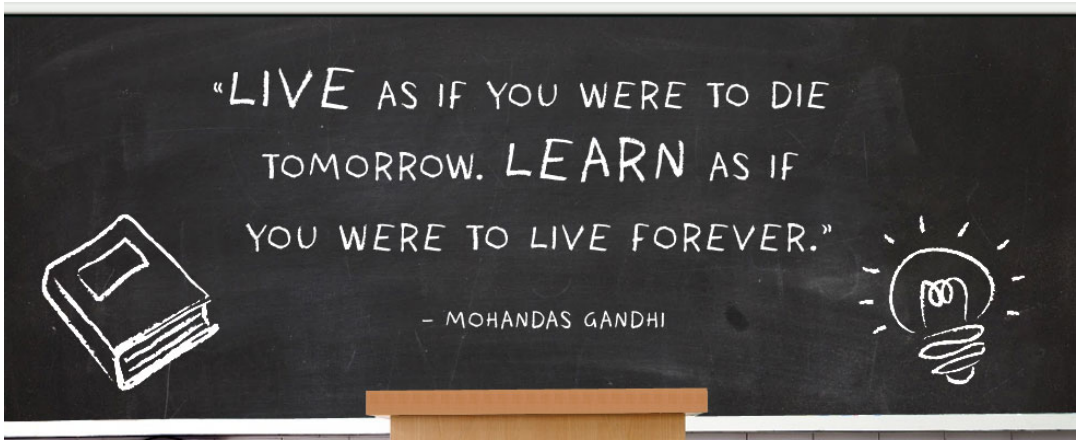
Vorlesung Softwaretechnik
SS 2019 / Teil II

Prof. Dr.-Ing. Samuel Kounev
M.Sc. Johannes Grohmann



Kontakt

- Prof. Dr.-Ing. Samuel Kounev

  Lehrstuhl für Informatik II – Software Engineering

  Am Hubland, Informatikgebäude, Raum A112

  97074 Würzburg

  Tel. +49 931 31 86601

  Email: samuel.kounev@uni-wuerzburg.de

  Website: http://se.informatik.uni-wuerzburg.de

- Sprechstunde: Nach Vereinbarung

Studienkoordinator / Übungsleiter

- M.Sc. Johannes Grohmann

  Lehrstuhl für Informatik II – Software Engineering

  Am Hubland, Informatikgebäude, Raum A110

  97074 Würzburg

  Tel. +49 931 31 84568

  Email: johannes.grohmann@uni-wuerzburg.de

- Sprechstunde: Nach Vereinbarung

Übungsleiter

- M.Sc. Thomas Prantl

  Lehrstuhl für Informatik II – Software Engineering

  Am Hubland, Informatikgebäude, Raum A118

  97074 Würzburg

  Tel. +49 931 31 82573

  Email: thomas.prantl@uni-wuerzburg.de

- Sprechstunde: Nach Vereinbarung

Lehrstuhl für Informatik II (March 2018)

Lehre am Lehrstuhl II

- **Bachelor**
  - Vorlesung “Softwaretechnik” (SWT)
  - Vorlesung “Betriebssysteme”
  - Vorlesung “Grundlagen der Programmierung (GdP)”
  - Softwarepraktikum, Programmierpraktikum
  - Seminar "Software Engineering“, "Programmierparadigmen"

- **Master**
  - Vorlesung “Performance-Engineering & Benchmarking”
  - Vorlesung “Software-Architektur“
  - Vorlesung ”Fortgeschrittenes Programmieren”
  - Vorlesung „Self-Aware Computing“
  - Praktikum "Software-Entwurf und -Qualität“

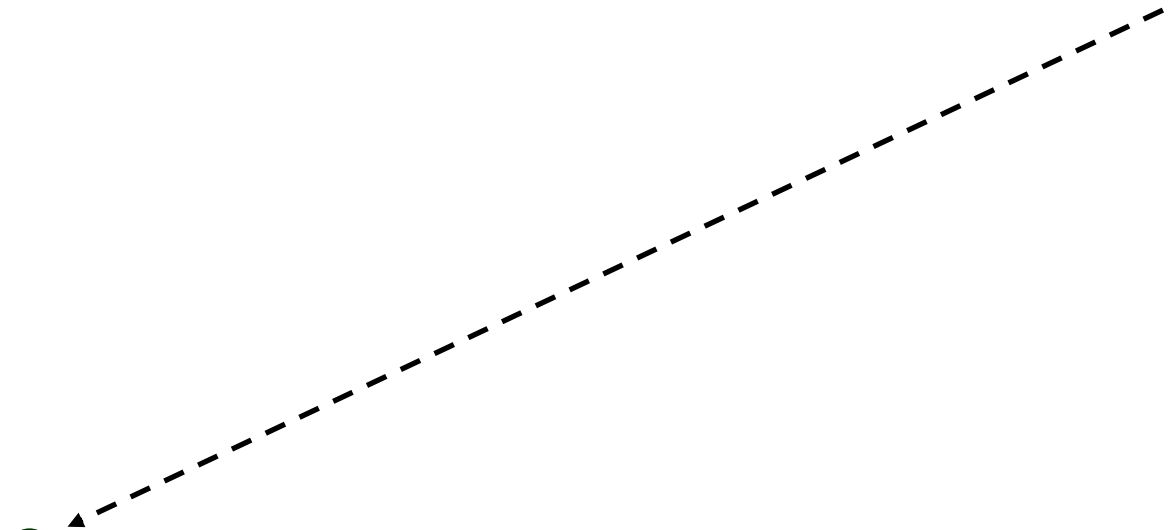
SWT-Vorlesungsübersicht

- **Teil I: Objektorientierte Softwareentwicklung mit UML**
  - Überblick über Software Engineering
  - Objektorientierter Softwareentwurf mit UML
  - Entwurf von graphischen Benutzungsoberflächen
  - Grundlagen von Datenbanken und objekt-relationale Abbildung

- **Teil II: Entwicklungsprozesse & Softwarearchitektur**
  - Softwareentwicklungsprozesse mit Fokus auf den Unified-Process
  - Anforderungsanalyse (engl. requirements engineering)
  - Softwarearchitektur
  - Entwurfsmuster

Vorlesungsmaterialien (Teil II)

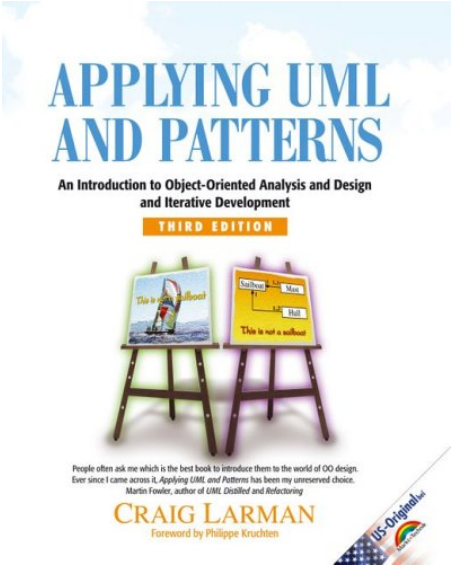
- Folienklassifizierung
  - Primäres Material → 
  - Sekundäres Material → 
  - Optionales Material → 



Literatur (Teil II)

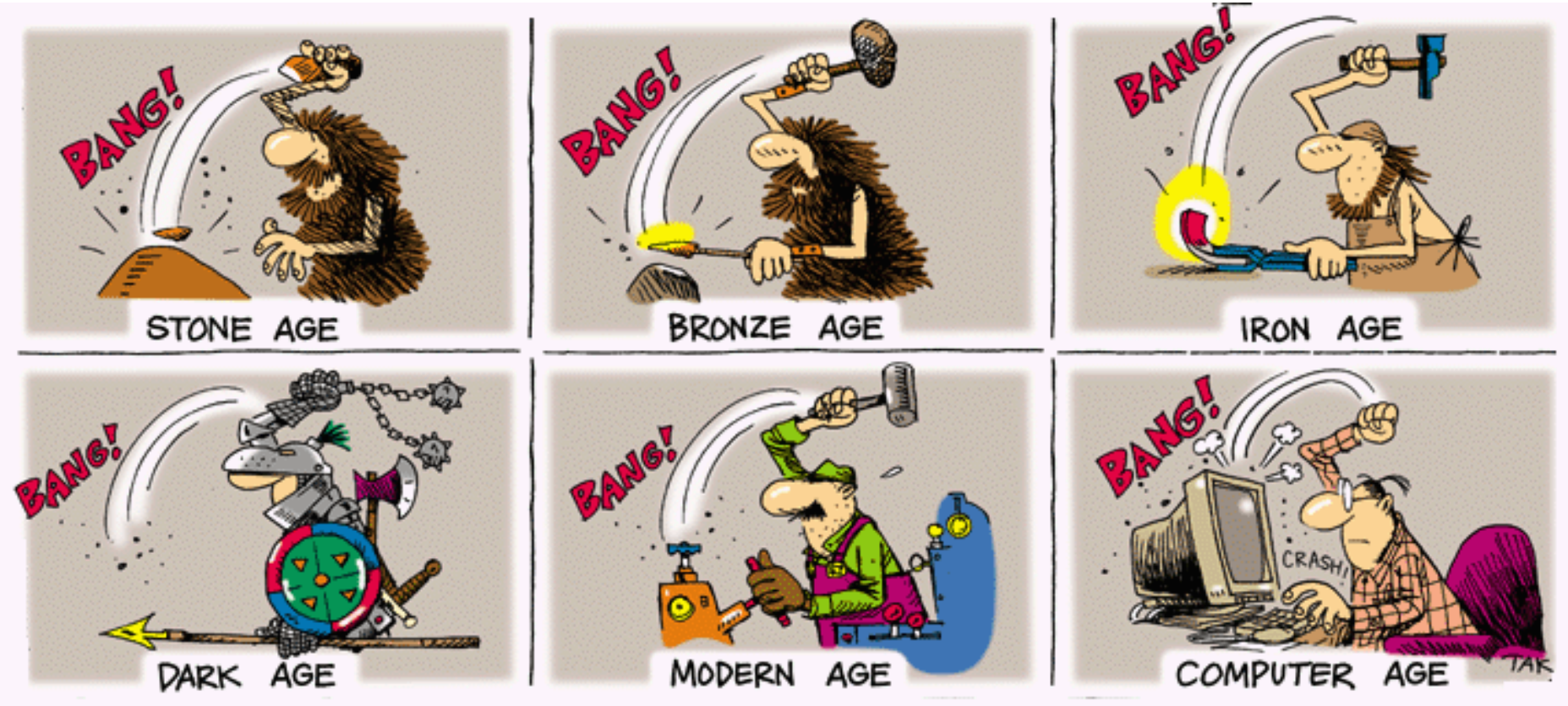
- Graig Larman “Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and iterative Development”, 3rd edition, Prentice Hall, 2005

- UML 2 und Patterns Angewendet, Mitp-Verlag, 2005



Fragen?

Der Computer als Werkzeug



Live-Feedback-System

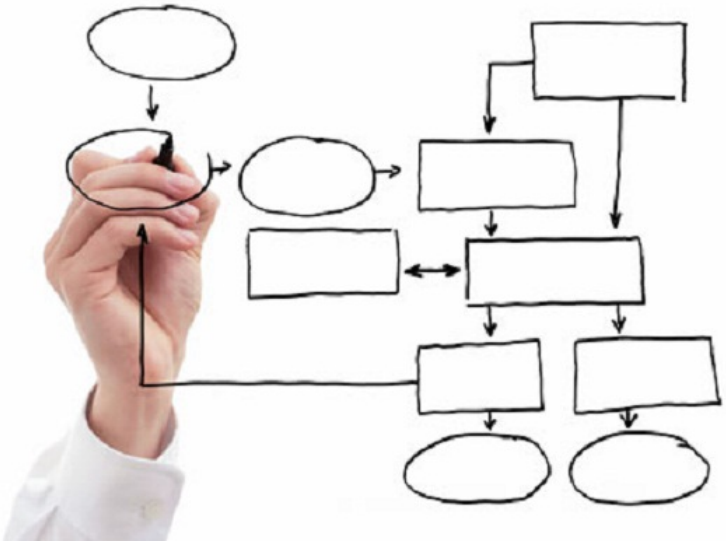
- PINGO = „Peer Instruction for very large groups“
- Wird für Live-Abstimmungen/Umfragen verwendet
- WICHTIG: Folgende Adresse „bookmarken“:

http://pingo.upb.de/4165

Softwaretechnik (Teil II)

1. Software Development Process Models

Prof. Dr.-Ing. Samuel Kounev



Acknowledgements

- Lecture materials from Prof. Dr. Oliver Hummel (University of Mannheim) have been used as a basis

- Further lecture materials that have been used
  - Craig Larman
  - Prof. Walter Tichy (KIT)

Objectives

- Know and understand the characteristics of the most important process models
  - Be able to describe their most important features and elements
- Be able to select an appropriate process model for a given project
  - Respectively to rule out inappropriate process models

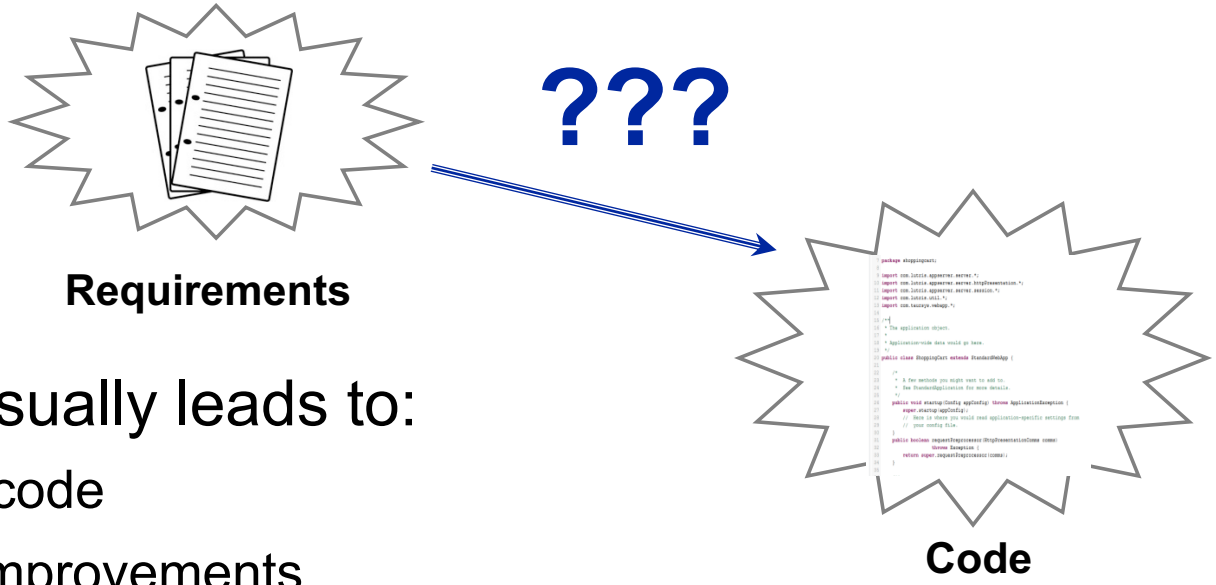
Roadmap

1. Introduction
2. The Waterfall Model
3. V Model
4. Spiral Model
5. Iterative and Incremental Development
6. (Rational) Unified Process
7. Software Prototyping
8. Agile Development

Appendix: Selection of process models

Code and Fix Approach

- Is it helpful?



