Introduction to Software Engineering

09 Software Lifecycle Modeling

Stephan Krusche, Pramod Bhatotia

05 July 2022
Technical University of Munich
https://ase.in.tum.de

Roadmap of the Lecture

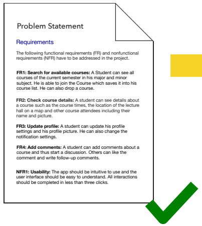
- **Context and assumptions**
  - We completed requirements elicitation, analysis, system & object design, and testing
  - You know the most important activities of model-based software engineering
  - You understand Scrum, UML diagrams, JavaFX, Gradle, REST, MVC, and patterns
- **Learning goals: at the end of this lecture you are able to**
  - Explain various software development lifecycle models:
    e.g. waterfall model, spiral model, unified process
  - Differentiate between iterative, incremental and adaptive development
  - Differentiate between activity- and entity-oriented models
  - Differentiate between defined and empirical process control
  - Model a software development lifecycle using UML activity diagrams

Course schedule (Garching)

#	Date	Subject
1	26.04.22	Introduction
2	03.05.22	Model-based Software Engineering
3	10.05.22	Requirements Analysis
4	17.05.22	System Design I
5	24.05.22	System Design II
6	31.05.22	Object Design I
	07.06.22	**Holiday (no lecture, no tutor groups)**
7	14.06.22	Object Design II
8	21.06.22	Testing
	28.06.22	**Guest Lecture SAP (no tutor groups)**
9	05.07.22	Software Lifecycle Modeling
10	12.07.22	Software Configuration Management
11	19.07.22	Software Quality Management
12	26.07.22	Project Management

Overview of model based software engineering

Customer



Problem Statement

Requirements

The following functional requirements (FR) and nonfunctional requirements (NFR) have to be addressed in the project.

FR1: Search for available courses: A Student can see all courses of the current semester in his major and minor subject. He is able to join the Course which saves it into his course list. He can also drop a course.

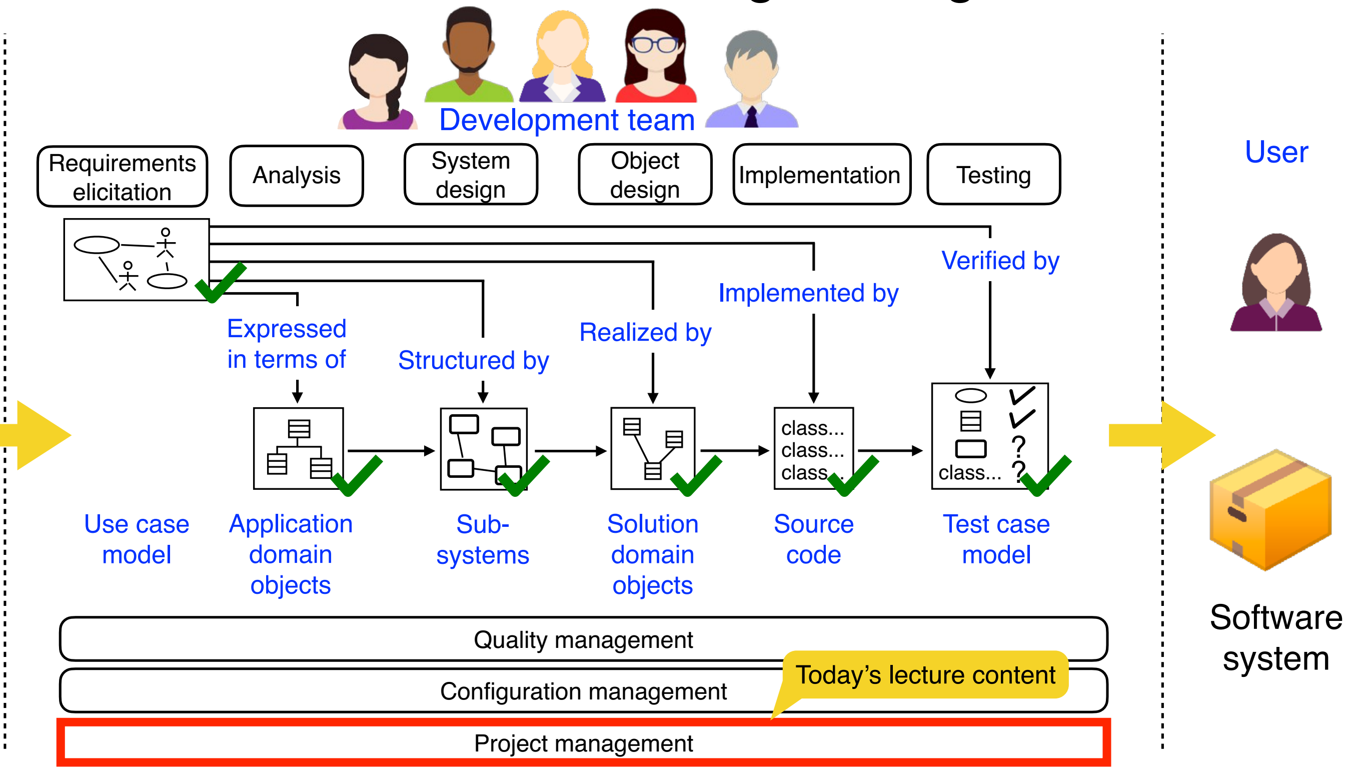
FR2: Check course details: A student can see details about a course such as the course times, the location of the lecture hall on a map and other course attendees including their name and picture.

FR3: Update profile: A student can update his profile settings and his profile picture. He can also change the notification settings.

FR4: Add comments: A student can add comments about a course and thus start a discussion. Others can like the comment and write follow-up comments.

NFR1: Usability: The app should be intuitive to use and the user interface should be easy to understand. All interactions should be completed in less than three clicks.

Problem statement

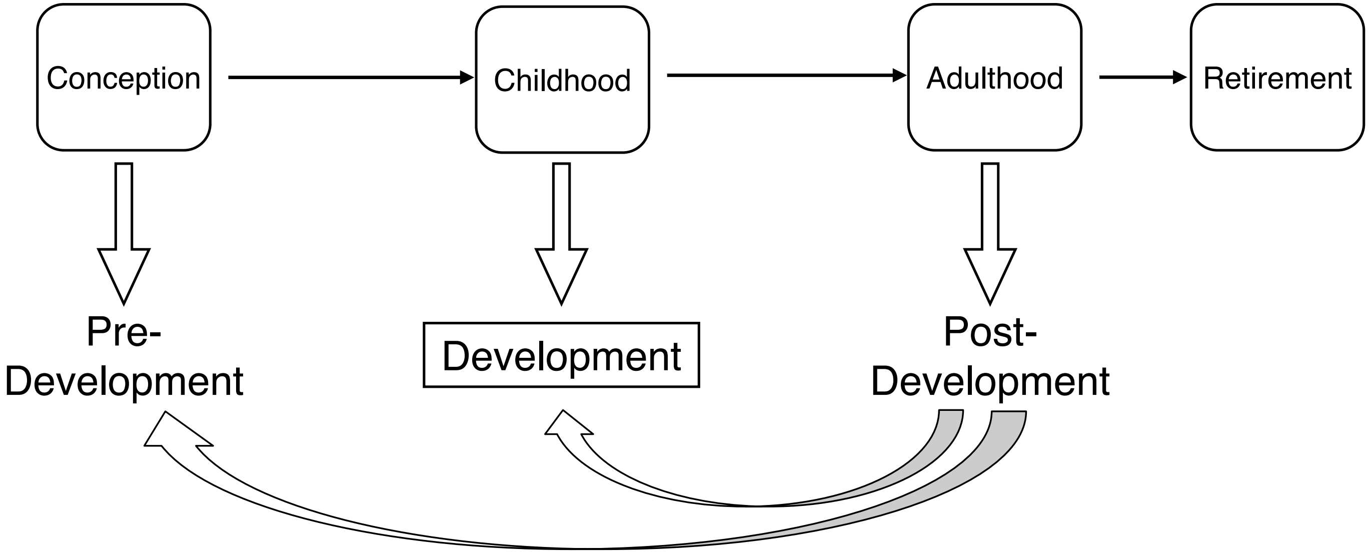


Outline

➡ **Software development as application domain**
  - UML activity diagrams
- Linear models
  - Waterfall model
  - V-model
- Iterative models
  - Spiral model
  - Unified process
- Agile model
  - Extreme programming
  - Scrum
  - Kanban

