Software Engineering

Lecture 04 – Software Processes

© 2015-20 Dr. Florian Echtler
Bauhaus-Universität Weimar
<florian.echtler@uni-weimar.de>

Software Processes

- SPs are “activities involved in producing a software system”
- SP models are “abstract representations of these processes”

(Definitions from [Sommerville2011])

- *There is no ideal process.*
- *One size does not fit all.*

Fundamental Activities

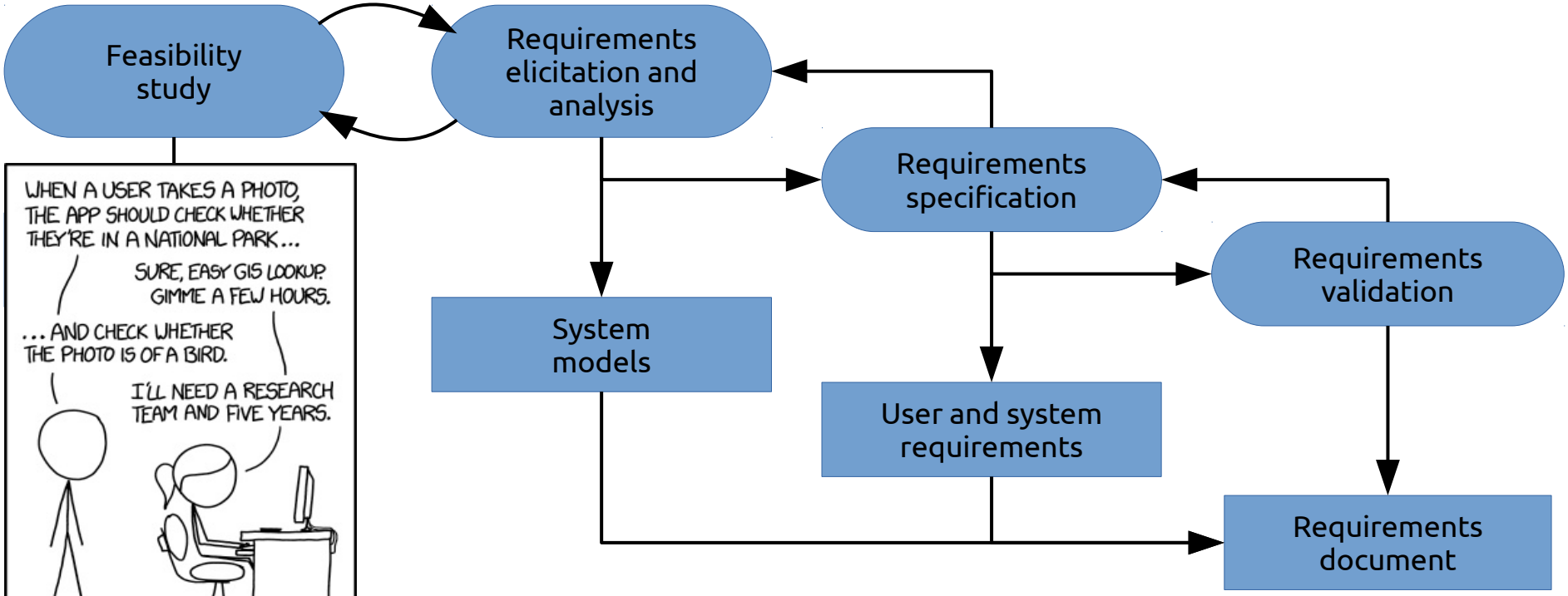
- 4 fundamental activities – software …
  - Specification
  - Design & Implementation
  - Validation
  - Evolution
- Major components:
  - Products (= outcomes of an activity)
  - Roles (= responsibilities of people involved)
  - Pre- and post-conditions

Software Specification

- Defines functionality of/constrains on the software product
- Also known as *requirements engineering*
- Nearly always the initial step
- Sub-activities:
  - Feasibility study
  - Requirements elicitation/analysis
  - Requirements specification
  - Requirements validation

Software Specification

Image source (FU): Sommerville, Software Engineering, Chapter 2, https://xkcd.com/1425/