- **Code and Fix** usually leads to:
  - badly structured code
  - non-systematic improvements
  - no real team work, as tasks cannot be planned
  - missing design and documentation
  - complications in maintenance
  - …
- Are structured processes the way out?

Software Development Processes

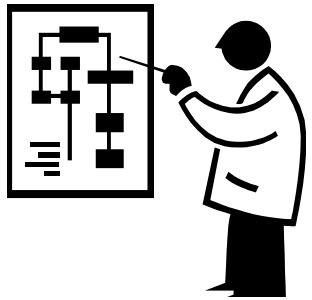
- A way out? → Software Development Processes
  - *Waterfall model*
  - *Spiral model*
  - *V model*
  - *RUP*
  - *…*
  - *Agile methods*

- Typical advantages of processes:
  - _____
  - _____
  - _____

➔ however, this depends on concrete project and selected process

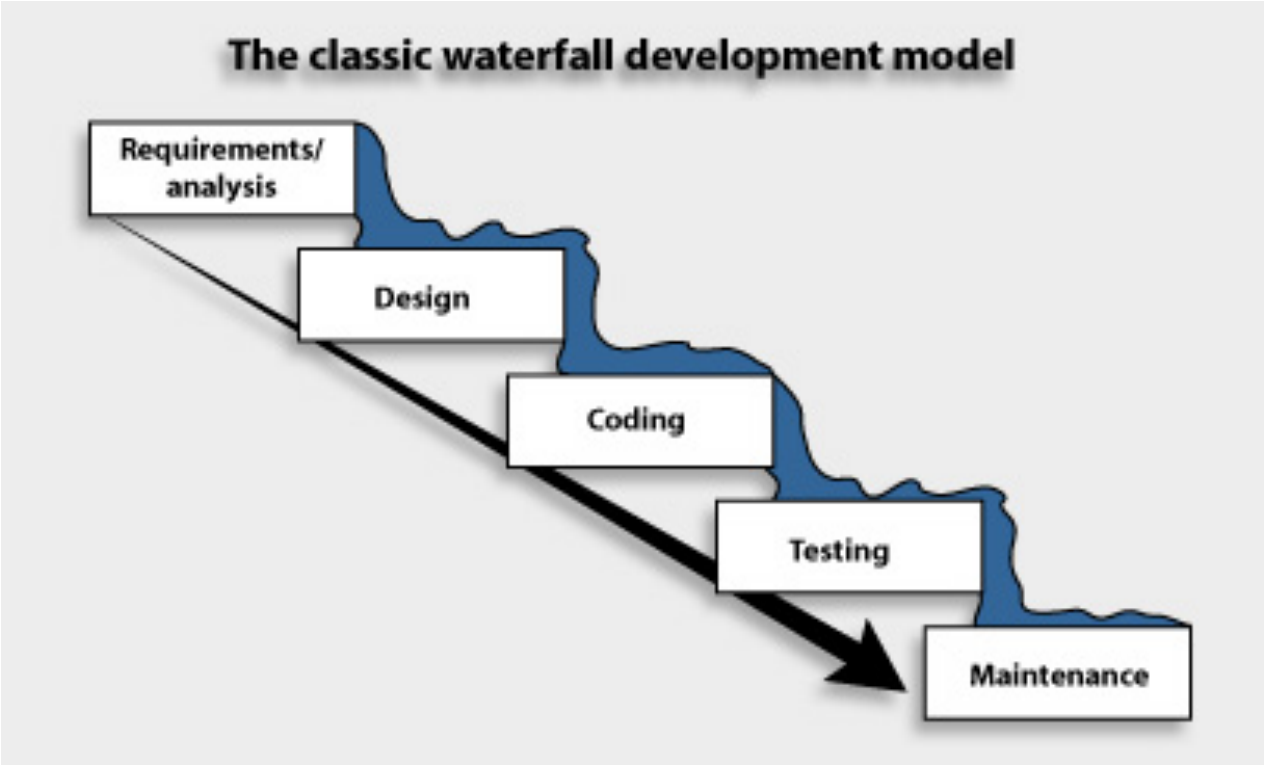
Software Process Model

- Def.: Abstract representation of a software dev. process
  - A description of a process from some particular perspective
  - Recommendations and guidelines for –
    - Which activities are to be carried out (phases, milestones)?
    - Who has to carry out what (roles, responsibilities)?
    - Which products are to be built until when (artifacts)?
    - Which techniques and tools are to be used?



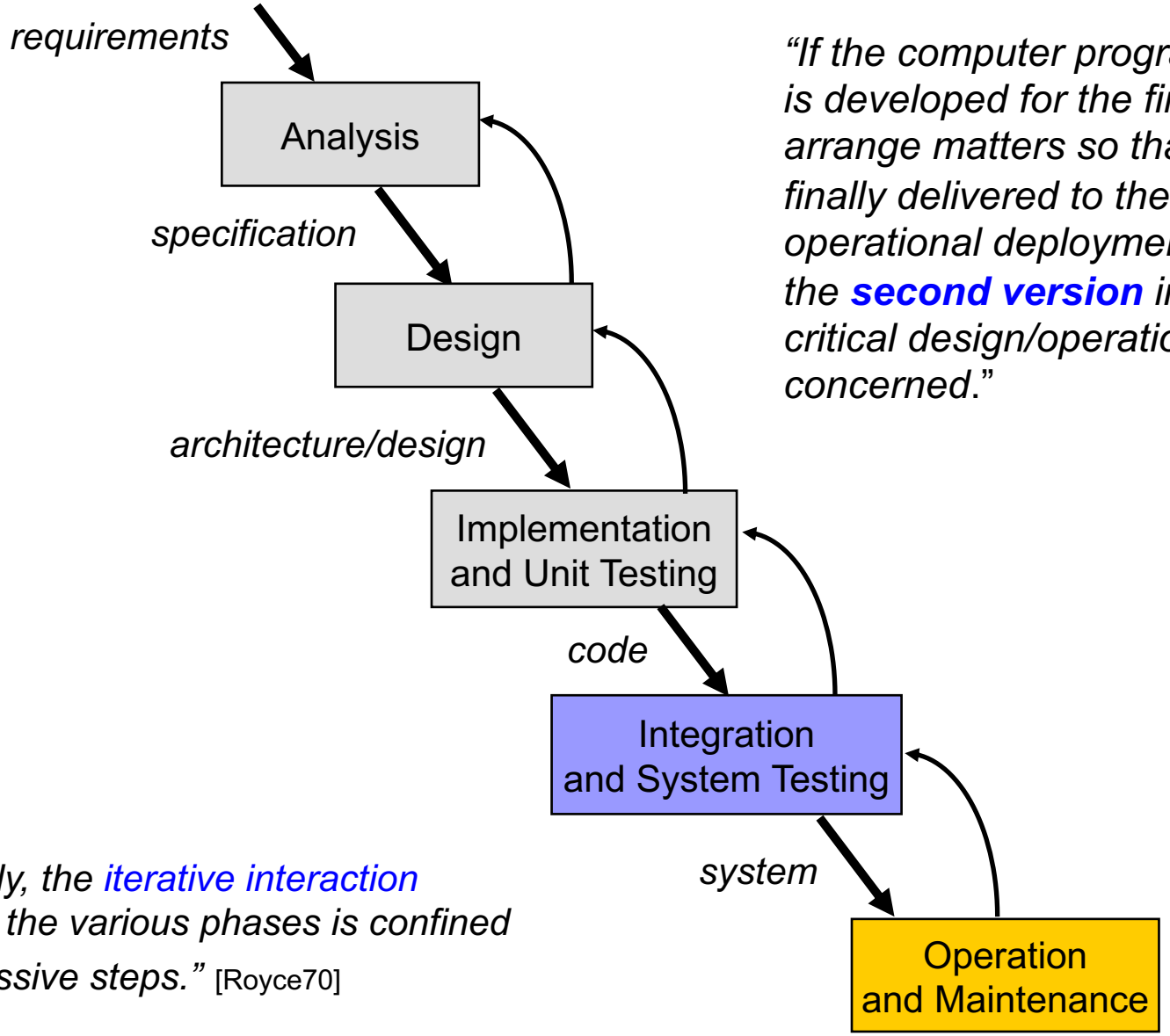
- Many supposed process models are rather pure life cycle models
  - i.e., merely describe the order of phases and transitions btw. them
  - e.g., Benington‘s phase model / the waterfall model, V model (XT)

Phase Model / Waterfall Model



- First described by **Herbert D. Benington** (as ***„stagewise model"***) at the Symposium on Advanced Programming Methods for Digital Computers, June, 29, 1956
- First formal definition by **Winston Royce**: Proceedings of IEEE WESCON: *Managing the Development of Large Software Systems*, August 1970

Phase Model / Waterfall Model (2)



*"If the computer program in question is developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the **second version** insofar as critical design/operation areas are concerned."*

[Royce70]

"Hopefully, the iterative interaction between the various phases is confined to successive steps." [Royce70]

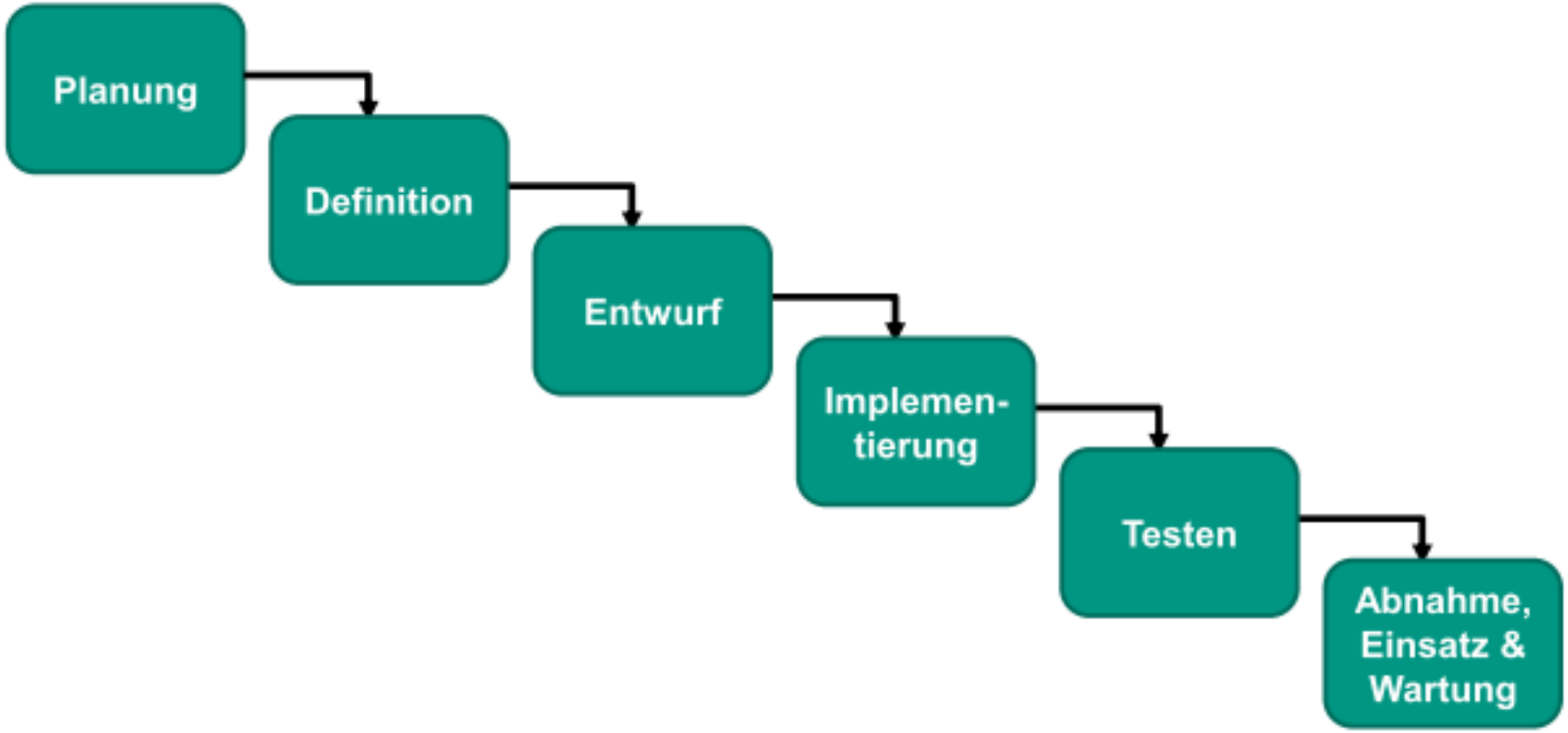
Waterfall Model Phases

- Analysis
  - Study, understand, and specify the system requirements (services, constraints, goals …)
- Design
  - Study, understand, and design the system architecture
  - Identify and describe the fundamental software system abstractions and their relationships
- Implementation and unit testing
  - Create individual code units and verify that each fulfills its responsibilities
- Integration and system testing
  - Integrate and test increasingly large modules, up to the level of the system as a whole
  - Deliver validated system to the customer
- Operation and maintenance
  - Install system and put it into practical use
  - Update software to correct errors, improve implementation, and satisfy new requirements

These phases/ activities can be found in basically all process models.

The interesting question is how to arrange them.