Software development lifecycle

Based on the metaphor of the life of a person



Definitions

- **Software development lifecycle**
  - Set of activities and their relationships to each other to support the development of a software system
  - Examples of activities: requirements elicitation, analysis, system design, implementation, testing, software configuration management, delivery
  - Examples of relationships: implementation must be done before testing, analysis, and system design can be done in parallel
- **Software development lifecycle model**
  - An abstraction representing the development of software for understanding, monitoring, or controlling the software

Typical software development lifecycle questions

- ➡ **Which activities** should we select for the software project?
- What are the **dependencies between activities**?
- How should we **schedule the activities**?

Review: example of software development activities

Requirements analysis	What is the problem?
System design	What is the solution?
Object design	What are the best mechanisms to implement the solution?
Implementation	How is the solution constructed?
Testing	Is the problem solved?
Delivery	Can the customer use the solution?
Maintenance	Are enhancements needed?

Software project management as a system

- We can handle the management of a software project like we handle a complex system
- We studied methods to model complex systems (UML, SysML,…)
- We can apply these methods to model software project management

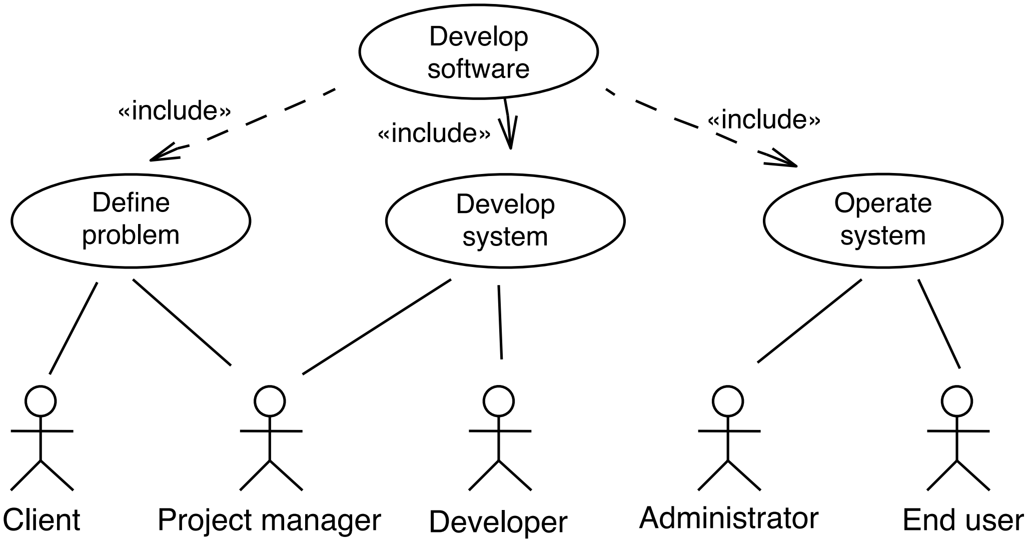
Norman Rockwell – “Triple Self Portrait”

Modeling a software development lifecycle

We can use the same modeling techniques we use for software development:

1. **Functional model** of a software development lifecycle
   - Scenarios
   - Use cases
2. **Structural model** of a software development lifecycle
   - Class diagrams
   - Object identification
3. **Dynamic model** of a software development lifecycle
   - Communication diagrams
   - Activity diagrams

Functional model of a simple lifecycle (use-case diagram)



Modeling a software development lifecycle

We can use the same modeling techniques we use for software development:

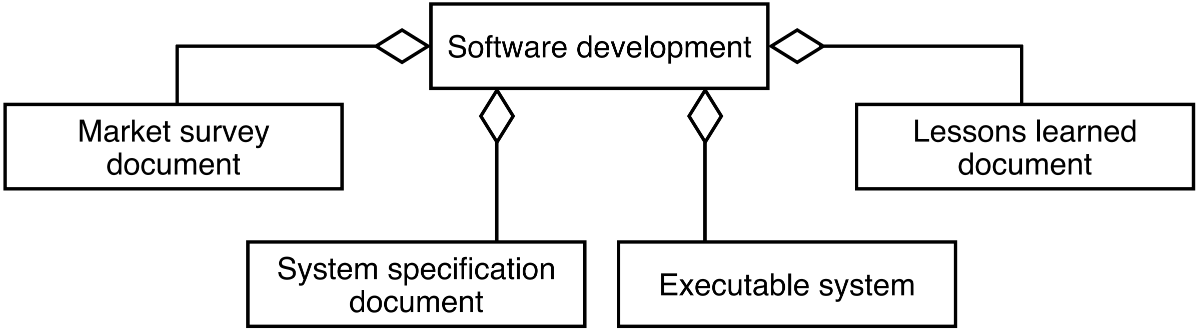
1. **Functional model** of a software development lifecycle
   - Scenarios
   - Use cases
2. **Structural model** of a software development lifecycle
   - Class diagrams
   - Object identification
3. **Dynamic model** of a software development lifecycle
   - Communication diagrams
   - Activity diagrams

Two major views of software development lifecycles

➡ 1) Entity-centered: software development as a set of deliverables

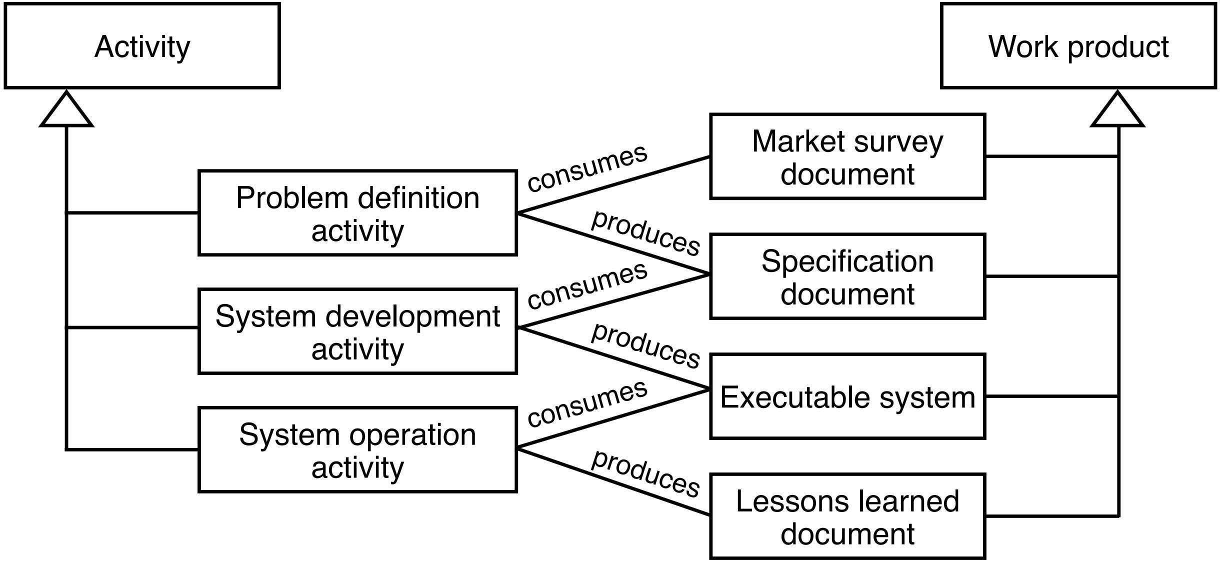
2) Activity-centered: software development as a set of activities

