Project Name: Smart sdlc - Ai Enhanced Software Development Lifecycle

Team ID : LIVIP2025IMID59931

Team Size : 4
Team Leader : Kanadam Sai Sowmya
Team Member : Kanala Sreenivasulu
Team Member : Kandukuri Shyam Kumar
Team Member : Katineni Hruday Kumarnaidu

Smartsdlc - Ai Enhanced Software Development Lifecycle

- **Smart SDLC** refers to an advanced, optimized approach to the traditional **Software Development Life Cycle (SDLC)**, integrating **automation, AI, analytic**, and Develops **practices** to improve efficiency, quality, and speed in software development.

❖ Introduction to Smart SDLC

- **Smart SDLC** is a modern, intelligent methodology for managing the end-to-end life cycle of software development. Unlike traditional SDLC models, which follow rigid, sequential steps (like Waterfall or even basic Agile), Smart SDLC uses **smart technologies and data-driven decisions** to streamline development, reduce errors, and respond to change more effectively.

At its core, SmartSDLC aim:

Automate repetitive tasks (e.g., testing, deployment, code review

Utilize real-time analytic to predict risks and improve planning

Leverage AI/ML for smarter decision-making in coding, testing, and resource management

Integrate Develops principles to ensure security from the start

Continuously deliver value with faster iteration cycles

- **Key Features of Smart SDLC**

Intelligent Planning: Uses historical data and AI to estimate timelines and resources accurately.

Automated Testing & Deployment: Integrates CI/CD pipelines for faster and error-free releases.

Collaborative Platforms: Enhances communication across developers, testers, and operations teams.

Security Integration (Develops): Security checks are embedded throughout the life cycle.

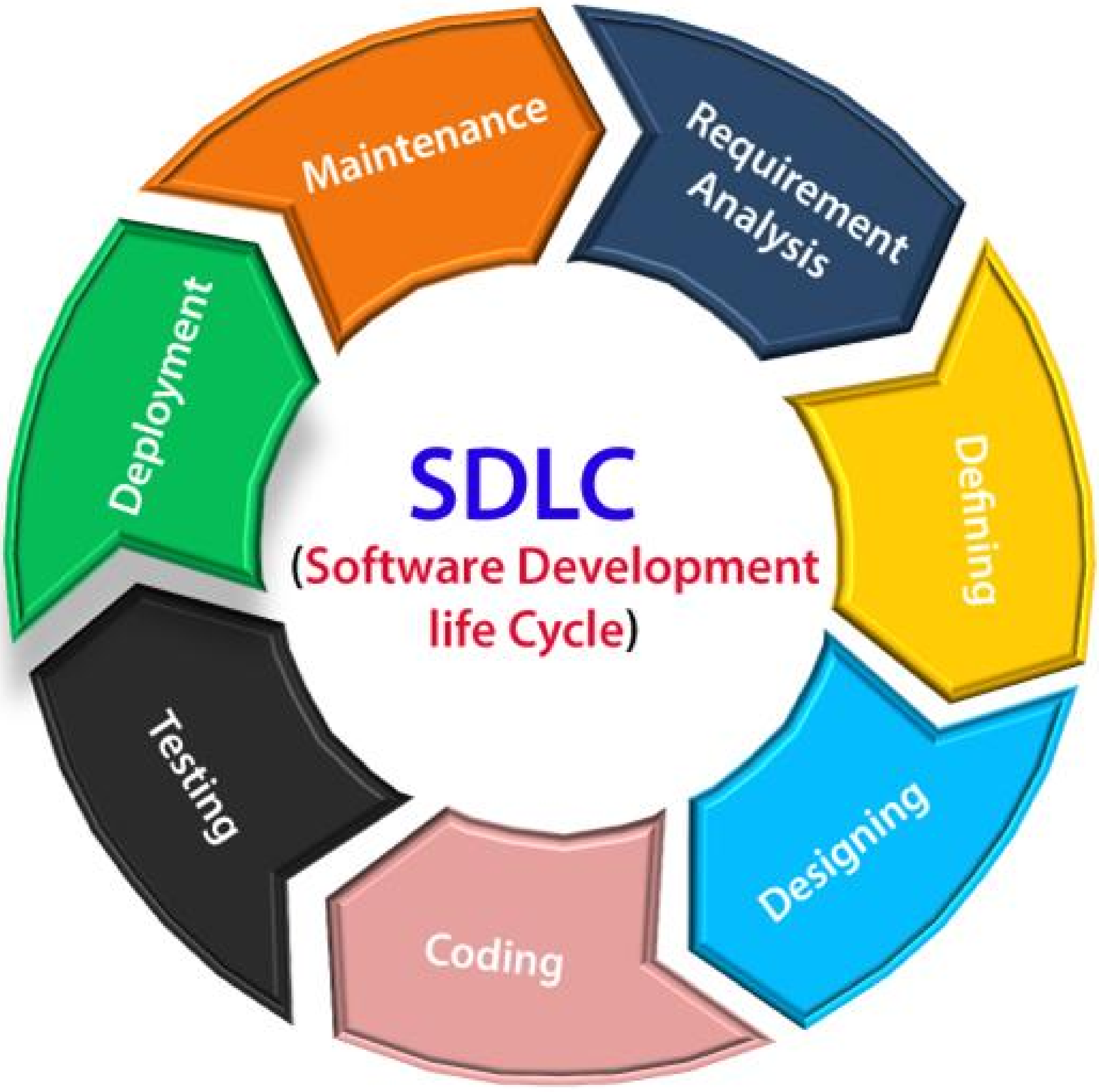
Continuous Feedback Loops: End-user and stakeholder feedback are continuously integrated into future iterations.