Das Wasserfall Modell



Waterfall Model Characteristics

- **Advantages**
  - A sequence of concrete activities with complete, well-defined products (inputs and outputs)
  - Simple process model for managers
- **Disadvantages**
  - Inflexible partitioning of the project into distinct stages
  - Insensitive to changing customer requirements
  - High-risk due to “big bang” integration
- **Applicability**
  - Only feasible when the requirements are well-understood
  - Requires lots of experience with development techniques and tools
  - Not well suited for complex and unprecedented projects

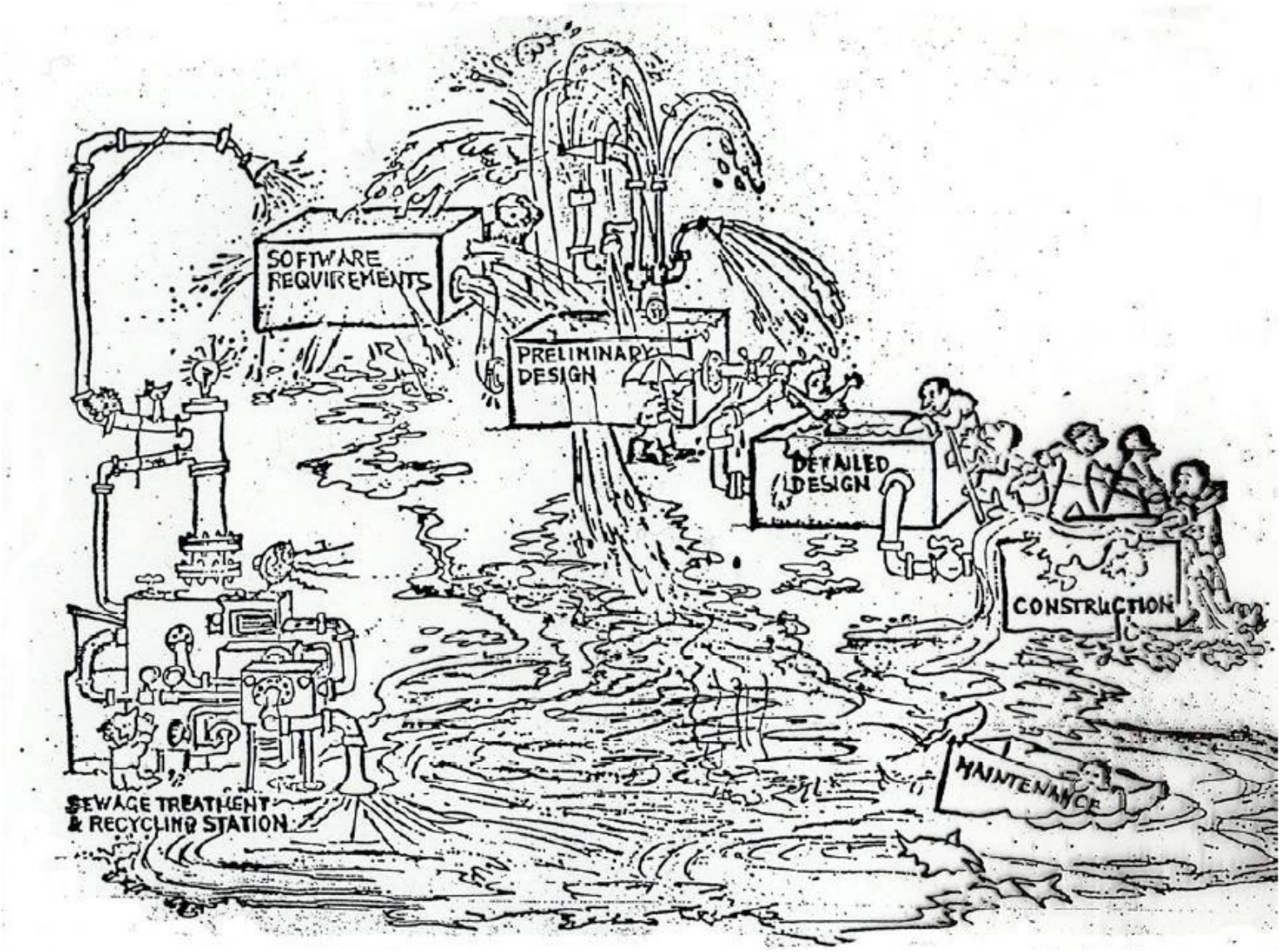
Waterfall Development Life Cycle

- Just a sequential sequence of activities

- How long can you look and plan ahead?
  - 5 days?
  - 5 weeks?
  - 5 months?
  - 5 years?

- On average: **35%** of the requirements of a software system change!
  - In recent years, the percentage is typically even higher!
  - And this without considering system evolution after deployment.

The Waterfall Model in Practice

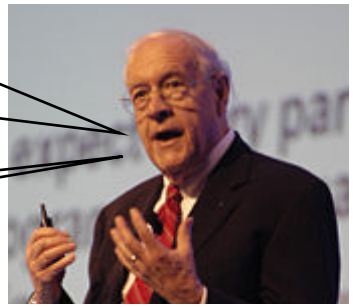


Waterfall Model

- Royce was rather an advocate of **iterative development** [Larman/Basili03]
  - He never expected his waterfall to be applicable for all kinds of systems (and without iterations)
- However, his simple model perceived lots of attention
  - It conveyed the illusion of an orderly, accountable and measurable process
  - Even became a quasi standard in the US until the late 1980s (DoD-Std-2167)

*"Much of present-day software acquisition procedure rests upon the assumption that one can specify a satisfactory system **in advance**, get bids for its construction, have it built, and install it. I think this assumption is **fundamentally wrong**, and that many software acquisition problems spring from that fallacy."* *[No Silver Bullet 1987]*

"The waterfall model is wrong!"
[ICSE keynote 1995]



Fred Brooks

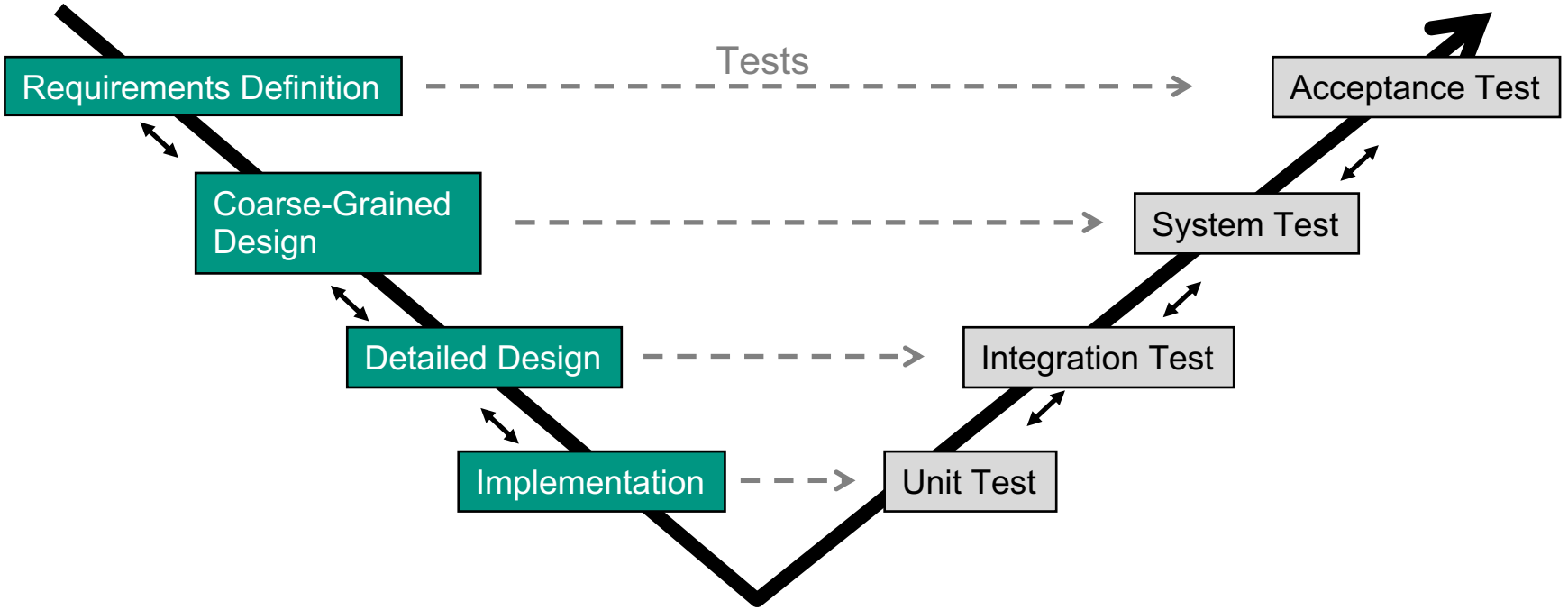
Roadmap

1. Introduction
2. The Waterfall Model
3. **V Model**
4. **Spiral Model**
5. Iterative and Incremental Development
6. (Rational) Unified Process
7. Software Prototyping
8. Agile Development

Appendix: Selection of process models

The V Model

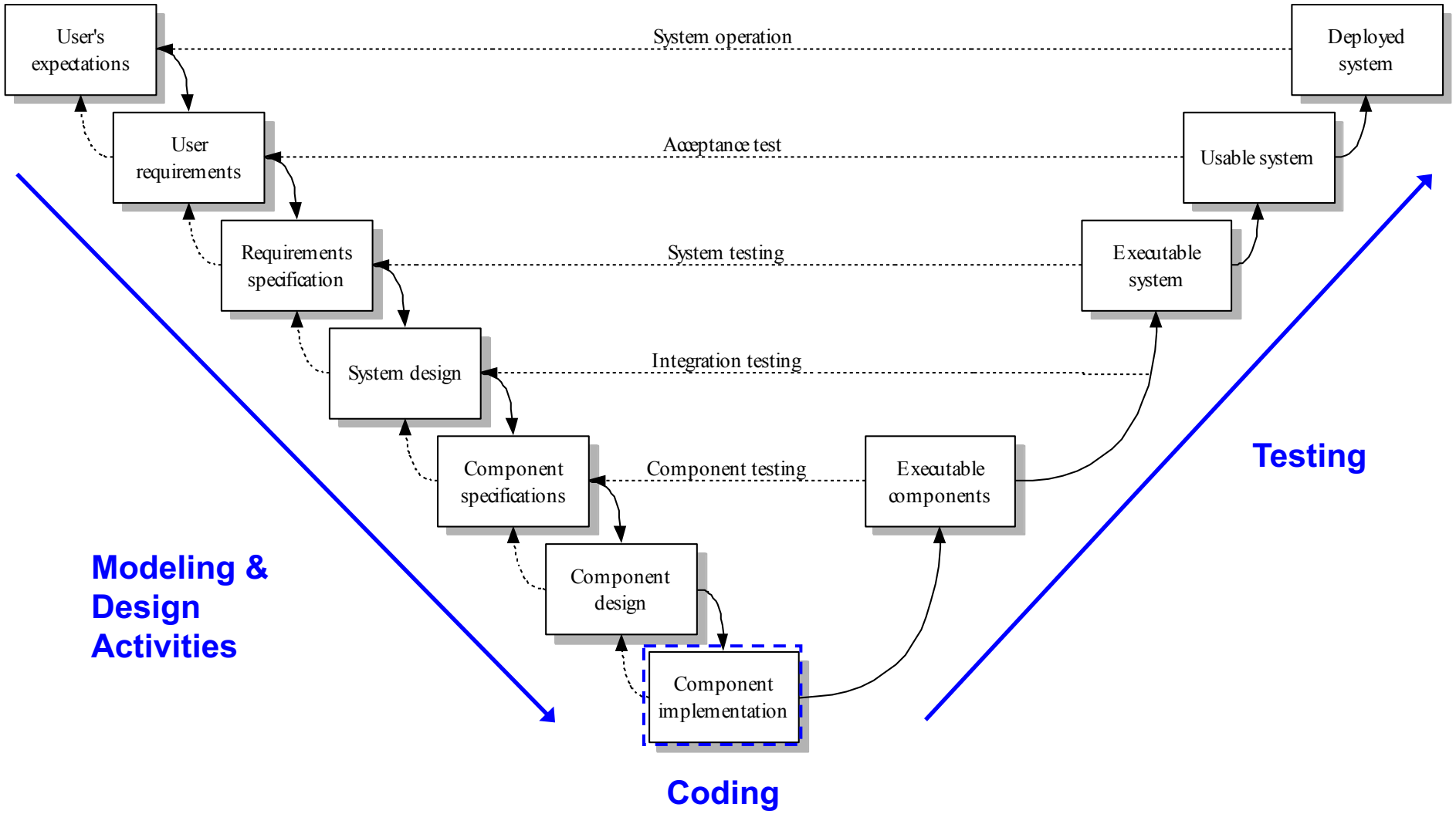
Adds an elaborated view on **verification** and **validation**



➔ Can be seen as (just) an explanation of how to relate testing to basic development phases

The V Model (2)

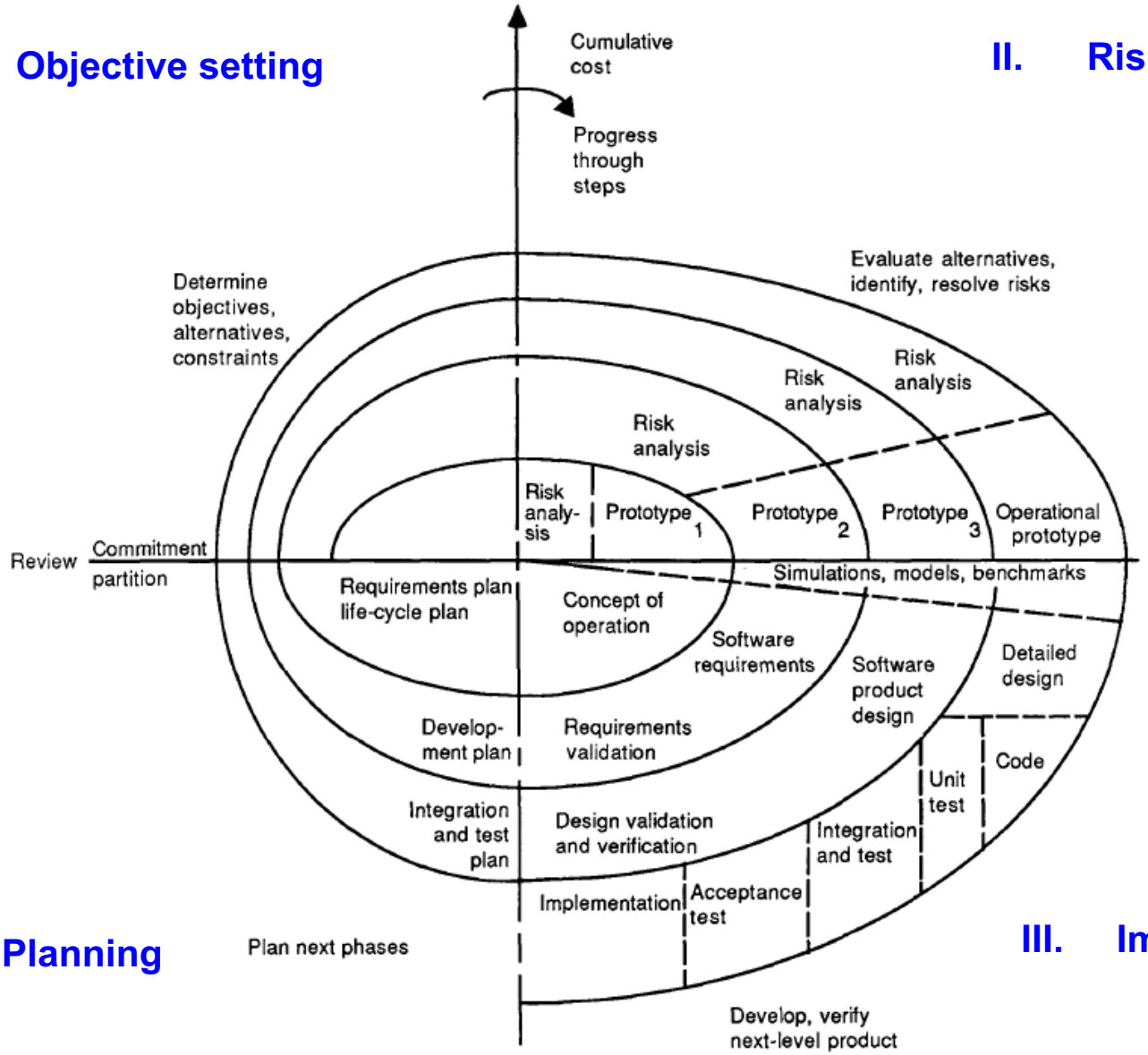
Adds an elaborated view on verification and validation