Entity centered view

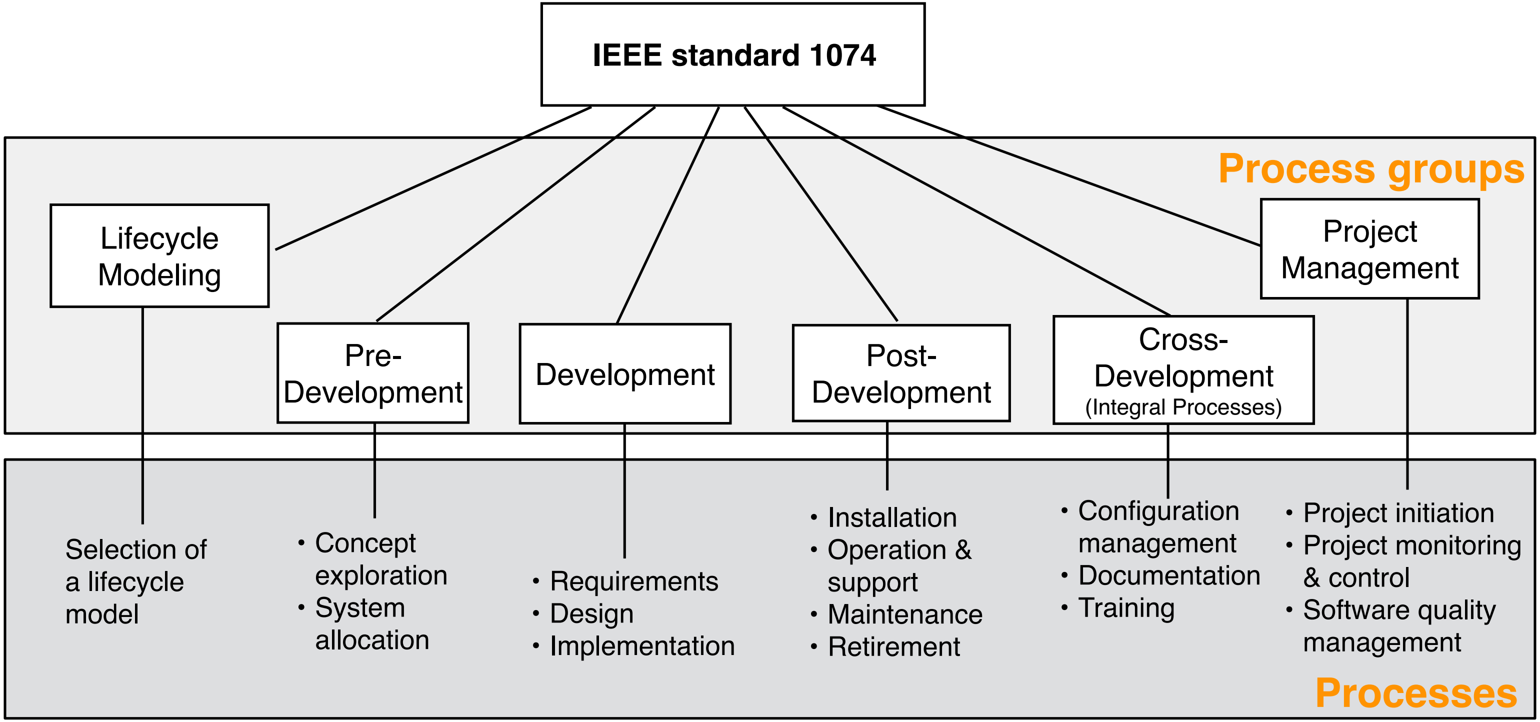


Software development consists of the creation of a set of deliverables

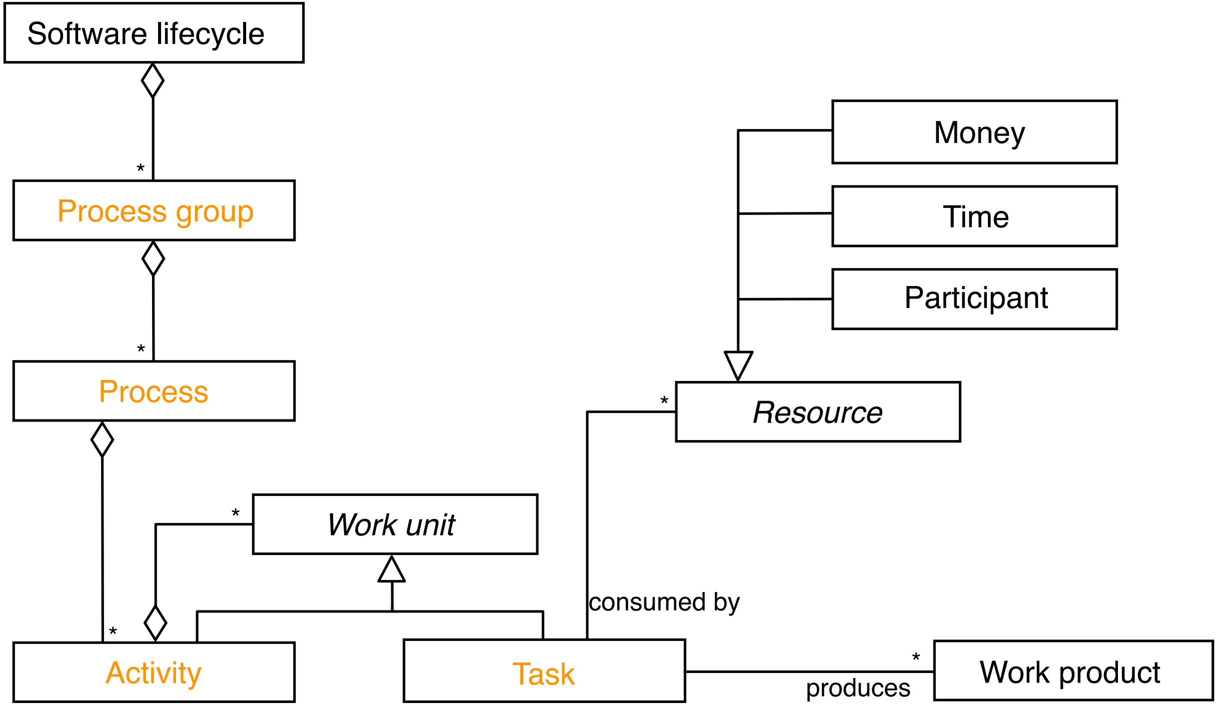
Combining activities and entities in a UML class diagram



