Chapter 8.1 - Process Models

Walter F. Tichy

What is a Software Process Model?

A software process model is an abstraction of the software development process. It defines the following:

- The activities to be performed
- The input and output of each activity
- The pre and post conditions for each activity
- The order in which the activities are performed

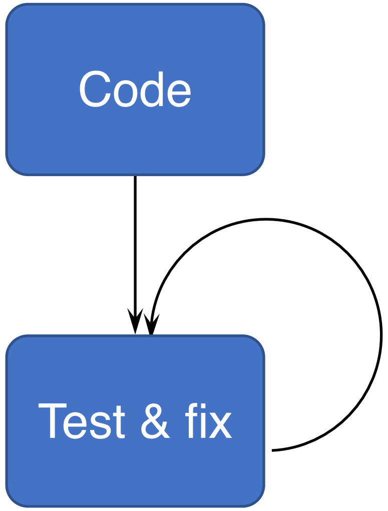
The goal of a software process model is to provide guidance for controlling and coordinating the activities to achieve the end product and objectives as effectively as possible.

Process Models Covered

1. Programming by trial and error
2. Waterfall model
3. V-Modell
4. Prototype models
5. Incremental models
6. Synchronize and stabilize
7. Agile methods
   - Extreme Programming
   - Scrum

Programming by Trial and Error

- Also called ”Code & Fix"
- Procedure
  - Create preliminary program
  - Think about requirements, design, testing, maintenance
  - “Improve” program accordingly
- Features
  - Fast (?), code developed without "useless" additional effort
  - Generates poorly structured code (“spaghetti code”) due to unsystematic improvements and lack of a design phase

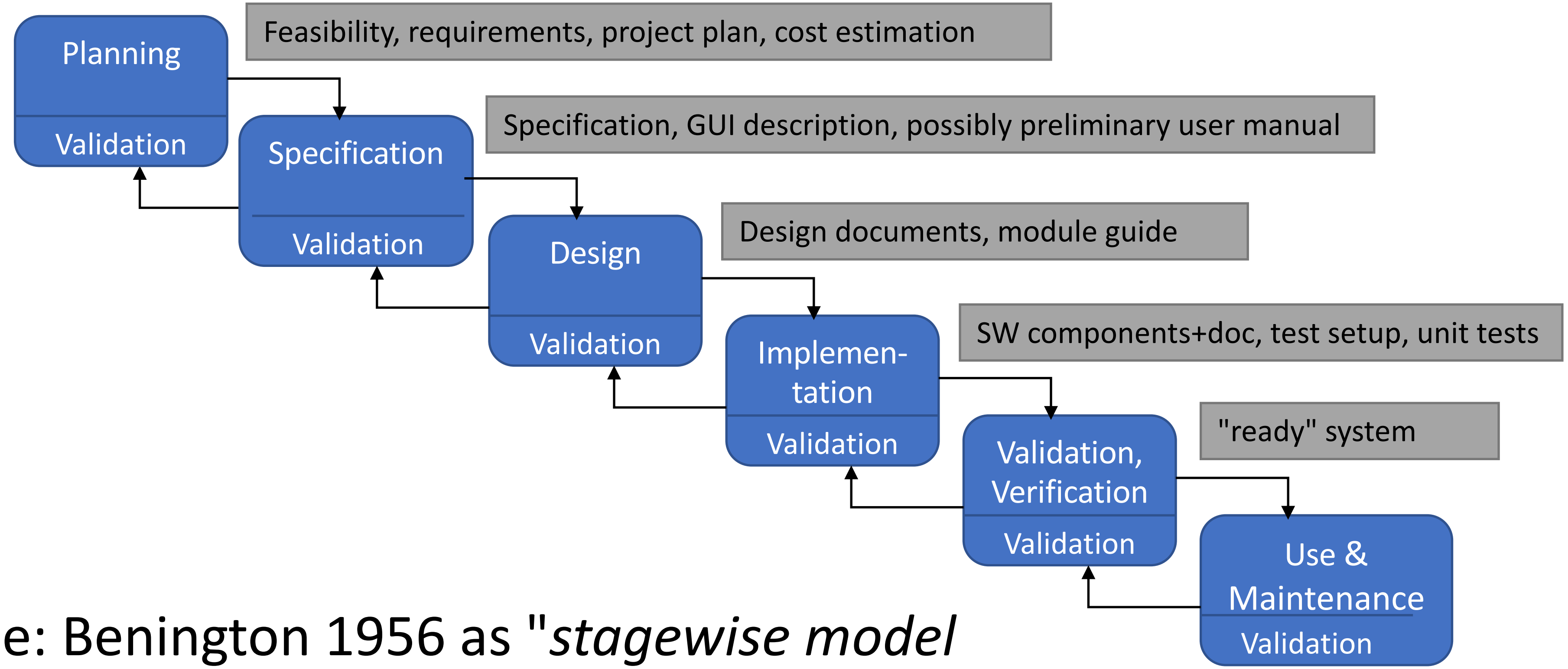


Programming by Trial and Error

- Problems
  - Insufficient task fulfillment due to lack of requirements analysis
  - Maintenance costly, as program is not prepared for it
  - Documentation non-existent
  - Completely unsuitable for teamwork, as no division of tasks is provided.

