1.7 Terms

Software

- organized collections of computer data and instructions
- *disembodied information machines* (D. Gelernter, Mirror Worlds)

Program

- solves isolated task
- developed by a single person

Software system

- multiple components
- developed by team

Programming in the Small

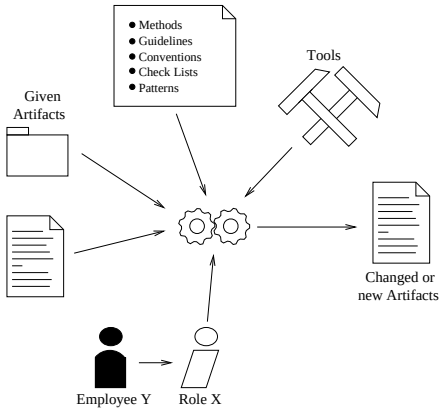
- development of a program or a component
- algorithmic aspects (sometimes)
- procedure:
  - **"stepwise refinement"** (N. Wirth),
  - **"structured programming"** (E. Dijkstra)
  - "structured control flow"
    (if-then-else, for, while, ...; no goto)
  - procedural decomposition, top-down
  - flat monolithic structure

Programming in the Large

- development of a **software system**:
  - **long life span**
  - **high probability of changes**
    (due to aging)
- **requirements** at first **fuzzy**
  - communication problem user $\leftrightarrow$ developer
  - understanding the problem
- **decomposition in components**
  (for programming in the small)
- **information hiding** (D.L. Parnas)
- promising approach:
  object-oriented analysis and design

1.8 Process Models

- process model: structured network of activities and artifacts
- an activity transforms artifacts



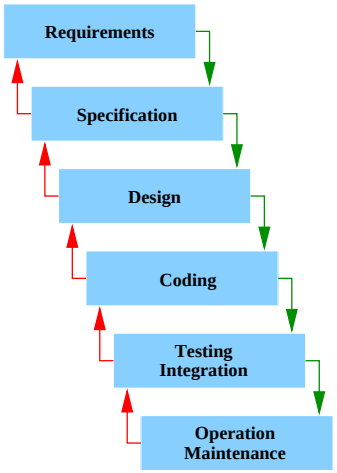
Phases

- Phases provide structure of process model
- Description of a phase
  - goals
  - activities
  - roles
  - required/new artifacts
  - patterns, guidelines, and conventions

Desiderata for Process Models

- the fewer phases, artifacts, roles, the better
- artifacts should cover standard case
- tool support
- quality assurance for each artifact
- traceability

1.9 The Classic: Waterfall Model



- early error correction is cheaper
  (e.g. after analysis phase 100 times cheaper than after deployment)
- hence, after every phase:
  check of previous phases
- potentially return to previous phase
- phases may overlap

1.9.1 Requirements Analysis

tractability

cost analysis

result:
decision on continuation of project

documents: (*artifacts*)

- **requirement specification** (Lastenheft)
- **cost estimation**
- **project plan**

1.9.2 Definition / Specification

starting point:
vague, incomplete, inconsistent requirements

result:
complete, consistent, unequivocal, accomplishable requirements

documents:

- system specification (Pflichtenheft)
- product model (e.g. OOA)
- GUI model
- user manual

- only **external behavior** of system
- **analysis of requirements**
- results in **system specification**
  - fixes the scope of the product
  - serves as basis for **contract** between customer and contractor
  - basis for **final acceptance**
  - contains
    - functionality
    - user interface
    - interfaces to other systems
    - performance (response time, space usage)
    - required hard and software
    - guidelines for documentation
    - time scheduling

1.9.3 Design

starting point: system specification / product model

- decomposition in components / subsystems
- fixes external behavior / interfaces of each component

result: software architecture (with specification of components)

1.9.4 Implementation and Testing

- translation of component specification to programming language
- compilation to machine language
- module testing

result: programmed system and testing protocols

1.9.5 Integration, system test, and deployment

- integration:
  - stepwise addition of single components
  - tested with data fixed in advance
    (functional requirements only)
- system test:
  - check of entire system (incl. hardware)
  - check of non-functional requirements
    (performance, GUI)
- deployment:
  - transfer of software system in its working environment

result: deployed product, protocol of final acceptance

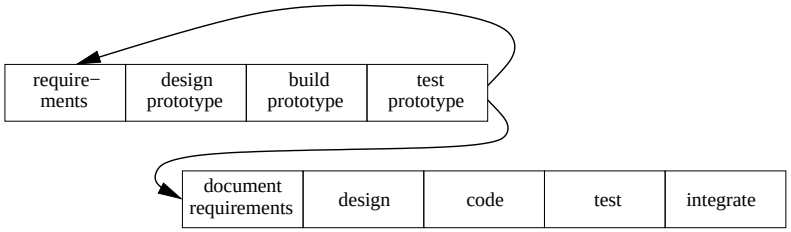
1.9.6 Maintenance

- bug fixes
- changes due to changes in requirements (incl. extensions)

result: maintained product

1.10 Prototyping Model

Lifecycle



Prototyping - Overview

Advantages:

- understanding the requirements for the user interface
- improves understanding between developer and client
- early testing of feasibility, usefulness, performance, etc.