IEEE Standard 1074 for software lifecycle activities

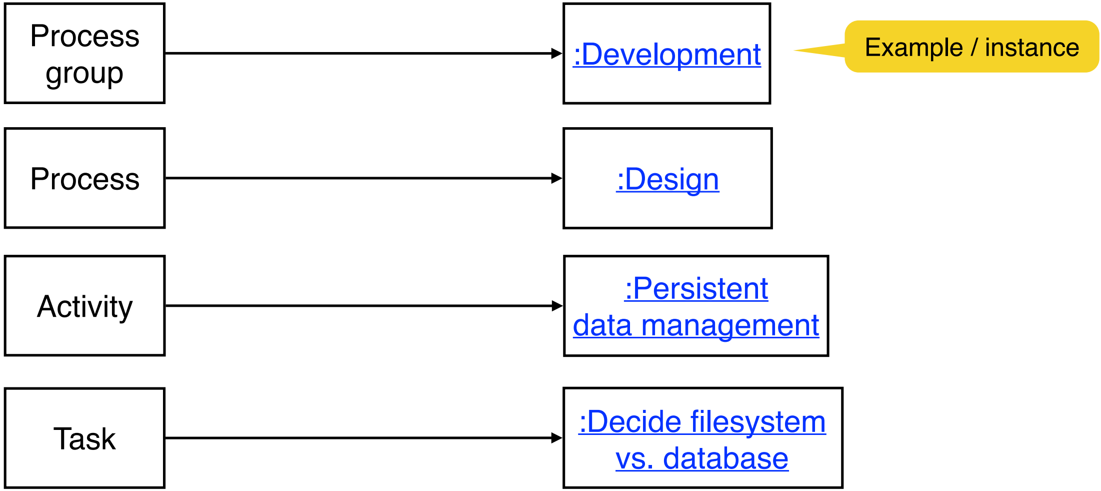


Object model of the IEEE 1074 Standard



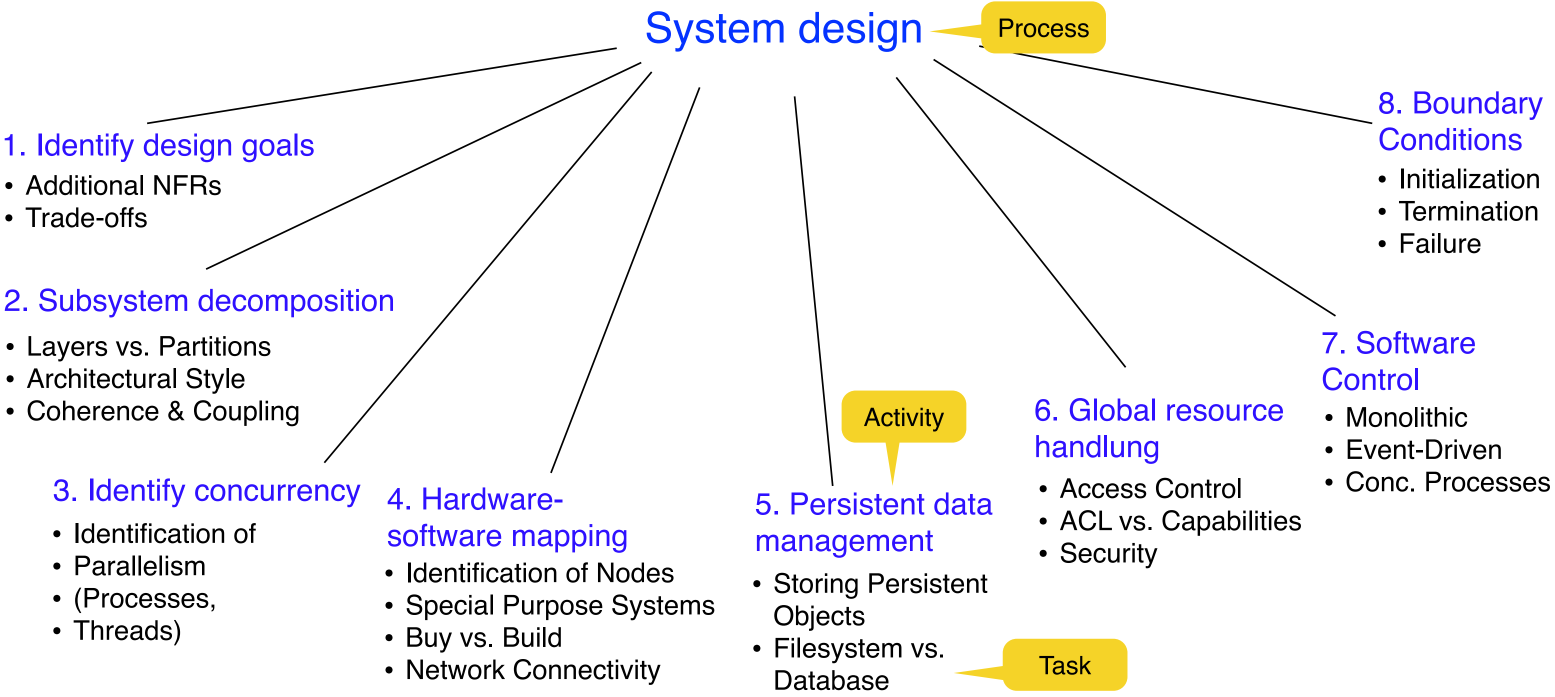
Process groups, processes, activities, and tasks

- Process group: consists of a set of processes
- Process: consists of activities
- Activity: consists of sub-activities (phases, steps,…) and tasks



Development

Process group



System design

Process

1. Identify design goals

- Additional NFRs
- Trade-offs

2. Subsystem decomposition

- Layers vs. Partitions
- Architectural Style
- Coherence & Coupling

3. Identify concurrency

- Identification of
- Parallelism
- (Processes,
- Threads)

4. Hardware-software mapping

- Identification of Nodes
- Special Purpose Systems
- Buy vs. Build
- Network Connectivity

Activity

5. Persistent data management

- Storing Persistent Objects
- Filesystem vs. Database

Task

6. Global resource handlung

- Access Control
- ACL vs. Capabilities
- Security

7. Software Control

- Monolithic
- Event-Driven
- Conc. Processes

8. Boundary Conditions

- Initialization
- Termination
- Failure

Tailoring

- There is no “one size fits all” software development lifecycle model that works for all possible software engineering projects
- Tailoring: adjusting the lifecycle model to fit a specific project

1) Naming: adjust the naming of activities
2) Cutting: remove activities that are not necessary for the project
3) Ordering: define the order of the activities in which they take place

Review: software development activities

Requirements analysis	What is the problem?
System design	What is the solution?
Detailed design	What are the best mechanisms to implement the solution?
Program implementation	How is the solution constructed?
Testing	Is the problem solved?
Delivery	Can the customer use the solution?
Maintenance	Are enhancements needed?

Tailoring example 1) adjusting the names of activities

Requirements analysis	What is the problem?
System design	What is the solution?
Object design	What are the best mechanisms to implement the solution?
Implementation	How is the solution constructed?
Testing	Is the problem solved?
Delivery	Can the customer use the solution?
Evolution	Are enhancements needed?

Tailoring example 2) cutting the number of activities

Requirements analysis	What is the problem?
System design	What is the solution?
Object design	What are the best mechanisms to implement the solution?
Implementation	How is the solution constructed?

Tailoring example 3) reordering the activities

Requirements analysis — What is the problem?

How is this approach called?

Test driven development (TDD)

Testing — Is the problem solved?

Implementation — How is the solution constructed?

Delivery — Can the customer use the solution?