- **Benefits of Smart SDLC**

- ❖ Faster time-to-market
- ❖ Improved software quality
- ❖ Lower development and maintenance costs
- ❖ Higher adaptability to changing requirements
- ❖ Enhanced team productivity

SmartSDLC -Ai Enhnaced Software Development Lifecycle

Diagram



Pre-requisites:

- Python 3.10 : https://www.python.org/downloads/release/python-3100/
- FastAPI : https://fastapi.tiangolo.com/
- Streamlit : https://docs.streamlit.io/
- IBM Watsonx AI & Granite Models:https://www.ibm.com/products/watsonx-ai/foundation-model

- LangChain :https://www.langchain.com/
- Uvicorn: https://www.uvicorn.org/
- PyMuPDF (fitz): https://pymupdf.readthedocs.io/en/latest/
- Git & GitHub: https://www.w3schools.com/git/git_intro.asp?remote=github
- Frontend Libraries

Activity 1: Model Selection and Architecture

- **Activity 1.1:**Presentation Layer (Front-end**)**
- **Activity 1.2:** Application Layer (Back-end Services)
- **Activity 1.3:**Data Layer
- **Activity 1.4:**Develops & CI/CD Layer
- **Activity 1.5:**AI/ML Layer (Optional but powerful)
- **Activity 1.6:** Security & Compliance Layer
- **Activity 1.7:** Integration Layer

Activity 2: Core Functionalities Development

- **Activity 2.1:** Requirements Analysis
- **Activity 2.2**:Feature Breakdown
- **Activity 2.3:** Define Data Structures
- **Activity 2.4:** Use AI for Assistance (Smart SDLC Element)
- **Activity 2.5:** .Unit Testing
- **Activity 2.5:**Documentation

Activity 3: App.py Development

- **Activity 3.1:** Project Example: Todo App with Smart SDLC Logic
- **Activity 3.2:** Sample core.py

Activity 4: Front-end Development

- **Activity 4.1:**Designing and Developing the User Interface
- **Activity 4.2:** Creating Dynamic Interaction with Back-end

Activity 5: Deployment

- **Activity 5.1:** Preparing the Application for Local Deployment
- **Activity 5.2 :** Testing and Verifying Local Deployment

Activity 6: Program and Output

Activity 6.1: Program

Activity 6.2: Output

Project Architecture For Smart SDLC System :

Milestone 1: Model Selection and Architecture

Activity 1.1: Presentation Layer (Front-end)

Purpose: Interface for developers, testers, project managers, and stakeholders.

