

**Name: Akash Yadav**

**SAP ID: 590014126**

**Batch: 12**

**Subject: Software Engineering**

**Topic: Disadvantages of Waterfall Model in SDLC.**

The standard flow usually looks like this:

1. **Requirements Analysis:** Gathering all possible requirements of the system to be developed.
2. **System Design:** Planning the programming languages, databases, and high-level architecture.
3. **Implementation (Coding):** The actual development of the software in small units.
4. **Integration & Testing:** Merging those units and testing the entire system for bugs.
5. **Deployment:** Delivering the product to the customer or the market.
6. **Maintenance:** Fixing issues that arise in the real world and updating the software.

Phases of waterfall model are ordered like:

- **Testing is the 4th step:** Because it comes after Implementation, you've already spent months (and a lot of budget) building the software before you even check if it actually works or meets the security standards we need in our specialization.
- **Deployment is the 5th step:** The client doesn't see anything until the very end. If they hate the UI design we did in Step 2, we don't find out until Step 5.
- **Sequential dependency:** You can't start testing until implementation is 100% done. If the coding phase gets delayed, the testing team gets squeezed for time.

Disadvantages of waterfall model consists of:

### **1. No Room for Change (Rigidity)**

The biggest headache with Waterfall is its linear nature. Once you move from the Requirements Analysis phase to Design, there's no going back. If the client realizes they need a new feature halfway through the coding phase, you're basically stuck. You have to finish the entire cycle before you can address changes.

## 2. The "Big Bang" Risk

In Waterfall model, you don't see a working product until the very end of the lifecycle. This is high-risk because if there was a misunderstanding during the initial requirement gathering, you won't realize it until the software is fully built. By then, fixing it is expensive and time-consuming.

## 3. Poor Requirement Clarity

It assumes that the customer knows exactly what they want right at the start. As we know from our own projects, requirements often evolve as you start seeing the UI or the logic come to life. Waterfall doesn't account for this discovery process.

## 4. Testing Happens Too Late

Testing only begins after the development is 100% complete. If you find a fundamental architectural flaw during the testing phase, you might have to scrap a massive chunk of code. In modern models like DevOps or Agile, we test continuously to avoid this.

## 5. High Maintenance Effort

Because changes are so hard to implement during the cycle, any missed requirements end up being pushed to the maintenance phase, making it bloated and difficult to manage.