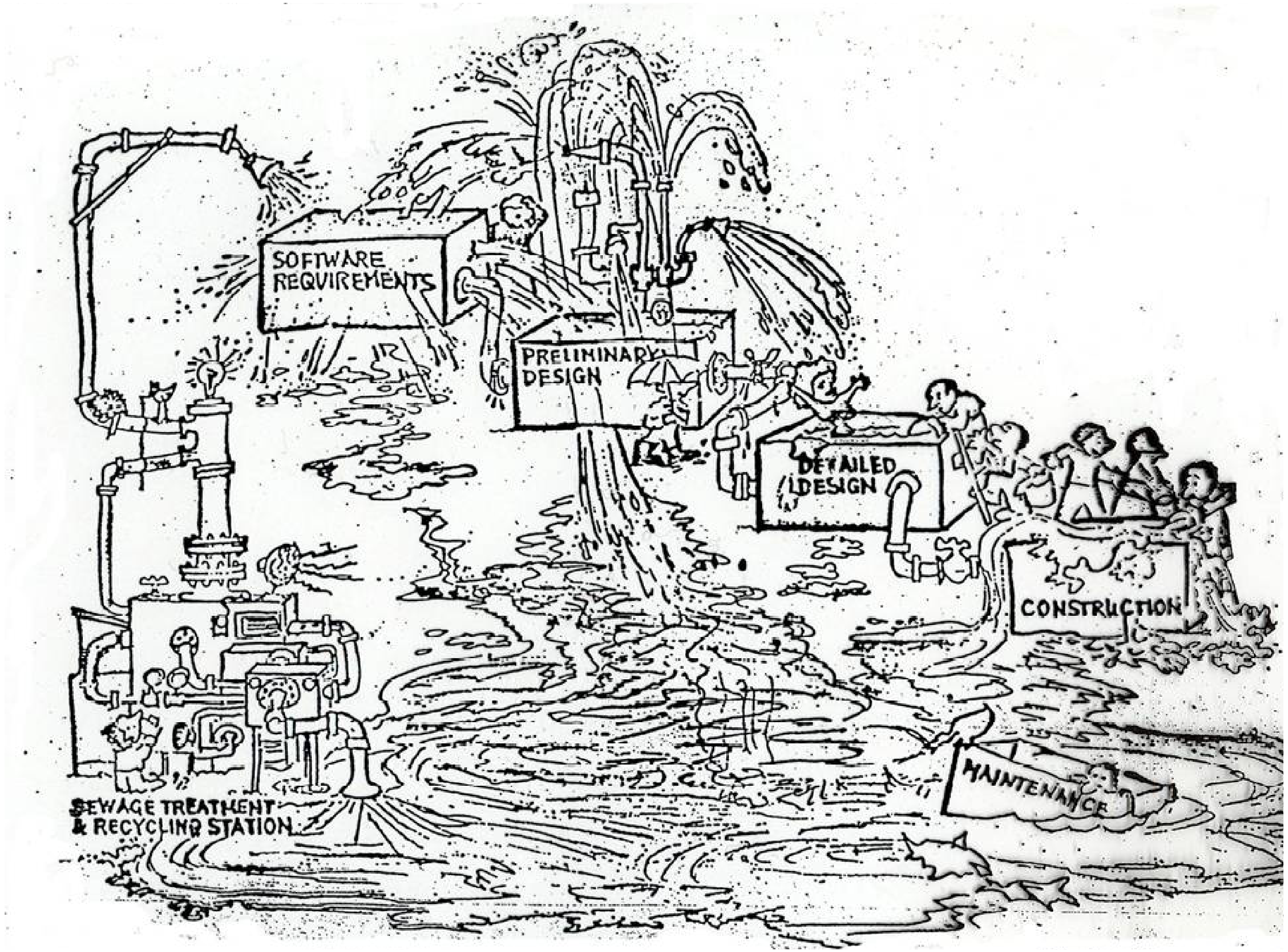
Waterfall Model

- Also "phase model“


Planning
Validation
Feasibility, requirements, project plan, cost estimation
Specification
Validation
Specification, GUI description, possibly preliminary user manual
Design
Validation
Design documents, module guide
Implementation
Validation
SW components+doc, test setup, unit tests
Validation, Verification
Validation
"ready" system
Use & Maintenance
Validation

- For the first time: Benington 1956 as "*stagewise model*
- Royce 1970 extension: feedback

Waterfall Model



Waterfall Model

- Procedure
  - All activities
  - are completed
  - in the order given

strictly
sequential
approach

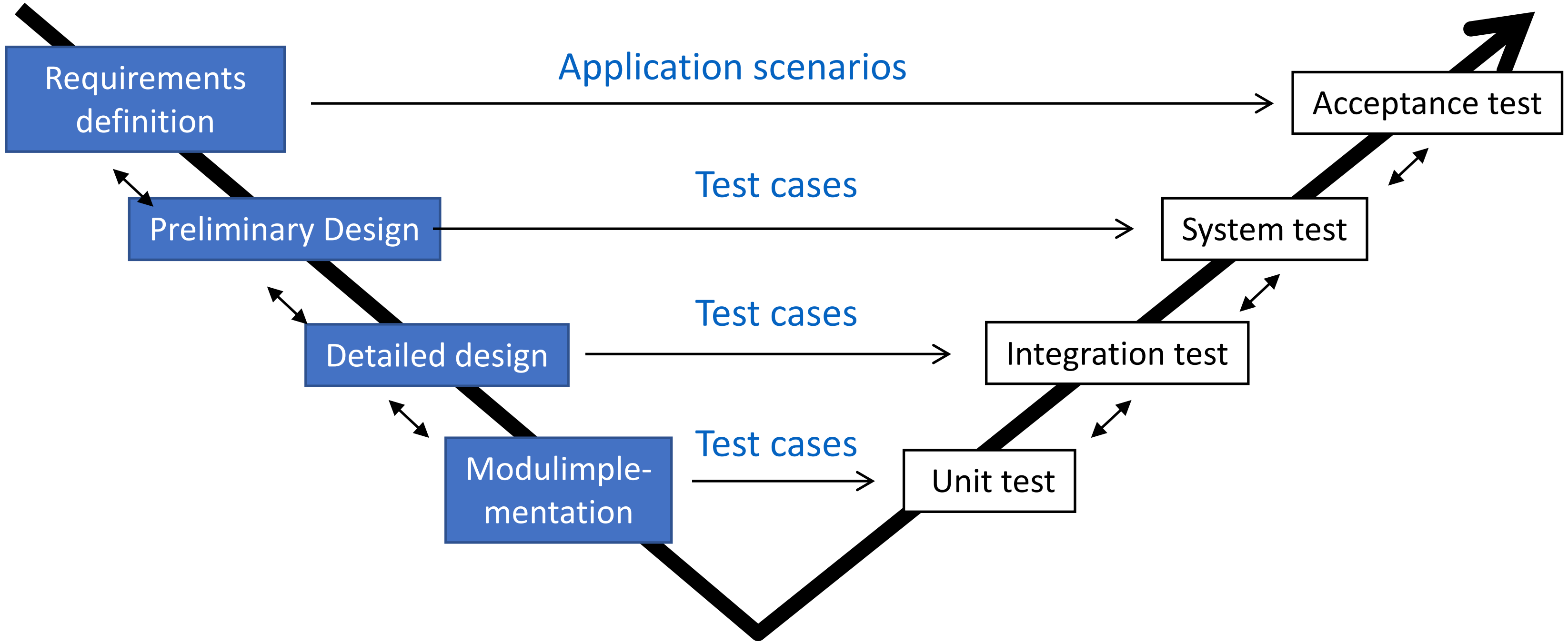
- At the end of each activity there is a finished set of documents
  → "document-driven" model
- Simple, understandable
- User participation only foreseen in the planning and specification phases

