The Waterfall Development Model

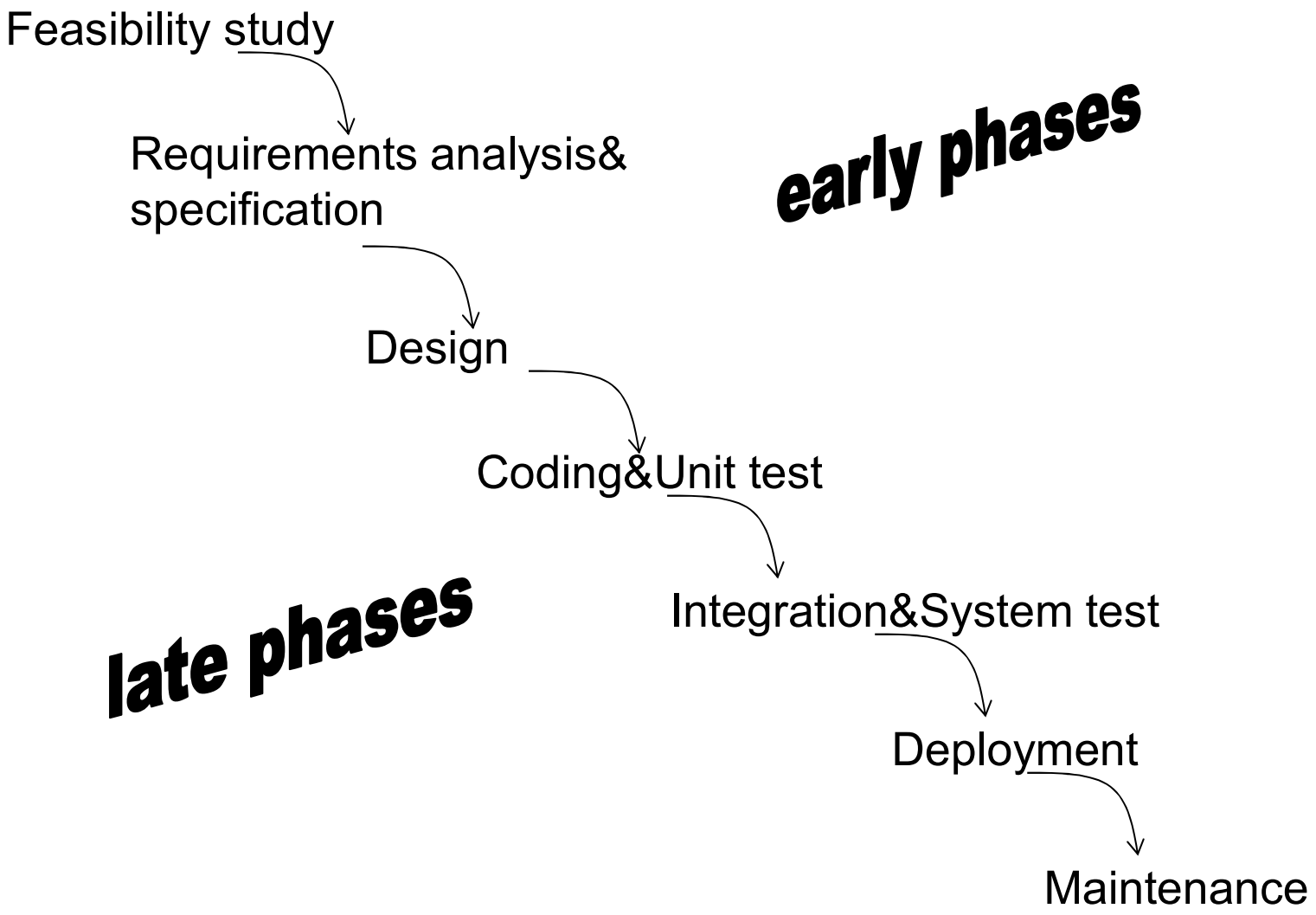
Sandro Morasca
Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate
Via Mazzini 5
I-21100 Varese, Italy
sandro.morasca@uninsubria.it

