

# Smart SDLC - AI Enhanced Software Development Lifecycle

Generative AI with IBM

## Software Development Life Cycle



# 1. Introduction

The Smart SDLC represents an innovative software development approach that incorporates Artificial Intelligence (AI) tools and automation throughout the entire Software Development Lifecycle. This advanced methodology distinguishes itself from conventional SDLC models by utilizing AI capabilities to enhance planning efficiency, minimize human errors, expedite delivery timelines, and achieve superior quality results.

## Development Team

- Preethi Saravanan
- Sathya Priya D
- Narmada G
- Devisri M

## 2. Project Overview

Smart SDLC is a revolutionary software development framework that seamlessly integrates AI-powered tools and automation across all phases of the traditional Software Development Lifecycle. This approach transcends conventional methodologies such as Waterfall, Agile, and DevOps by leveraging artificial intelligence to enhance project planning, minimize errors, accelerate development cycles, and deliver superior outcomes.

## Key Features

**Conversational Interface** The platform's intelligence is driven by its conversational interface, enabling developers, quality assurance professionals, and stakeholders to engage with AI through natural language interactions. The system autonomously manages documentation, code generation, testing procedures, and deployment processes.

**Policy Summarization** Comprehensive policy frameworks within Smart SDLC ensure responsible AI implementation, safeguarding data integrity, maintaining quality standards, and preserving human oversight throughout the development process.

**Resource Forecasting** Advanced AI algorithms intelligently predict and allocate human resources, timeline requirements, budget considerations, tools, and infrastructure needs to ensure project completion within specified parameters and with appropriate team support.

**Eco-Tip Generator** Environmental consciousness is integrated through AI-generated sustainability recommendations at each development stage, promoting eco-friendly practices and energy-efficient software solutions.

**Citizen Feedback Loop** The system implements continuous feedback collection and analysis mechanisms, utilizing AI to process user input and enhance software intelligence, responsiveness, and user-centricity.

**KPI Forecasting** Predictive analytics monitor and forecast project success metrics including timeline adherence, cost management, quality assurance, and user satisfaction, enabling proactive issue resolution.

**Anomaly Detection** Continuous monitoring systems identify unusual patterns, errors, or potential risks across all development phases, facilitating rapid problem resolution and maintaining software reliability.

**Multimodal Input Support** The platform accommodates diverse interaction methods including text, voice commands, visual inputs, and sketches, making software development more intuitive and accessible to all stakeholders.

**Streamlit to Gradio UI Integration** The system utilizes Streamlit for comprehensive dashboards and project monitoring, while Gradio powers AI-driven conversational interfaces, creating an interactive and user-friendly development environment.

### 3. Architecture

#### Frontend Architecture

The frontend provides an AI-enhanced, multimodal user interface featuring integrated dashboards, intelligent chatbots, and feedback collection systems that facilitate seamless user interaction with the Smart SDLC platform.

#### Backend Architecture

The backend serves as the AI-powered processing engine, incorporating databases, business logic, and cloud infrastructure to handle inputs, execute AI models, and deliver results to the frontend interface.

#### LLM Integration

Large Language Model integration connects advanced AI capabilities to the development workflow, enabling conversational intelligence and automating complex tasks including requirement analysis, code generation, testing, and feedback processing.

**Vector Database Architecture** The vector storage system maintains project knowledge in vector format, allowing AI to access historical requirements, code repositories, bug reports, and feedback data for enhanced development speed and accuracy.

**ML Modules** Specialized machine learning components function as domain experts for specific areas including requirements analysis, coding assistance, testing automation, forecasting, and feedback processing, collaborating to optimize the development process.

## 4. Setup Instructions

### Prerequisites

- Python 3.8 or higher for AI/ML module support
- Streamlit or Gradio for frontend user interface development
- TensorFlow or PyTorch for machine learning model implementation
- LangChain or Transformers for Large Language Model integration
- Vector database solution (FAISS, Pinecone, or Weaviate)

### Installation Process

1. Install Python 3.8+ on your development environment
2. Execute `pip install -r requirements.txt` to install required dependencies
3. Configure a virtual environment for project isolation
4. Install Streamlit or Gradio for frontend interface
5. Set up and configure TensorFlow or PyTorch for ML model support

## 5. Directory Structure

- Frontend    User interface components
- Backend    API services and business logic
- ML\_modules    AI/ML intelligent modules
- LLM\_integration    Large Language Model configuration
- Vector\_store    Vector database and embeddings management
- Tests    Quality assurance and testing frameworks
- Configs    System configuration files
- Docs    Project documentation

## 6. Application Deployment

### Project Initialization

1. Clone the repository from GitHub using terminal
2. Navigate to the project directory: `cd Smart-SDLC-AI`
3. Create and activate a virtual environment
4. Install dependencies: `pip install -r requirements.txt`
5. Initialize database and vector store configurations
6. Configure API keys for OpenAI/LLM and database services
7. Launch backend server: `python backend/api/main.py`
8. Start frontend interface: `streamlit run frontend/app.py` or `gradio app.py`

## Frontend Deployment (Streamlit)

Access the frontend directory, execute Streamlit, open browser interface, and utilize the Smart SDLC user interface.

