Software Engineering

Lecture 02: Processes

Peter Thiemann

University of Freiburg, Germany

SS 2014

Terms

Software
- Organized collection of computer data and instructions

Component
- Solves isolated task
- Developed by a single person

SW System
- Multiple components
- Developed by a team

Programming in the Small

- Development of a system comprised of a small number of “mind-sized” components
- **Requirements** often clear
- Sometimes algorithmic aspects
- Procedure for a single component:
  - Procedural decomposition, top-down
  - **“stepwise refinement”** (N. Wirth),

Programming in the Large

- Development of a **software system** comprised of many components
- **Requirements** at first **fuzzy**
- Size or complexity dictate . . .
  - decomposition in a large number of components
  - development in a team
  - size determines duration, but beware of Brook's law!

Brook's law: Adding manpower to a late SW project makes it later

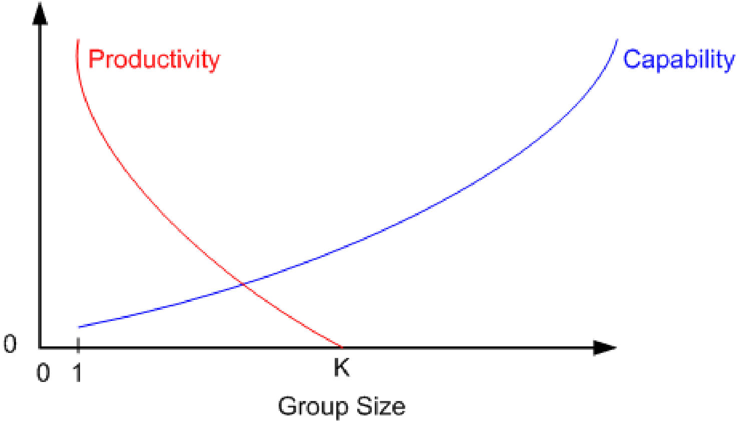


Image: `http://bulldozer00.files.wordpress.com/2010/11/productivity-and-capability.png?w=595`

Issues Arising with Programming in the Large

- **Requirements** need to be investigated
    - Communication problem customer ↔ developer
    - Understanding the problem
- **Design** of the system is significant task
    - **Decomposition in components** (interfaces, contracts)
    - **Information hiding** (D.L. Parnas)
    - Design for maintenance
        - **Long life span**
        - **High probability of changes** (aging)
    - Promising approach: **object-oriented analysis and design**
- **Construction** of components: programming in the small
- **Testing** required on many levels

Conclusion

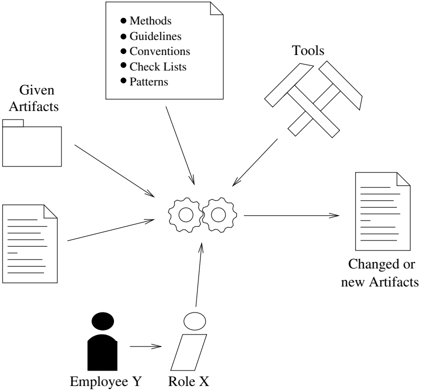
- Programming in the large is a structured approach to all activities in the development of a software system
- Unfortunately, . . .

Conclusion

- Programming in the large is a structured approach to all activities in the development of a software system
- Unfortunately, . . .
  - there are many overall approaches (process models)
  - there are many techniques with similar goals

Process Models

- **Process Model**: structured network of **activities** and **artifacts**
- An activity transforms a set of artifacts into new artifacts



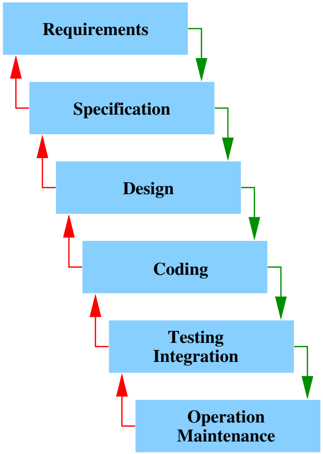
Phases

- 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

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

Requirements Analysis

tractability

cost analysis

result:

decision on continuation of project

documents: (*artifacts*)

- **Requirement specification** (Lastenheft)
- **Cost estimation**
- **Project plan**

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**

Definition / Specification (cont'd)

- Only **external behavior** of system
- **Analysis of requirements**
  - functional / non-functional requirements
  - prioritization
- Main outcome: **system specification**
  - fixes the scope of the product
  - serves as basis for **contract** between customer and contractor
  - basis for **final acceptance**
  -
    - functionality
    - user interface
    - interfaces to other systems
    - performance (response time, space usage)
    - required hard and software
    - guidelines for documentation
    - time scheduling
    - quality

Design

starting point: system specification / product model

- Decomposition in components / subsystems
- Logical interfaces of each component
- Choice of technologies

result: **Software architecture** (with specification of components)

Implementation and Testing

starting point: Software architecture

- Coding of component specifications
- Compilation to machine language
- Unit testing up to component level

result: implemented components and testing protocols

Integration, system test, and deployment

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

result: deployed product, protocol of final acceptance

Maintenance

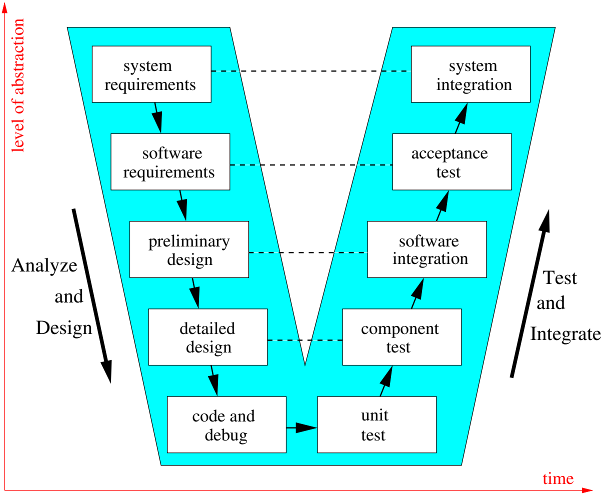
- Supervision
- Bug fixes
- Changes due to changes in requirements (incl. extensions)

result: maintained product

Concrete Process Models

1. V-Model
2. Prototyping model
3. Phased models (evolutionary, incremental, spiral)
4. Unified Software Process
5. Agile development techniques