Waterfall Model

- Problems
  - No cross-phase feedback provided
    → Troubleshooting and correction problematic
  - Parallelization potential not properly exploited
    → Market launch is unnecessarily delayed
  - Forces precise specification of even poorly understood user interfaces and functions
    → Design, implementation and testing of later useless code.
  - Not suitable for changing requirements
- Waterfall model is useful if requirements will not change during development, for instance in government projects.
- Many practitioners unfortunately still use Code&Fix.
- In this lecture, the waterfall model is a pedagogical model, where the individual activities are clearly separated and can therefore be studied and learned in "pure form". The model also shows that SW development is much more than just coding.

"V-Modell 97"

- V like German „Vorgehensmodell“ (procedural model)
- Each activity has its own test step



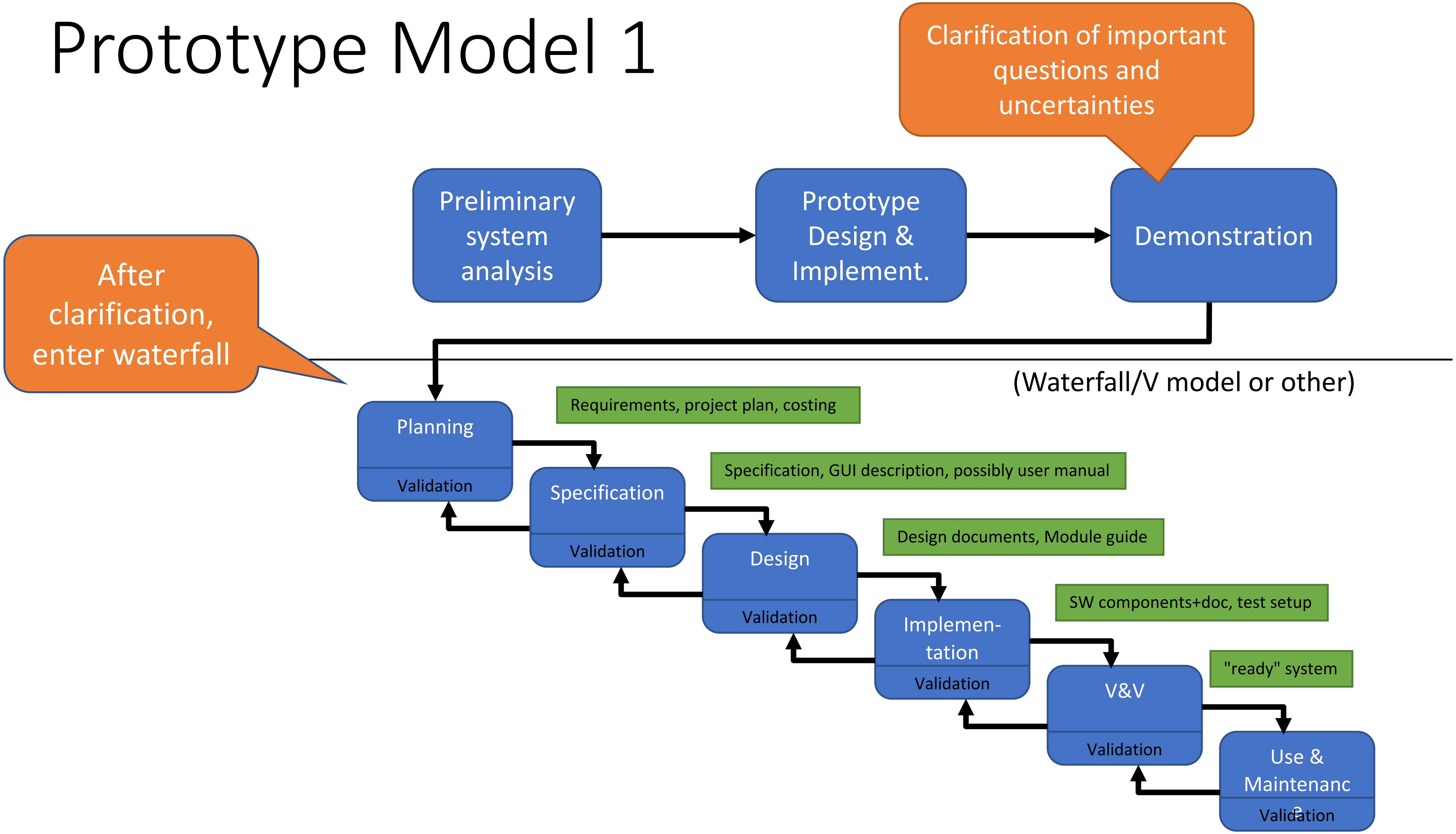
Properties of Waterfall-based Models

- Managers love waterfall models
  - Nice milestones
  - No need to look back (linear system)
  - Always one activity at a time
  - Easy to check progress during development: 90% coded, 20% tested
- However, software development is non-linear
  - While a design is being developed, problems with requirements are identified
  - While a program is being coded, design and requirement problems are found
  - While a program is tested, coding errors, design errors and requirement errors are found.

Prototype Model

- Suitable for systems for which a complete specification cannot be produced without exploratory development or experimentation
- The prototype (limited functional system) can strengthen morale and trust between supplier and customer
- Frederick P. Brooks in "The Mythical Man-Month" (Ch. 11, p. 116): *Plan to throw one away; you will, anyhow.*
- Important: **DISCARD THE PROTOTYPE!**

Prototype Model 1



Clarification of important questions and uncertainties

Preliminary system analysis

Prototype Design & Implement.

Demonstration

After clarification, enter waterfall

(Waterfall/V model or other)