Modeling a software development lifecycle

We can use the same modeling techniques we use for software development:

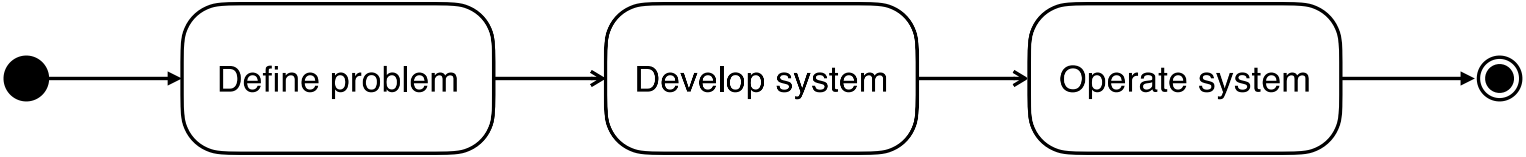
1. **Functional model** of a software development lifecycle
   - Scenarios
   - Use cases
2. **Structural model** of a software development lifecycle
   - Class diagrams
   - Object identification
3. **Dynamic model** of a software development lifecycle
   - Communication diagrams

   ➡ Activity diagrams

Outline

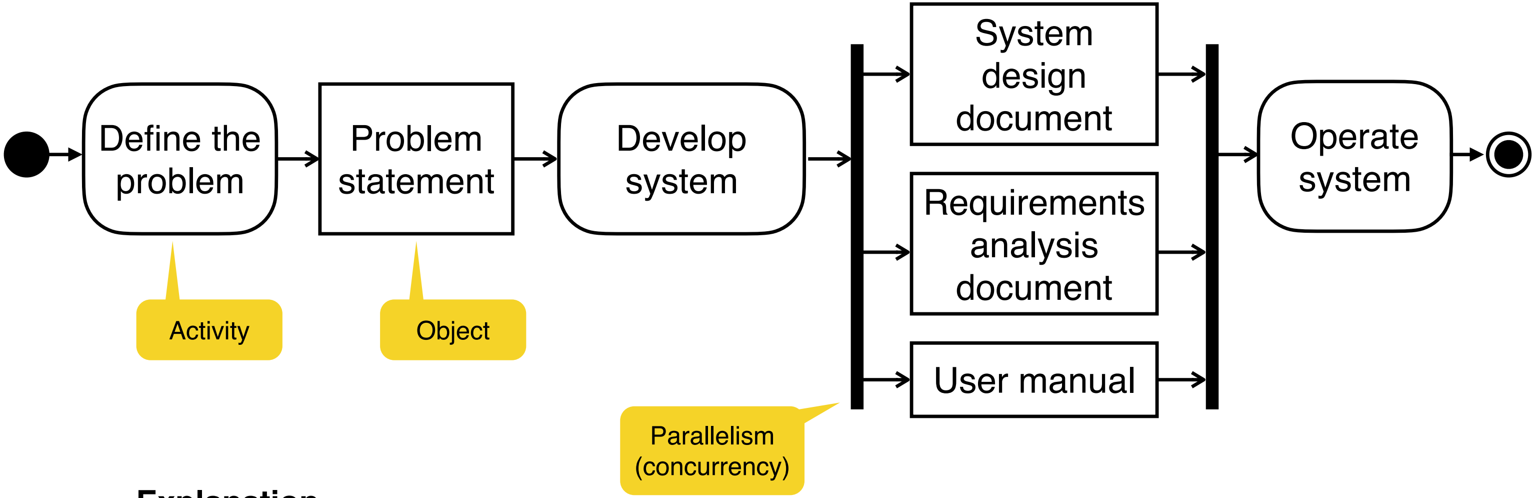
- Software development as application domain
- ➡ **UML activity diagrams**
- Waterfall model
- V-model
- Spiral model
- Unified process
- Limitations of linear and iterative models
- Extreme programming
- Scrum
- Kanban

UML activity diagram for the same lifecycle model



Explanation
In this model, software development goes through a linear progression of activities called problem definition, system development, and system operation

UML activity diagram for a lifecycle model with objects



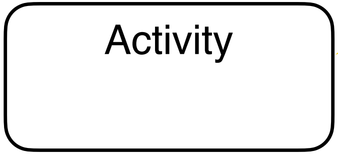
Explanation

In this exemplary model, software development goes through a linear progression of states called problem definition activity, software development activity, and system operation activity The system operation activity can only start when the artifacts system design document, requirements analysis document, and user manual have been created

UML activity diagrams

- Activity diagrams consist of nodes and edges
- **Nodes** describe activities and objects
  - Control nodes
  - Executable nodes
    - Most prominent: action node
  - Object nodes
    - E.g. a document
- An **edge** is a directed connection between nodes

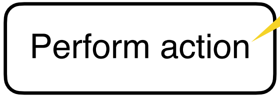
UML activity diagram elements



Activity

Large: can be decomposed into smaller ones

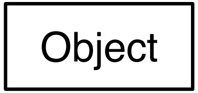
Activity node



Perform action

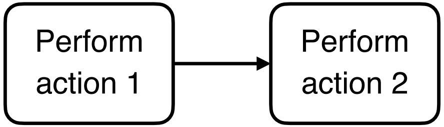
Small

Action node



Object

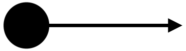
Object node



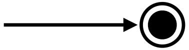
Perform action 1

Perform action 2

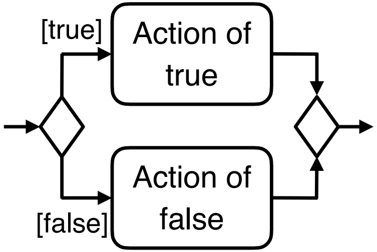
Control flow



Initial node



Final node



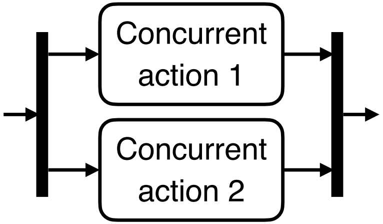
[true]

Action of true

[false]

Action of false

Decision and merge node (conditions)

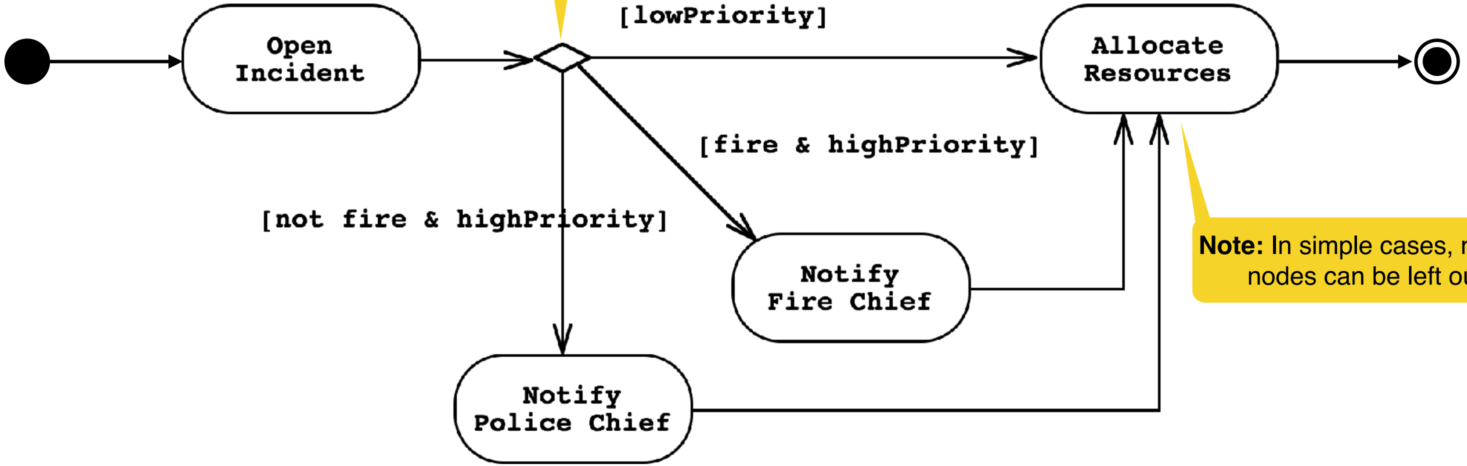


Concurrent action 1

Concurrent action 2

Fork and join node (concurrency)