Tech Stack: React.js / Angular / Vue.js

Features:

Dashboard for project tracking

Code and commit viewer

Build & test results

Alerts and notifications

Activity 1.2: Application Layer (Backend Services)

Purpose: Core business logic, orchestration of SDLC processes

Tech Stack: Node.js / Python (Flask/Django) / Java (Spring Boot) / .NET Core

Modules:

Project Management Module: Sprint planning, backlog, task assignment

Code Analysis Module: Static & dynamic code analysis using AI

Testing Module: Integration with test automation tools (e.g., Selenium, JUnit, Postman)

CI/CD Orchestration: Jenkins, Git Lab CI, or GitHub Actions integration

AI Insights Engine: ML-based predictions for defects, delivery times, bottlenecks

Security Scanner: Develops checks via tools like Reasonable, Snaky, or OWASP ZAP

Activity 1.3: Data Layer

Purpose: Store and manage structured and unstructured project data

Databases:

Relational: PostgreSQL / MySQL (for users, tasks, builds, releases)

No SQL: MongoDB / Elasticsearch (for logs, unstructured test results, analytic)

Data Lake / Warehouse (optional): For big data analytic and historical metrics

Activity 1.4: Develops & CI/CD Layer

Purpose: Automate code build, test, and deployment

Components:

Version Control: Git (GitHub, GitLab, Bitbucket)

CI/CD Pipelines: Jenkins, GitHub Actions, Circles, or Git Lab CI

Containerization: Docker

Orchestration: Kubernetes or Docker Swarm

Monitoring & Logging: Prometheus, Grafana, ELK Stack

Activity 1.5:AI/ML Layer (Optional but powerful)

Purpose: Predictive analytic and automation

Functions:

Bug prediction from commit history

Estimation of sprint velocity and developer productivity

Code suggestion & optimization

Anomaly detection in builds/tests

Activity 1.6:Security & Compliance Layer

Purpose: Ensure secure coding and regulatory compliance

Components:

Static Application Security Testing (SAST)

Dynamic Application Security Testing (DAST)

Dependency Scanning (e.g., OWASP Dependency-Check)

Compliance Checks (e.g., GDPR, SOC2)

Activity1.7: Integration Layer

Purpose: Connect with third-party tools and API s

Tools Supported:

Jira / Trello (task management)

Slack / MS Teams (notifications)

GitHub / Bitbucket / GitLab (code repositories)

TestRail / Zephyr (test management)

Milestone - 2:Core Functionalities Development

Activity 2.1: Requirements Analysis

Understand **what the user needs** (features, functions, limitations).

Example: "User should be able to create, update, and delete tasks."

Activity 2.2: Feature Breakdown

Split the core functionality into **smaller, manageable tasks**.

Example for a Todo App:

Add new task

View all tasks

Edit a task

Delete a task

Activity 2.3: Define Data Structures

Decide how to **store and manage data** (variables, lists, databases).

Example:

```
tasks = []
```

Activity 2.4: Use AI for Assistance (Smart SDLC Element)

Use AI tools or code suggestions to:

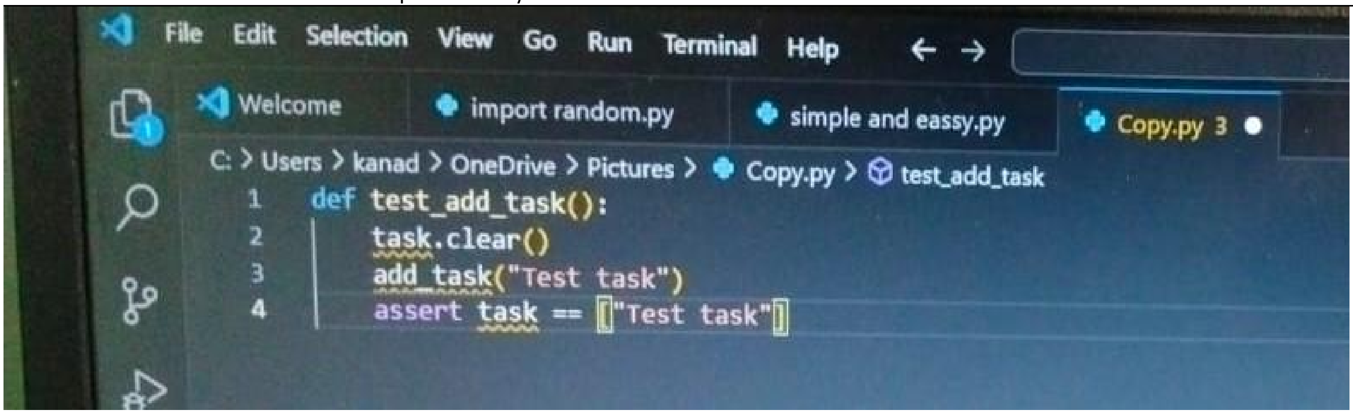
Generate boilerplate code

Recommend efficient algorithms

Detect bugs early

Activity 2.5:Unit Testing

Write tests to check each feature works correctly.

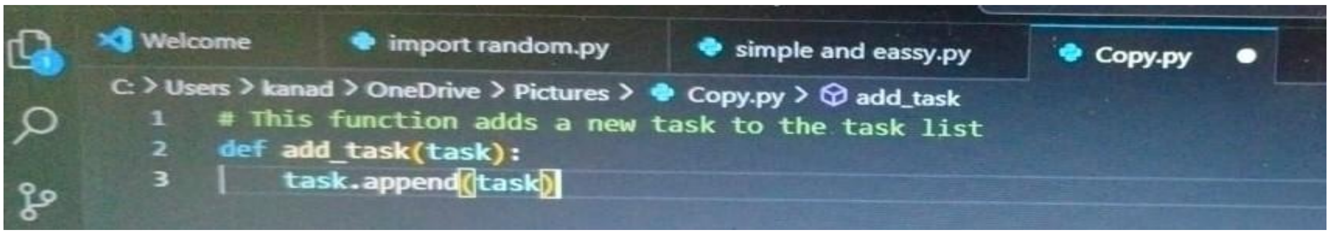

File Edit Selection View Go Run Terminal Help
Welcome
import random.py
simple and eassy.py
Copy.py 3
C: > Users > kanad > OneDrive > Pictures > Copy.py > test_add_task

```
def test_add_task():
    task.clear()
    add_task("Test task")
    assert task == ["Test task"]
```

Activity 2.6:Documentation

Explain how each function works for future developers (or yourself)

Example:


Welcome
import random.py
simple and eassy.py
Copy.py
C: > Users > kanad > OneDrive > Pictures > Copy.py > add_task

```
# This function adds a new task to the task list
def add_task(task):
    task.append(task)
```

Milestone - 3:Main.py Development

This milestone will:

Integrate your core functions (like add_task, view_tasks, etc.)

Provide a user interface (CLI)

Include basic AI logic for enhanced functionality (optional)

