

The software lifecycle and its documents

Supplementary material for Software Architecture course

B. Meyer, May 2006

Lifecycle models

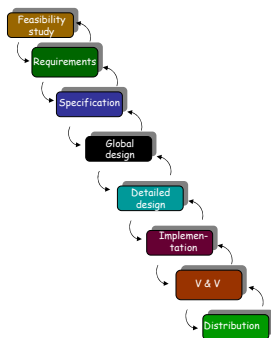
Origin: Royce, 1970, Waterfall model

Scope: describe the set of processes involved in the production of software systems, and their sequencing

"Model" in two meanings of the term:

- Idealized description of reality
- Ideal to be followed

Lifecycle: the waterfall model



Lifecycle phase: Requirements

Objective:

Gather user needs to define system functionality

Actors:

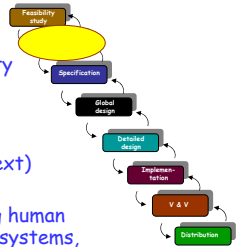
Customers
Development team
Other "stakeholders" (see next)

Techniques:

Interviews, study of existing human processes, study of existing systems,

Products:

Requirements document
QA plan (in particular test plan)
Draft user's manual



Properties of successful requirements

- | | |
|--|---|
| <ul style="list-style-type: none">▪ Correct▪ Complete▪ Consistent▪ Unambiguous▪ Feasible▪ Relevant▪ Abstract | <ul style="list-style-type: none">▪ Traceable▪ Delimited▪ Readable▪ Modifiable▪ Testable▪ Prioritized▪ Endorsed |
|--|---|

Difficulties of requirements

- Natural language and its imprecision
- Formal techniques and their abstraction
- Users and their vagueness
- Customers and their demands
- The rest of the world and its complexity

Goals of performing requirements

Source: OOSC

- Understand problem or problems that the eventual software system, if any, should solve
- Prompt relevant questions about problem & system
- Provide basis for answering questions about specific properties of problem & system
- Decide what system should do
- Decide what system should not do
- Ascertain that system will satisfy the needs of its stakeholders
- Provide basis for development of system
- Provide basis for V & V* of system

*Validation & Verification, including testing

Possible requirements stakeholders

- Clients (tailor-made system)
- Customers (product for general sale)
- Clients' and customers' customers
- Users
- Domain experts
- Market analysts
- Unions?
- Legal experts
- Purchasing agents
- Software developers
- Software project managers
- Software documenters
- Software testers
- Trainers
- Consultants

IEEE 830-1998

"IEEE Recommended Practice for Software Requirements Specifications"

Approved 25 June 1998 (revision of earlier standard)

Descriptions of the content and the qualities of a good software requirements specification (SRS).

Goal: "The SRS should be correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable, traceable."

IEEE Standard 830-1998

Recommended practice for Software Requirements Specifications

Recommended document structure:

1. Introduction
 - 1.1 Purpose
 - 1.2 Scope
 - 1.3 Definitions, acronyms, and abbreviations ← Glossary!
 - 1.4 References
 - 1.5 Overview
2. Overall description
 - 2.1 Product perspective
 - 2.2 Product functions
 - 2.3 User characteristics
 - 2.4 Constraints
 - 2.5 Assumptions and dependencies
3. Specific requirements
 - Appendixes
 - Index

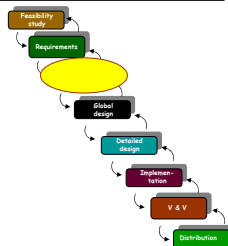
Lifecycle phase: Specification

Objective:
Define precise blueprint for system functionalities

Actors:
Development team

Example techniques:
UML, abstract data types, formal specification languages, Petri nets, State Charts, contracts

Products:
Specification document
Revised draft user's manual



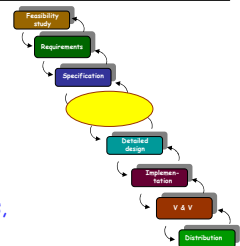
Lifecycle phase: Global Design

Objective:
Define system architecture

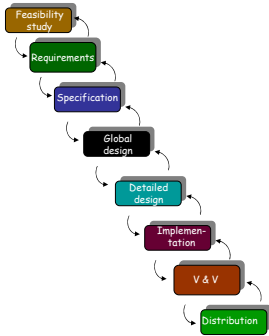
Actors:
Development team

Example techniques:
Design language, design patterns, high-level programming language

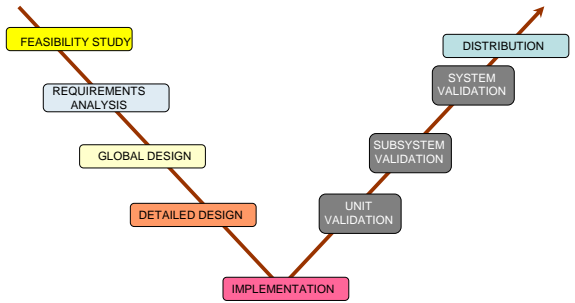
Products:
Specification document



Lifecycle: the waterfall model



V-shaped

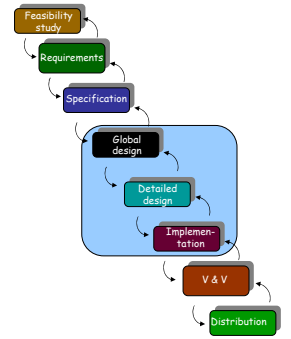


Arguments for the waterfall

(After B.W. Boehm: *Software engineering economics*)

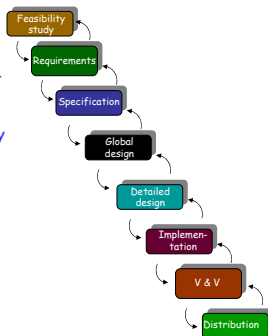
- The activities are necessary
 - (But: merging of middle activities)
- The order is the right one.

