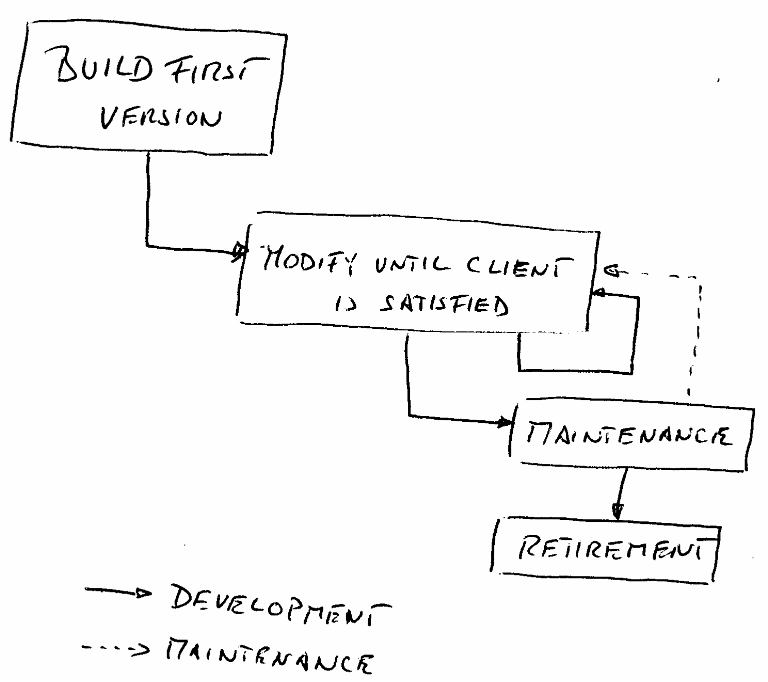
Software Life-Cycle Models

Software Life-Cycle Models (Software Process Models)

1. Define the phases that are part of the software development process for a product;
2. Define how the development process moves from one phase to another.

The second part of the above definition distinguishes life-cycle models from software processes.

Build-and-Fix Model



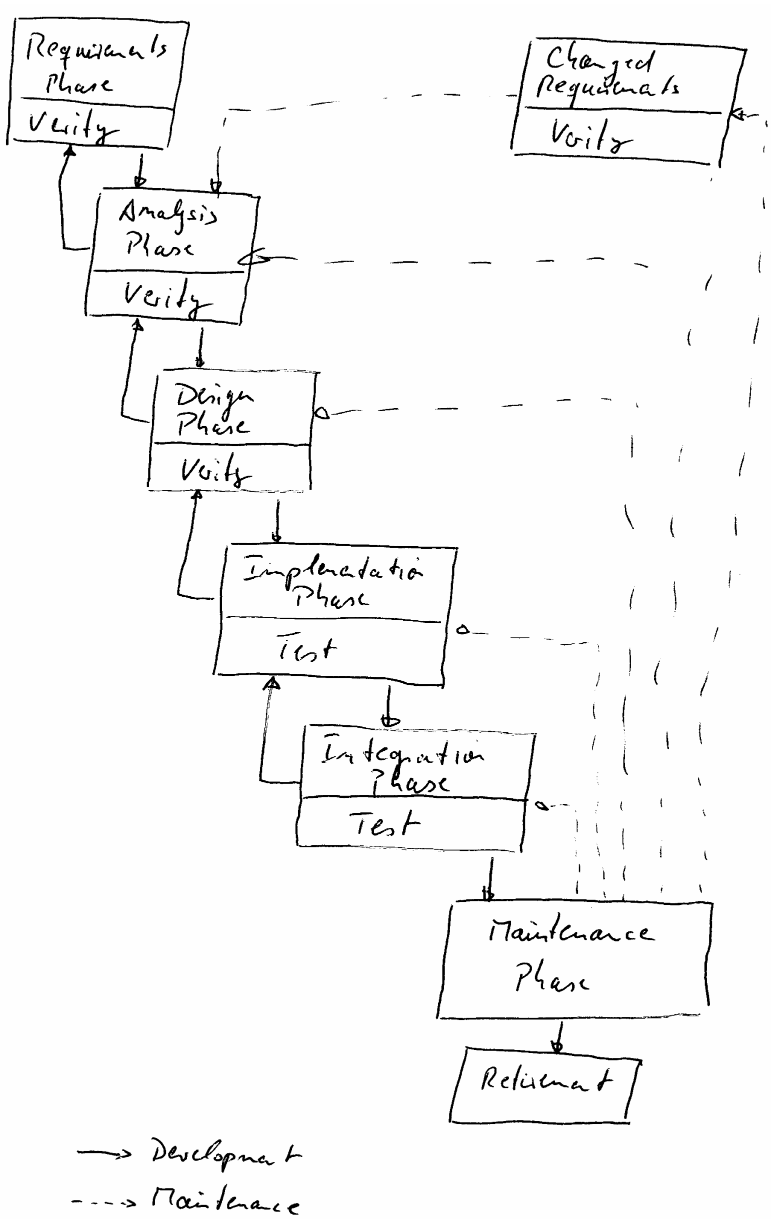
Problems

- No requirements phase
- No analysis phase
- No design phase (!)
- Everything is done during implementation → change is very expensive.
- Radical requirement shifts could lead to a complete fresh start.

