The Saxena Model

# Shortcomings of the Traditional Waterfall Model

The Waterfall Model, while foundational in project management and software development, has several inherent drawbacks that often impede optimal progress. Here are the common problems associated with the Traditional Waterfall Model:

* No Feedback Loops Early On: Issues are often not discovered until the late stages of development, leading to costly fixes and rework.
* Rigid Structure: The model does not accommodate easy revisitation of earlier phases, making adjustments cumbersome.
* Poor Risk Management: Risks are typically not addressed until the implementation phase, which can jeopardize the project's success.
* Lack of Customer Involvement Midway: Customers have limited opportunities to verify progress and influence changes during the project's midpoint.
* Not Adaptive: Changing requirements once development starts is difficult, leading to potential misalignments with evolving customer needs.

# Step 2: The Saxena Model

To address these shortcomings, the saxena Model introduces enhancements aimed at improving communication, risk management, and customer involvement throughout the project lifecycle.

## Saxena Model (Additions are in CAPS)

* Communication
* Project initiation
* Requirements gathering -> Add Customer Feedback Loop
* Planning
* Estimating
* Scheduling
* Tracking -> Add Risk Assessment
* Modeling
* Analysis
* Design -> Add Design Review + Prototyping
* Construction
* Code
* Test
* Peer Validation -> Add Incremental Builds
* Deployment
* Delivery
* Support
* Feedback
* TAM -> Add Post-Deployment Retrospective

# Step 3: UML Diagram

# A diagram of a process AI-generated content may be incorrect.Step 4: Python Implementation (saxena.py)

Here’s a clean Python class that takes user inputs and outputs your Waterfall model:

def display\_menu():

    print("\n--- To-Do List Manager ---")

    print("1. View To-Do List")

    print("2. Add Task")

    print("3. Remove Task")

    print("4. Exit")

def view\_tasks(tasks):

    if not tasks:

        print("Your to-do list is empty.")

    else:

        print("\nYour To-Do List:")

        for idx, task in enumerate(tasks, 1):

            print(f"{idx}. {task}")

def add\_task(tasks):

    task = input("Enter the task to add: ")

    tasks.append(task)

    print(f"'{task}' has been added to your to-do list.")

def remove\_task(tasks):

    view\_tasks(tasks)

    if tasks:

        try:

            task\_num = int(input("Enter the task number to remove: "))

            removed = tasks.pop(task\_num - 1)

            print(f"'{removed}' has been removed from your list.")

        except (IndexError, ValueError):

            print("Invalid task number. Please try again.")

def main():

    tasks = []

    while True:

        display\_menu()

        choice = input("Choose an option (1-4): ")

        if choice == "1":

            view\_tasks(tasks)

        elif choice == "2":

            add\_task(tasks)

        elif choice == "3":

            remove\_task(tasks)

        elif choice == "4":

            print("Exiting To-Do List Manager. Goodbye!")

            break

        else:

            print("Invalid option. Please choose from 1 to 4.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Execution

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

References

Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner’s Approach. McGraw-Hill Education.

Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.