Requirements Specification

- User requirements: (“Lastenheft”)
  - Statements in natural language (+ diagrams)
  - What services is the system expected to provide?
  - What constraints is it expected to observe?
  - Often part of the call for bids (“Ausschreibung”)
- System requirements: (“Pflichtenheft”)
  - Detailed description of functions/services
  - Defines exactly what is to be implemented
  - Often part of the contract

User requirements

- “Lastenheft” → customer's view
- Clear, rigid structure, e.g. from [Balzert2009]
    - Goals
    - Application area
    - Functions
    - Data
    - (additional) Services
    - Quality requirements
    - Appendix

System requirements

- “Pflichtenheft” → developer's view
- Extension of UR with additional sections
    - Goals
    - Application area
    - **Product environment**
    - Functions
    - Data
    - (additional) Services
    - Quality **Goals**
    - **Test scenarios**
    - **Development environment**
    - Appendix

Requirements Specification (2)

Source (FU): Sommerville, Software Engineering, Chapter 4

- User requirements example (“feature wishes”)
  - The patient management system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.
- System requirements example (“testable”)
  - On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated. The system shall automatically generate the report for printing after 17:30 on the last working day of the month.
  - A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs. If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.
  - Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Requirements Specification (3)

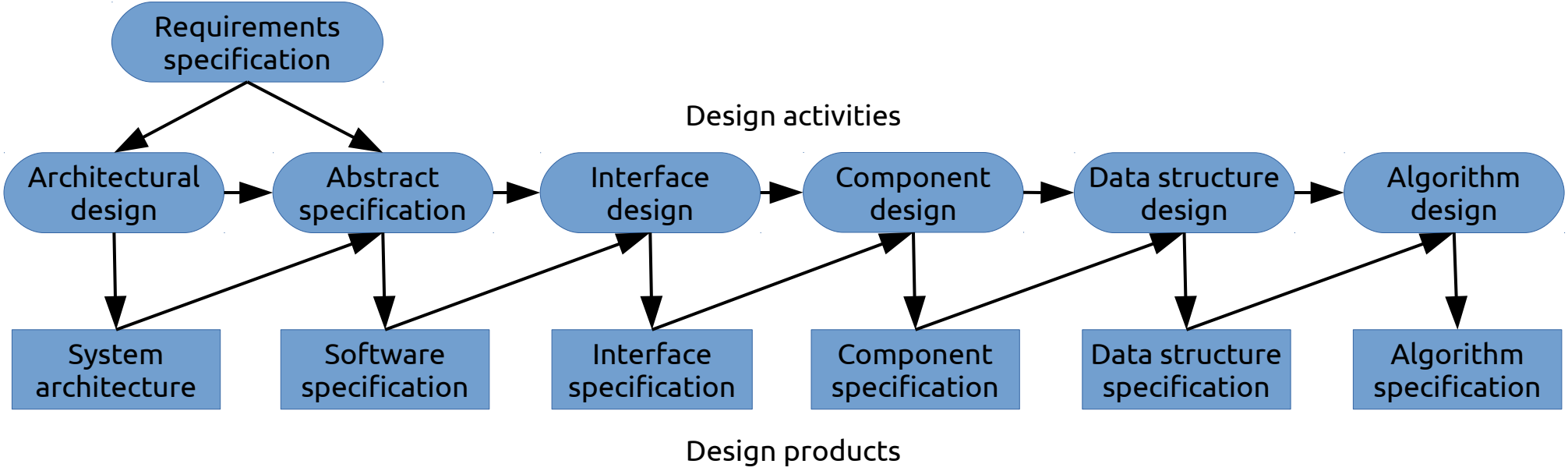
- Functional requirements
  - What should the system (not) do?
  - E.g. “A user shall be able to search the appointment lists for all clinics.”
- Non-functional requirements
  - Reliability, response time, security, ease of use …
  - E.g. “The system shall be available during normal working hours (Mo-Fr, 8:30 – 17:30). Downtime during NWH shall not exceed 5 sec. per day.”
  - “The system shall conform to ISO Standard XYZ.”