Merging of middle activities

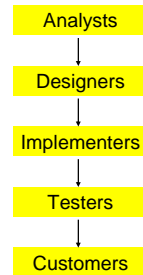


Problems with the waterfall

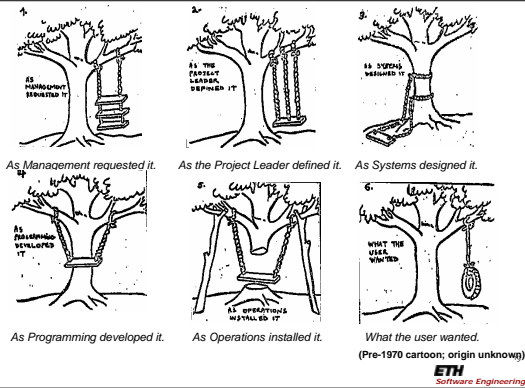
- Late appearance of actual code.
- Lack of support for requirements change – and more generally for extendibility and reusability
- Lack of support for the maintenance activity (70% of software costs?)
- Division of labor hampering Total Quality Management
- Impedance mismatches
- Highly synchronous model



Quality control?



Impedance mismatches



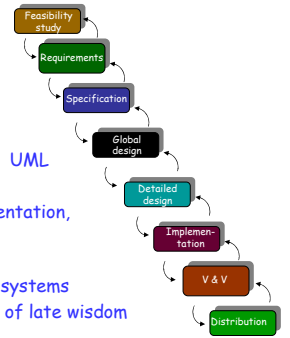
Problems with the waterfall

Separate tools:

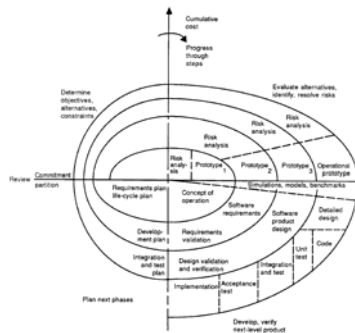
- Programming environment
- Analysis & design tools, e.g. UML

Consequences:

- Hard to keep model, implementation, documentation consistent
- Constantly reconciling views
- Inflexible, hard to maintain systems
- Hard to accommodate bouts of late wisdom



The Spiral model (Boehm)



"Prototyping" in software?

In other engineering fields, "prototyping" means one of:

- 1. Smaller-scale model (e.g. water-processing plant...)
- 2. Fully functional product, but custom-built, before attempting mass-production (e.g. car...)

In software:

- 1 is dubious
- 2 has no equivalent in ordinary circumstances (except for reuse, see next)

"Prototyping" in software

The term is used in one of the following meanings:

- Experimentation:
 - Requirements capture
 - Try specific techniques: GUI, implementation ("buying information")
- Pilot project
- Incremental development
- Throw-away development

(Fred Brooks, *The Mythical Man-Month*: "Plan to throw one away, you will anyhow").

Risks of "throw-away" prototyping

- If it removes an important constraint, it may be useless
- If you can envision shipping it, it's not an experiment!
- What if you have to ship anyway?

Risks of the spiral and prototyping



M.C Escher:
Waterfall

Extreme Programming and "agile" methods

Reaction against over-constraining models

Basic ideas:

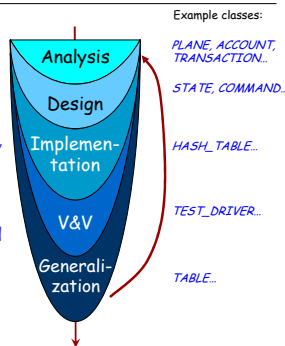
- Short development cycles
- No full initial specification; refine specs as you go
- Test-Driven Development
- Involve customers
- Recommended practices, e.g. Pair Programming

The cluster model

Seamless development:

Single notation, tools, concepts, principles throughout
Continuous, incremental development
Keep model, implementation and documentation consistent

Reversibility: go back and forth



Example classes:

PLANE, ACCOUNT,
TRANSACTION...

STATE, COMMAND...

HASH_TABLE...

TEST_DRIVER...

TABLE...

Generalization

Prepare for reuse

For example:

- Remove built-in limits
- Remove dependencies on specifics of project
- Improve documentation, contracts...
- Extract commonalities and revamp inheritance hierarchy

Few companies have the guts to provide the budget for this

The cluster model

Cluster 1

Cluster 2

Cluster n



Reversibility

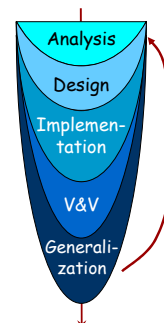




Diagram Tool

- System diagrams can be produced automatically from software text
- Works both ways: update diagrams or update text - other view immediately updated

No need for separate UML tool

Metrics Tool

Profiler Tool

Documentation generation tool

...