Planning

Validation

Requirements, project plan, costing

Specification

Validation

Specification, GUI description, possibly user manual

Design

Validation

Design documents, Module guide

Implementation

Validation

SW components+doc, test setup

V&V

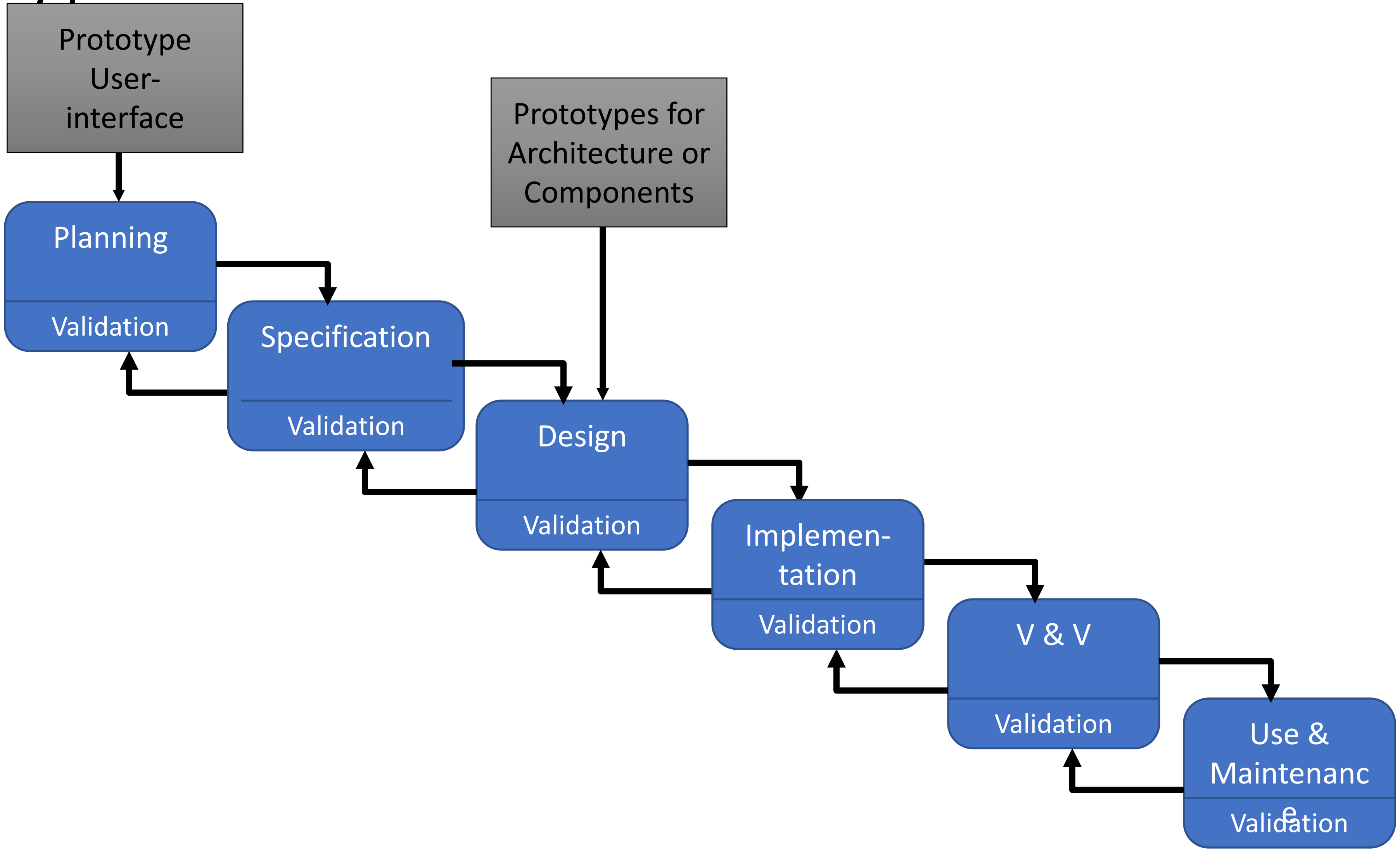
Validation

"ready" system

Use & Maintenance

Validation

Prototype Model 2



Incremental Model

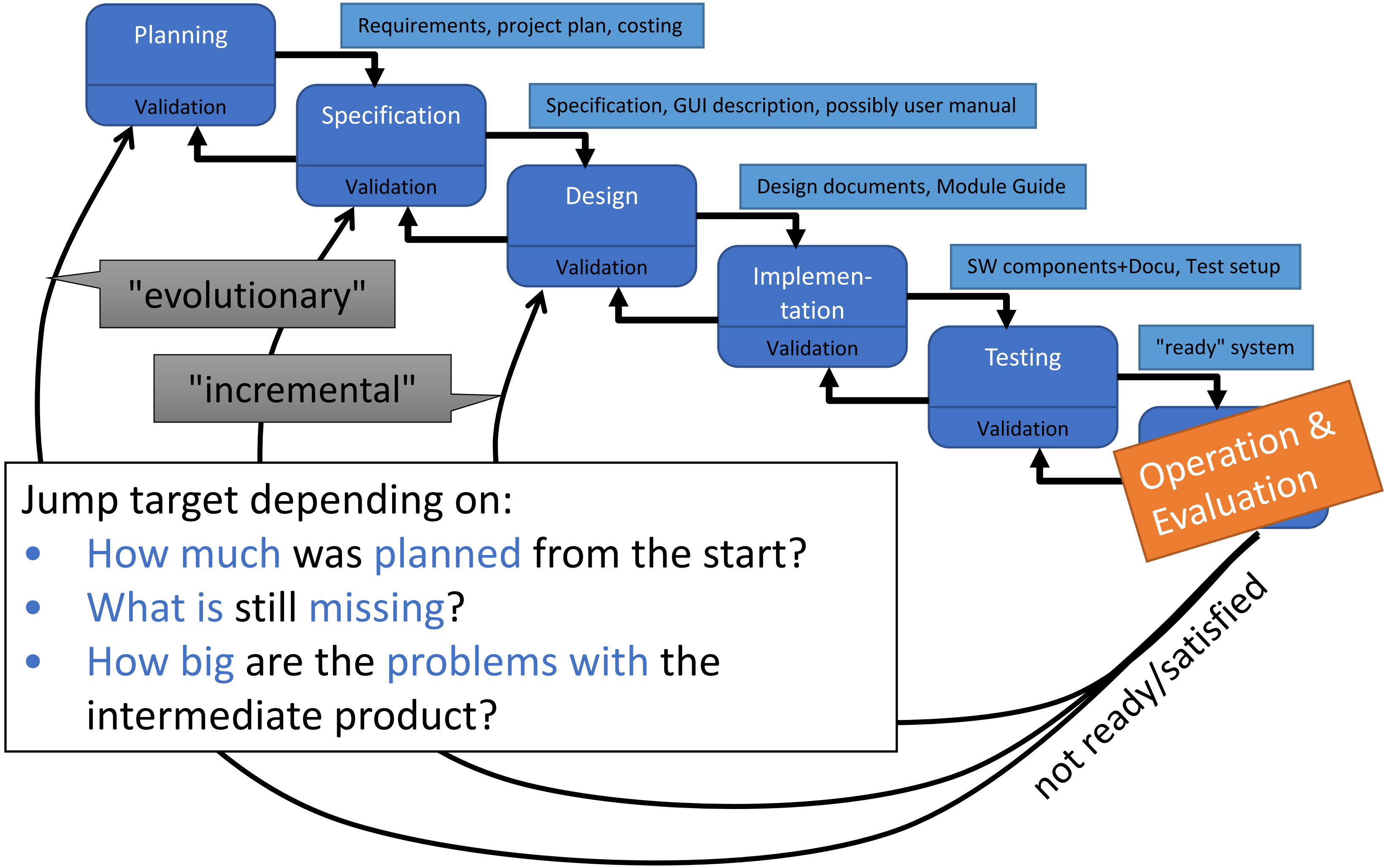
- Also "*successive versions*" - extension of the prototype idea
- Idea: At least parts of the functionality can be clearly defined and realized
- Therefore: functionality is created step by step and "added" to the product
- Same advantages and areas of application as prototype model
- Attempts to reuse more than the prototype model

Incremental Model

- Different approaches in the literature for planning and analysis phase
  - Evolutionary: plan and analyze only the part that will be added next (n-fold waterfall model)
    - Risk that the next part cannot be integrated due to structural difficulties and therefore parts have to be done again
  - Incremental: Plan and analyze everything, then iterate n times through incremental design, implementation, and testing phases
    - Requires complete planning and analysis, which is actually what this model is designed to circumvent....

→ Mixed forms and flexibility appropriate

Incremental Model



Planning

Validation

Requirements, project plan, costing

Specification

Validation

Specification, GUI description, possibly user manual

Design

Validation

Design documents, Module Guide

Implementation

Validation

SW components+Docu, Test setup

Testing

Validation

"ready" system

Operation & Evaluation

"evolutionary"

"incremental"