Software Design & Implementation

- Conversion from system specification to executable system
- Sub-activities:
  - Design
    - e.g. architecture, interface, component, database, …
    - Often involves graphical models, UML
  - Implementation
    - Often interleaved with design (depending on SPM)
    - Also involves testing & debugging

Software Design & Implementation

Image source (FU): Sommerville, Software Engineering, Chapter 2

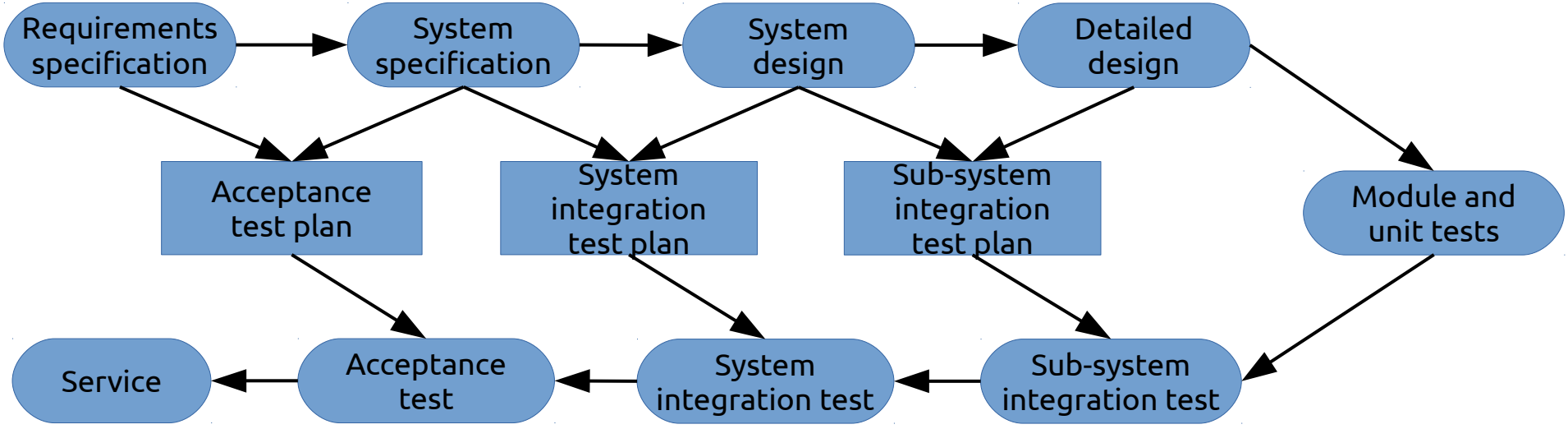


Software Validation

- Verification and validation (V&V)
- Validation → “Are we building the right product?” [Boehm79]
  - User testing
  - Acceptance testing (with user-supplied data)
- Verification → “Are we building the product right?” [Boehm79]
  - Unit testing (with developer-supplied data)
  - System testing

