Lecture Notes on OOP

by Yuqun Zhang

1 Requirement engineering

Requirement engineering is one element of the *Waterful model*.

- Requirement engineering
- Design
- Implementation
- Testing (Verification)
- Maintenance

1.1 Waterfall model (AI generated)

V3.1> Requirement engineering is one element of the **Waterful model**.

- Requirement engineering
- Design
- Implementation
- Testing (Verification)
- Maintenance

explain this

The Big Picture: The Waterfall Model

The Waterfall Model is a **linear and sequential** approach to software development. The name "waterfall" comes from the idea that progress flows steadily downwards (like a waterfall) through these phases.

The key characteristic is that each phase must be **100% complete** and its output (documentation, design specs, code) approved before the next phase can begin. There is no going back to a previous phase easily; if a mistake is found late, it is very costly and disruptive to fix.

Explanation of Each Phase in the List

Your list outlines the five core phases of the traditional Waterfall Model:

1. Requirement Engineering (a.k.a. Requirements Gathering & Analysis)

- **What it is:** This is the **first and most crucial phase**. The development team works with the client and end-users to understand, document, and define **exactly what the software must do**. The goal is to capture all functional requirements (features) and non-functional requirements (performance, security, etc.).
- **Output:** A detailed **Software Requirements Specification (SRS)** document. This document is the foundation for all subsequent phases and is signed off by all stakeholders.

2. Design

- **What it is:** Using the SRS document from the previous phase, system architects and designers create the **blueprint for the system**. This phase defines *how* the system will be built. It involves:
  - **System Design:** Defining the overall system architecture, hardware setup, and software components.
  - **Detailed Design:** Defining the logic of each module, database structure, algorithms, and user interfaces.
- **Output: Design Document Specification (DDS)**. This is the roadmap for the programmers.

3. Implementation (a.k.a. Coding & Development)

- **What it is:** Programmers and developers start writing the **actual code** for the system based on the design documents created in the previous phase. This is where the software is physically built.
- **Output:** The **source code** and the executable program (the software itself).

4. Testing (Verification & Validation)

- **What it is:** The software built in the implementation phase is rigorously tested against the requirements defined in the *first* phase (the SRS). Testers check for bugs, errors, and any mismatches between the requirement and the implemented feature.
  - **Verification:** "Are we building the product **right**?" (Does the code meet the design specs?)
  - **Validation:** "Are we building the **right product**?" (Does the final product meet the user's needs from the SRS?)
- **Output: Test reports, bug reports**, and a stable, tested version of the software ready for release.

5. Maintenance

- **What it is:** After the software is deployed and being used by customers, the **long-term phase** begins. The development team handles issues that arise, provides support, and releases updates. This includes:
  - Fixing newly discovered bugs.
  - Adapting the software to new environments (e.g., a new operating system).
  - Adding new features or enhancements requested by the user.
- **This phase continues for the entire lifespan of the software.**

`V3.1>` why do we call it waterful model?

Excellent question! The name "Waterfall Model" is a very intentional metaphor. It's called that because of the way the development process is visualized and how it ideally functions.
Here's the breakdown of why the name fits so well:

1. The Visual Metaphor: Flowing Downward

Imagine a multi-tiered waterfall. The water flows from one pool at the top down to the next pool below it, then to the next, and so on. It doesn't naturally flow back up.
The Waterfall Model is drawn and conceptualized in the same way:
Each phase represents a distinct pool or level. The output (documentation, code, etc.) from one phase "flows down" and becomes the input for the next phase.

2. The Sequential and Non-Overlapping Nature

Just like water can't easily flow back up a waterfall, the classic Waterfall Model is designed to be **sequential and non-iterative**. This means:

- You must **100% complete** one phase before you begin the next.

- There is **no going back**. Once you are in the Design phase, you shouldn't still be discovering new requirements. Once you are Coding, you shouldn't be redesigning the core system.

This rigid, one-way direction is the core reason for the "waterfall" name. The process is meant to march steadily forward in one direction only.

3. The "Freeze" and Documentation

Each phase in the waterfall ends with a formal review and approval (often called a "freeze"). For example:

- The **Requirements phase** ends when the Requirements Specification document is finalized and signed off by the client.
- The **Design phase** ends when the Design Document is approved.

This approved document then "cascades" down to the next team. This formal hand-off of documentation from one team to the next is another part of the flowing-downward metaphor.