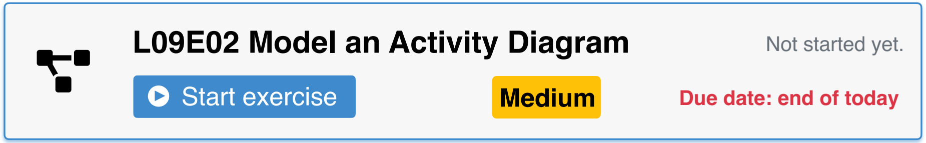
Example of a decision node



Predicate

Decision

Note: In simple cases, merge nodes can be left out



L09E02 Model an Activity Diagram

Not started yet.

▶ Start exercise

Medium

Due date: end of today

🕓 10 min

🏆 4 pts

- **Problem statement**
  - Model an activity diagram for the scenario **Pass EIST exam** (next slide)
  - Insert actions and objects as needed
- Hints
  - Use action nodes and control flow for sequential actions
  - Use decision and merge nodes for conditional actions
  - Use fork and join nodes for concurrent messages

Same scenario as in L03E03

Visionary scenario for L09E02

1) Name: Pass EIST exam

2) Participating actors

Peter: Student,
Stephan: Lecturer

3) Flow of events

1. Peter enrolls in the EIST course and starts the course
2. Stephan prepares the final exam
3. Peter takes the final exam
4. Stephan corrects the final exam
5. If Peter has passed the exam, he receives a certificate
6. Peter evaluates Stephan at the end

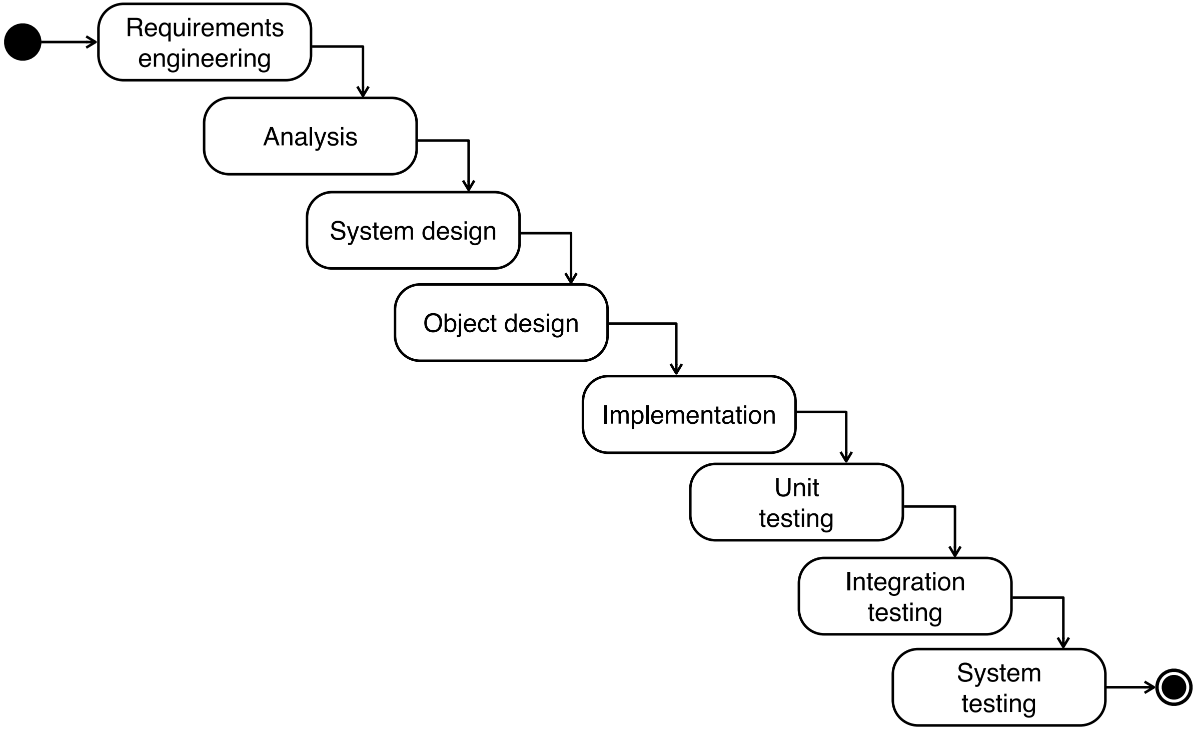
Outline

- Software development as application domain
- UML activity diagrams
- ➡ **Waterfall model**
- V-model
- Spiral model
- Unified process
- Limitations of linear and iterative models
- Extreme programming
- Scrum
- Kanban