Software Validation

Image source (FU): Sommerville, Software Engineering, Chapter 2



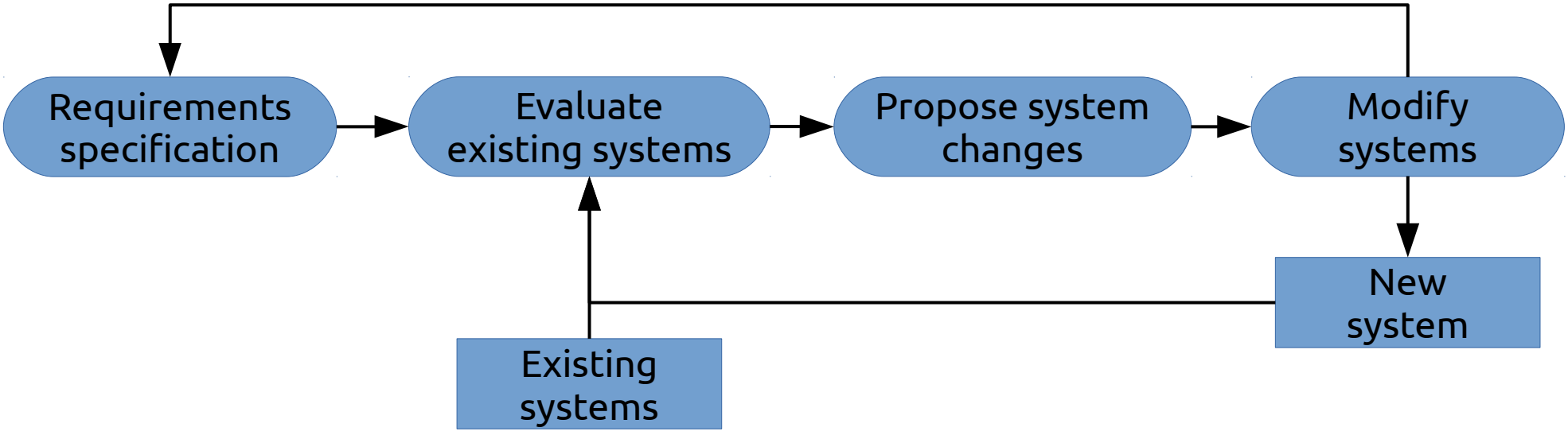
Software Evolution

- Also known as software maintenance
- Often higher costs than initial development
- Integral part of the entire software process

Software Evolution

Image source (FU): Sommerville, Software Engineering, Chapter 2

- Beware of “not-invented-here” (NIH) syndrome



“Plan-driven” vs. “Agile”

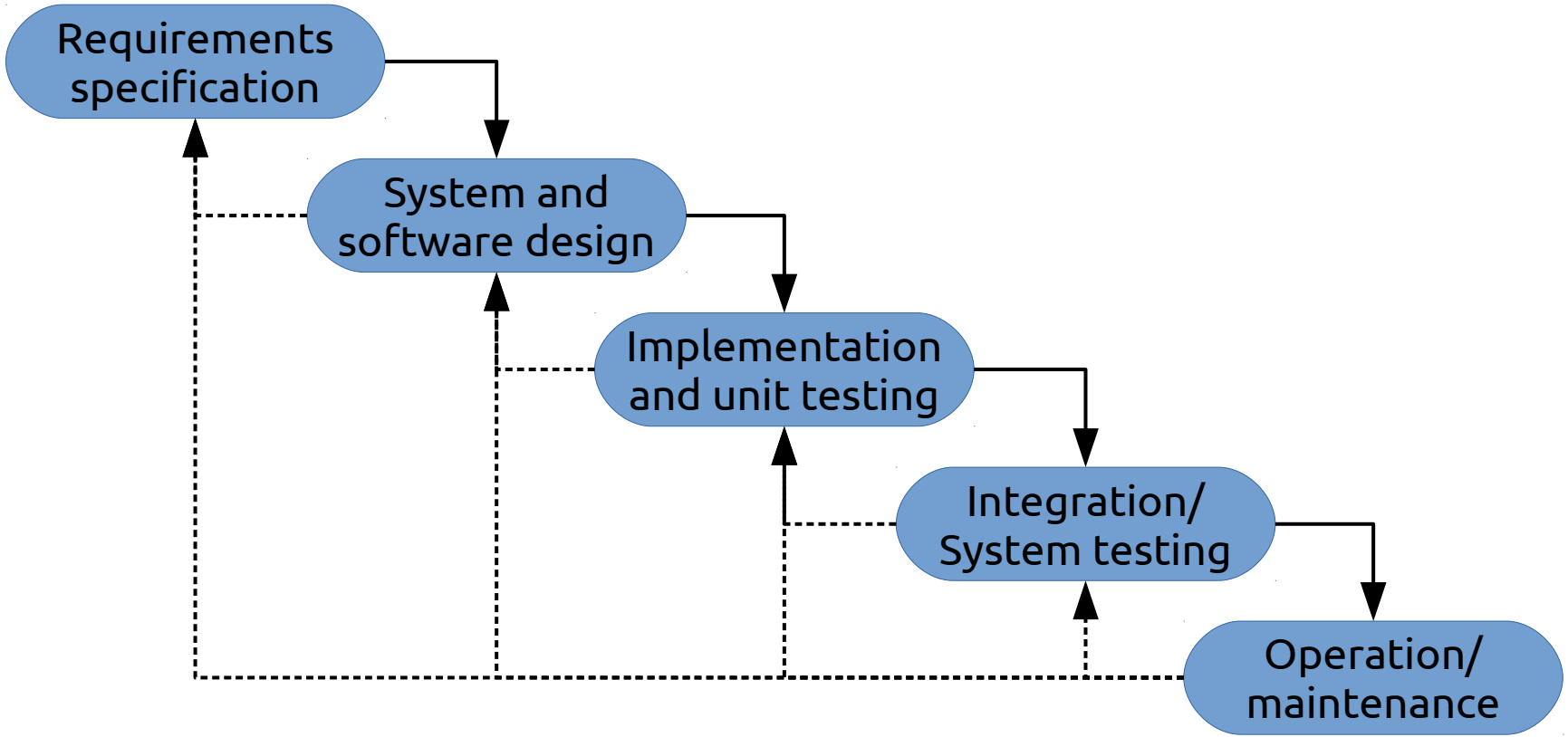
- Plan-driven processes:
  - All steps/activities planned in advance
  - Progress measured against this plan
- Agile processes:
  - Incremental planning
  - Easier adaptation to change
  - More difficult to measure progress
- Many hybrid methods