The Spiral Model

I. Objective setting

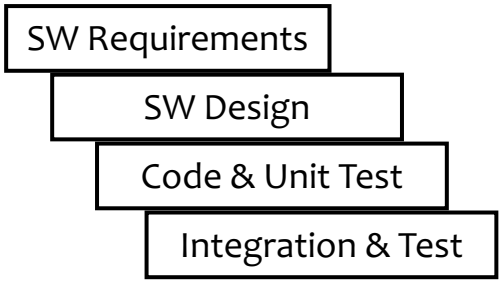
II. Risk assessment and reduction



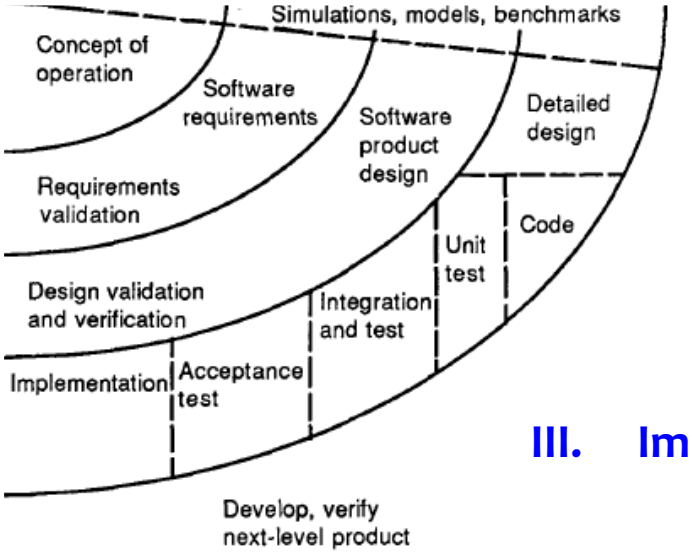
IV. Planning

III. Implementation and validation

The Spiral Model (2)



➔ the original spiral model incorporates a waterfall model



III. Implementation and validation

The Spiral Model

- Development process is represented as a spiral [Boehm88]
  - Rather than as a sequence of activities with backtracking
- Emphasis is on **risk analysis and mitigation**
  - **Boehm hypothesis**
    Project risks can be resolved or mitigated by addressing them early
  - Risks are explicitly assessed and resolved throughout the process
- No fixed phases such as specification or design
  - Loops in the spiral are chosen depending on what is required
  - A "meta process model" that defines risk analysis and mitigation cycles
    - The actual development activities need to come from a dedicated process model
    - In its original form, development activities are ordered sequentially

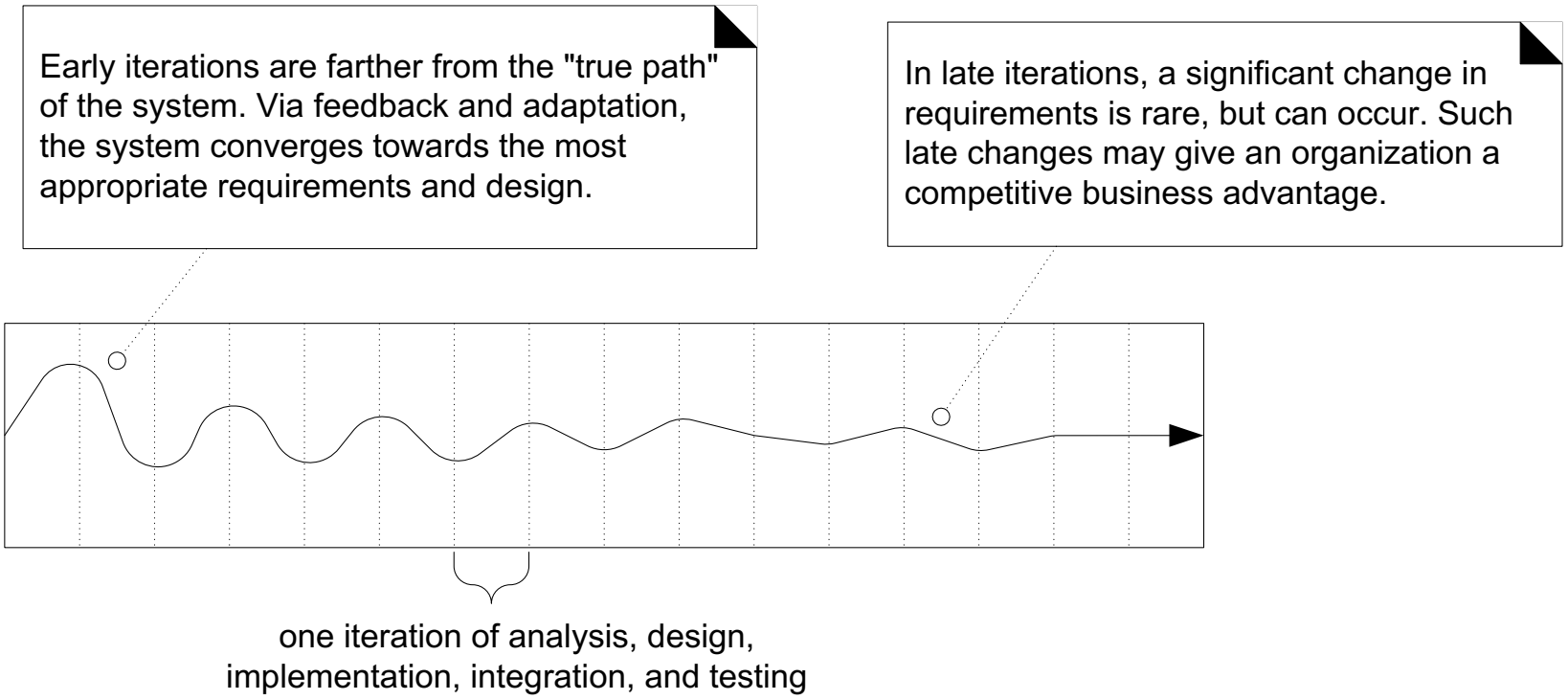
Roadmap

1. Introduction
2. The Waterfall Model
3. V Model
4. Spiral Model
5. **Iterative and Incremental Development**
6. (Rational) Unified Process
7. Software Prototyping
8. Agile Development

Appendix: Selection of process models

Iterative Development

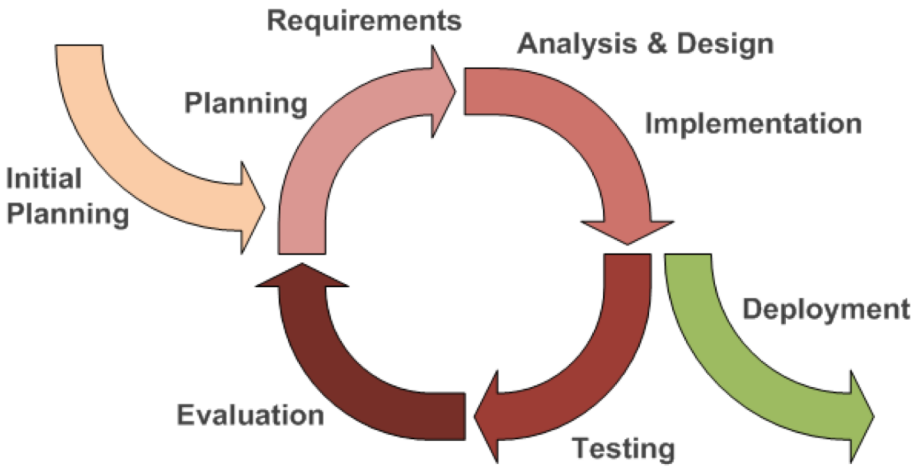
- A project is split into a series of short, fixed length mini-projects → **iterations**
- The outcome of each is a tested, integrated, executable *partial* system
  - i.e., each iteration includes its own analysis, design, implementation and testing



Iterative Development: Key Aspects

- **Typical assumptions**
  - Successive enlargement → incremental development
  - Successive refinement, cyclic feedback and adaptation → evolutionary and adaptive development

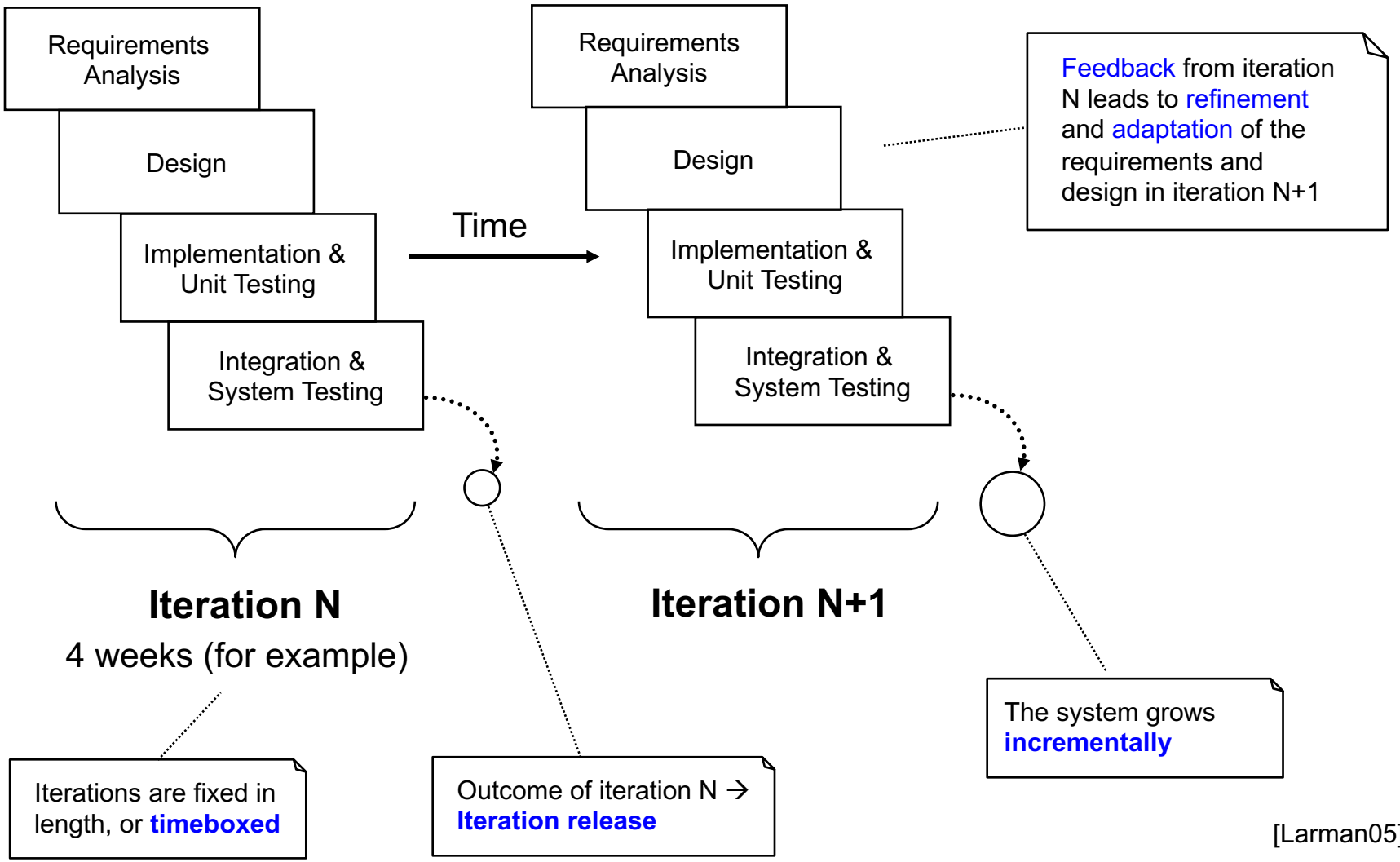
- **Terms often used interchangeably with Iterative Dev.**
  - Iterative & Incremental Development → **IID**
  - Evolutionary Development
  - Adaptive Development



An Increment

- Each **increment** contains a part of the required functionality
  - The highest priority requirements are included in early increments
- **Risk-driven** iterative development
  - Chooses the riskiest, most difficult features for the early iterations
- **Client-driven** iterative development
  - The choice of features for the next iteration comes from the client
  - Whatever they perceive as the highest business value to them
- Normally, a combination of both is recommended

Iterative, Incremental & Evolutionary Development



[Larman05]

Comparison of Life Cycles

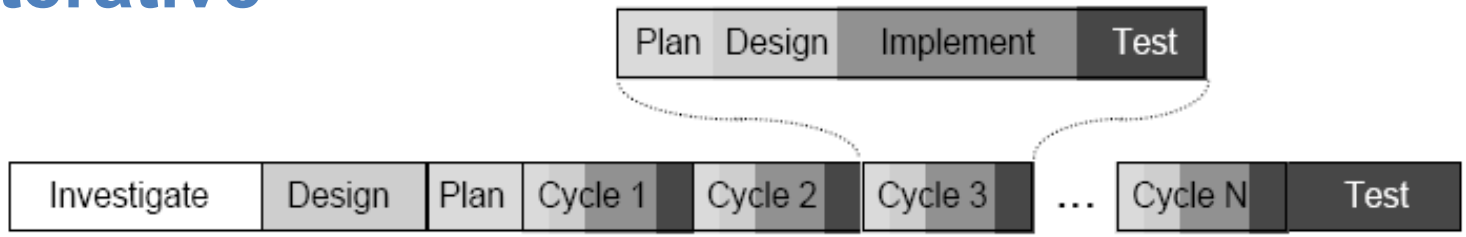
Sequential

e.g., 1 year

Waterfall Development Life Cycle

Iterative

usually 2-4 weeks



Incremental Development Life Cycle

Iterative Development Characteristics

- **Advantages**
  - Early rather than late attention for high risks
  - Early visible progress
  - Early feedback
  - Managed complexity
  - Experience from iterations can be used to improve the process itself
- **Disadvantages**
  - Additional overhead involved in planning and integration
  - Refactoring may be needed as additional requirements are addressed
  - Cost estimation is claimed to be harder

"You should use iterative development only on projects that you want to succeed"
-- Martin Fowler

