**PROJECT REPORT**

# TITLE: SMART SDLC – AI-POWERED SOFTWARE-DEVELOPMENT-LIFE-CYCLE ASSISTANT USING IBM GRANITE

## 1. INTRODUCTION

### 1.1 PROJECT OVERVIEW

SMART SDLC is an intelligent assistant that automates and augments every major SDLC phase—requirements analysis, code generation, bug fixing, test-case creation, code summarization and conversational help—using IBM Watsonx Granite 13B and a Streamlit-based user interface.

### 1.2 PURPOSE

The system accelerates software delivery while improving code quality. It enables developers to prototype faster, eliminate repetitive tasks and gain instant insight into legacy code from a single dashboard.

## 2. IDEATION PHASE

### 2.1 PROBLEM STATEMENT

Developers spend significant time reading old code, writing boilerplate and tracking down bugs. Existing tools address only fragments of this workflow. A unified large-language-model copilot can remove those bottlenecks.

### 2.2 EMPATHY MAP CANVAS

SAYS "How do I fix this bug quickly?" "Can the AI write my unit tests?"

THINKS "Will the generated code be safe and maintainable?"

DOES Searches Stack Overflow, copies snippets, writes ad-hoc scripts

FEELS Stressed by deadlines, frustrated by repetitive tasks

PAINS Manual debugging, boilerplate coding, unclear legacy logic

GAINS Faster turnaround, fewer errors, clearer understanding

2.3 BRAINSTORMING

Standalone ideas such as bug fixer, test generator and story extractor were combined into one end-to-end assistant covering the complete SDLC.

## 3. REQUIREMENT ANALYSIS

3.1 CUSTOMER JOURNEY MAP

4. Start the Streamlit web application.

5. Select a module: Requirement, Code Generation, Bug Fix, Tests, Summary or Chat.

6. Provide input by uploading a PDF or code snippet.

7. The Granite model produces stories, code, fixes or answers.

8. Review the output, copy or download as needed.

9. Switch modules or end the session.

3.2 SESSION REQUIREMENTS

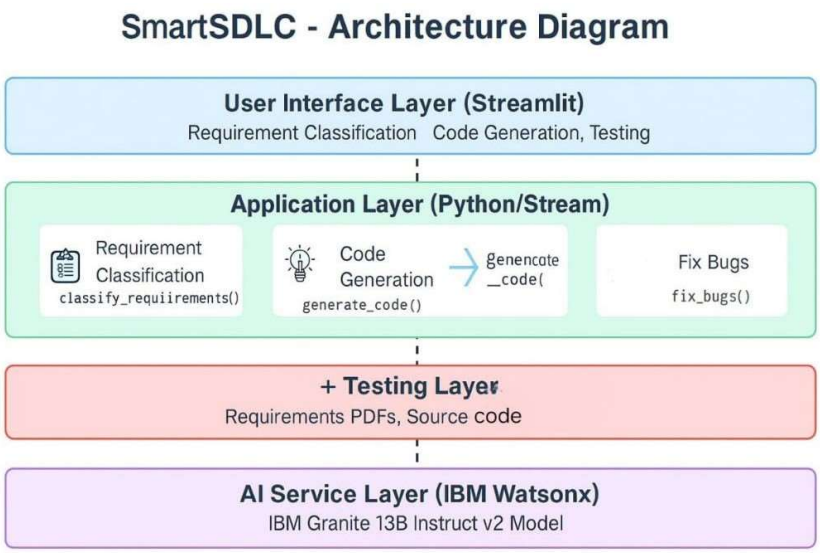
a) Upload PDF and code files.

b) Receive real-time AI responses.

c) Download generated assets.

d) Preserve chat history within the session.

3.3 DATA FLOW DIAGRAM




3.4 TECHNOLOGY STACK

Frontend Streamlit

Backend Python 3.11

AI Service IBM Watsonx Granite 13B Instruct v2

PDF Parsing PyMuPDF

Environment Management virtualenv and .env secrets

## 4. PROJECT DESIGN

4.1 PROBLEM–SOLUTION FIT

Teams need faster, higher-quality delivery. Embedding Granite LLMs inside daily tools provides intelligent automation that meets this need.