V-Model *“Entwicklungsstandard für Systeme des Bundes”*

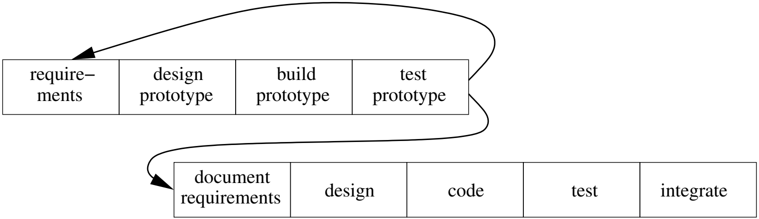


V-Model

- Builds on waterfall model
- Emphasizes validation connections between late phases and early phases
- Objectives
  - risk minimization
  - quality assurance
  - cost reduction
  - communication between stakeholders
- Current instance: V-Model XT

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:

- customers treat the prototype as the product
- a prototype is **not** a specification
- significant user involvement

Phased Models

Evolutionary Development

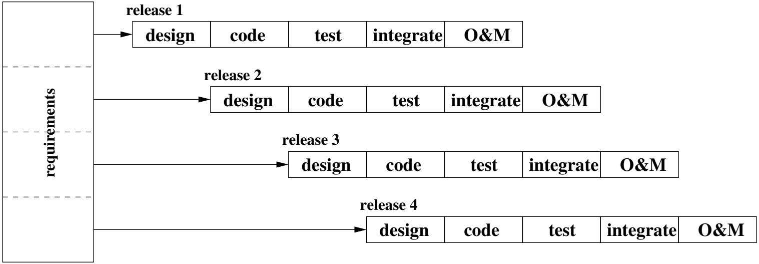
1. model **core requirements**
2. design and implement
3. deploy
4. feedback from customer
5. revise/extend requirements
6. revise/extend design
7. revise/extend implementation
8. iterate from 3 until all requirements met

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

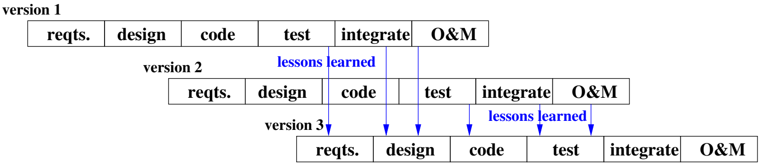
Incremental Development

(each iteration adds more functionality)



Evolutionary Development

(each iteration incorporates new requirements)



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

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

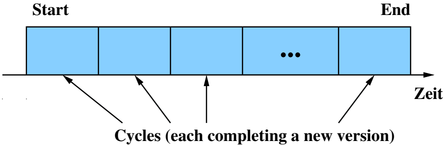
- Architecture developed in parallel to use cases (mutual dependency)

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

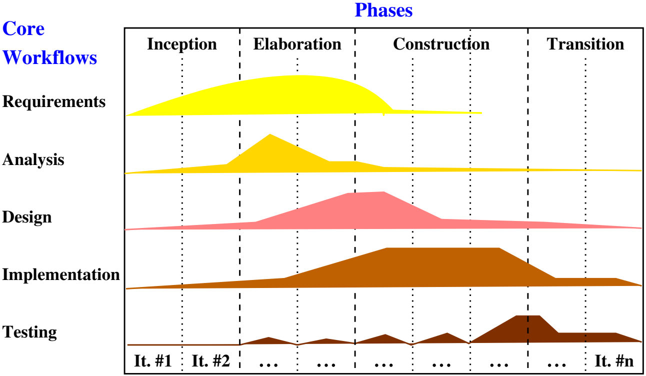
Structure of the Unified Software Process

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



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

Main-Workflows and Phases



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

Inception Phase

- 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
- **GOAL:** rough vision of the product

Elaboration Phase

- specify (most) use cases in detail
- design initial architecture
- implement most important use cases
- plan activities and resources for remaining project
- consider risks
- **GOAL:** prototype (proof-of-concept for architecture)

Construction Phase

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

Transition Phase

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

Agile Development Techniques

Extreme Programming (XP, Kent Beck 1999)

- frequent releases
- short development cycles
- pair programming
- unit testing w tests developed before the code
- features (requirements) exemplified by tests
- features implemented when needed
- features implemented serve as progress marks
- ahead-of-time design deemphasized
- stakeholder involvement

Agile Development Techniques

Scrum (Hirotaka Takeuchi and Ikujiro Nonaka 1986)

- Flexible approach to development in a self-organizing team
- Incremental process
- Requirements organized in a **product backlog**
- Development structured into **Sprints** (2-4 weeks of intense development)
  - **Sprint backlog**: requirements chosen for a sprint (frozen)
  - **Burndown chart**: progress meter
- Communication structure: sprint planning, daily standup meetings, sprint review
- Team structure: Product owner, Scrum master, Team (Stakeholders, Managers)

Summary

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