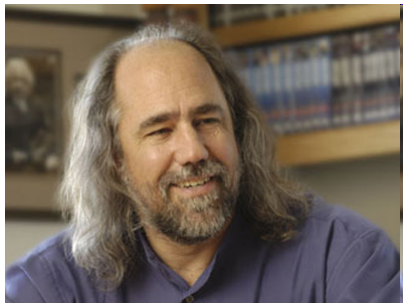
Roadmap

1. Introduction
2. The Waterfall Model
3. V Model
4. Spiral Model
5. Iterative and Incremental Development
6. **(Rational) Unified Process**
7. Software Prototyping
8. Agile Development

Appendix: Selection of process models

The Rational Unified Process (RUP)

- Developed by “Rational Software”
- Development started in the mid 1990s
- Mainly developed by the „Three Amigos“
  - Rumbaugh, Jacobson, and Booch



Grady Booch

- P. Kruchten lead the RUP development team
- As a byproduct the UML emerged
- Rational was strategically well positioned, having –
  - the UML, the RUP, and tools & services to support both

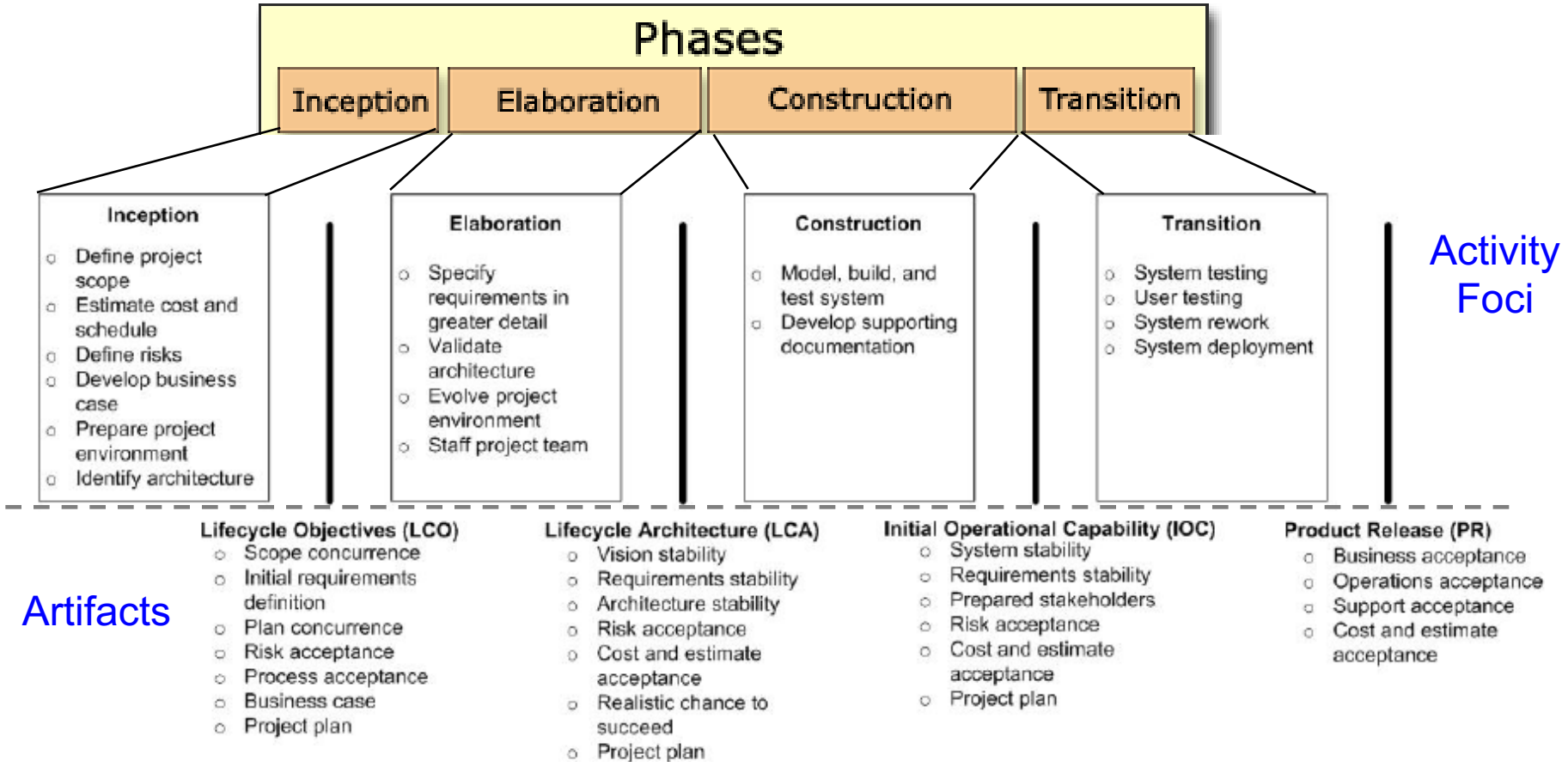
  ➔ acquired by IBM in 2003

The Rational Unified Process (RUP)

- A disciplined approach to assign **tasks** and **responsibilities** within an organization in order to develop specific products
- **Roles** → Who?
  - Defines a set of skills and responsibilities
- **Activities / Tasks** → How?
  - Describes work packages that need to be carried out by a role
  - Implemented in disciplines (a.k.a. workflows) in each iteration
- **Artifacts / Work Products** → What?
  - Results of a task, e.g., models and documents

Phases at a Glance

The RUP phases, their objectives and their concluding milestones [Ambler05]



RUP Supporting Disciplines

	Project Management	Environment	Configuration and Change management
Activities (57)	1. Conceive new project 2. Evaluate project scope and risk 3. Develop software development plan 4. Monitor and control project 5. Plan for next iteration 6. Manage iteration 7. Close out phase 8. Close out project	1. Prepare environment for project 2. Prepare environment for an iteration 3. Prepare guidelines for an iteration 4. Support environment during an iteration	1. Plan project configuration and change control 2. Create a project configuration management enviror 3. Change and deliver configuration items 4. Manage baselines and releases 5. Monitor and report configuration status 6. Manage change requests
Artifacts (117)	1. Test plan 2. Software architecture document 3. Iteration assessment 4. Business case 5. Software development plan 6. Iteration plan 7. Problem resolution plan 8. Risk management plan 9. Product acceptance plan 10. Measurement plan 11. Work order 12. Status assessment 13. Project measurements 14. Review record 15. Requirements Attributes 16.Vision 17. Risk list 18. Change requests	1. Development case 2. Development organization assessment 3. Project specific templates 4. Manual style guide 5. Use case modeling guidelines 6. Requirements management plan 7. Business modeling guidelines 8. User interface guidelines 9. Test guidelines 10. Design guidelines 11. Programming guidelines 12. Tools 13. Tool support assessment 14. Tool guidelines 15. Support environment	1. Project measurements 2. Deployment unit 3. Configuration audit fundings 4. Configuration management plan 5. Project repository 6. Change request 7. Workspace 8. Work order (integration) 9. Work order (completed) 10. Workspace (development)
Roles (38)	1. Project manager	1. Process engineer 2. Technical writer 3. System analyst 4. Business process analyst 5. User interface designer 6. Test designer 7. Architect 8. Tool specialist 9. System administrator	1. Configuration manager 2. System integrator 3. Change control manager 4. Project member

Every comprehensive process model should define these

Otherwise, we prefer to call it a lifecycle model (e.g., the waterfall model is one)

RUP Engineering Disciplines

	Business Modeling	Requirements	Analysis and Design	Implementation	Test	Deployment
Activities (57)	1. Assess business status 2. Describe current business 3. Identify business process 4. Refine business process definitions 5. Design business process realizations 6. Refine roles and responsibilities 7. Explore process automation 8. Develop a domain modeling	1. Analyse the problem 2. Understand stakeholder needs 3. Define the system 4. Manage the scope of the system 5. Refine the system definition 6. Manage changing requirements	1. Define a candidate architecture 2. Refine the architecture 3. Analyse behaviour 4. Design components 5. Design real time components 6. Design the database 7. Perform architectural synthesis	1. Structure the implementation model 2. Plan the integration 3. Implement components 4. Integrate each subsystem 5. Integrate the system	1. Plan test 2. Design test 3. Implement test 4. Execute tests in integration test stage 5. Execute tests in system test stage 6. Evaluate test	1. Plan deployment 2. Develop support material 3. Manage acceptance tests 4. Produce deployment unit 5. Package product 6. Provide access to download site 7. Beta test product
Artifacts (117)	1. Support specifications 2. Business glossary 3. Business rules 4. Business use case model 5. Business object model 6. Target organization assessment 7. Business vision 8. Business architecture document 9. Supplemantary business specification 10. Business use case 11. Business use case realization 12. Organization unit 13. Business entity 14. Business worker 15. Business modelling guidelines 16. Review record 17. Analysis model	1. Software architecture document 2. Requirements measurement plan 3. Stakeholder requests 4. Glossary 5. Vision 6. Use case model 7. Supplementary specifications 8. Use case 9. Software requirements specification 10. User interface prototype 11. Use case storyboard	1. Component 2. Reference architecture 3. Software architecture document 4. Use case realization 5. Analysysis model 6. Design model 7. Design subsystem 8. Design package 9. Design class 10. Interface 11. Capsule 12. Protocol 13. Data model 14. Deployment model 15. Integration build plan 16. Test component	1. Integration build plan 2. Component 3. Implement subsystem 4. Software architecture document 5. Integration build plan 6. Test component	1. Change requests 2. Test plan 3. Test model 4. Test case 5. Test procedure 6. Test script 7. Test class 8. Test packages 9. Test component 10. Test subsystem 11. Test results 12. Test evaluation summary 13. Workload analysis document	1. Installation component 2. End-user artifacts 3. Support material 4. Deployment plan 5. Release notes 6. Bill of materials 7. Training material 8. Test results 9. Change request 10. Development infrastructure 11. Development unit 12. Product
Roles (38)	1. Business process analyst 2. Business designer 3. Stakeholders 4. Business reviewer	1. System analyst 2. Use case specifyer 3. User interface designer	1. Architect 2. Designer 3. Database designer 4. Capsule designer	1. Architect system integrator 2. System integrator 3. Code reviewer 4. Implementer	1. Test designer 2. Designer 3. Implementer 4. Tester	1. Implementer 2. Technical writer 3. Deployment manager 4. Graphic artist 5. Course developer

- RUP is a heavyweight process: All in all over **5000 page** documentation!
- It is intended to be tailored for each specific project
  - Select only those elements that are necessary and helpful

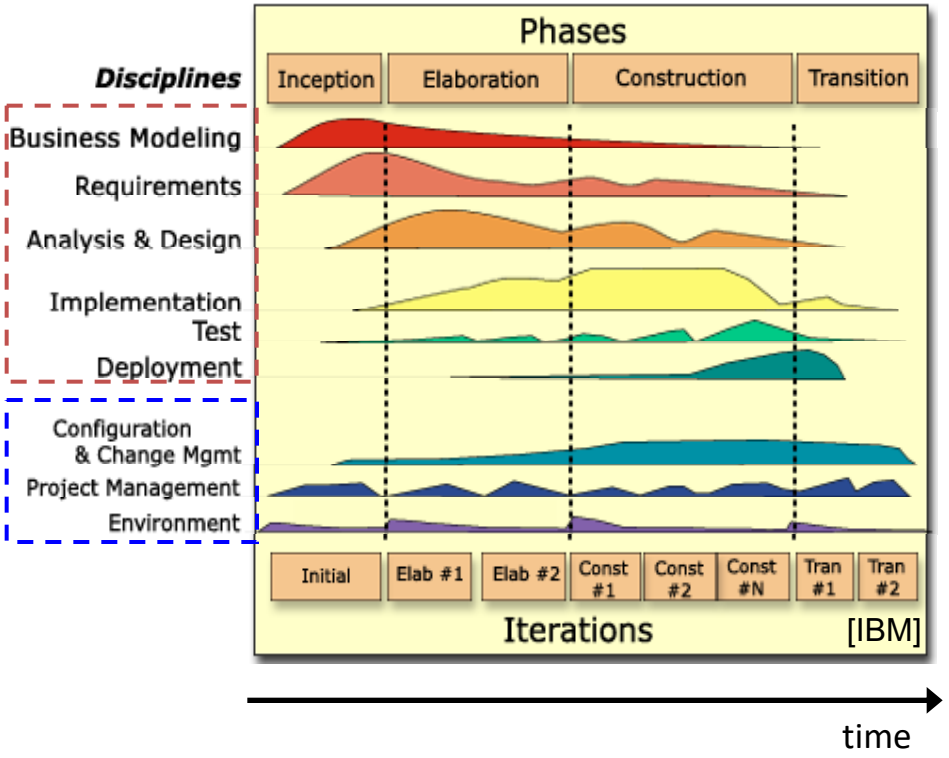
Roles in a Project ☺

RUP
- 57 activities
- 117 artifacts
- 38 roles

Best Practice Iterative Process

The **Unified Process (UP)** defines:

- **4 abstract phases**
  - to be concluded with a milestone
  - not equivalent to waterfall phases!
- **9 disciplines**
  - 6 engineering + 3 supporting
  - here we find our “mini waterfalls”
- **6 best practices**
  - develop software iteratively
  - manage requirements
  - use component-based architectures
  - model software visually
  - verify software quality
  - control changes to software



UP vs. RUP

- **UP can be seen as**
  - abstract/generic model that **captures the essential elements** common to most realizations
  - generic name for a family of models that meet certain criteria

- **RUP can be seen as**
  - **refinement of UP**, i.e., adds details including guidelines, templates, and tool assistance
  - trademark of IBM → both the process model as well as related tools

- Roughly speaking, UP can also be considered as a „**subset** of RUP“
  - The essential parts adopted by the community and established as best practices

UP Phases

1. Inception

- approximate vision, business case
- scope, vague estimates (e.g., time, cost, risks)

2. Elaboration

- refined vision, iterative implementation of core architecture
- resolution of high risks, identification of most requirements and scope
- more realistic estimates

