team Size: 4

Team Leader: Anamika Kumari Chouhan (22P31A0575)

Team member: Anushka Rajput (22P31A0578)

Team member: Nitin Kumar (22P31A05A9)

Team member: Vaishnavi Verma (22P31A05B8)

College: Aditya College of Engineering & Technology (Surampalem, A.P.)

Project Overview:

SmartSDLC is a full-stack, AI-driven platform that transforms the traditional Software Development Lifecycle (SDLC) by integrating Natural Language Processing (NLP), generative AI, and intelligent automation into each phase of development. Built using Python, Streamlit for the frontend, and optionally FastAPI for backend services, it leverages advanced AI models from Hugging Face or IBM Watsonx.

The platform allows users to upload unstructured requirement documents, which are automatically classified into SDLC phases and converted into structured user stories. It can generate clean, production-ready code from natural language prompts, create automated test cases, fix buggy code, and summarize complex logic into human-readable documentation. A built-in AI chatbot provides real-time support, helping users with SDLC-related queries. Overall, SmartSDLC reduces manual effort, accelerates development, and enhances software quality through intelligent automation.

Core Objectives:

- Automate multiple SDLC stages using generative AI
- Enable seamless translation of natural language into code, test cases, and documentation
- Provide real-time chatbot support to assist users with SDLC-related queries
- Improve software quality and development speed through AI-enhanced workflows

Scenarios:

Scenario 1: Requirement Upload & Classification

Purpose: This scenario automates the initial phase of the software lifecycle — gathering and organizing requirements.

How it works:

- The user uploads a PDF file that contains raw, unstructured requirements (like business goals or client expectations).
- The backend extracts text from the PDF using a library called PyMuPDF (also called fitz).
- AI processes each sentence and classifies it into categories like:
    - Requirements
    - Design
    - Development
    - Testing
    - Deployment
- These are displayed as structured user stories, which help in planning and assigning tasks.

Why it's useful: It saves time by replacing manual requirement analysis and helps developers better understand what needs to be built.

Scenario 2: AI Code Generator

Purpose: To quickly turn user stories or plain-text instructions into working code.

How it works:

- The user types a prompt like "Create a login system using Python and Flask."
- The prompt is sent to the Hugging Face or Watsonx generative model.
- The AI returns relevant, clean, and formatted code.
- The frontend displays the output with syntax highlighting for readability.

Why it's useful: Speeds up development, especially for repetitive or boilerplate code, and is helpful for junior developers or prototyping.

Scenario 3: Bug Fixer

Purpose: To help developers identify and fix coding errors without manually debugging.

How it works:

- The user pastes a snippet of buggy code (e.g., Python, JavaScript).
- The AI model analyzes the code and finds syntax or logical errors.
- It returns a corrected version, sometimes with explanations.

Why it's useful: Reduces the time spent on debugging and helps new developers learn best practices by example.

Scenario 4: Test Case Generator

Purpose: To automate the creation of test cases for validating code logic.

How it works:

- The user provides a code snippet or functional description (like "a function that calculates the area of a circle").
- The AI generates unit tests using frameworks such as unittest or pytest (for Python).
- These tests can be directly copied into the codebase for validation.

Why it's useful: Ensures better test coverage, promotes quality assurance, and saves time writing tests manually.

Scenario 5: Code Summarizer

Purpose: To generate human-readable explanations of source code.

How it works:

- The user submits a block of code.
- The AI reads the code and outputs a plain-English summary.
- It explains what the code does, what its inputs and outputs are, and how it might be used.

Why it's useful: Helpful for documentation, code reviews, or when onboarding new developers to understand existing logic.

Scenario 6: Floating AI Chatbot Assistant

Purpose: To assist users in real-time with SDLC questions and navigation.

How it works:

- A chatbot (using LangChain + Hugging Face or Watsonx) is embedded in the frontend.
- The user can ask questions like:
    - "How do I write a unit test?"
    - "What are the phases of SDLC?"
    - "What does this error mean?"
- The chatbot understands the question and gives an informative, AI-generated response.

Why it's useful: Acts like a virtual mentor or project assistant. Very helpful for new developers or non-technical users who need quick answers.

Purpose:

The purpose of the **SmartSDLC – AI-Enhanced Software Development Lifecycle** project is to **automate and optimize key phases of the software development lifecycle using AI technologies**, specifically IBM Watsonx Granite models and LangChain. It aims to:

1. **Reduce manual effort** in requirement analysis, code generation, testing, bug fixing, and documentation.
2. **Improve developer productivity and code quality** through AI-assisted features.
3. **Accelerate SDLC workflows** with faster and more intelligent tooling.
4. **Enable non-technical users** to interact with SDLC tasks using a simple Streamlit-based interface.
5. **Demonstrate the power of AI integration** in modern software engineering using cloud-native, modular architecture.