## Backend Deployment (FastAPI)

Navigate to backend directory, launch FastAPI server, access API documentation, and prepare backend for Smart SDLC UI integration.

## 7. API Documentation

The backend API infrastructure enables frontend and module interactions with Smart SDLC AI capabilities, including requirements analysis, code generation, automated testing, anomaly detection, KPI and resource forecasting, sustainability recommendations, and citizen feedback processing.

## 8. Authentication Framework

The authentication system provides multiple security benefits:

- Prevents unauthorized access to sensitive project information
- Protects AI models, source code, and feedback data
- Implements role-based access controls for enhanced security
- Establishes trust among users and project stakeholders

Smart SDLC authentication ensures verified user access through secure login mechanisms, token-based authentication, and role-based permissions to protect sensitive data and AI operations.

## 9. User Interface Design

The user interface provides several advantages:

- Makes AI functionality accessible to non-technical personnel
- Delivers real-time updates and visual analytics
- Facilitates cross-functional collaboration
- Enhances usability, efficiency, and decision-making capabilities

## 10. Testing Framework

Smart SDLC implements AI-powered testing methodologies that automatically generate test cases, identify defects, monitor system performance, and provide actionable feedback to ensure high-quality, reliable software delivery. The platform enhances traditional SDLC through AI automation, integrating LLM capabilities, ML modules, anomaly detection, KPI and resource forecasting, multimodal input support, and citizen feedback mechanisms for smarter, faster, and more reliable development processes.

## 11. Known Issues and Limitations

While Smart SDLC represents a significant technological advancement, several known limitations exist:

- Response Latency: Complex LLM queries may experience extended response times, causing minor interaction delays
- AI-Generated Content: Occasionally incomplete or requiring manual refinement for AI-produced code and suggestions
- Multimodal Input Challenges: Voice commands and image uploads may perform inconsistently in noisy environments or with unclear inputs
- Vector Store Retrieval: Rare instances of failed historical project data retrieval affecting AI context awareness



- False Positives: Automated testing and anomaly detection may occasionally flag normal operations as issues
- API Limitations: Extensive LLM API usage may encounter request limits
- Access Control Issues: Some users may experience restrictions due to misconfigured role-based access settings
- Infrastructure Dependencies: System performance relies on stable cloud infrastructure and consistent internet connectivity

The development team continuously monitors and enhances the system to improve reliability and operational efficiency.

## 12. Future Enhancement Roadmap

Smart SDLC's future development includes several strategic improvements:

**Advanced AI Integration:** Incorporation of more sophisticated LLMs and specialized AI models for enhanced code generation, requirement analysis, and testing recommendations with improved speed and accuracy.

**Real-time Collaboration:** Implementation of seamless collaborative features enabling simultaneous work by developers, testers, and project managers on shared projects.

**Enhanced Multimodal Capabilities:** Expansion of input methods including advanced voice recognition, sketch-to-code conversion, and video analysis for improved platform accessibility.

**Predictive Analytics Enhancement:** Strengthened project risk assessment, cost prediction, and resource management through historical data analysis and advanced machine learning algorithms.

**DevOps Integration:** Complete automation pipeline integration from development to production deployment, incorporating comprehensive DevOps toolchains.

# Project Screenshot

Health Ai.ipynb - C...  
b.research.google.com

Health Ai.ipynb  
File Edit View Insert Runtime Tools Help  
Commands + Code + Text ▶ Run all Copy to Drive  
Connect T4

```
( ) |pip install transformers torch gradio PyPDF2 -q  
( ) |pip install transformers torch gradio PyPDF2 -q  
( )  
  
import gradio as gr  
import torch  
from transformers import AutoTokenizer, AutoModelForCausalLM  
import PyPDF2  
import os  
  
# -----  
# Load Model and Tokenizer  
# -----  
# Using a smaller model for demonstration purposes.  
# You can replace with a better model, e.g., "EleutherAI/gpt-neo-1.3B"  
model_name = "gpt2"  
  
tokenizer = AutoTokenizer.from_pretrained(model_name)  
model = AutoModelForCausalLM.from_pretrained(model_name)  
  
# Check if CUDA is available and move the model to the appropriate device  
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  
model = model.to(device)  
  
# -----  
# Text Generation Helper  
# -----  
def generate_response(prompt, max_length=300):  
    # Encode the prompt with attention mask and handle potential truncation  
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512).to(device)  
    attention_mask = inputs["attention_mask"]  
  
    with torch.no_grad():  
        # Generate text using the model, ensuring pad_token_id is set  
        outputs = model.generate(  
            input_ids=inputs["input_ids"],  
            attention_mask=attention_mask,  
            max_length=max_length,  
            temperature=0.7,  
            top_p=0.9,  
            do_sample=True,  
            pad_token_id=tokenizer.eos_token_id # Set pad_token_id  
        )  
  
        # Decode the generated text, skipping special tokens and removing the prompt  
        response = tokenizer.decode(outputs[0], skip_special_tokens=True)  
        # Ensure the response doesn't include the original prompt  
        if response.startswith(prompt):  
            response = response[len(prompt):].strip()  
        return response.strip()  
  
# -----  
# PDF Text Extraction  
# -----  
def extract_text_from_pdf(pdf_file_path):  
    if pdf_file_path is None:  
        return ""  
  
    text = ""  
    try:  
        with open(pdf_file_path, 'rb') as file:  
            reader = PyPDF2.PdfReader(file)  
            for page in reader.pages:  
                text += page.extract_text() + "\n"  
    except FileNotFoundError:  
        return "Error: PDF file not found."  
    except PyPDF2.errors.PdfReadError:  
        return "Error: Could not read PDF file. It might be corrupted or encrypted."  
    except Exception as e:  
        return f"Error reading PDF: {str(e)}"  
  
    return text  
  
# -----  
# Eco Tips Generator  
# -----  
def eco_tips_generator(problem_keywords):  
    if not problem_keywords:  
        return "Please provide keywords to generate eco tips."  
    prompt = f"Generate practical and actionable eco-friendly tips for sustainable living related  
    return generate_response(prompt, max_length=200)  
  
# -----  
# Policy Summarization  
# -----  
def policy_summarization(pdf_file, policy_text):  
    content = ""
```