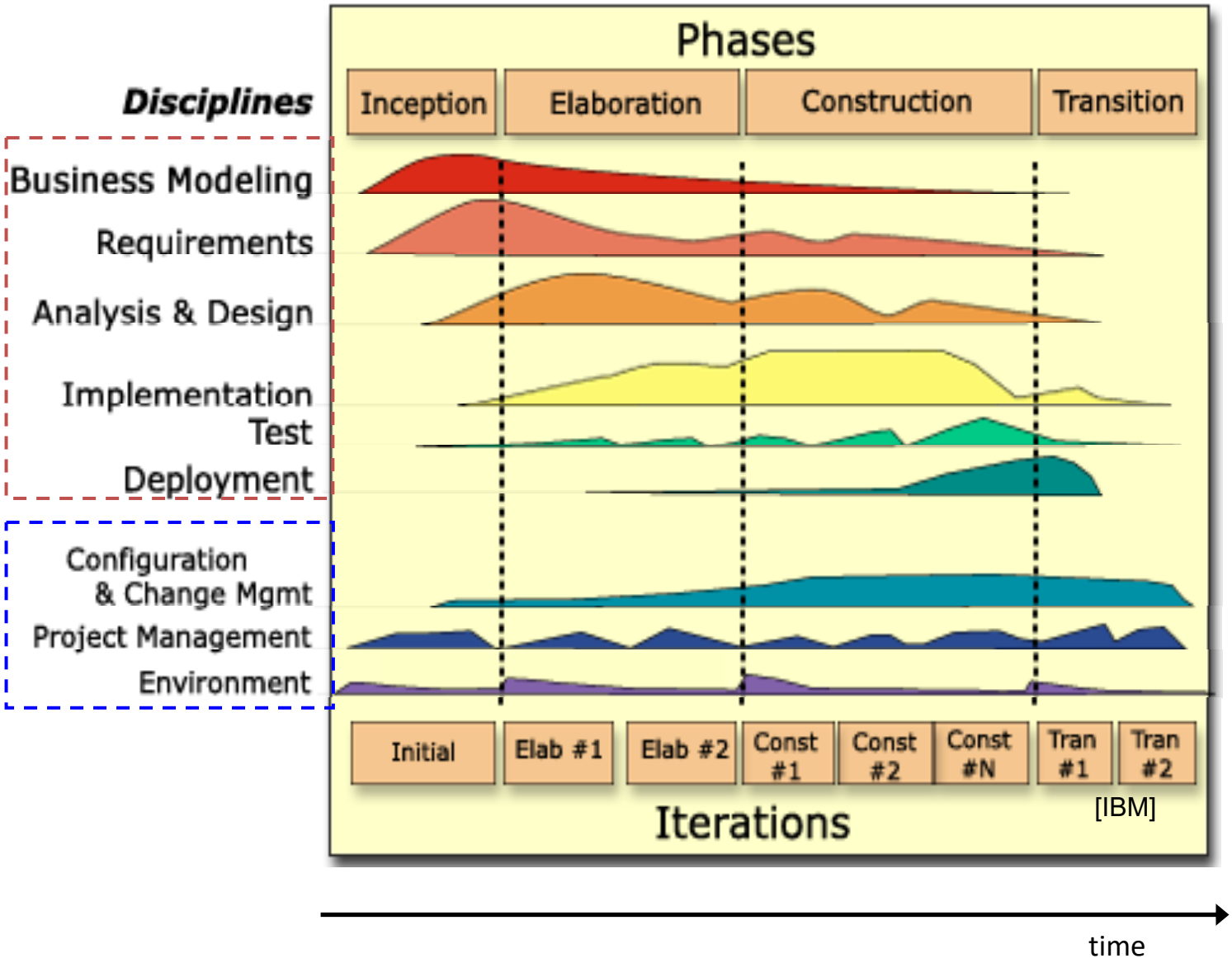
3. Construction

- iterative implementation of the remaining lower risk & easier elements
- preparation for deployment

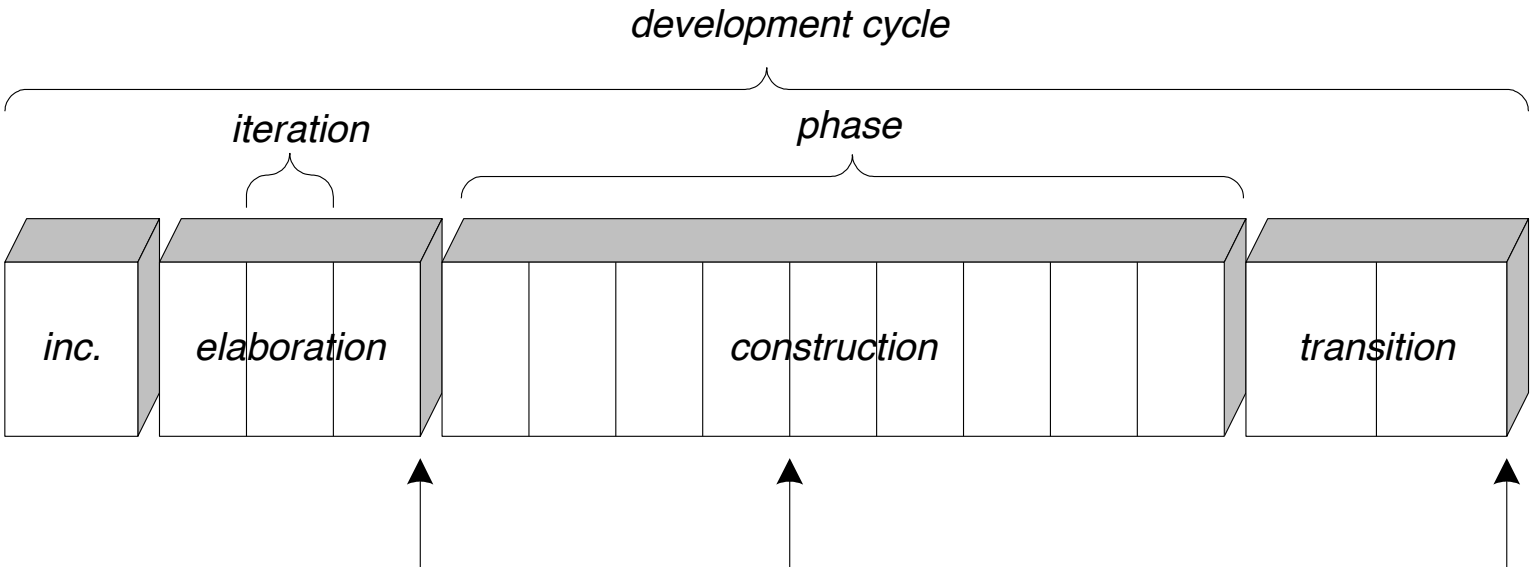
4. Transition

- beta tests, deployment

UP Overview


Phases
Disciplines
Inception
Elaboration
Construction
Transition
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration
& Change Mgmt
Project Management
Environment
Initial
Elab #1
Elab #2
Const #1
Const #2
Const #N
Tran #1
Tran #2
Iterations
[IBM]
time

UP Phases (2)



milestone

An iteration end-point when some significant decision or evaluation occurs.

release

A stable executable subset of the final product. The end of each iteration is a minor release.

increment

The difference (delta) between the releases of 2 subsequent iterations.

final production release

At this point, the system is released for production use.

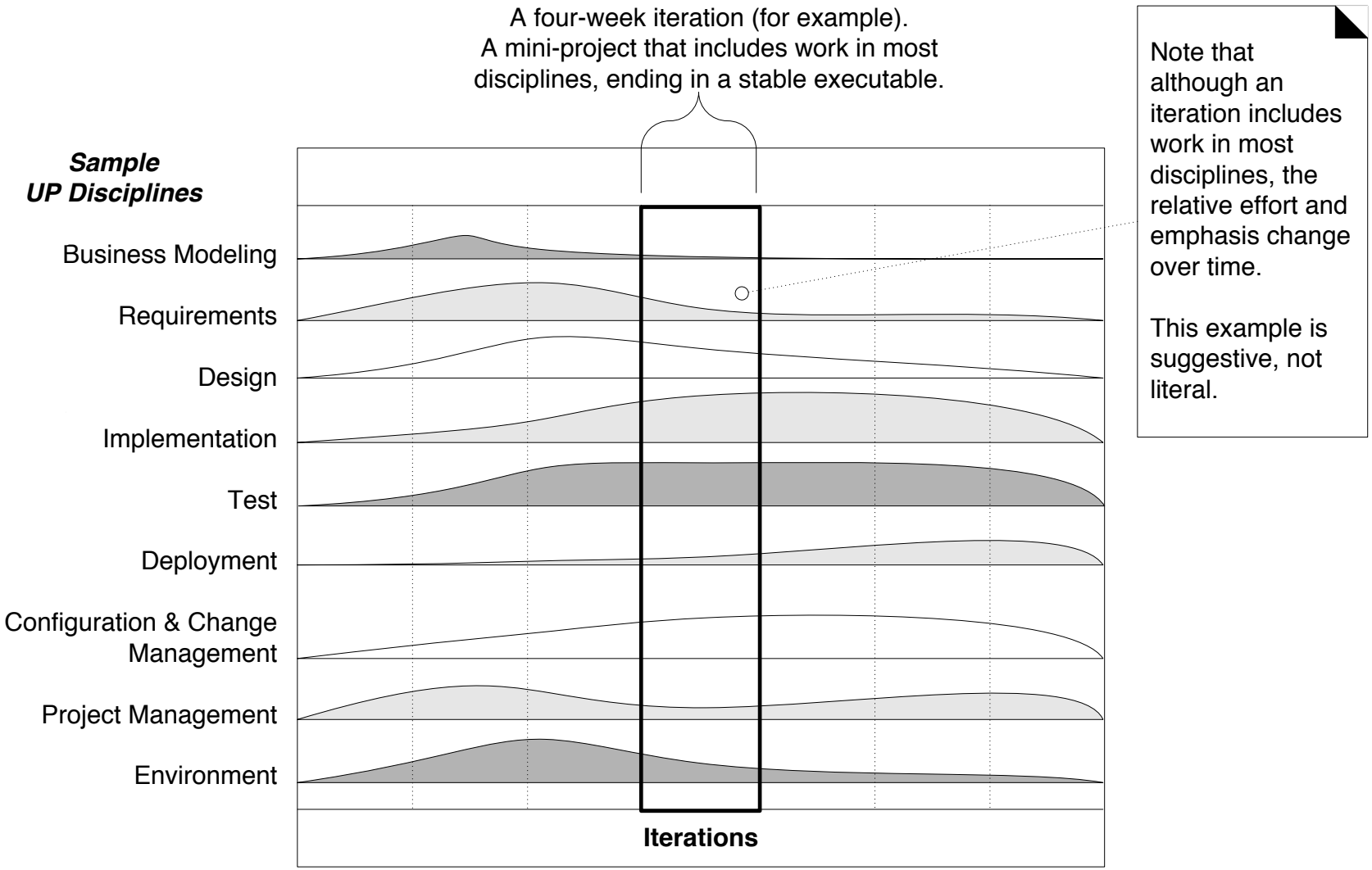
UP Disciplines

- Def: UP Discipline → A set of activities (and related artifacts) in one subject area, e.g., requirements analysis
- Def: UP Artifact → Any work product: code, Web graphics, database schema, text documents, diagrams, models, etc.
- All activities and artifacts are *optional*
- **Def: Development Case** → A short document summarizing the choice of practices and artifacts for a given project, for example:

Discipline	Practice	Artifact	Incep.	Elab.	Const.	Trans.
		Iteration →	I1	E1..En	C1..Cn	T1..T2
Business Modeling	agile modeling req. workshop	Domain Model		s		
Requirements	req. workshop vision box exercise dot voting	Use-Case Model	s	r		
		Vision	s	r		
		Supplementary Specification	s	r		
		Glossary	s	r		
Design	agile modeling test-driven dev.	Design Model		s	r	
		SW Architecture Document		s		
		Data Model		s	r	

s = start
r = refine

UP Disciplines (2)



Iteration Length and Timeboxing

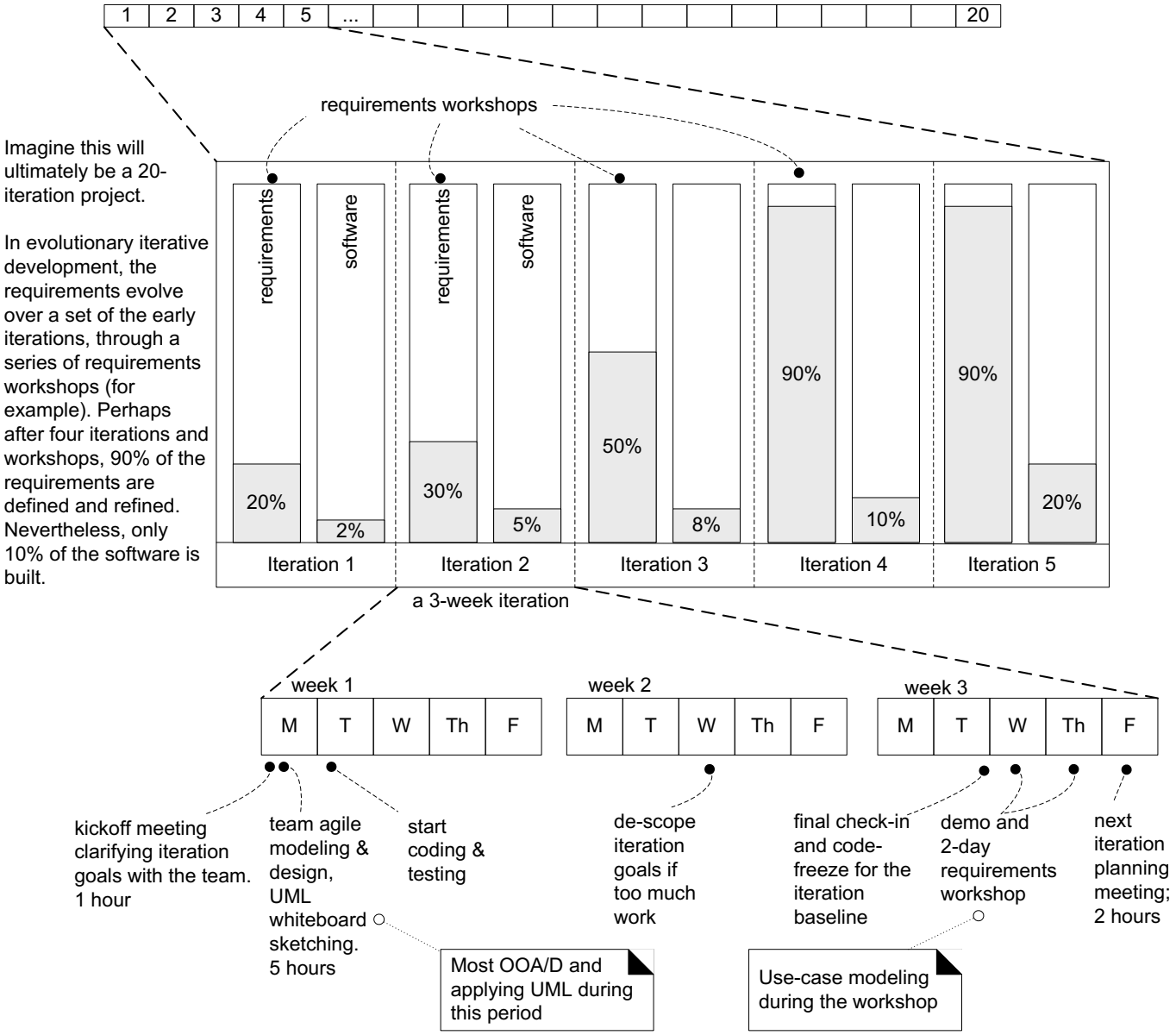
- Short iterations allow for rapid feedback and adaptation
- Long iterations increase project risk
- Iterations are normally fixed in length (**time-boxed**)

- The UP recommends:
  - Use **short time-boxed** iterations (btw. 2 and 4 weeks)
  - If meeting deadline seems to be difficult
    - remove tasks or requirements from the iteration
    - include them in a future iteration

Inception is not the Requirements Phase!