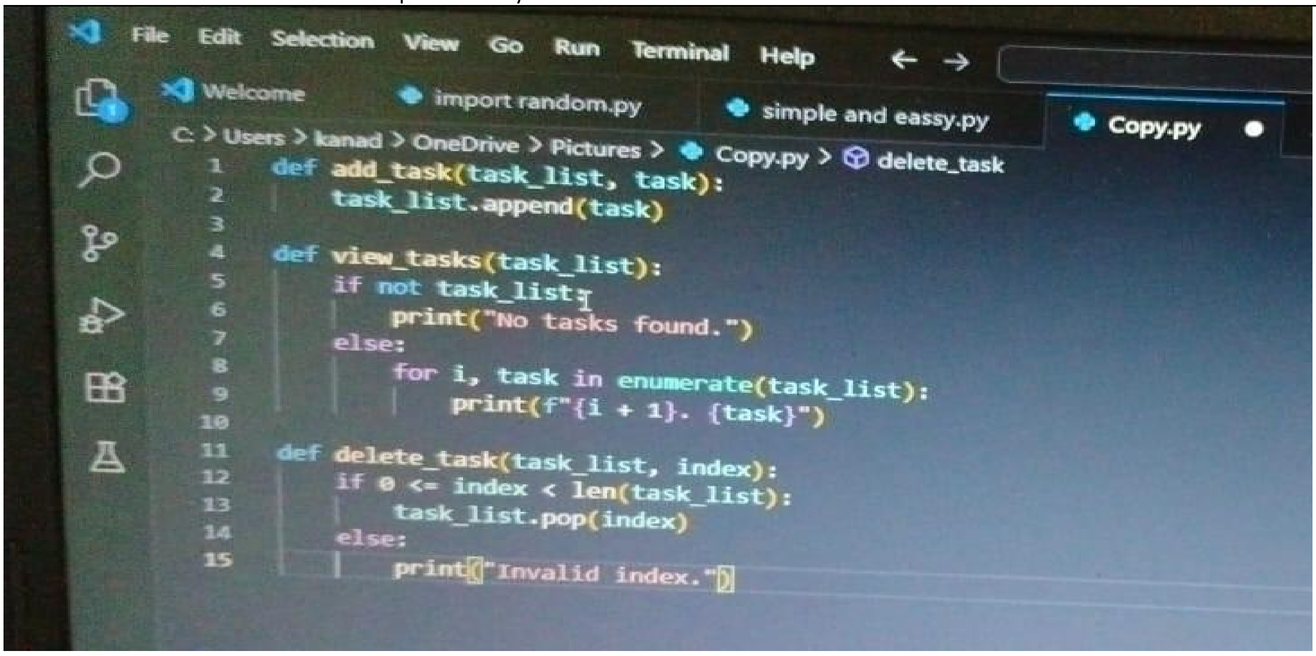
Be simple and easy to extend

Activity 3.1: Project Example: Todo App with Smart SDLC Logic

File Structure

```
Smart sdlc_project/
│
├── core.py          # Core functions
├── AI_helper.py       # Simulated AI suggestions (optional)
└── main.py           #  Main program (this milestone)
```

Activity 3.2: Sample core.py (Milestone 3):



```python
def add_task(task_list, task):
    task_list.append(task)

def view_tasks(task_list):
    if not task_list:
        print("No tasks found.")
    else:
        for i, task in enumerate(task_list):
            print(f"{i + 1}. {task}")

def delete_task(task_list, index):
    if 0 <= index < len(task_list):
        task_list.pop(index)
    else:
        print("Invalid index.")
```

Milestone - 4:Fronted Development

Activity 4.1: Designing and Developing the User Interface

1. Set Up the Base Streamlit Structure

- Create a main Home.py file that acts as the dashboard and entry point of the app.
- Add a welcoming hero section with a Lottie animation, a title, and a tagline.
- Organize features in a grid layout with clean navigation links to modular pages.

2. Design a Responsive Layout Using Streamlit Components

- Use st.columns(), st.container(), st.markdown() for layout control and consistency.
- Apply custom CSS styling for better fonts, backgrounds, and card shadows.
- Design a flexible layout that works well across different screen sizes and resolutions.

3. Create Separate Pages for Each Core Functionality

1. Build feature-specific modules under the pages/ directory:

- Upload_and_Classify.py: Upload PDF and classify requirements.
- Code_Generator.py: Convert user prompts into working code.
- Test_Generator.py: Generate test cases.
- Bug_Fixer.py: Automatically fix buggy code.
- Code_Summarize.py: Summarize code into documentation.

2. Connect each page to corresponding FastAPI endpoints via api_client.

Feedback.py: Collect user feedback.

Activity 4.2: Creating Dynamic Interaction with Backend

1. Integrate Fast-API with Streamline for Real-Time Content

- Use requests.post() or requests.get() inside api_client.py to interact with backend routes like /ai/generate-code, /ai/upload-pdf, etc.
- Ensure user inputs like uploaded files or prompt text are formatted and passed properly.
- Display AI responses using st.code(), st.success(), and markdown blocks for clarity.

2: Embed a Smart Floating Chat bot

- Add a minimal inline chat bot in Home.py using a Streamline form.
- Use the /chat/chat endpoint to send and receive real-time responses.
- Enhance interaction by showing emoji-based avatars and session memory.

Milestone 5: Deployment

Activity 5.1: Preparing the Application for Local Deployment

1. Set Up a Virtual Environment

- Create a Python virtual environment to manage dependencies and avoid conflicts with other projects.
- Activate the environment and install dependencies listed in requirements.txt to ensure all libraries (FastAPI, Streamlit, Watsonx SDK, etc.) are available.
- 2: Configure Environment Variables
- Set environment variables for sensitive data such as IBM Watsonx API key, model IDs, and database URLs.
- Create a .end file in your project root to securely store and load these settings during runtime.
- WATSONX_API_KEY=your_ibm_key_here
- WATSONX_PROJECT_ID=your_project_id
- WATSONX_MODEL_ID=granite-20b-code-instruct
- These values are loaded using python-dote inside your back-end (e.g., config.py).