Example Software Processes

- Waterfall model
- Other “traditional” models
- Incremental development
- Reuse-oriented development

© Scott Adams, Inc./Dist. by UFS, Inc.

Waterfall Model



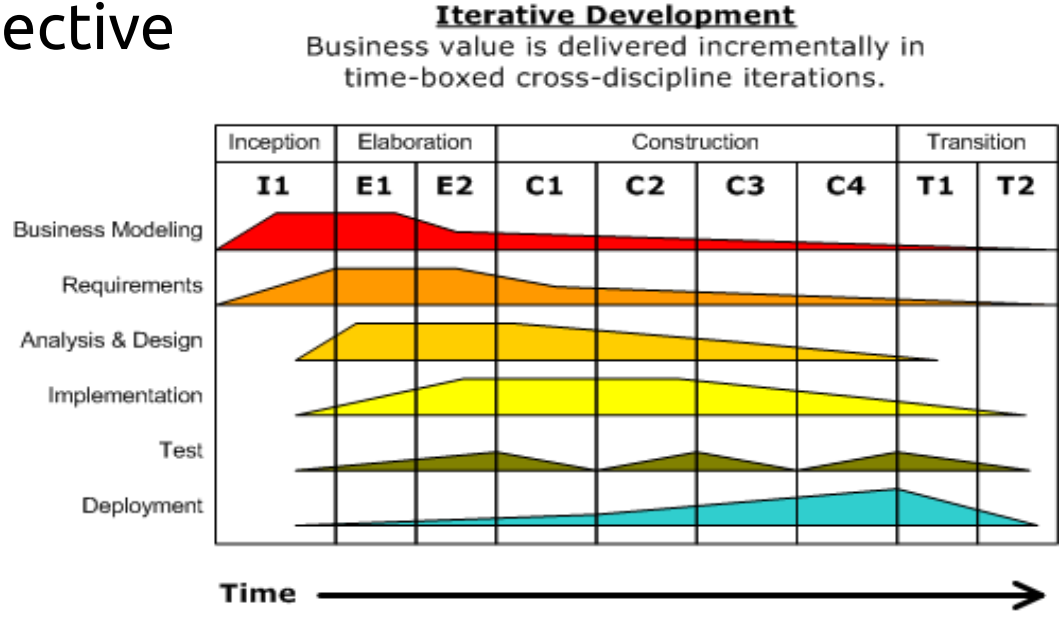
Waterfall Model

- Derived from general systems engineering
- Phases …
  - have clear lifetimes (plan-driven)
  - end with “sign-off” on specific documents
    → iterations are expensive
- Used in formal development (proof-driven, for safety-critical systems)
- Very inflexible