- Trying to discover all requirements during Inception is waterfall mentality!
- Most requirements analysis normally occurs during the elaboration phase
- Inception is typically only one week (or a few weeks) long
- Major Inception questions
  - Why should we do this? (business case)
  - Is it feasible? Should we buy or build?
  - What are the rough expected costs? (order of magnitude estimation)
  - Should we proceed or stop?
- Major Inception goals
  - Analyze important use cases (~10%) and critical non-functional requirements
  - Justify the project
- Main question answered
  - Do the stakeholders have basic agreement on the vision of the project, and is it worth investing in serious investigation (→ elaboration)?

UP Example in Pictures



UP Iteration Instantiation Example

- How much of which activity should be performed at what time?
  - A two day (timeboxed) requirements workshop is held before iteration 1
    - Start with high-level requirements collection (half a day)
    - Just use case or feature names plus key non-functional aspects
    - Ask chief architect and important business people to pick about the 10% most important use cases; in terms of architecture significance, business value and risk
    - Refine the selected use cases in the remaining time
  - Select the most important aspects of these use cases to be implemented in a planning meeting at the beginning of iteration 1
  - Do iteration 1 (time-boxed to e.g. 3 weeks)
    - The goal is to learn and to establish collaboration guidelines in the team
    - Do modeling, programming and testing, and a demo of the 1st increment
    - Do another requirements workshop (review old and refine new use cases)
    - And plan the next iteration

Roadmap

1. Introduction
2. The Waterfall Model
3. V Model
4. Spiral Model
5. Iterative and Incremental Development
6. (Rational) Unified Process
7. **Software Prototyping**
8. Agile Development

Appendix: Selection of process models

Software Prototyping

- A prototype is an initial version of a system (or part of it) used to demonstrate concepts and try out design options
  - Typically used in sequential process models
- A prototype can e.g. be used in:
  - The requirements engineering process to help with requirements elicitation and validation
  - Design processes to explore options and develop a User Interface (UI) design
  - The testing process to run back-to-back tests
- **Boehm's second law**
  Prototyping (significantly) reduces requirement and design errors, especially for user interfaces

Prototypes

- Prototypes can be realized in different ways, e.g., as –
  - UI sketches on paper or in Powerpoint (“slideware”)
  - Executable prototypes
    - Adaptation of a legacy system
    - Simplified development from scratch
    - Mock-ups
    - Executable requirements notation (e.g., SDL)
  - Trade-off between effort and expected benefit

- Potential benefits of applying prototyping
  - Improved system usability
  - A closer match to users’ real needs
  - Improved design quality and maintainability
  - Eventually reduced development effort

Prototyping vs. Iterative & Incremental Dev.

- Prototyping can sometimes conflict with IID
  - Since prototypes may be “abused” as system increments
  - IID aims to deliver a working system to end-users
    - Normally starts with those requirements that are well understood
  - Prototyping aims to validate or derive the system requirements
    - Starts with those requirements that are poorly understood
- Prototypes should be discarded after development
  - The prototype’s structure is usually degraded through rapid change
  - Will probably not meet normal organizational quality standards
  - Normally undocumented and meant to be thrown away!

Frederick P. Brooks in „The Mythical Man-Month“ (Ch. 11, p. 116): „Plan to throw one away; you will, anyhow.“

Roadmap

1. Introduction
2. The Waterfall Model
3. V Model
4. Spiral Model
5. Iterative and Incremental Development
6. (Rational) Unified Process
7. Software Prototyping
8. **Agile Development**

Appendix: Selection of process models

Agile Development

- A **lightweight approach** to software development
  - Goal → early and continuous delivery of useful software components
- A philosophy that encompasses many concrete methods
  - **SCRUM** [Pichler07], **Extreme Programming** (XP) [Beck04], **Crystal** [Cockburn04], **Development Method** (DSDM), **Feature-Driven Development** (FDD), **Pragmatic Programming**,...
  - Captured in Agile Software Development Manifesto [http://agilemanifesto.org]
- Four main value statements
  - **Individuals and interactions** over processes and tools
  - **Working software** over comprehensive documentation
  - **Customer collaboration** over contract negotiation
  - **Responding to change** over following the plan

Agile Development Principles

- Highest priority to satisfy the customer through early and continuous delivery
  - Deliver working software frequently
- Welcome changing requirements, even late in development
  - Harness change for customer's competitive advantage
- Business people and developers must work together daily throughout the project
  - Most effective communication method: face-to-face conversation
- Build projects around motivated individuals
  - give them the environment and support they need
  - then trust them to get the job done

Agile Development Principles (2)

- Working software is the primary measure of progress
- Agile processes promote sustainable development
  - the sponsors, developers, and users should be able to maintain a constant pace indefinitely
- Continuous attention to technical excellence and good design enhances agility
- Simplicity — the art of maximizing the amount of work not done — is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective

Agile Development Characteristics

- **Advantages**
  - Simple management model
  - Early feedback is guaranteed
  - Rapid response to change is feasible
- **Disadvantages**
  - Require frequent refactoring cycles
  - Sometimes used as an excuse when there exists no process at all
  - Not always feasible for large distributed projects
  - May lead to systems with a suboptimal maintainability
- **Applicability**
  - Small or medium-size systems
  - As long as the development team can work closely with the customer

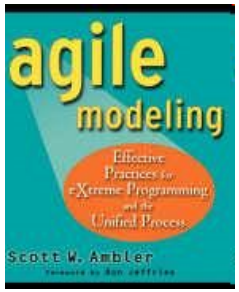
Three Ways to Apply UML

- UML as a sketch
  - Informal and incomplete diagrams (often hand sketched on whiteboards)
- UML as a blueprint
  - Relatively detailed design diagrams used either for (1) **reverse engineering** to visualize and better understand existing code or (2) **forward engineering** → code generation
- UML as a programming language
  - Complete executable specification of a software system in UML. Executable code is automatically generated.

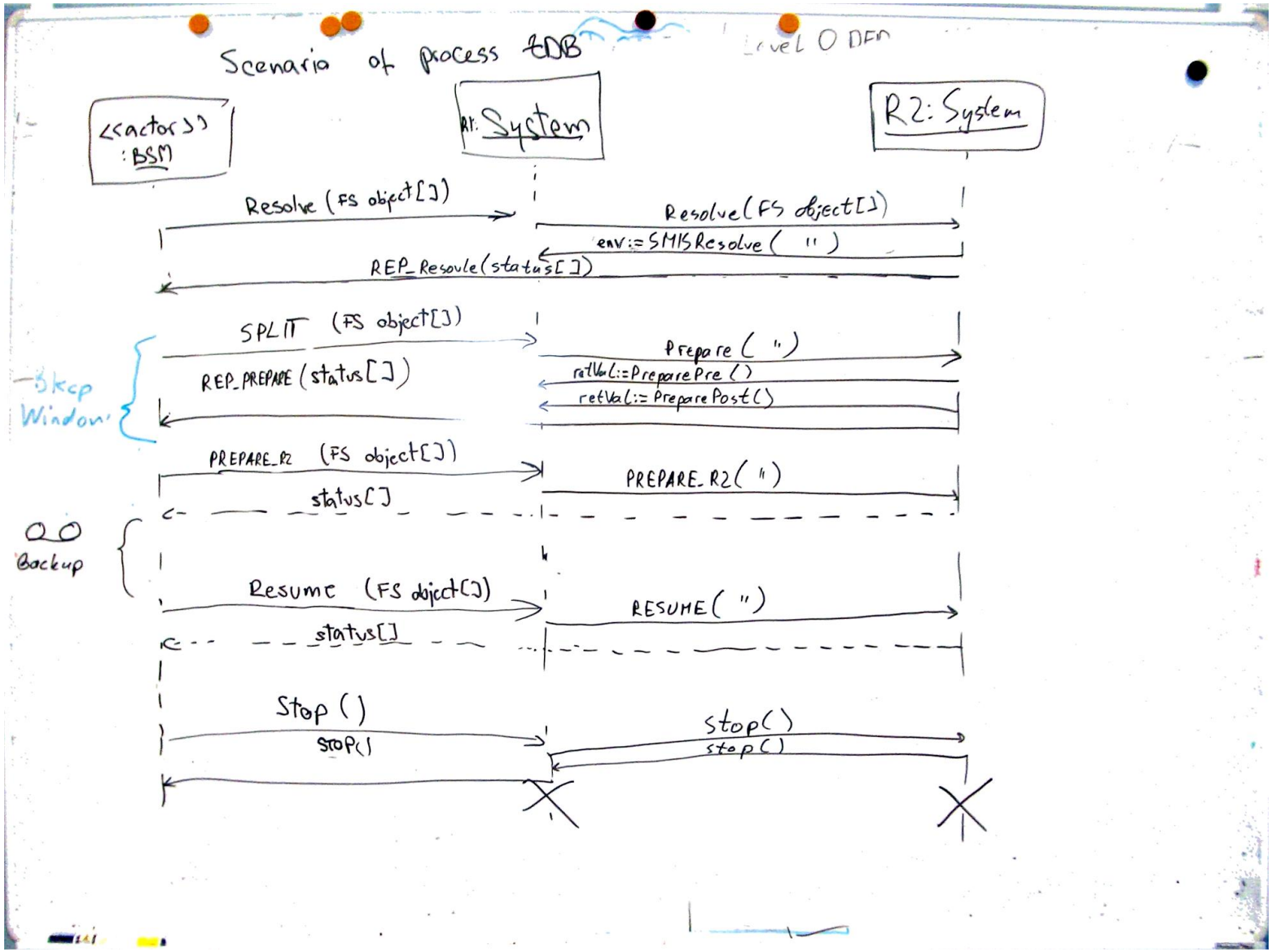
Agile UP & Agile Modeling

- Agile Unified Process (UP)
  - Combines best practices from UP and the agile world
  - The way UP was intended to be applied by its creators

- Main principles
  - Adapt UP to the specific project
  - Select UP activities and artifacts that add value
  - Apply UP with **Agile Modeling** practices
    - Term coined by Scott Ambler
    - Emphasizes using UML as a sketch



UML as a Sketch



Agile Modeling

“In preparing for battle I have always found that plans are useless, but planning indispensable”

-- General Dwight David Eisenhower

34th President of the United States

Agile Modeling Practices

- Adopting an agile method does not mean avoiding any modeling
- The purpose of modeling is primarily to support understanding and communication, not documentation!
- Don't model or apply the UML to all or most of the software design. Focus on unusual, difficult parts.
- Use the simplest tool possible, e.g., sketching on whiteboards
- Don't model alone, model in pairs, or triads…
- Create models in parallel, e.g., static view and dynamic view
- Use "good enough" simple notation
- Know that all models will be inaccurate
- Developers themselves should do the OO design modeling

You Know …

- … you did not understand iterative development when
  - you try to fully define –
    - most of the requirements before starting design and implementation
    - most of the design or architecture before starting to program
  - you spend weeks in UML modeling before programming
  - you think –
    - inception = requirements
    - elaboration = architecture and design
    - construction = implementation
  - you believe a suitable iteration length is 3 month rather than 3 weeks
  - you try to plan a project in detail from start to finish
    - i.e., you speculatively try to predict & plan all iterations
  - you defer testing until near the end of the project

Self Assessment

1) *The six basic activities in every software development project are*

➔ _____, _____, _____

_____, _____, _____

2) *A comprehensive software development process should define –*

➔ _____, _____, _____

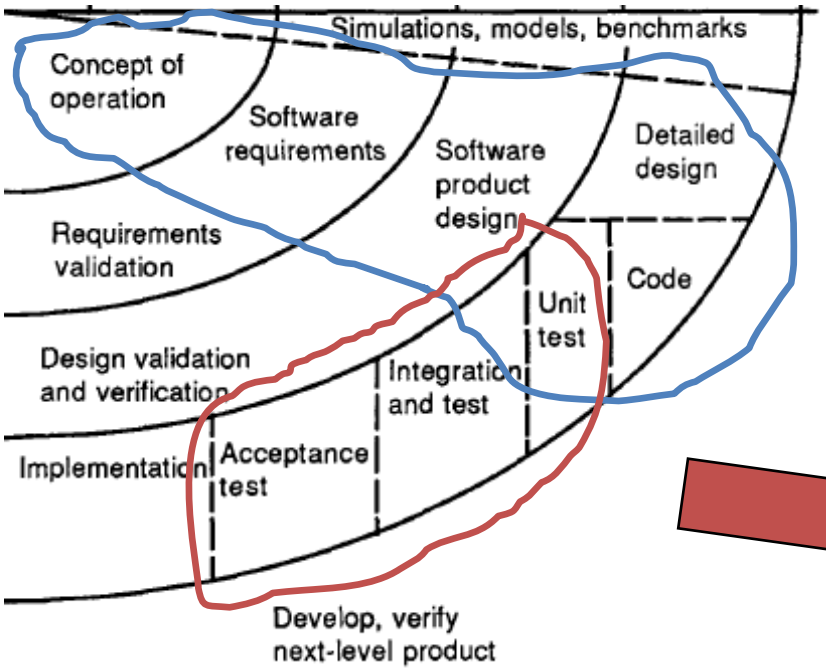
3) *Name the central problems of the Waterfall Model*

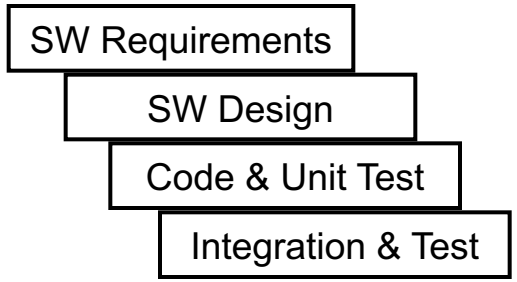
➔ _____

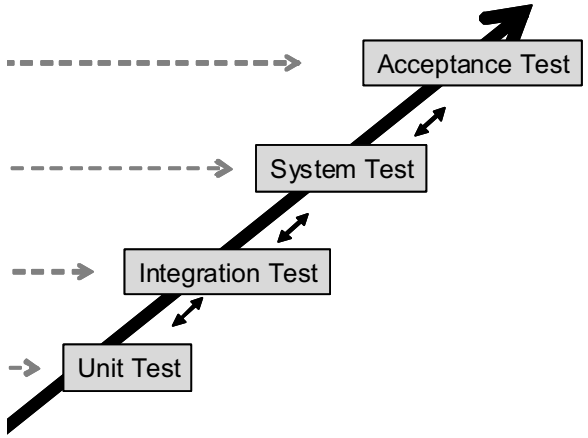
_____

Is the spiral model genuinely iterative?

