SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation:

Domain: Generative AI with IBM Cloud
Model: SDLC (Software Development Life Cycle)

---

1. Introduction

- **Project Title:** SmartSDLC – AI-Enhanced Software Development Lifecycle
- **Team Members:**
  - Medimi Srujana – Prompt Engineer
  - Gurajarla Varshini – AI Workflow Designer
  - Yalamati Om Sai Ram – Data Processor &
  - Tammisetti Indu N L R Padmanjani - Integration Specialist

---

2. Project Overview

- **Purpose:**
  SmartSDLC is designed to revolutionize traditional software development workflows by integrating IBM Watsonx-based generative AI into key SDLC phases such as requirements analysis, code generation, debugging, testing, and documentation.
- **Goals:**
  - Reduce manual effort in requirement classification
  - Automatically generate clean, production-ready code from prompts
  - Enable error detection and code summarization
  - Streamline software testing with AI-generated test cases
  - Provide real-time development help via an AI chatbot assistant

- **Core Philosophy:**
  Automate. Accelerate. Enhance SDLC using Generative AI.

---

3. Platform Capabilities (Scenarios)

- **Scenario 1: Requirement Upload & Classification**
  - Users upload raw PDF documents.
  - AI (via PyMuPDF + Watsonx Granite-20B) extracts and classifies sentences into SDLC phases.
  - Output: Structured user stories grouped by phase.
- **Scenario 2: AI Code Generator**
  - Input: Natural language prompt or user story.
  - Output: Production-ready code in supported languages.
  - AI Model: Watsonx with language context switching.
- **Scenario 3: Bug Fixer**
  - Input: Faulty code snippet.
  - Output: Error-free optimized version with explanation.
  - Languages Supported: Python, JavaScript.
- **Scenario 4: Test Case Generator**
  - Input: Code or requirement.
  - Output: AI-generated unit/integration test cases using unittest or pytest.
- **Scenario 5: Code Summarizer**
  - Input: Any code snippet/module.
  - Output: Human-readable documentation with purpose, logic, and usage.
- **Scenario 6: Floating AI Chatbot Assistant**
  - Integrated via LangChain.
  - Real-time help with SDLC queries.
  - Examples: “How to write test cases?”, “Explain DevOps.”

---

4. Architecture

- **Input Layer:**
  - PDF/Text/Code inputs provided by user via Web Interface or CLI.
- **AI Engine (Core Layer):**
  - IBM Watsonx Granite-13B/20B for:
    - NLP-based classification
    - Prompt-to-code
    - Error detection
    - Summarization
  - LangChain for chatbot memory and context
- **Output Renderer:**
  - Summaries, code, test cases, and classifications presented in structured formats.
- **Support Modules:**
  - PyMuPDF for PDF extraction
  - Prompt templates for scenario-specific workflows

---

5. Prerequisites

- Python 3.10+
- IBM Cloud account with Watsonx access
- IBM API Key & endpoint
- PyMuPDF, LangChain, OpenAI/WX SDK
- VS Code / Jupyter Notebook

---

6. Setup & Installation

1. **Clone the repo:**

bash

CopyEdit

git clone https://github.com/your-org/smartsdlc

cd smartsdlc

2. **Create virtual environment:**

bash

CopyEdit

python -m venv venv

source venv/bin/activate

3. **Install dependencies:**

bash

CopyEdit

pip install -r requirements.txt

4. **Set environment variables in .env:**

ini

CopyEdit

IBM_API_KEY=your_watsonx_api_key

IBM_ENDPOINT=https://us-south.ml.cloud.ibm.com

---

7. Core Workflows and Sample Code

Requirement Classification (Python)

python

CopyEdit

import fitz # PyMuPDF

from ibm_watsonx_ai.foundation_models import generate_classification

```
def classify_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    full_text = " ".join([page.get_text() for page in doc])
    classification = generate_classification(full_text)
    return classification
```

⚙️ **Prompt-to-Code Generator**

```
python
CopyEdit
prompt = "Build a REST API using Flask"
response = generate_code(prompt, model="granite-20b")
print(response)
```

---

8. AI Integration Details

Module	IBM Watsonx Role	Output
Requirement Classifier	Classifies sentences by SDLC phase	JSON with grouped phases
Code Generator	Converts prompt to runnable code	Python, JS, etc.
Bug Fixer	Fixes code errors	Cleaned-up source code
Test Case Generator	Converts requirement into tests	pytest/unittest code
Summarizer	Converts code into explanation	Markdown/Plain text
Chatbot Assistant	Provides interactive Dev help	Human-like chat response

---

9. User Interface

While SmartSDLC doesn't use a full frontend/backend stack, here’s how the output is presented:

Scenario	Input Method	Output Format
PDF Upload	CLI / Interface	JSON → Table
Code Prompt	Prompt Box	Syntax-highlighted code
Bug Fix	Code Snippet	Error-free code + explanation
Chatbot	Text	Conversational replies

---

10. Testing Strategy

- **Unit Testing:** Python unittest and pytest for input/output validation
- **Mock Testing:** Watsonx responses mocked with sample payloads
- **Integration Testing:** End-to-end testing of each scenario using CLI interface

---

11. Screenshots

SmartSDLC

Requirement Upload and Classification

Drop PDF document here or click to upload

Phase	User Story
Requirements	The system shall allow users to regisiter, login, and manage their profiles.
Development	Implement the user authentication module using JotT tokens.
Testing	Write unit tests to validate the functionality of autrientication module.
Deployment	Deploy the application to a cloud-based enviro-nment.

AI Code Generator

Enter prompt

Generate

```
def square(x.x)
    return x*x
```

Bug Fixer

```
1  pintx()
```

Fix

```
def square(x:x)
    return x * x
```

Test Case Generator

Enter code or requirement

Generate

```
import excepptions._test

class TestCaseMostErater_test

def TestCaseTestCase()
```

Code Summarizer

Enter code

Summarize

This function defines the square function, which takes an integer x and returns the square of x.

---

12. Known Issues

- Requires high-quality internet for model calls.
- Watsonx rate-limiting during peak load.
- Multi-language code generation is still experimental.

---

13. Future Enhancements

- Add voice-based prompts (STT input)
- Fine-tune a custom SDLC model using Watson Studio
- Add CI/CD integration for test and deploy automation
- Export all outputs to Confluence/Jira automatically