Problems:

- users treat the prototype as the solution
- prototype is only a partial specification
- significant user involvement

1.11 Phased Models

Evolutionary Development

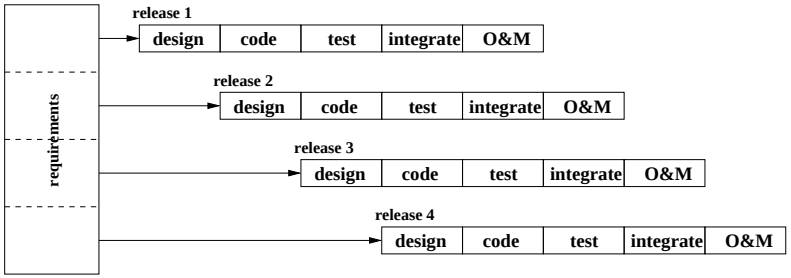
1. model core requirements
2. design and implement
3. deploy
4. feedback from customer
5. revise/extend requirements
6. revise/extend design
7. revise/extend implementation
8. iterate from 3 until all requirements met

Incremental Development

1. model all requirements
2. design and implement only core requirements
3. deploy
4. feedback from customer
5. revise requirements
6. design further requirements
7. implement further requirements
8. iterate from 3 until all requirements met

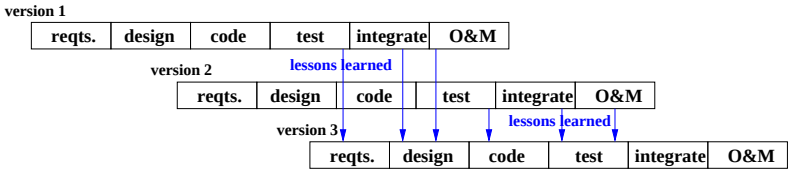
1.11.1 Incremental Development

(each iteration adds more functionality)

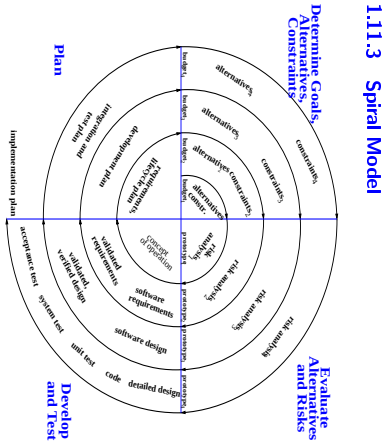


1.11.2 Evolutionary Development

(each iteration incorporates new requirements)



1.11.3 Spiral Model



1.11.4 Comments on Phased Models

- Incremental development
  - avoids 'big bang' implementation
  - but assumes all requirements known up-front
- Evolutionary development
  - allows for lessons from each version to be incorporated into the next
  - but: hard to plan for versions beyond the first;
    lessons may be learned too late
- Spiral model
  - incorporates prototyping and risk analysis
  - but: cannot cope with unforeseen changes
    not clear how to analyze risk

1.12 Agile Development Techniques

1.12.1 Extreme Programming (XP)

- Kent Beck 1999
- frequent releases
- short development cycles
- pair programming
- unit testing w tests developed before the code
- features specified by tests
- implement features when needed
- clear progress marks
- don't spend much time on design
- stakeholder involvement

1.12.2 Scrum

- Flexible approach to development; incremental process
- Adaptability to changing requirements
- 1986, Hirotaka Takeuchi and Ikujiro Nonaka