not ready/satisfied

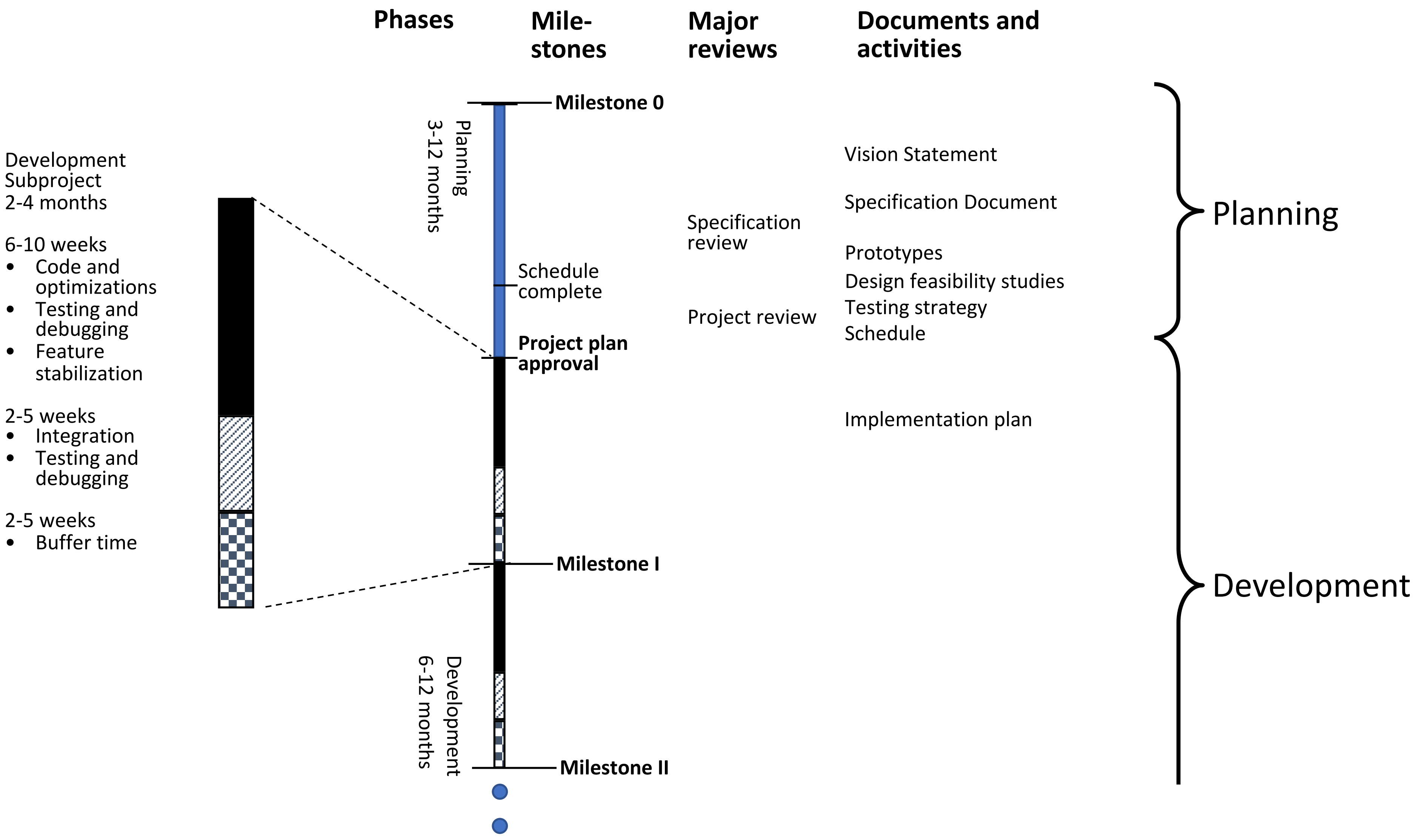
Jump target depending on:

- How much was planned from the start?
- What is still missing?
- How big are the problems with the intermediate product?