Other “traditional” models

Image source (CC): https://en.wikipedia.org/wiki/Rational_Unified_Process

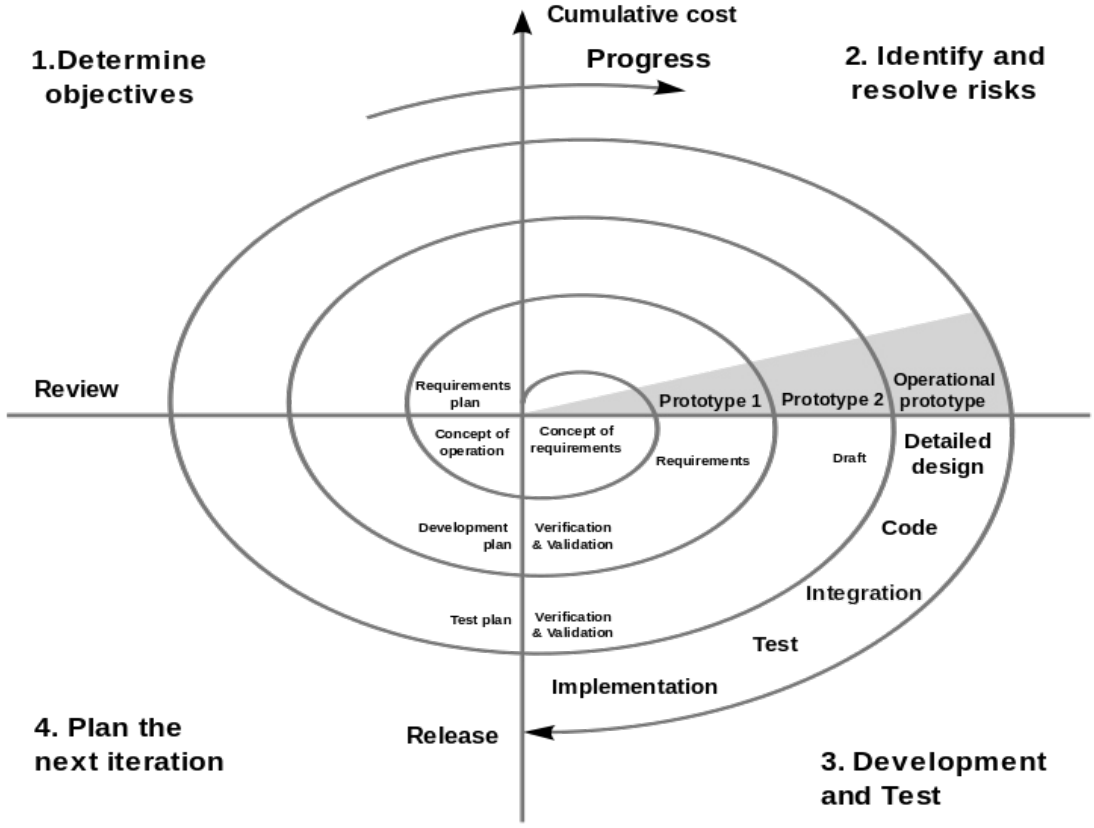
- Rational Unified Process (RUP) [Krutchen03]
- Derived from work on UML
- Focus on business perspective
- Focus on iteration



Other “traditional” models

Image source (PD): https://en.wikipedia.org/wiki/Spiral_model

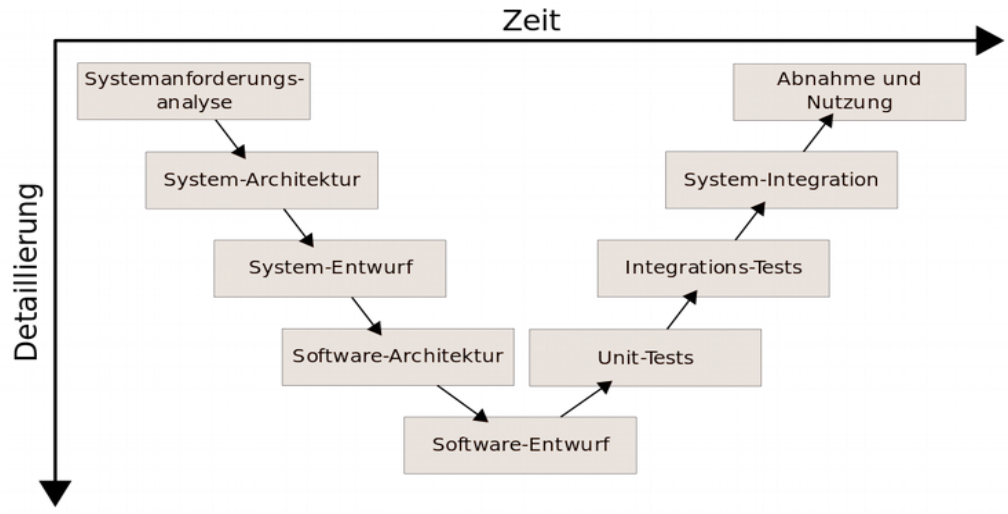
- Spiral Model [Boehm88]
- 4 sectors for each cycle
- Focus on risk assessment