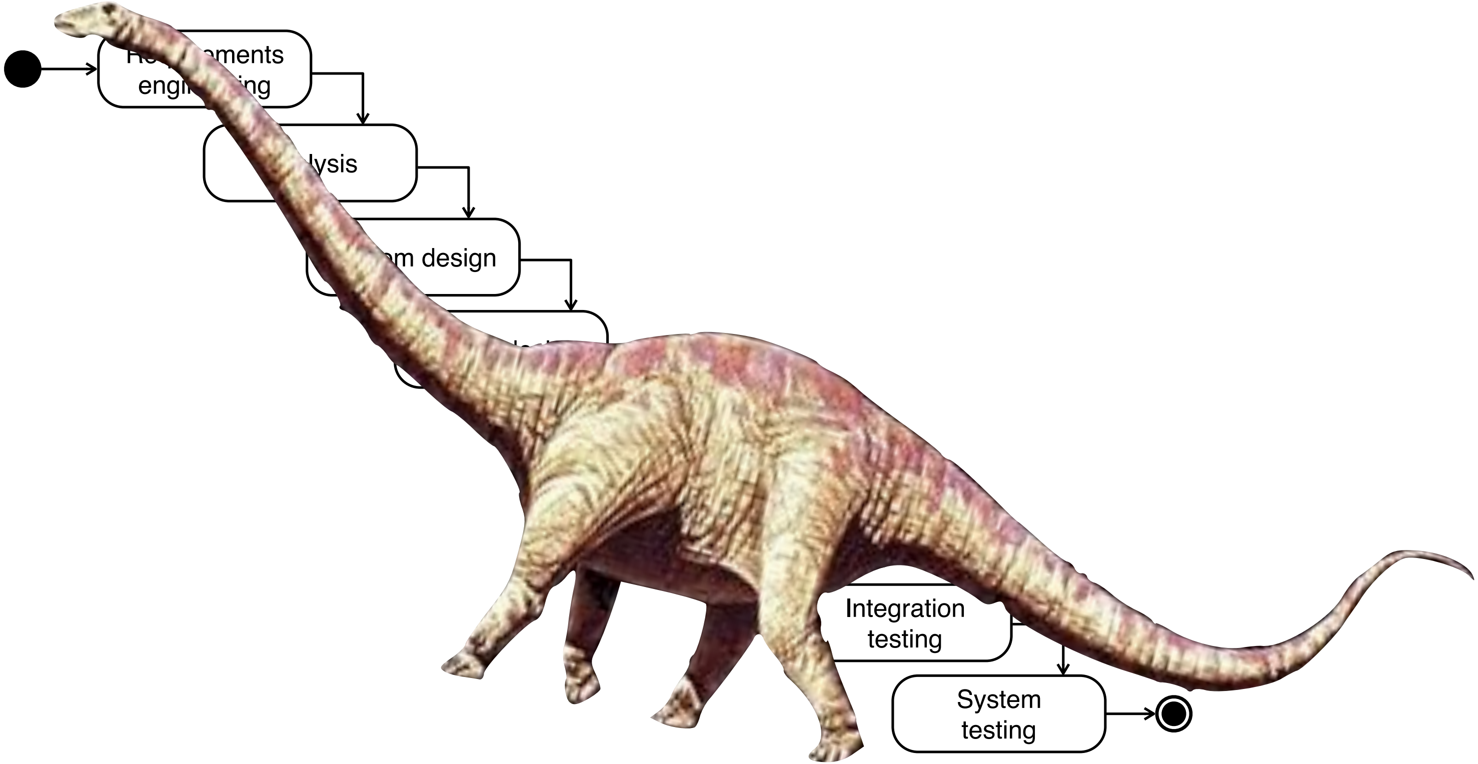
Waterfall model

- First described in 1970 by Royce
- Activity-centered lifecycle model: **sequential** execution of activities
  - Assumes that software development can be scheduled as a step-by-step process that transforms user needs into code
  - You cannot turn back once an activity is completed
- **Verification**: ensures that no activity introduces unwanted or deletes mandatory requirements
- Simplistic view of software development: measures progress by the number of tasks that have been completed

The Waterfall model



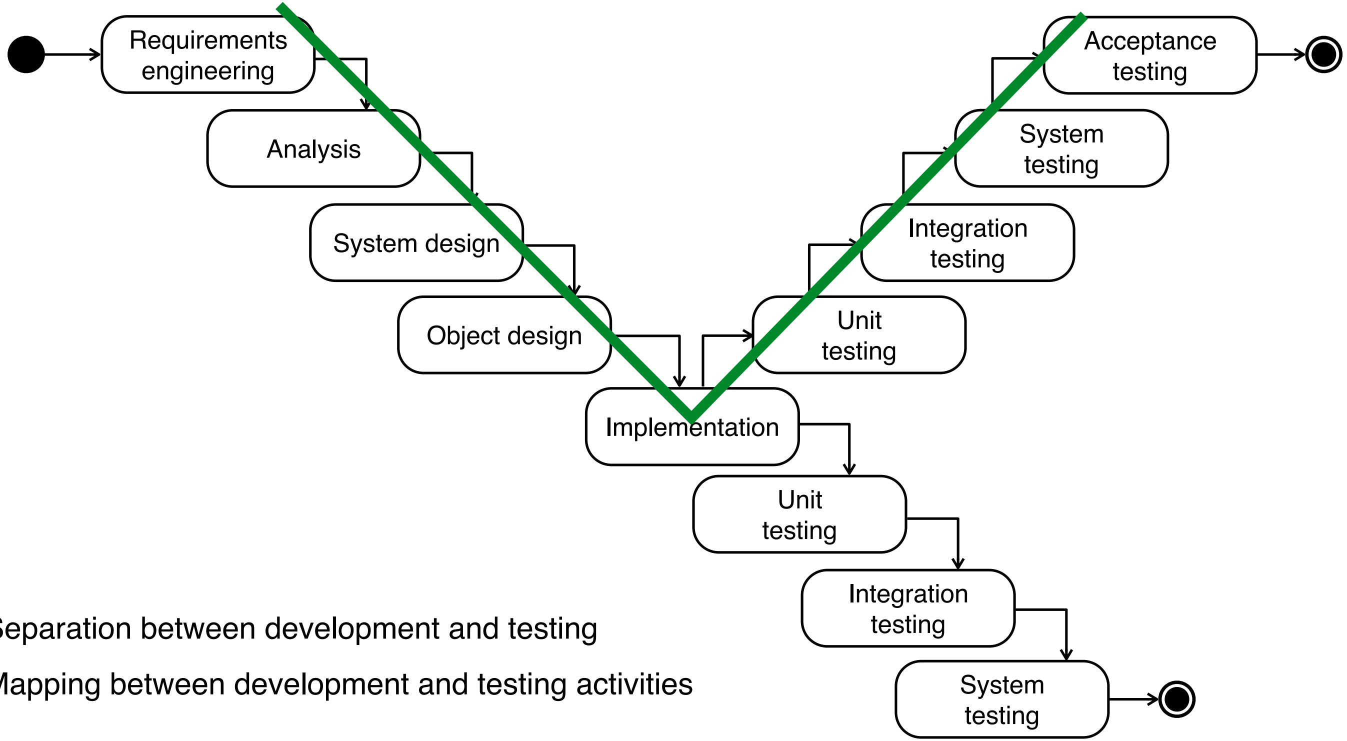
The Waterfall model is a dinosaur



Outline

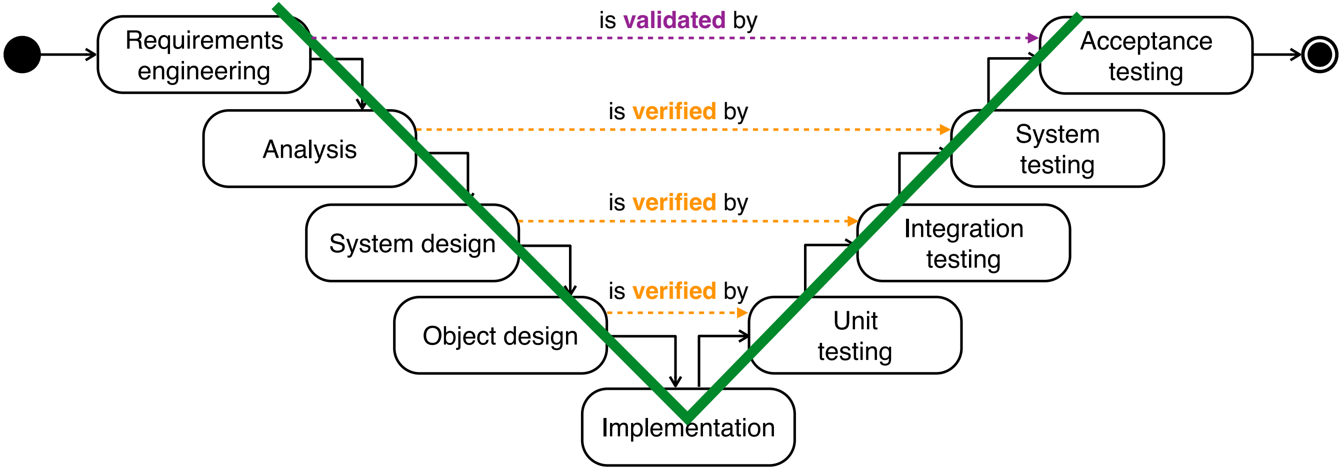
- Software development as application domain
- UML activity diagrams
- Waterfall model
- ➡ **V-model**
- Spiral model
- Unified process
- Limitations of linear and iterative models
- Extreme programming
- Scrum
- Kanban

From the Waterfall model to the V-model



- Separation between development and testing
- Mapping between development and testing activities

V-model



- **Validation:** assurance that a product, service, or system meets the needs of the customer and other identified stakeholders (often involves acceptance and suitability with **external** customers)
- **Verification:** evaluation whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition (often an **internal** process)