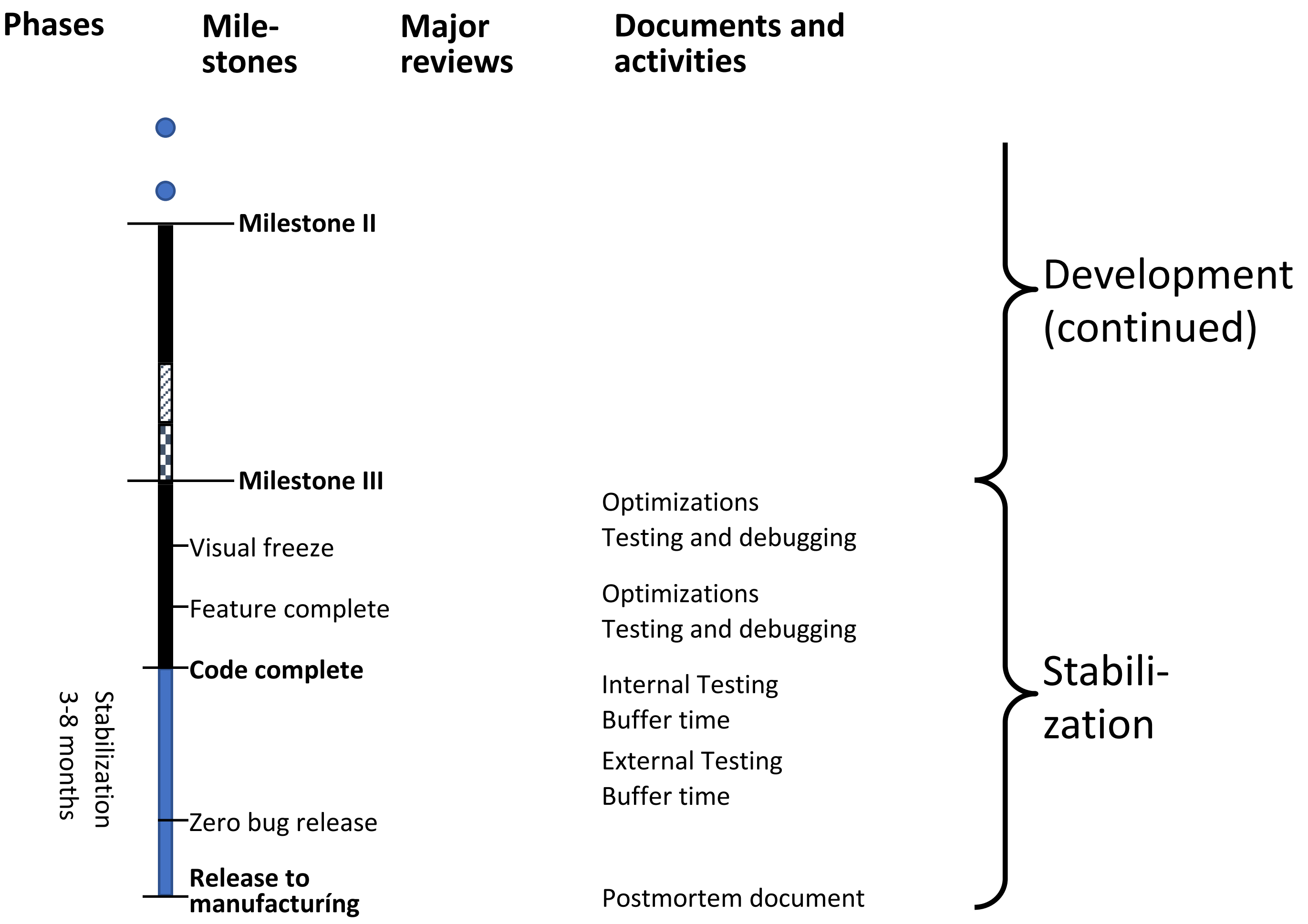
Synchronize and Stabilize

- Also called "Microsoft model" (see [CusSel95])
- Approach
  - Organize the 200 programmers of a project (e.g. Windows 95) into "small hacker teams".
    → Freedom for own ideas/designs
  - But: Synchronize regularly (nightly)
  - And Stabilize Regularly (Milestones, 3 Mon.)
- Three phases
  - Planning phase
  - Development phase in 3 subprojects
  - Stabilization phase

Synchronize and Stabilize



Synchronize and Stabilize



Synchronize and Stabilize: Planning Phase

≥ 30% changes during project duration

- **Vision statement**: Managers (marketing professionals) identify and prioritize product features based on extensive collections of customer wishes
- **Specification**: Managers and developers define functions, architecture and component dependencies based on the desired vision
- **Schedule and team structure**: division of tasks into "product function groups" with each
  - 1 Manager
  - 3-8 developers
  - Exactly as many testers as developers (work 1:1 in parallel with developers)
- **Duration**: 3 -12 months (depending on complexity)

Synchronize and Stabilize: Development Phase

- Tasks
  - Managers coordinate further development of the specification
  - Developers design, code and remove bugs
  - Testers work in parallel with their developer
- Three subprojects → Three milestones
  - First third of the planned functionality, most important functions
  - Second third
  - Last third: least important functions

Synchronize and Stabilize Development Phase

- Checkout, editing, compiling & testing
- Checking in (when required, usually 2x per week)
- Nightly, complete (re-)compilation of all sources.
- Automatic regression testing
- Sanctions (characteristics depending on the team) for causing errors during integration. These errors must be fixed immediately, as they hold up other developers!
- At the end of each subproject all detected errors are eliminated (subproject stabilization)

Synchronize and Stabilize: Stabilization Phase

- Tasks
  - Managers coordinate beta testers and collect feedback
  - Developers stabilize code
  - Testers isolate errors
- Tests
  - Internal testing (within Microsoft)
  - External tests (tests at "beta testers")
- Preparation for delivery
  - Burn finished product onto the "master" disc
  - Prepare documentation for printing
- Duration: 3 - 8 months

Synchronize and Stabilize: Schedule

- Planning: 3 - 12 Mon.
- Each of the 3 subprojects: 2 - 4 mon., whereby
  - 6 - 10 weeks coding, optimizing, testing, debugging and stabilizing the functionality
  - 2 - 5 weeks integration, testing and debugging
  - 2 - 5 weeks buffer time
- Stabilization: 3 - 8 Mon.

Development: 6 - 12 Mon.

12 - 32 months total

Synchronize and Stabilize

- Pro
  - Effective through short product cycles
  - Prioritization according to functions
  - Natural modularization by functions
  - Progress possible even without complete specification
  - Many developers work in small teams and thus just as effectively as a few
  - Feedback can be incorporated at an early stage

Synchronize and Stabilize

- Contra
  - Unsuitable for some types of software - architecture problems, poor fault tolerance, possibly real-time problems
  - Undefined team process: ad-hoc processes in each team, no learning across team boundaries
  - Every 18 months 50% of the code has been revised (code instability)

Synchronize and Stabilize Comparison with Waterfall Model

Sync-and-Stabilize	Waterfall
Product development and testing done in parallel	Separate phases done in sequence
Vision statement and evolving specification	Complete "frozen" specification and detailed design before building the product
Features prioritized and built in 3 or 4 milestone subprojects	Trying to build all pieces of a product simultaneously
Frequent synchronizations (daily builds) and intermediate stabilizations (milestones)	One late and large integration and system test phase at project's end
"Fixed" release and ship dates and multiple release cycles	Aiming for feature and product "perfection" in each phase
Customer feedback continuous in the development process	Feedback primarily after development as inputs for future projects
Product and process design so large teams work like small teams	Working primarily as a large group of individuals in separate functional departments

Linear and Prototype Models have Limitations

- Neither of these models deal well with frequent change
    - The waterfall and prototype model assumes that once you are done with a phase, all issues covered in that phase are closed and cannot be reopened
    - The Iterative and Sync&Stabilize model can deal with change between phases, but do not allow change within a phase
- What do you do if change is happening more frequently?