Variables Terminal

Health Ai.ipynb - C...  
b.research.google.com

Health Ai.ipynb  
File Edit View Insert Runtime Tools Help  
Commands + Code + Text ▶ Run all Copy to Drive  
Connect T4

```
( ) return generate_response(prompt, max_length=200)  
( )  
  
# -----  
# Policy Summarization  
# -----  
def policy_summarization(pdf_file, policy_text):  
    content = ""  
  
    if pdf_file is not None:  
        # Gradio provides a file object or path depending on version/configuration  
        # We need the path to open the file  
        pdf_file_path = pdf_file.path if hasattr(pdf_file, 'name') else str(pdf_file)  
        content = extract_text_from_pdf(pdf_file_path)  
        if content.startswith("Error"): # Handle potential PDF extraction errors  
            return content  
  
    if not content and policy_text:  
        content = policy_text  
  
    if not content:  
        return "No policy content provided (either PDF or text)."  
  
    summary_prompt = f"Summarize the following policy document and extract the most important poi  
    # Using a slightly larger max_length for summaries  
    return generate_response(summary_prompt, max_length=600)  
  
# -----  
# Gradio Interface  
# -----  
with gr.Blocks() as app:  
    gr.Markdown("🌱 Eco Assistant & Policy Analyzer")  
  
    with gr.Tab("📄 Eco Tips Generator"):  
        with gr.Row():  
            with gr.Column():  
                keywords_input = gr.Textbox(  
                    label="Environmental Problem/Keywords",  
                    placeholder="E.g., plastic, solar, water waste, energy saving...",  
                    lines=3  
                )  
                generate_tips_btn = gr.Button("Generate Eco Tips")  
  
            with gr.Column():  
                tips_output = gr.Textbox(label="Sustainable Living Tips", lines=8)  
  
        generate_tips_btn.click(  
            eco_tips_generator,  
            inputs=keywords_input,  
            outputs=tips_output  
        )  
  
    with gr.Tab("📄 Policy Summarization"):  
        with gr.Row():  
            with gr.Column():  
                pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".pdf"])  
                policy_text_input = gr.Textbox(  
                    label="Or paste policy text here",  
                    placeholder="Paste policy document text...",  
                    lines=8  
                )  
                summarize_btn = gr.Button("Summarize Policy")  
  
            with gr.Column():  
                summary_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)  
  
        summarize_btn.click(  
            policy_summarization,  
            inputs=[pdf_upload, policy_text_input],  
            outputs=summary_output  
        )  
  
# -----  
# Launch App  
# -----  
# Set enable_queue=True for better performance with multiple requests  
app.launch(share=True)
```


```
⚙ /usr/local/lib/python3.12/dist-packages/
The secret `HF_TOKEN` does not exist in
To authenticate with the Hugging Face Hu
You will be able to reuse this secret in
Please note that authentication is recom
warnings.warn(


tokenizer_config.json: 100% ██████████ 26.0/26.0 [00:00<00:00, 2.91kB/s]
config.json: 100% ██████████ 665/665 [00:00<00:00, 79.6kB/s]
vocab.json: 100% ██████████ 1.04M/1.04M [00:00<00:00, 1.50MB/s]
merges.txt: 100% ██████████ 456k/456k [00:00<00:00, 1.96MB/s]
tokenizer.json: 100% ██████████ 1.36M/1.36M [00:00<00:00, 1.51MB/s]
model.safetensors: 100% ██████████ 548M/548M [00:13<00:00, 46.4MB/s]
generation_config.json: 100% ██████████ 124/124 [00:00<00:00, 10.6kB/s]


Colab notebook detected. To show errors
* Running on public URL: https://6bb31e9
```

1:56

5G+

 **Eco Assistant & Policy Analyzer**

 Eco Tips Generator

 Policy Summarization

Environmental Problem/Keywords

E.g., plastic, solar, water waste, energy saving...

Sustainable Living Tips

Generate Eco Tips

Thank you...