Other “traditional” models

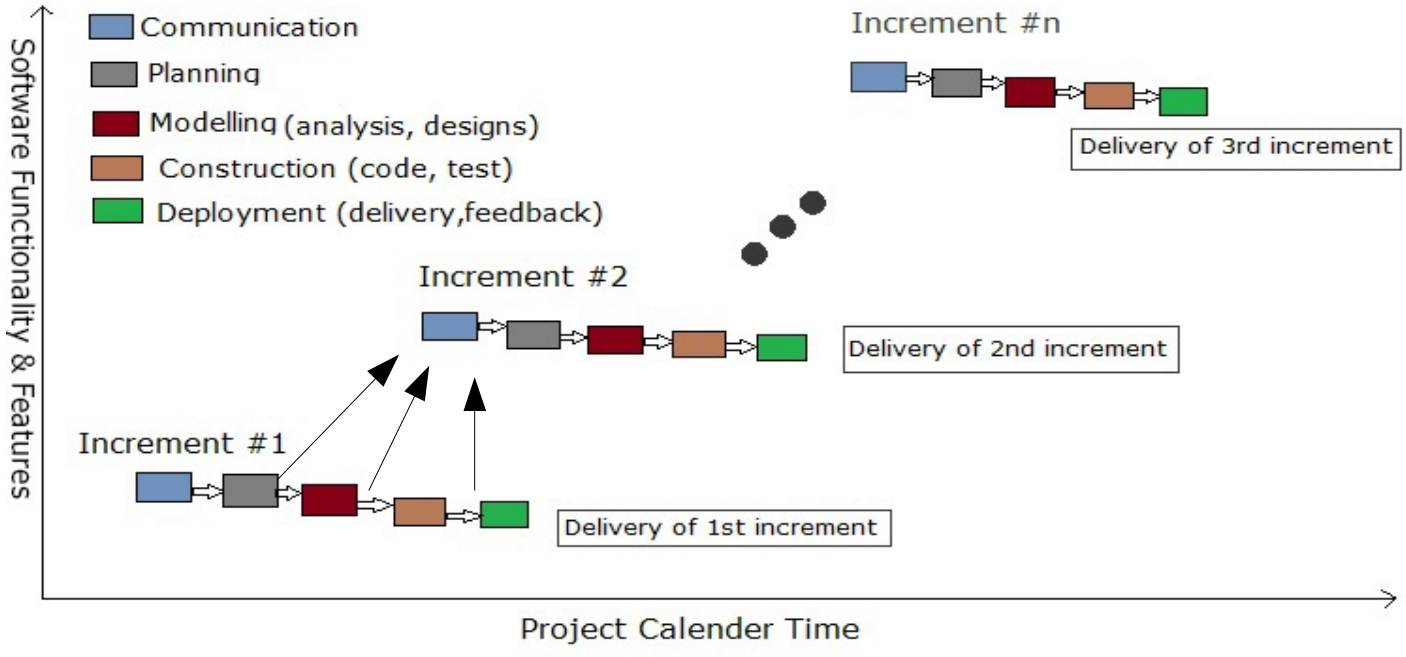
Image source (CC): https://de.wikipedia.org/wiki/V-Modell

- V Model [Boehm79], based on Waterfall
- Often used by German government
- Current variant: “V-Modell XT”



