SOFTWARE ENGINEERING LAB TASK 6

08-01-2025

HU22CSEN0100999

Eshwar Deshmukh Chavan

Aim:

Create a weather prediction model using a quadratic equation, following the Waterfall model approach.

Waterfall Model:

The Waterfall model is a step-by-step software development process where each phase is completed before moving to the next.

Phases:

Requirement Analysis

- Collect weather data (e.g., temperature, humidity).
- Define the model's requirements and outputs.

Design

- Create the quadratic formula: y=ax2+bx+cy = ax^2 + bx + cy=ax2+bx+c.
- Plan the process using flowcharts and data structures.

Implementation

- Write the code for the quadratic model.
- Use Python for better flexibility and visualization.

Verification

- Test the model using past weather data.
- Check if predictions are accurate.

Deployment

- Deliver the system for use in real-world scenarios.
- Ensure it's user-friendly and reliable.

Maintenance

- Update the model as needed for accuracy.
- Regularly review its performance.

Steps Of Implementation:

1. Import Libraries

   Imports NumPy for calculations and Matplotlib for plotting.

   1. Import Libraries

   ```
   import numpy as np
   import matplotlib.pyplot as plt
   ```

2. Define Quadratic Function

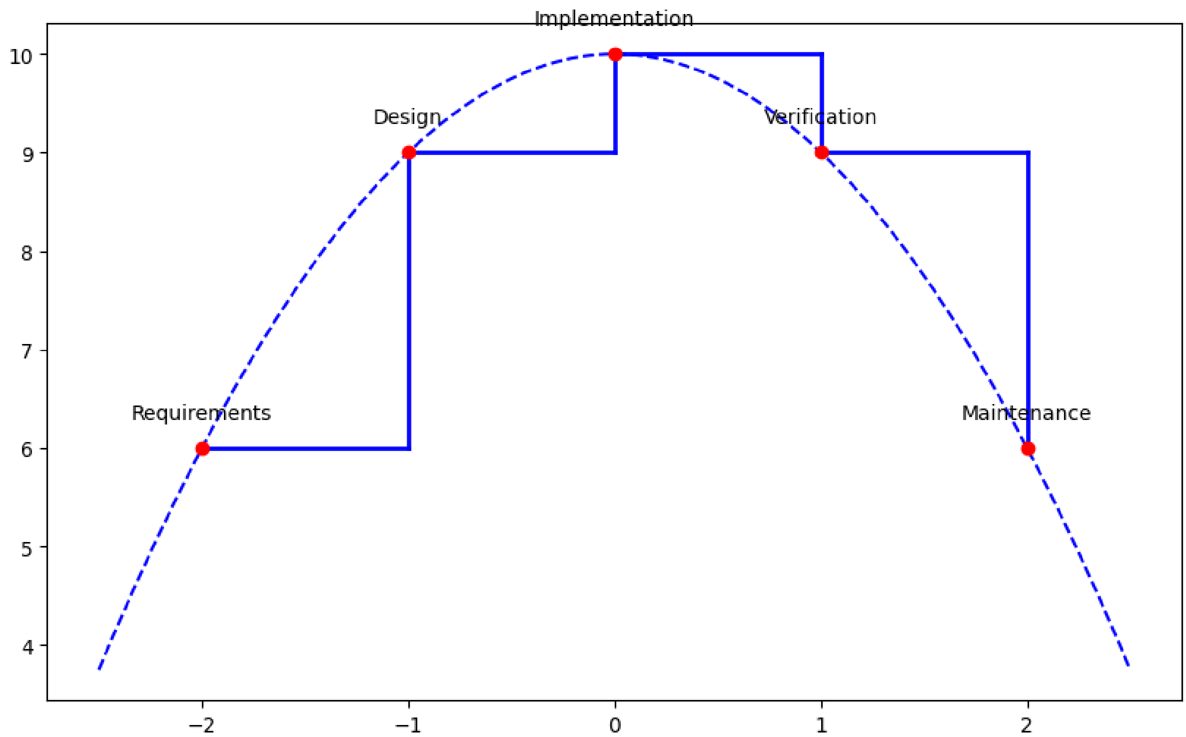
   Defines a quadratic function y=ax2+bx+c to model the stages of the Waterfall Model.

   ```
   def quadratic(x, a=-1, b=0, c=10):
       return a * x**2 + b * x + c
   ```

3. Waterfall Model Visualization with Quadratic Curve

Plots a quadratic curve representing the stages, with markers and vertical lines for the waterfall effect.

```python
stages = ["Requirements", "Design", "Implementation", "Verification", "Maintenance"]

# Generate equally spaced x positions
x_positions = np.linspace(-2, 2, len(stages))

# Compute y positions using the quadratic function
y_positions = quadratic(x_positions)  # Assuming quadratic function is defined elsewhere

# Plot the quadratic curve
x_curve = np.linspace(-2.5, 2.5, 500)
y_curve = quadratic(x_curve)

plt.figure(figsize=(10, 6))
plt.plot(x_curve, y_curve, color="blue", linestyle="--", label="Quadratic Curve")

# Add markers and connect them with vertical lines to create the waterfall effect
for i in range(len(stages)):
    if i > 0:
        # Draw a vertical drop to simulate a waterfall
        plt.plot([x_positions[i - 1], x_positions[i]], [y_positions[i - 1], y_positions[i - 1]], color="blue", linewidth=2)  # Horizontal segment
        plt.plot([x_positions[i], x_positions[i]], [y_positions[i - 1], y_positions[i]], color="blue", linewidth=2)  # Vertical drop

    # Place markers and labels
    plt.scatter(x_positions[i], y_positions[i], color="red", zorder=5)
    plt.text(x_positions[i], y_positions[i] + 0.3, stages[i], ha="center", fontsize=10)
```



4. Style and Show the Plot

Adds titles, labels, grid, and a legend, then displays the plot.

```
plt.plot(x_curve, y_curve, linestyle="--", color="green", label="Quadratic Curve")
plt.title("Model Representation Using Quadratic Curve", fontsize=12)
plt.xlabel("Stages of the Waterfall Model", fontsize=12)
plt.ylabel("Value", fontsize=12)
plt.axhline(0, color="black", linewidth=0.5, linestyle="--", alpha=0.7)
plt.axvline(0, color="black", linewidth=0.5, linestyle="--", alpha=0.7)
plt.grid(color="gray", linestyle="--", linewidth=0.5, alpha=0.7)
plt.legend()
plt.show()
```