⇒ We need a more complete life-cycle model:

- Game plan
- Phases other than implementation & maintenance
- Project milestones that are *predictable* and *measurable*

Waterfall Model



Observations:

- All phases are present in this model and we are told how they are connected
- You can move back and forth between phases as often as necessary – a misconception about the waterfall model

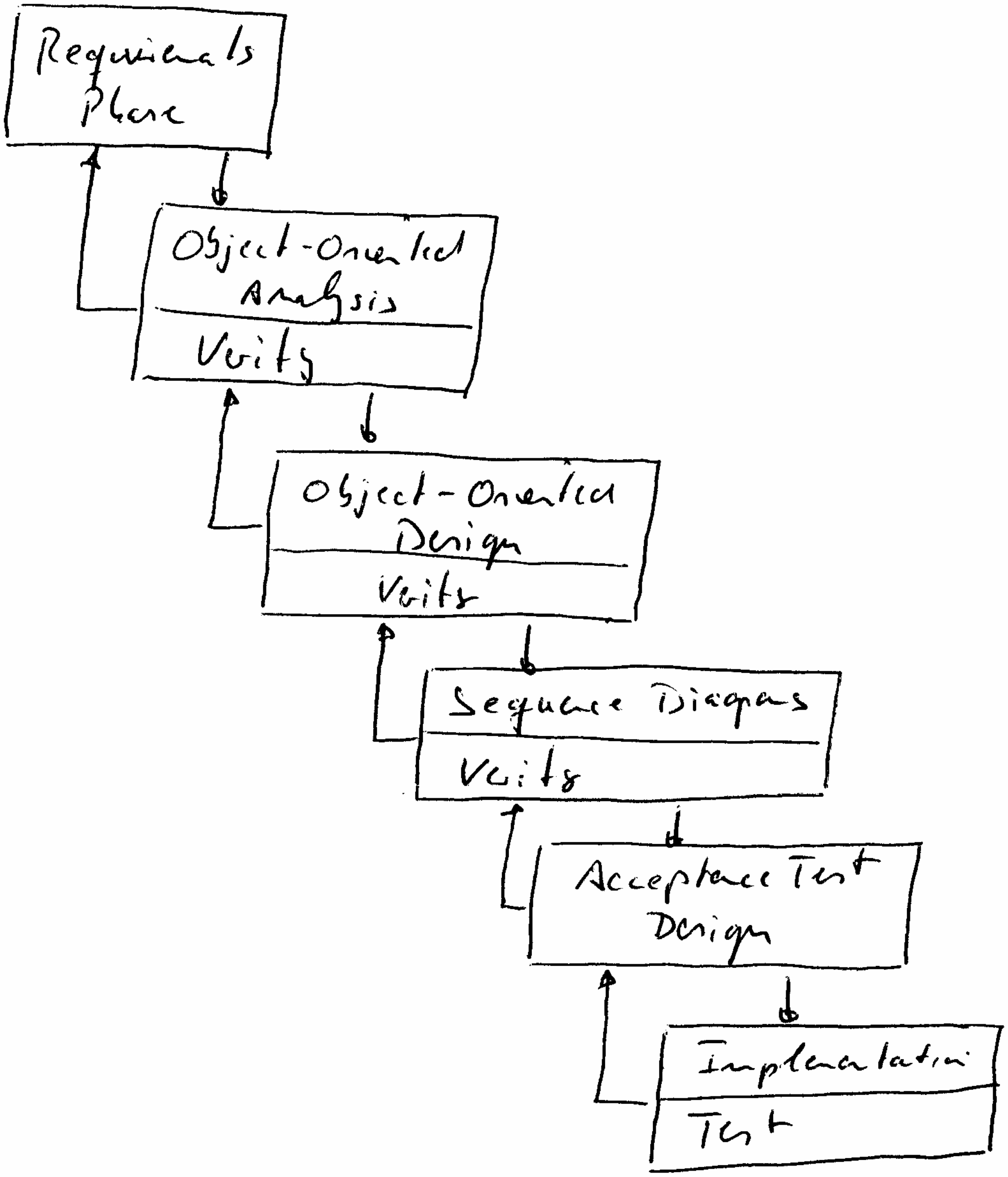
Notes on the Waterfall Model

- Characterized by
  - Feedback loops
  - Documentation driven
- Advantages
  - Documentation is properly generated
  - Natural generation of project milestones
  - Maintenance easier
- Disadvantages
  - First functionality is only seen very late in the game
  - That means the real litmus test whether the right product is being built happens at the end of the life-cycle model – before that that all just paper – difficult to visualize, especially for the customer.

⇒ The disadvantages can be mitigated by a modified waterfall model: rapid prototyping

- Here a rapid prototype is substituted for the requirements phase in the standard waterfall model above.
- A rapid prototype is a working *model* functionally equivalent to a *subset* of the product.
- A prototype should **NEVER** become the product!

Our Life-Cycle Model


Requirements Phase
Object-Oriented Analysis
Verify
Object-Oriented Design
Verify
Sequence Diagrams
Verify
Acceptance Test Design
Implementation
Test