4.2 PROPOSED SOLUTION

Layer 1 User interface: individual Streamlit pages per module

Layer 2 Core logic: Python helpers for PDF handling, code cleanup and API calls

Layer 3 AI layer: cached Granite model accessed with secure credentials

4.3 SOLUTION ARCHITECTURE

UI Layer  Sidebar navigation, chat window, file widgets

Application Logic  app.py and pages route requests

Helper Layer  watson.py, pdf_utils.py, cleaning.py

AI Layer  Granite service with retry and rate-limit control

(Placeholder for architecture diagram)

## 5. PROJECT PLANNING AND SCHEDULING

Week 1 (12 Jun – 19 Jun)  Idea finalisation, Streamlit skeleton, PDF ingestion

Week 2 (20 Jun – 26 Jun)  Granite API integration, module logic, unit tests

Week 3 (27 Jun – 03 Jul)  Bug-fix loop, UI polish, report creation

Week 4 (04 Jul – 10 Jul)  Final demonstrations, documentation, deployment script

## 6. FUNCTIONAL AND PERFORMANCE TESTING

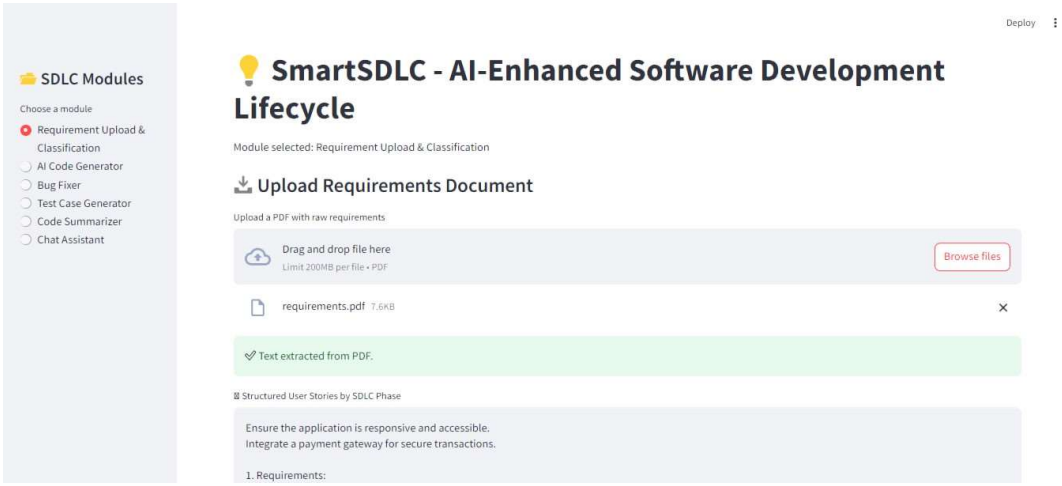
Unit testing  PDF parser and code-cleanup utilities

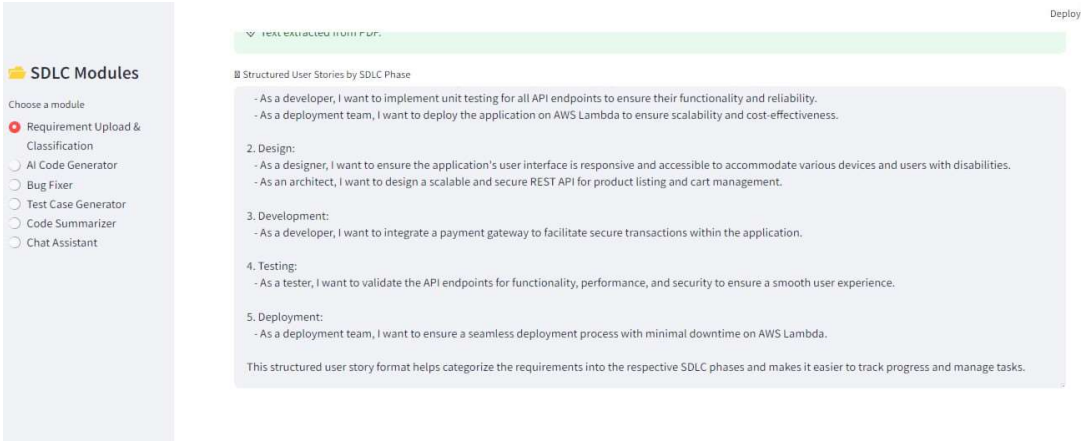
Integration testing  End-to-end Streamlit to Granite response