- *i.e., does it go through the waterfall multiple times?*







III. Implementation and validation

➔ the original spiral model basically incorporates a waterfall and a V model, **but is not a genuine IID model**

Summary

- The **waterfall model** was predominant up until the 1990s
  - Makes sense only if the requirements are well understood and people are experienced in technology and domain
- **Iterative and incremental development** is generally assumed to be the better approach
- The Unified Process combined with agile development practices provides a good balance between systematic planning and process flexibility
- But remember: there is no process model for all occasions!
  - Models must be tailored to the specific challenges of a project

Book References

- [Balzert00] H. Balzert
  Lehrbuch der Softwaretechnik (3. Auflage)
  Spektrum akademischer Verlag, 2009, 2011.

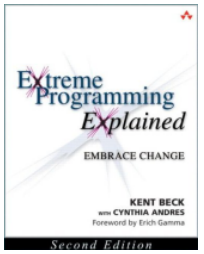
- [Beck] K. Beck
  Extreme Programming Explained: Embrache Change (2nd edition)
  Addision-Wesley, 2004.

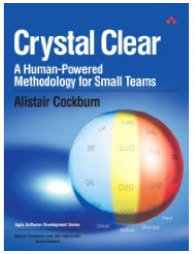
- [Bunse/v. Knethen08] C. Bunse & A. v. Knethen
  Vorgehensmodelle kompakt (2. Auflage)
  spektrum akademischer Verlag, 2008.

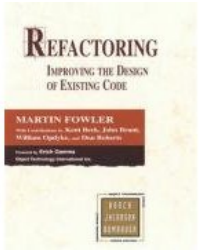
- [Cockburn] A. Cockburn
  Crystal Clear: A Human-Powered Methodology for Small Teams
  Addison-Wesley, 2004.

- [Fowler18] M. Fowler
  Refactoring: Improving the Design of Existing Code
  2nd edition, Addison-Wesley, 2018.

- [Pichler07] R. Pichler
  Scrum – Agiles Projektmanagement erfolgreich einsetzen
  dpunkt, 2007.

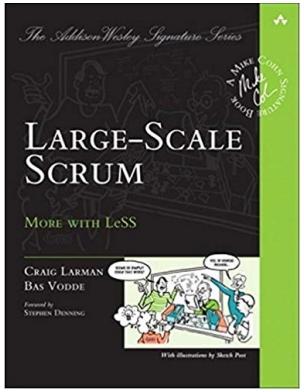






Book References

- „Large-Scale Scrum: More with Less“ by Craig Larman und Bas Vodde, Addison-Wesley, Aug. 2016



Article References

- [Benington56] H. D. Benington.
  Production of Large Computer Programs
  Proceedings of the ONR Symposium on Advanced Programming Methods for Digital Computers 1956.

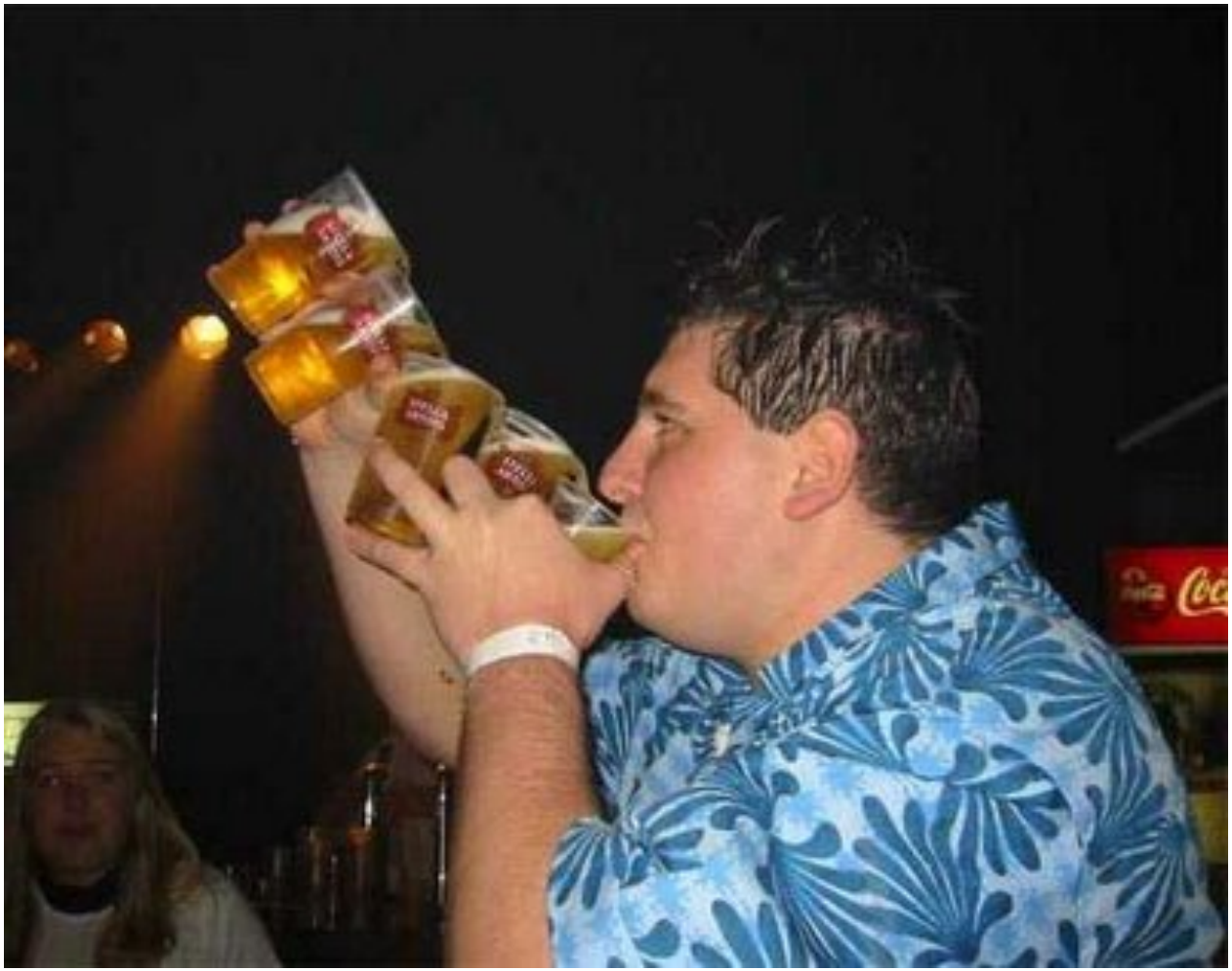
- [Boehm88] B.W. Boehm
  A spiral model of software development and enhancement
  IEEE Computer, May 1988.

- [Denert/Hesse80] E. Denert & W. Hesse
  Projektmodell und Projektbibliothek: Grundlagen zuverlässiger Software-Entwicklung und Dokumentation
  Informatik-Spektrum 4/1980.

- [Larman/Basili03] C. Larman & V. Basili
  Iterative and Incremental Development
  http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01204375

- [Royce70] Winston W. Royce
  Managing the Development of Large Software Systems
  Proceedings of the IEEE WESCON Conference, 1970.

Without Comment



Appendix

Terminology for Process Models I

- Sequential
  - Development activities are allocated to phases which are carried out sequentially
  - Transition to a subsequent phase requires delivery of specific assets
  - Returning to earlier phases possible, but considered exception rather than the rule
- Repetitive
  - (Some) activities in the process are carried out repeatedly
- Iterative
  - A project is split into a series of short, fixed length mini-projects → iterations
- Incremental
  - The system and other artifacts grow in “discrete” increments

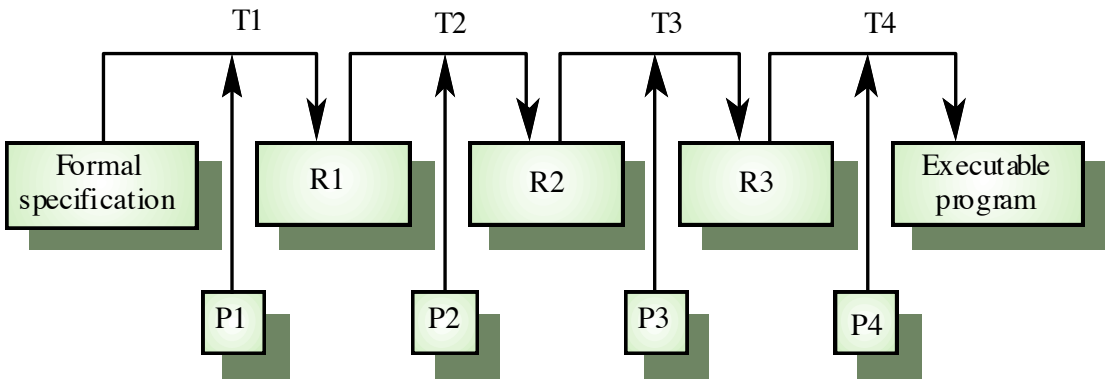
Terminology for Process Models II

- Recursive
  - Recursive models hierarchically decompose a system into sub-systems of sub-systems, etc.
- Evolutionary
  - Repetitive approaches that blur the boundaries between the phases in the waterfall loop
  - They often include a risk-driven feature selection and close customer involvement
- Agile
  - Is iterative and incremental with close customer involvement
  - Additionally, often advocates test-driven development and other agile techniques
- Prototypical
  - At certain points in the process prototypes can be developed
  - Feedback from these prototypes is used in the development process

Terminology for Process Models III

- Model-Driven
  - Initially meant the generation of code from system models (e.g., UML-based)
  - However, in industry the term is often used to simply indicate that models are used

- Formal
  - Aims at deriving a provable specification from the requirements
    - That is, it can be formally checked for correctness and completeness
  - Can be considered a specialization of model-driven approaches
    - Code is ideally (automatically) generated through various transformations



How to find and use a process model?

The basic approach comprises four steps

1. Characterization
   - Assess the underlying conditions for the project, e.g. –
     - Previous experience of the people with processes, methods and tools
     - Resource limitations (people, schedule, budget)
     - Company standards
2. Selection
   - Compare the conditions with existing models
   - Select the best suited model, if necessary evaluate it further (e.g., with a pilot study)
3. Adoption
   - Tailor the model to the environment, i.e., to existing processes and available resources
4. Application
   - Apply the model within the concrete project
   - The experience made is collected and used to improve the model for future projects

Quality Criteria for Process Models I

Denert [Denert/Hesse80] lists the following quality criteria for a universally applicable process model (cited after [Bunse/v. Knethen08]):

1. **Completeness**
   - Describes all phases of software development
   - Covers project management, quality assurance and configuration management
2. **Systematics**
   - Defines a unified terminology and method that allow a precise application
3. **Modularity**
   - The project is divided into manageable units
     - They have precise guidelines and results
     - And are thus plannable and controllable
4. **Customizability**
   - Can be adapted to organizational and technical characteristics

Quality Criteria for Process Models II

5. Generality

- Is scalable and thus usable for systems of various sizes
- Is applicable/usable in different application domains

6. Tool support

- Tools that support the activities of the process model are available

7. Documentation

- How is the model documented?
- As a (hand)book (that no-one really reads?) or in an electronic format?

➔ The following **selection criteria** can be derived

- If results contradict a lot, this may indicate a high development risk
- Due to conflicting project constraints

Selection Criteria: Project-Related I

1. What is the expected effort of the project in person months?

   The larger the project the more support is required by the process model to manage complexity

   - small (1-6 person months)
     - Basically all kinds of models can be used
     - However, avoid models that impose too much management and modeling effort
   - medium (7-12 person months)
     - The process model needs some means to manage complexity
     - The usage of iterative models should make sense here --> it divides development in smaller increments that allow frequent quality assessment
   - large (13+ person month)
     - Projects of this size require significant management efforts
     - The use of a recursive process model might be helpful, as it allows the natural decomposition in increments that can be built on top of each other

Selection Criteria: Project-Related II

2. How many people are supposed to work on the project?
   - The more people participating in the project, the more planning and monitoring activities need to be supported
   - A clear separation of concerns and clear milestones become more important with a growing number of people
     - small (1-10 persons)
       - Basically all models can be used, however, those that are straightforward are probably sufficient, e.g., agile models, such as XP or Scrum
     - medium (11-25 persons)
       - An iterative model that has a clear separation of concerns and thus allows a distribution of tasks is probably best suited here
       - The model should be well documented, e.g., RUP
     - large (26+ persons)
       - Again, iterative models are probably best suited
       - However, this time the model needs a clear support for planning and monitoring activities, few models support that fully

Selection Criteria: Project-Related III

3. Does the project require distributed teams?
   - Requires a clear separation of concerns and clear milestones
   - Agile projects are typically required to be non-distributed

4. What is the main priority in the project?

It should be supported by the process model

- Schedule
  - Iterative models guarantee that at least „something“ can be delivered on time
- Functionality
  - The elicitation and documentation of requirements should be well supported
  - As well as planning and controlling mechanisms
- Quality
  - Choose a model that supports systematic quality assurance activities explicitly

Selection Criteria: Team-Related

1. Experience of the project manager?
   - Choose a simple (sequential or agile?) model if not well experienced, or at least one that is well documented

2. What is the technical experience of the people planned for the project?
   - Especially in regard to the concepts (such as OO or frameworks used) that are supposed to be used
   - Choose a prototypical (perhaps an iterative) approach if little experience is available --> Plan to throw one version away!

3. How much experience do the developers have in the application domain?
   - Choose an iterative (or even an agile) model if little experience is available
     - In order to enable early feedback and tight customer involvement
     - Try to avoid distributed development teams

Selection Criteria: Development Constraints

1. Is the system developed from scratch?
   - Many process models target only development from scratch and ignore existing environments
   - Choose a model that also supports the modeling of interfaces to existing systems
2. What is the desired degree of reuse?
   - Reuse gains more and more importance, as it is supposed to reduce effort and increase quality of software development
   - However, systematic reuse needs to be supported by the process model, as it requires the integration of existing components into the system
   - Choose a model that supports reuse
3. What tool support is required or available?
   - Many process models are quite complex, however, tools don't always exists that support their application or they may be too expensive
   - If this applies to your project, choose a simpler (e.g., agile) model

Selection Criteria: Requirements-Related

1. Are the requirements stable and well-understood?
   - Changing requirements and constraints are a common issue in software development
   - They call for iterative/agile approaches aiming to minimize the effects of changes
2. Are presentable intermediate versions necessary?
   - If software is developed for a specific customer, it is often necessary to present working prototypes early in the development process
   - This can also help to clarify unclear requirements
   - Iterative (agile) approaches are generally better suited for this challenge
3. Are concrete milestones required by the customer?
   - Milestones typically require a clear separation of activities in the model
   - This calls for a model that can be mapped to a sequential process