Activity 5.2: Testing and Verifying Local Deployment

1: Launch and Access the Application Locally

- Start the Fast API back end using Unicorn: unicorn app.main:app --reload
- Run the Streamline front end: streamline run front end/Home.py

Open your browser and navigate to:

- Streamlit UI: http://localhost:8501
- FastAPI Swagger Docs: http://127.0.0.1:8000/docs
- Test each Smart SDLC feature (requirement upload, code generation, bug fixing, chat bot, feedback) to ensure everything is connected and functioning correctly.
- Run the Web Application
- Now type “streamlit run ????home.py” command
- Navigate to the local host where you can view your web page

Milestone 6: Program And Output

Activity 6.1: Program

```
        import random

# SmartSDLC class to handle each phase
class SmartSDLC:
    def __init__(self, project_name):
        self.project_name = project_name

    def planning(self):
        print("\nPlanning Phase:")
        ai_suggestion = random.choice(["Use Python and java", "Use Node.js and Express"])
        print(f"AI suggests: {ai_suggestion}")

    def design(self):
        print("\n Design Phase:")
        print("Designing simple and scalable architecture...")

    def implementation(self):
        print("\n Implementation Phase:")
        print("Writing clean and efficient code with AI help...")

    def testing(self):
        print("\n Testing Phase:")
        bugs = random.randint(0, 6)
        print(f"AI found {bugs} bug(s).")
        return bugs

    def deployment(self):
        print("\n Deployment Phase:")
        print("Application deployed to the cloud successfully.")

    def maintenance(self):
        print("\n Maintenance Phase:")
        suggestion = random.choice(["No issues detected", "Performance improvement needed"])
        print(f"AI Monitoring: {suggestion}")

    def run(self):
        print(f"\n Starting SMART SDLC for project: {self.project_name}")
        self.planning()
        self.design()
        self.implementation()

        bugs = self.testing()
        if bugs == 0:
            self.deployment()
        else:
            print("Fixing bugs with AI help...")
            self.implementation()
            self.testing()
            self.deployment()

        self.maintenance()
        print(f"\n Project '{self.project_name}' completed using SmartSDLC.\n")


# Run the process
if __name__ == "__main__":
    sdlc = SmartSDLC("Simple App")
    sdlc.run()
```

Activity 6.2:Output

Starting SMART SDLC for project: Simple App

Planning Phase:

AI suggests: Use Python and java

Design Phase:

Designing simple and scalable architecture...

Implementation Phase:

Writing clean and efficient code with AI help...

Testing Phase:

AI found 1 bug(s).

Fixing bugs with AI help...

Implementation Phase:

Writing clean and efficient code with AI help...

Testing Phase:

AI found 3 bug(s).

Deployment Phase:

Application deployed to the cloud successfully.

Maintenance Phase:

AI Monitoring: No issues detected

Project 'Simple App' completed using Smart SDLC.

Milestone -7: Conclusion

The Smart SDLC platform represents a significant advancement in the automation of the Software Development Life cycle by integrating AI-powered intelligence into each phase—from requirement analysis to code generation, testing, bug fixing, and documentation. By leveraging cutting-edge technologies like IBM Watson x, Fast API, Lang Chain, and Streamline, the system demonstrates how generative AI can streamline traditional software engineering tasks, reduce manual errors, and accelerate development timelines.

The platform’s modular architecture and intuitive interface empower both technical and non-technical users to interact with SDLC tasks efficiently. Features such as requirement classification from PDF s, AI-generated user stories, code generation from natural language, auto test case generation, smart bug fixing, and integrated chat assistance illustrate the power of AI when applied thoughtfully within a development framework.

Overall, Smart SDLC not only improves productivity and accuracy but also sets the foundation for future enhancements like CI/CD integration, team collaboration, version control, and cloud deployment. It is a step toward building intelligent, developer-friendly ecosystems that support modern agile development needs with smart automation at its core.