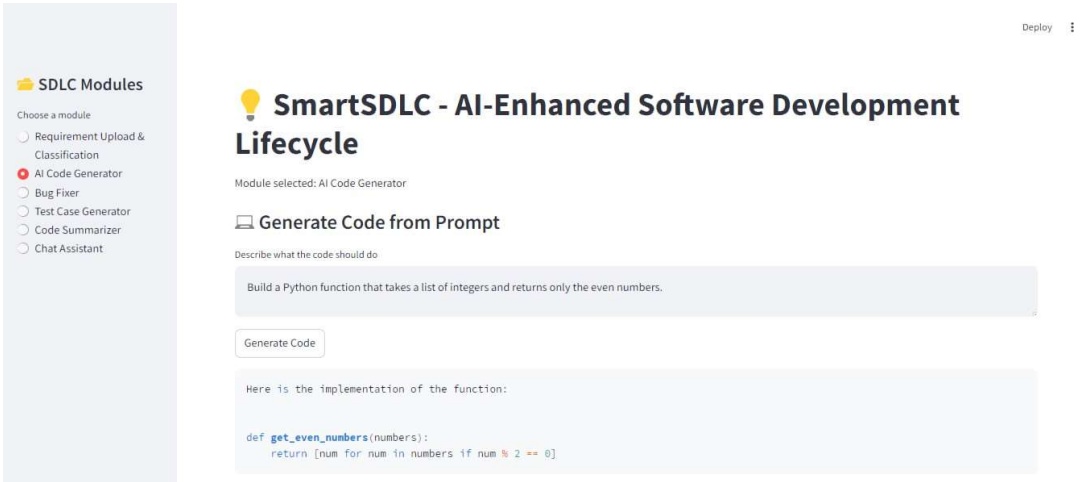
Manual testing  Real project PDFs and GitHub codebases

Error handling  Network drops, oversized files, API-quota exhaustion

## 7. RESULTS


SDLC Modules
Choose a module
Requirement Upload & Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Chat Assistant
Deploy
SmartSDLC - AI-Enhanced Software Development Lifecycle
Module selected: Requirement Upload & Classification
Upload Requirements Document
Upload a PDF with raw requirements
Drag and drop file here
Limit 200MB per file • PDF
Browse files
requirements.pdf 7.6KB
Text extracted from PDF.
Structured User Stories by SDLC Phase
Ensure the application is responsive and accessible.
Integrate a payment gateway for secure transactions.
1. Requirements:


SDLC Modules
Choose a module
Requirement Upload & Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Chat Assistant
Deploy
Structured User Stories by SDLC Phase
- As a developer, I want to implement unit testing for all API endpoints to ensure their functionality and reliability.
- As a deployment team, I want to deploy the application on AWS Lambda to ensure scalability and cost-effectiveness.
2. Design:
- As a designer, I want to ensure the application's user interface is responsive and accessible to accommodate various devices and users with disabilities.
- As an architect, I want to design a scalable and secure REST API for product listing and cart management.
3. Development:
- As a developer, I want to integrate a payment gateway to facilitate secure transactions within the application.
4. Testing:
- As a tester, I want to validate the API endpoints for functionality, performance, and security to ensure a smooth user experience.
5. Deployment:
- As a deployment team, I want to ensure a seamless deployment process with minimal downtime on AWS Lambda.
This structured user story format helps categorize the requirements into the respective SDLC phases and makes it easier to track progress and manage tasks.


SDLC Modules
Choose a module
Requirement Upload & Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Chat Assistant
Deploy
SmartSDLC - AI-Enhanced Software Development Lifecycle
Module selected: AI Code Generator
Generate Code from Prompt
Describe what the code should do
Build a Python function that takes a list of integers and returns only the even numbers.
Generate Code
Here is the implementation of the function:
def get_even_numbers(numbers):
return [num for num in numbers if num % 2 == 0]

Deploy

SDLC Modules

Choose a module

- Requirement Upload & Classification
- AI Code Generator
- Bug Fixer
- Test Case Generator
- Code Summarizer
- Chat Assistant

# 💡 SmartSDLC - AI-Enhanced Software Development Lifecycle

Module selected: Bug Fixer

## 🛠 Fix Buggy Code

Paste your buggy code here

```
def add_numbers(a, b)
  return a + b
```

Fix Code

```
Here is the corrected code:

def add_numbers(a, b):
    return a + b
The corrected code is:
The colon (:) was missing at the end of the function definition line in the erroneous code. This colon indicates the start of the
```

Deploy

SDLC Modules

Choose a module

- Requirement Upload & Classification
- AI Code Generator
- Bug Fixer
- Test Case Generator
- Code Summarizer
- Chat Assistant

# Lifecycle

Module selected: Test Case Generator

## 🧪 Generate Test Cases

Paste the function or requirement for test generation

```
def multiply(x, y):
  return x * y
```