Incremental Development

Image source (CC): https://en.wikipedia.org/wiki/Incremental_build_model



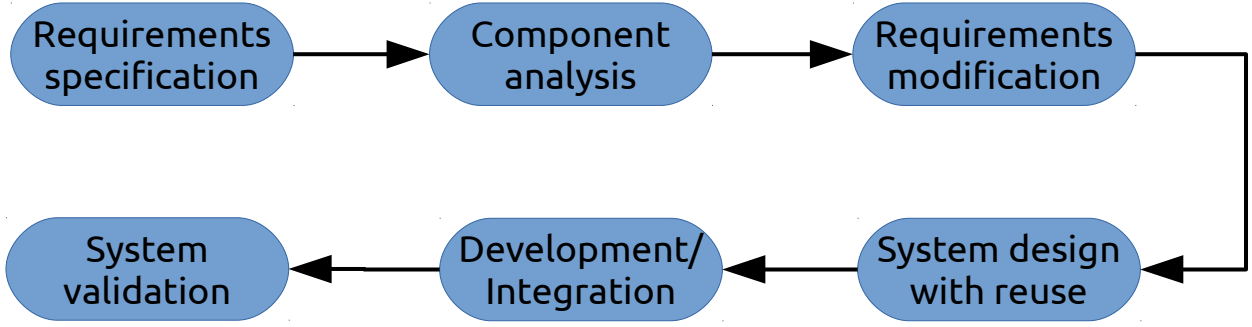
Incremental Development

- Also known as evolutionary development
- Basis for many agile processes
- Quickly develop multiple prototypes
- Test with end users, refine, repeat
- Very useful for GUI applications

Reuse-oriented Development

Image source (FU): Sommerville, Software Engineering, Chapter 2

- Focus on reuse/adaptation of existing components
- Needs requirement compromises
- Often used in Web context (e.g. LAMP stack)



Reacting to Change

- Change *will* happen during the SP
- Two strategies for coping with change:
  - Change avoidance → Prototyping
  - Change tolerance → Incremental Delivery

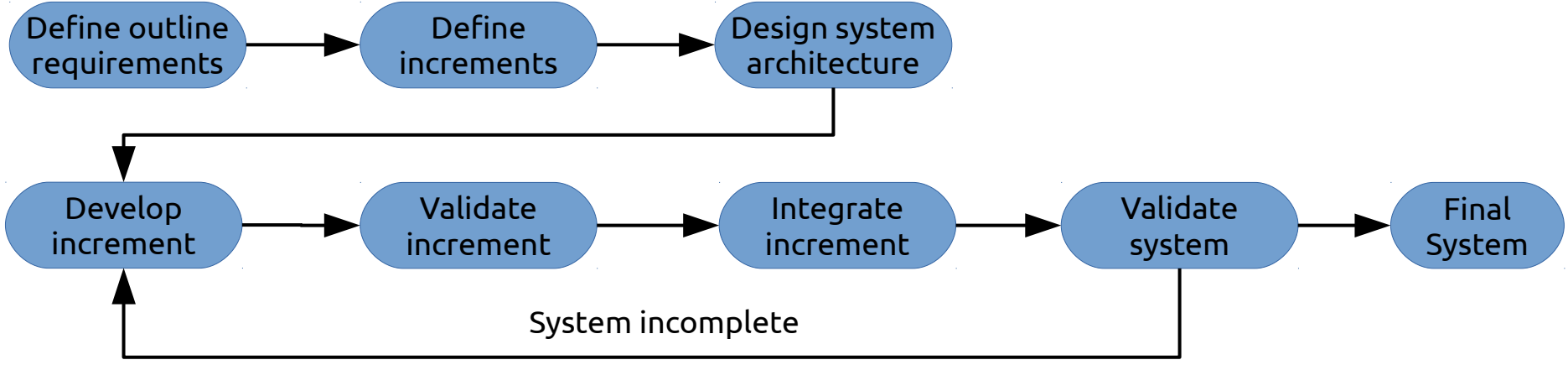
Prototyping

- Quickly create prototypes and expose to users
- Useful during all phases
- Initial prototypes may not even contain code (e.g. paper prototypes, mockups, Wizard-of-Oz)
- Be prepared to throw entire prototypes away

Incremental Delivery

Image source (FU): Sommerville, Software Engineering, Chapter 2

- Order functionality by importance
- Deliver most important functions to the end user first
- Either plan-driven (increments defined in advance) or agile (inc. defined on-the-fly)



Questions/suggestions?

Image source (FU): http://www.paragoninnovations.com/ng4/guide.shtml