Software Lifecycle Models

- In some cases, no reference model:
  - code&fix
- The traditional “waterfall” model
  - identify phases and activities
  - force linear progression from a phase to the next
  - no returns (they are harmful)
    - better planning and control
  - standardize outputs (artifacts) from each phase

software like manufacturing

A Waterfall Model



Studio di fattibilità

Feasibility Study

- Cost/benefits analysis Analisi dei costi e dei benefici
- Determines whether the project should be started (e.g., buy vs make), possible alternatives, needed resources
- Produces a Feasibility Study Document
  - Preliminary problem description
  - Scenarios describing possible solutions
  - Costs and schedule for the different alternatives

Feasibility Study

- In practice, the feasibility study is subject to
  - time pressure
  - cost pressure: we are not even sure that the customer will accept our offer
- Consequences
  - alternatives may not be investigated
  - risks are not assessed right

Analisi dei requisiti e delle specifiche

Requirements Analysis and Specification

- Analyze the domain in which the application takes place
- Identify requirements Identifica i requisiti
- Derive specifications for the software
  - Requires an interaction with the user
  - Requires an understanding of the properties of the domain
- Produces a Requirements Analysis and Specification Document (RASD)

The 5 Wh's

- Who
  - who will use the system
- Why
  - why should it be developed + why will the users use it
- What (vs How)
  - what will it provide
- Where
  - where will it be used, on which architecture
- When
  - when and how long will it be used

Chi dovrebbe usare il software

Come deve essere sviluppato

RASD

- Required properties
  - Precise
  - Complete
  - Consistent
  - Understandable
  - Modifiable
- May include
  - Preliminary User Manual
  - System Test Plan

RASD

- Functional requirements
- Non-functional requirements
- Requirements on the development and maintenance process

Design

- Defines the software architecture
  - Components (modules)
  - Relations among components
  - Interactions among components
- Goal
  - Support concurrent development, separate responsibilities
- Produces the Design Document

Coding&Unit Test

- Each module is implemented using the chosen programming language
- Each module is tested in isolation by the module's developer
- Programs include their documentation

Integration&System Test

- Modules are integrated into (sub)systems and integrated (sub)systems are tested
- This phase and the previous may be integrated in an incremental implementation scheme
- Complete system test needed to verify overall properties
- Sometimes we have *alpha test* and *beta test*

Effort Distribution

- From 125 projects within HP
  - 18% requirements and specification
  - 19% design
  - 34% coding
  - 29% testing
- typical variations around 10%

Deployment

- The goal is to distribute the application and manage the different installations and configurations at the clients' sites

Maintenance

- All changes that follow delivery
- Unfortunate term: software does not wear out
  - if a failure occurs, the cause was there
- Often more than 50% of total costs
  - Recent survey among EU companies
    - 80% of IT budget spent on maintenance

Maintenance

- It includes different types of change: correction + evolution
  - corrective maintenance ≈ 20%
  - adaptive maintenance ≈ 20%
  - perfective maintenance ≈ 50%

Other Activities

- Some activities are carried out along the entire lifecycle
- Documentation
- Verification
- Management

Folk Data on Errors

- Systematic inspection techniques can discover up to 50-75% of errors
- Modules with complex control flow are likely to contain more errors
- Often tests cover only about 50% of code
- Delivered code contains 10% of the errors found in testing
- Early errors are discovered late, and the cost of removal increases with time
- Eliminating errors from large and mature systems costs more (4-10 times) than in the case of small and new systems
- Error removal causes introduction of new errors
- Large systems tend to stabilize to a certain defect level

Why Evolution?