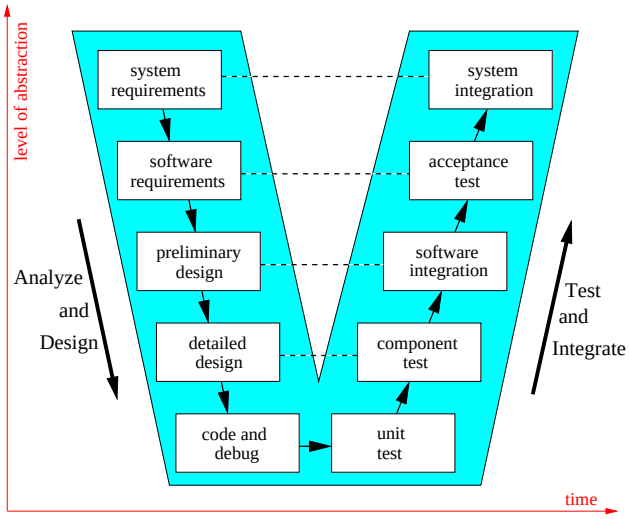
Roles Product owner, Scrum master, Team; Stakeholders, Managers

Sprint 2-4 weeks of intense development; goal: working increment that implements the sprint backlog; sprint backlog frozen during a sprint; self organization; burn down chart

Sprint Backlog requirements chosen for a sprint

Product Backlog as yet unimplemented requirements

1.13 V-Model "Entwicklungsstandard für Systeme des Bundes"



1.14 The Unified Software Process

Use-Case Driven

- Which user-visible processes are implemented by the system?
- Analysis, design, implementation, and testing driven by use-cases

Architecture centric

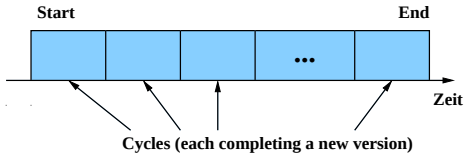
- Architecture developed in parallel to use cases (mutual dependency)

Iterative and Incremental

- eliminate risks first
- checkpoint after each iteration
- on failure of an iteration step, only current extension needs to be reconsidered
- small steps speed up project
- easy stepwise identification of the requirements

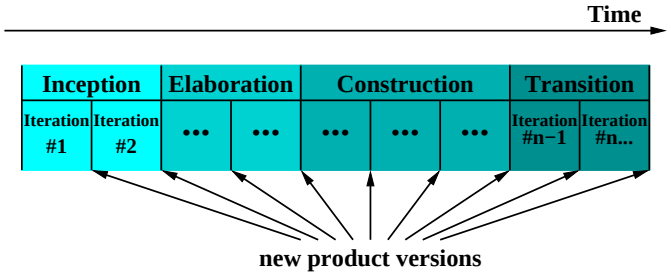
1.14.1 Structure of the Unified Software Process

- sequence of cycles
- after each cycle: product release with code, manuals, UML models, and test cases

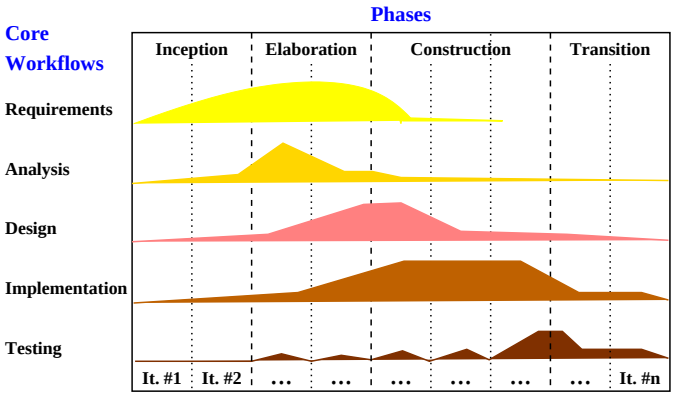


- cycle consists of 4 phases:
  Inception, Elaboration, Construction, Transition
- each phase consists of iterations

Cycle



Main-Workflows and Phases



- each phase ends with a mile stone
- each phase processes all workflows (with varying intensity)

1.14.2 Inception Phase

- GOAL: rough vision of the product
- functionality of system from users' perspective
  most important use cases (stakeholder needs)
- preliminary sketch of suitable architecture
- project plan and cost
- identify most important risks (with priorities)
- plan elaboration phase

1.14.3 Elaboration Phase

- specify (most) use cases in detail
- design architecture
- prototype (proof-of-concept for architecture)
- implement most important use cases
- result: initial architecture
- plan activities and resources for remaining project
- use cases and architecture stable?
- risk management?

1.14.4 Construction Phase

- implement system
- high resource needs
- small architectural changes
- **GOAL:** system ready for customer (small errors acceptable)

1.14.5 Transition Phase

- deliver beta-version to customer
- address problems (immediately or in next release)
- train customer
- hotline

1.15 Summary

- Software has unique problems with far-reaching consequences
- Creating software systems requires structured process models
- Classic process phases: waterfall model
- Further process models: prototyping, evolutionary, incremental, spiral, agile, V-model, unified