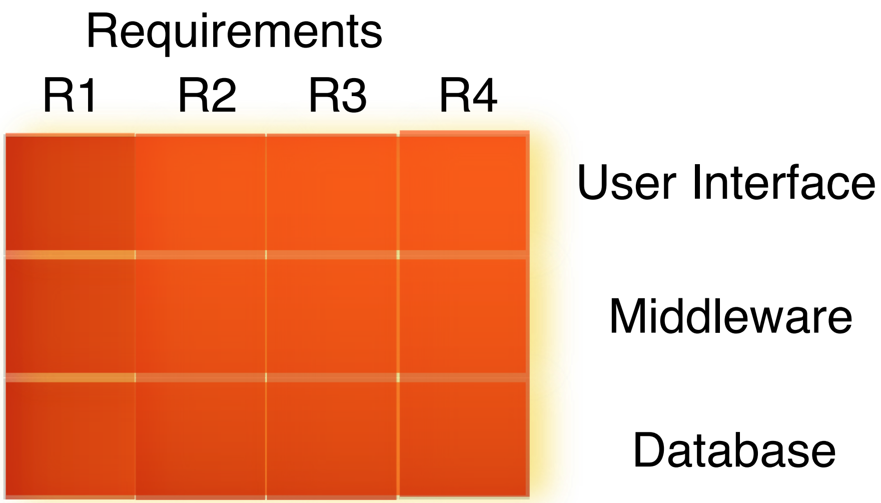
Incremental vs Iterative vs Adaptive

- **Incremental** means to “**add onto something”**
  - Incremental development extends your product
- **Iterative** means to “**re-do something”**
  - Iterative development debugs and improves your product
- **Adaptive** means **“react to changing requirements”**
  - Adaptive development improves the reaction to changing customer needs.

Incremental, Iterative, Adaptive Development

- **Incremental** means to "**add onto something**"
    - Incremental development extends your **product**

Bottom-up integration
Top-down integration
Vertical integration



Requirements

R1 R2 R3 R4

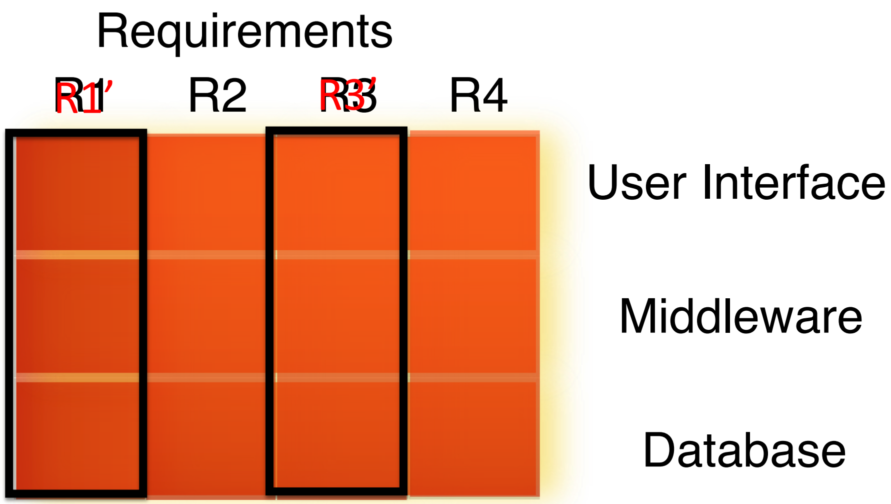
User Interface

Middleware

Database

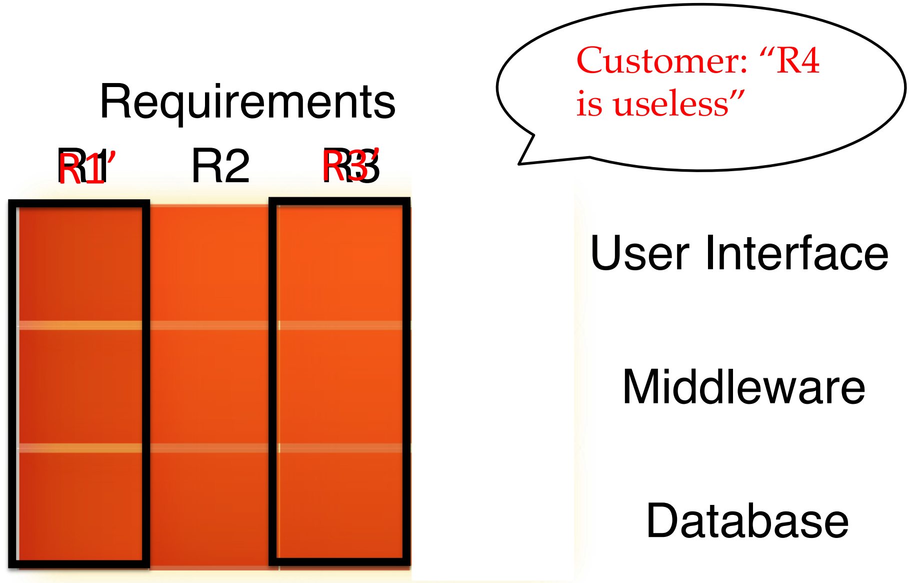
Incremental, Iterative, Adaptive Development

- **Incremental** means to “**add onto something**”
  - Incremental development extends your **product**
- **Iterative** means to “**re-do something**”
  - Iterative development debugs and improves your **product**



Incremental, Iterative, Adaptive Development

- **Incremental** means to “**add onto something”**
  - Incremental development extends your **product**
- **Iterative** means to “**re-do something”**
  - Iterative development debugs and improves your **product**
- **Adaptive** means to **“react to changing requirements”**
  - Adaptive development improves the reaction to changing customer needs



“People don’t know what they want until you show it to them.” - Steve Jobs.

Literature

[CusSel95] M. A. Cusumano, R. W. Selby, 1997, ACM, "How Microsoft builds software," at http://portal.acm.org/citation.cfm?id=255698

[MalPal99] Malik, S., Palencia, J., "Synchronize and Stabilize vs. Open-Source "at http://www.cs.toronto.edu/~smalik/downloads/paper_314.pdf.

Chapter 8.2 - Agile Processes

Walter F. Tichy