- Context changes (adaptive maintenance)
  - EURO vs national currencies
- Requirements change
  - New demands caused by introduction of the system
  - Survey among EU companies indicates that 20% of user requirements are obsolete after 1 year
- Wrong specifications (requirements were not captured correctly or domain poorly understood)
- Requirements not known in advance

How to Face Evolution

- Likely changes must be anticipated
- Software must be designed to accommodate future changes reliably and cheaply

This is one of the main goals of software engineering

Correction vs. Evolution

- Distinction can be unclear, because specifications are often incomplete and ambiguous
- This causes problems because specs are often part of a contract between developer and customer
  - early frozen specs can be problematic, because they are more likely to be wrong

Software Changes

- Good engineering practice
  - first modify design, then change implementation
  - apply changes consistently in all documents
- Software is very easy to change
  - often, under emergency, changes are applied directly to code
  - inconsistent state of project documents
- *Software maintenance is (almost) never anticipated and planned*
  - *this causes disasters*

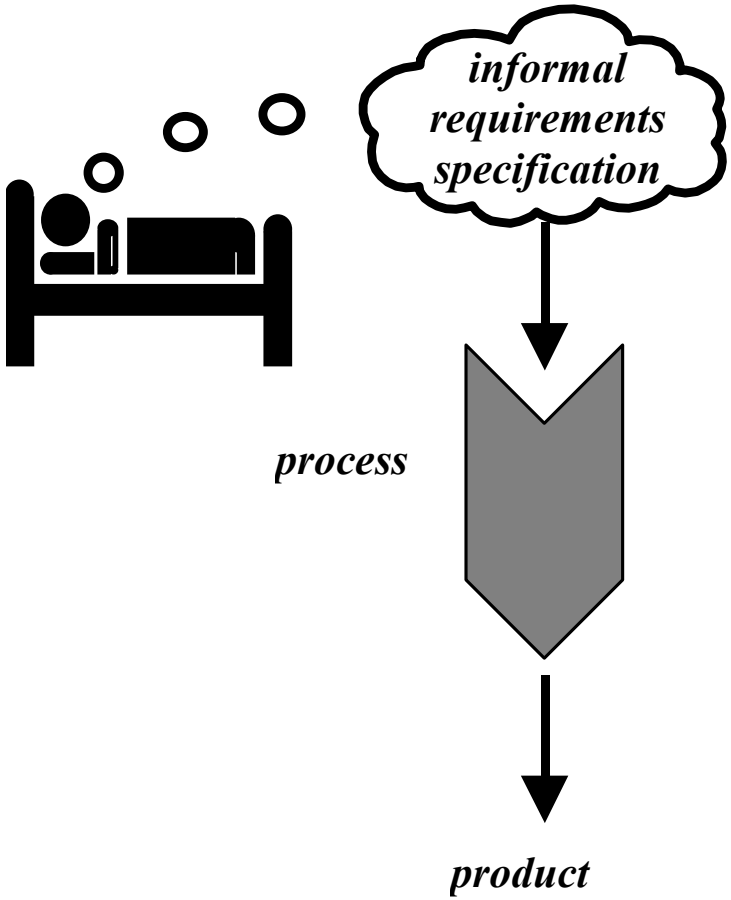
Waterfall Lifecycles

- Many variations exist
- Each organization tends to define “its own”
- Sample cases
  - software developed for personal use
  - customer (user) belongs to same organization
  - custom software developed by sw house
  - application for the market