Generate Test Cases

```
1. Test case for multiplying positive integers
2. Test case for multiplying negative integers
3. Test case for multiplying a positive integer by a negative integer
4. Test case for multiplying zero by any integer
5. Test case for multiplying a large integer by another large integer
6. Test case for multiplying by zero (edge case)

Here are the test cases using Python's unittest:

```python
```

Deploy

SDLC Modules

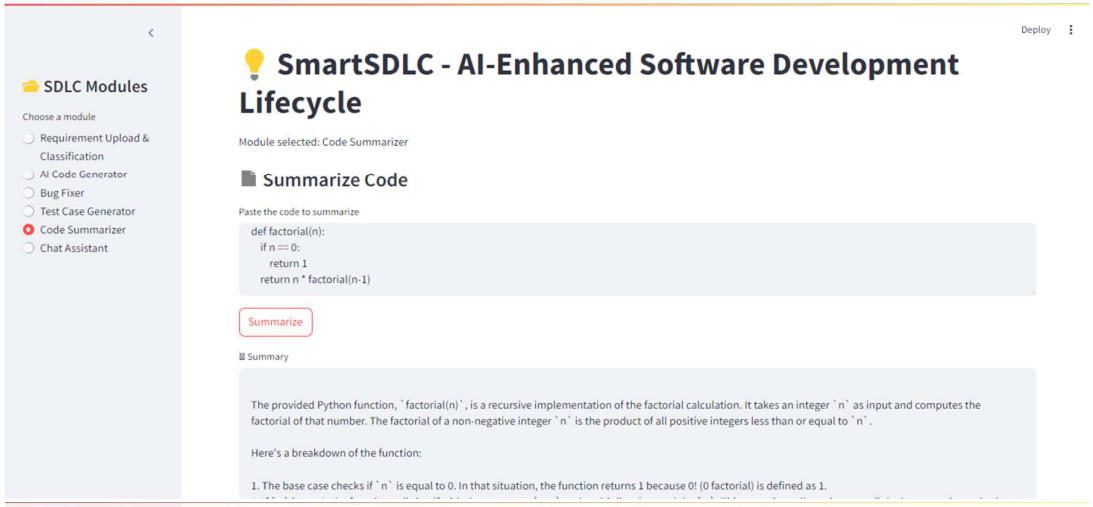
Choose a module

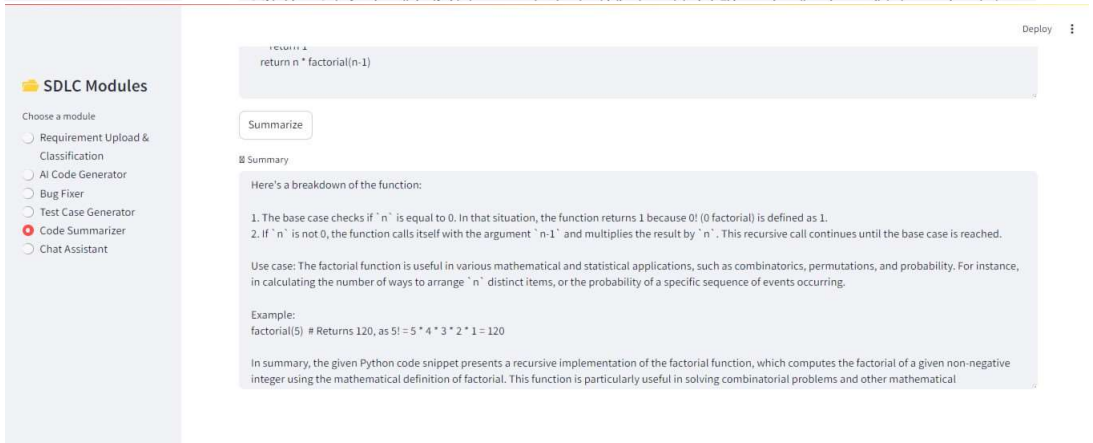
- Requirement Upload & Classification
- AI Code Generator
- Bug Fixer
- Test Case Generator
- Code Summarizer
- Chat Assistant

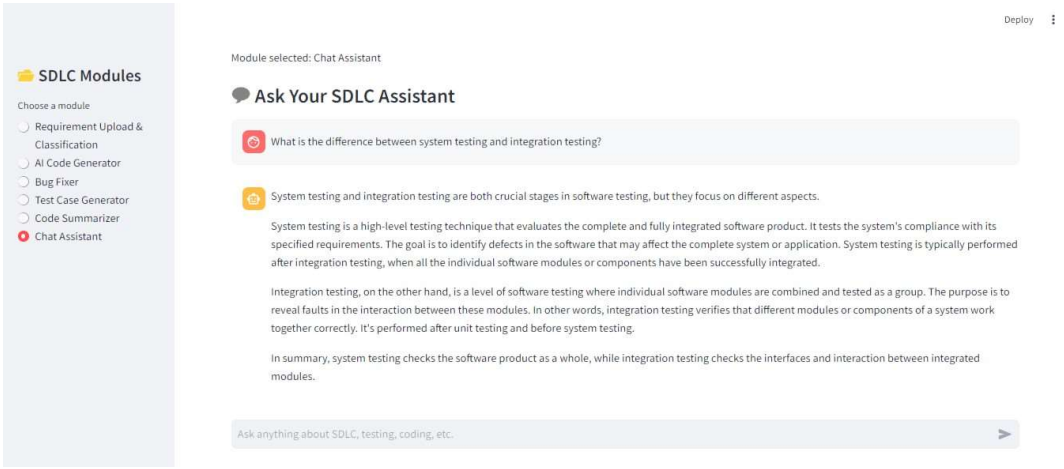
```
Here are the test cases using Python's unittest:

```python
import unittest

def multiply(x, y):
    return x * y

class TestMultiply(unittest.TestCase):

    def test_multiply_positive(self):
        self.assertEqual(multiply(5, 3), 15)

    def test_multiply_negative(self):
        self.assertEqual(multiply(-5, -3), 15)

    def test_mixed_signs(self):
        self.assertEqual(multiply(-5, 3), -15)

    def test_multiply_zero(self):
        self.assertEqual(multiply(0, 10), 0)

    def test_large_integers(self):
        self.assertEqual(multiply(123456789, 987654321), 121932631112635269)

    def test_multiply_by_zero(self):
        self.assertEqual(multiply(5, 0), 0)
```