Who Coined the Term?

It's important to know that the term "Waterfall" wasn't originally a positive one.

- The sequential model was first formally described in a 1970 paper by **Winston W. Royce**. However, Royce *did not call it the "Waterfall Model."* In fact, he presented it as an example of a flawed, risky model that needed iteration to be successful.
- The term **"Waterfall"** was later popularized in a 1976 paper by T.E. Bell and T.A. Thayer. They used the term to describe this sequential, documentation-driven approach that had become common in government and large-scale software contracts.

Key Takeaway

We call it the **Waterfall Model** because it visualizes software development as a steady, **one-way, cascading flow** of work from one distinct phase to the next, much like water flowing down a series of rocky tiers.
This is in direct contrast to **Agile** or **Iterative** models, which are more like a **spiral** or a series of rapid cycles, where you constantly revisit and refine earlier phases.

`V3.1>` who is Frederick Brooks?

Of course. **Frederick Phillips Brooks Jr.** (often called **Fred Brooks**) is one of the most influential figures in the history of computing and software engineering. He is best known for his seminal work, *The Mythical Man-Month*, and his leadership on the groundbreaking IBM System/360 project.
Here's a breakdown of who he is and why he matters:

---

1. Project Lead for IBM System/360

Before his famous writing, Brooks was a top-level hardware and software architect at IBM. In the early 1960s, he was appointed the **project manager** for the development of the **IBM System/360** family of computers and its **OS/360** operating system.

- **Why it was revolutionary:** The System/360 was the first family of computers designed to cover a complete range of applications, from small to large, both commercial and scientific. They were all compatible with each other, meaning software written for a small model would run on a large one. This concept of "architecture" separating hardware from software was a paradigm shift.
- **The challenge:** The project was massively complex, hugely over budget, and behind schedule. The experience of managing this monumental effort led directly to his most famous insights.

2. Author of *The Mythical Man-Month* (1975)

This book is a collection of essays based on his experiences on the System/360 project. It is arguably the most famous book on software project management ever written. Its central ideas are now foundational to the field:

- **Brooks's Law:** The book's most famous assertion is **"Adding manpower to a late software project makes it later."**
  - **Reasoning:** The reasoning is that new team members require time for existing members to train them, which diverts resources from actual development. Communication overhead increases exponentially with the number of people (`n(n-1)/2` communication paths), leading to more complexity and more potential for miscommunication and error.
- **The "Man-Month" is a Mythical Unit:** The title criticizes the common mistake of equating person-power with time. He argues that effort (man-months) and calendar time are not interchangeable in the world of software design. You cannot, for example, assume that nine women can produce a baby in one month.
- **The Second-System Effect:** The tendency of an architect, after a first successful, simple system, to design a second system that is overloaded with features and baroque complexity, often leading to failure.
- **Conceptual Integrity:** He argued that for a system to be truly great, it must have a unified and coherent design vision, preferably from one or a very small number of minds. This is more important than adding countless features suggested by many people.
- **"No Silver Bullet" (1986):** In a later, equally famous essay added to the book, he made the provocative claim that **"There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity."** He argued that the essence of software work—the complexity of conceptualizing and designing the system—is inherently difficult and cannot be solved by any single technological magic bullet (like object-oriented programming or AI, though he has updated his views slightly over time).

---

3. Academic and Educator

After his time at IBM, Brooks moved to the **University of North Carolina at Chapel Hill** in 1964. There, he founded the university's computer science department and chaired it for 20 years. He was a beloved professor and a pioneer in computer graphics and virtual reality research long before they were mainstream fields.

Summary: His Legacy

Fred Brooks is revered not just as a manager or an author, but as a **deep thinker** who articulated the fundamental, human-centric challenges of software engineering that are just as relevant today as they were 50 years ago.

- **Key Work:** *The Mythical Man-Month*
- **Famous Law:** "Adding manpower to a late software project makes it later."
- **Famous Essay:** "No Silver Bullet"
- **Major Project:** Manager of IBM System/360 and OS/360.
- **Role:** Computer architect, project manager, author, and educator.

He received the highest honors in computing, including the **Turing Award** (1999) and the **National Medal of Technology** (1985). He passed away on November 17, 2022, leaving behind a legacy that continues to shape how we think about building complex systems.

V3.1>