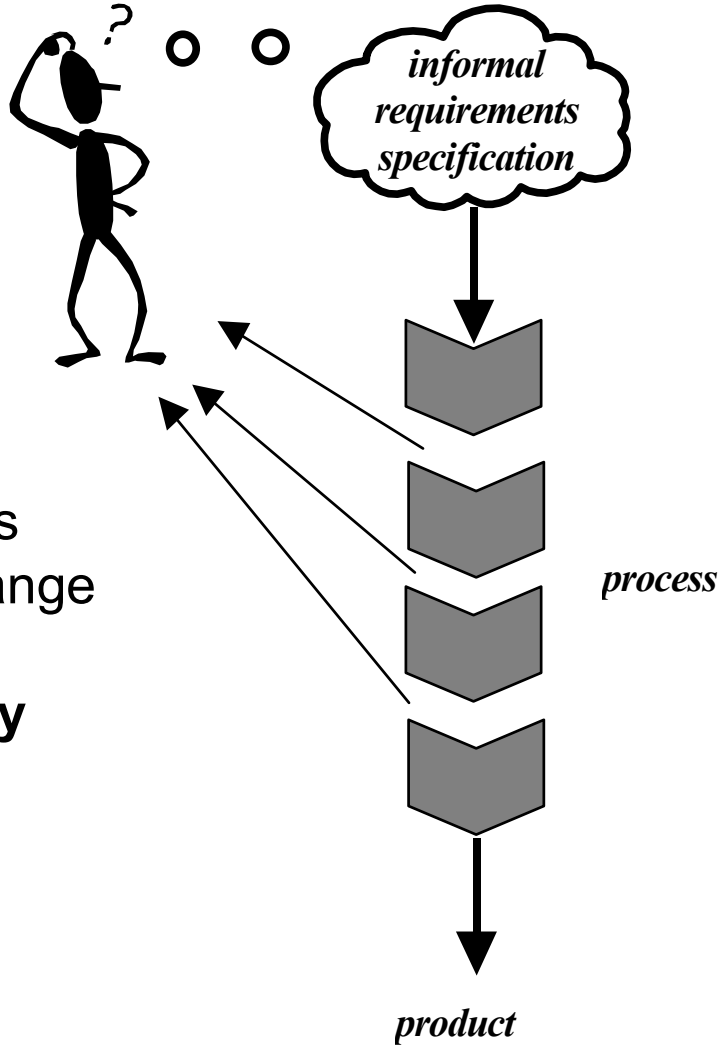
Waterfall Can Be Harmful

- “Waterfall” requires that the domain be understood and requirements be known and stable
- This happens in only a few cases
- Recycling cannot be eliminated
  - it is part of our problem

Waterfall Is “Black Box”

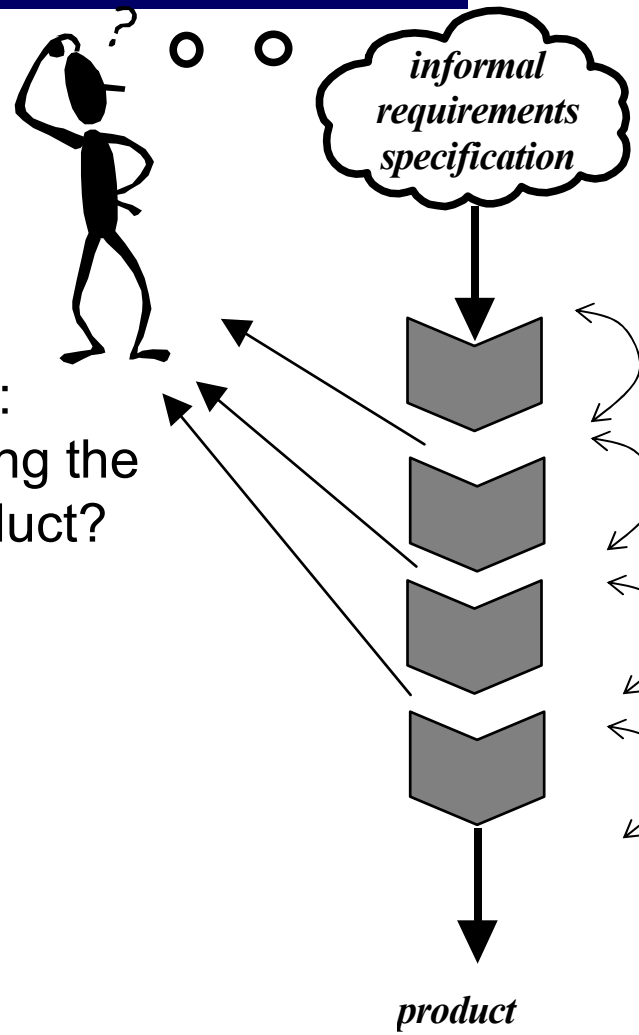


Need for Transparency



- Transparency allows early check and change via **feedback**
- It supports **flexibility**

Verification and Validation



Validation:
are we doing the
“right” product?

Verification:
are we doing
the product right?