Deploy
SDLC Modules
Choose a module
Requirement Upload & Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Chat Assistant
self.assertEqual(multiply(5, 0), 0)
if __name__ == '__main__':
unittest.main()
And here are the test cases using pytest:
python
def multiply(x, y):
return x * y
def test_multiply_positive():
assert multiply(5, 3) == 15
def test_multiply_negative():
assert multiply(-5, -3) == 15
def test_mixed_signs():
assert multiply(-5, 3) == -15
def test_multiply_zero():
assert multiply(0, 10) == 0
def test_large_integers():
assert multiply(123456789, 987654321) == 121932631112635269


Deploy
SDLC Modules
Choose a module
Requirement Upload & Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Chat Assistant
SmartSDLC - AI-Enhanced Software Development Lifecycle
Module selected: Code Summarizer
Summarize Code
Paste the code to summarize
def factorial(n):
if n == 0:
return 1
return n * factorial(n-1)
Summarize
Summary
The provided Python function, `factorial(n)`, is a recursive implementation of the factorial calculation. It takes an integer `n` as input and computes the factorial of that number. The factorial of a non-negative integer `n` is the product of all positive integers less than or equal to `n`.
Here's a breakdown of the function:
1. The base case checks if `n` is equal to 0. In that situation, the function returns 1 because 0! (0 factorial) is defined as 1.


Deploy
SDLC Modules
Choose a module
Requirement Upload & Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Chat Assistant
return n * factorial(n-1)
Summarize
Summary
Here's a breakdown of the function:
1. The base case checks if `n` is equal to 0. In that situation, the function returns 1 because 0! (0 factorial) is defined as 1.
2. If `n` is not 0, the function calls itself with the argument `n-1` and multiplies the result by `n`. This recursive call continues until the base case is reached.
Use case: The factorial function is useful in various mathematical and statistical applications, such as combinatorics, permutations, and probability. For instance, in calculating the number of ways to arrange `n` distinct items, or the probability of a specific sequence of events occurring.
Example:
factorial(5) # Returns 120, as 5! = 5 * 4 * 3 * 2 * 1 = 120
In summary, the given Python code snippet presents a recursive implementation of the factorial function, which computes the factorial of a given non-negative integer using the mathematical definition of factorial. This function is particularly useful in solving combinatorial problems and other mathematical




## 8. ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- Complete SDLC coverage in one tool
- Faster prototype-to-production pipeline
- High-quality code and summaries from Granite 13B
- Open and extensible foundation

DISADVANTAGES

- No user authentication yet
- Limited language support beyond Python (road-map)
- Internet connectivity required for AI service

## 9. CONCLUSION

SMART SDLC demonstrates that generative AI can streamline software engineering. By pairing Streamlit's simplicity with Watsonx power, it reduces development time and improves reliability, forming a basis for enterprise adoption.