V-model (explanation)

- Horizontal arrows show information flow between activities of the same abstraction level

  ➡ The layout of the activities has no semantics in UML

1) **Higher levels** of abstraction of the V-model deal with the requirements in terms of elicitation and operation

   ➡ The client acceptance activity **validates** the understanding of the user against the requirements

2) **The middle part** focuses on mapping the understanding of the problem into a software architecture

   ➡ Integration tests **verify** components and subsystems against the preliminary design

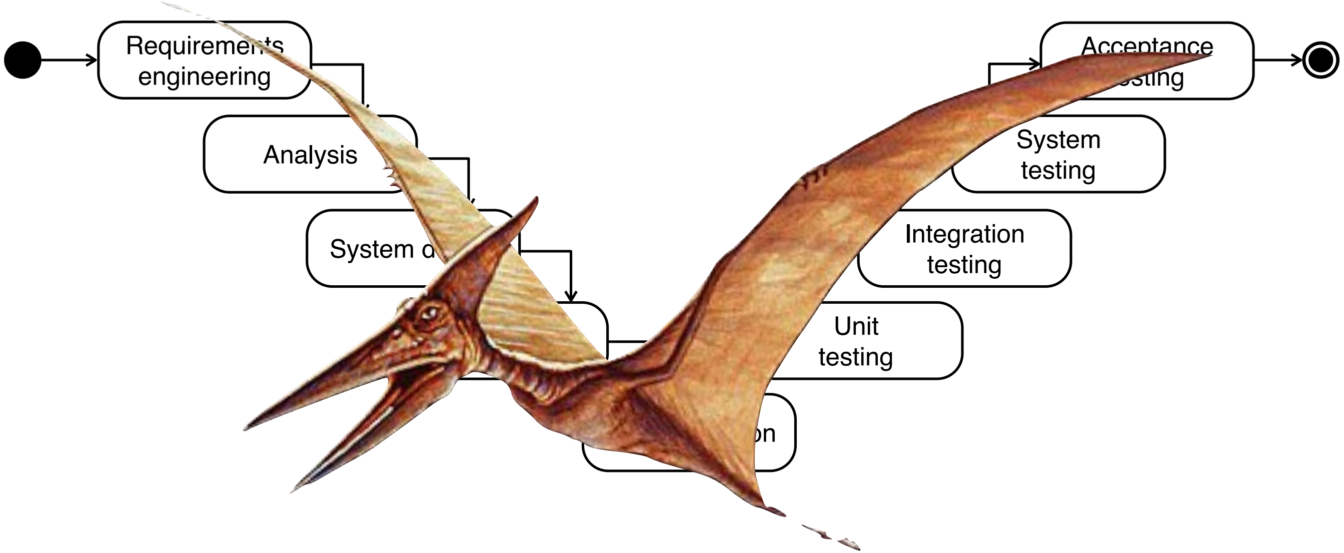
3) **The lower levels** focus on details such as the assembly and coding of software components

   ➡ Unit tests **verify** classes and methods against their description in the detailed design

Validation vs. verification

- Validation
    - Assurance that a product, service, or system meets the needs of the customer and other identified stakeholders
    - Often involves acceptance by **external** customers
    - **Informally: “Have you built the right thing?"**
- Verification
    - Evaluation whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition
    - Often an **internal** process
    - **Informally: “Have you built the thing right?"**

The V-model is also a dinosaur



Properties of sequential models

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

The alternative: Embrace iteration

lithograph by the Dutch artist M.C.Escher, first printed in October 1961.

Outline

- Software development as application domain
- UML activity diagrams
- Waterfall model
- V-model
- ➡ **Spiral model**
- Unified process
- Limitations of linear and iterative models
- Extreme programming
- Scrum
- Kanban

Spiral model

- The Spiral model was proposed by Barry Boehm in 1987 to deal with the problems of the Waterfall model
- It is an **iterative** model with **4 major activities**

1) Determine **objectives**, **alternatives,** and **constraints**
2) **Evaluate** alternatives and **identify** risks, **resolve** these risks by assigning priorities to them
3) **Develop** a series of **prototypes** for the identified risks starting with the highest risk: use a linear model for each prototype development
4) If a risk has successfully been resolved, **evaluate the results** of the iteration and **plan the next iteration**

➡ If the risk could not be resolved, the project is terminated

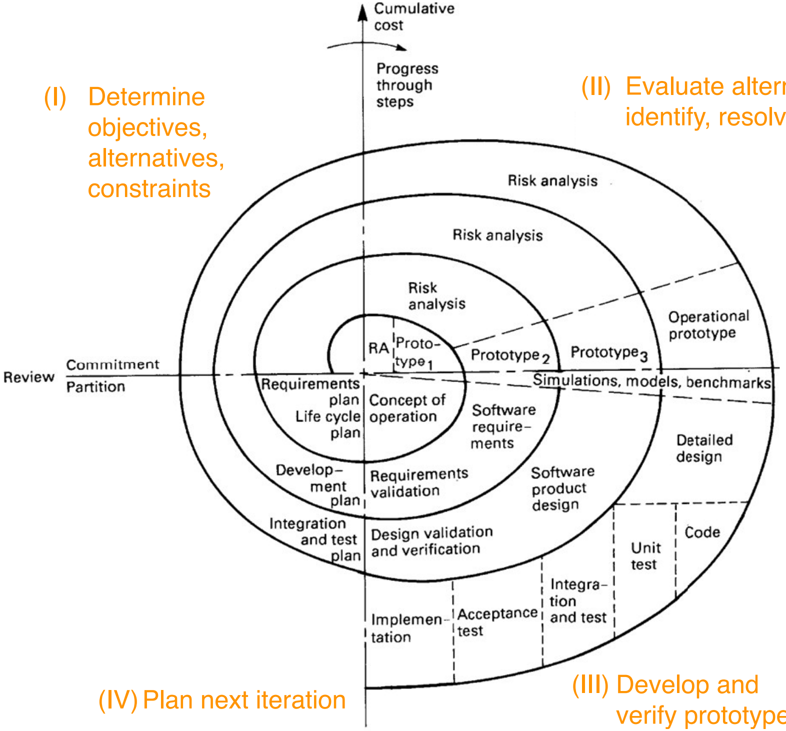
➡ The 4 activities are applied in 9 iterations

Iterations in the Spiral model

1. Concept of operations
2. Software requirements
3. Software product design
4. Detailed design
5. Code
6. Unit test
7. Integration and system test
8. Acceptance test
9. Implementation

- For each iteration go through these activities
    1. Determine objectives, alternatives, and constraints
    2. Evaluate alternatives, identify, and resolve risks
    3. Develop and verify prototype
    4. Plan next iteration

Visualization of the Spiral model



(I) Determine objectives, alternatives, constraints

(II) Evaluate alternatives, identify, resolve risks

(III) Develop and verify prototype

(IV) Plan next iteration

Outline

- Software development as application domain
- UML activity diagrams
- Waterfall model
- V-model
- Spiral model
- ➡ **Unified process**
- Limitations of linear and iterative models
- Extreme programming
- Scrum
- Kanban

Unified process

- Developed by Booch, Jacobson, and Rumbaugh
- **Iterative** and **incremental** lifecycle model built on the idea of cycles in the lifetime of a software system
- Each cycle consists of 2 **stages** (engineering, production) and 4 **phases** (inception, elaboration, construction, transition)
- Each phase can consist of several iterations
- Several workflows are performed in parallel: management, environment, requirements, design, implementation, assessment, deployment

The 2 stages in the unified process

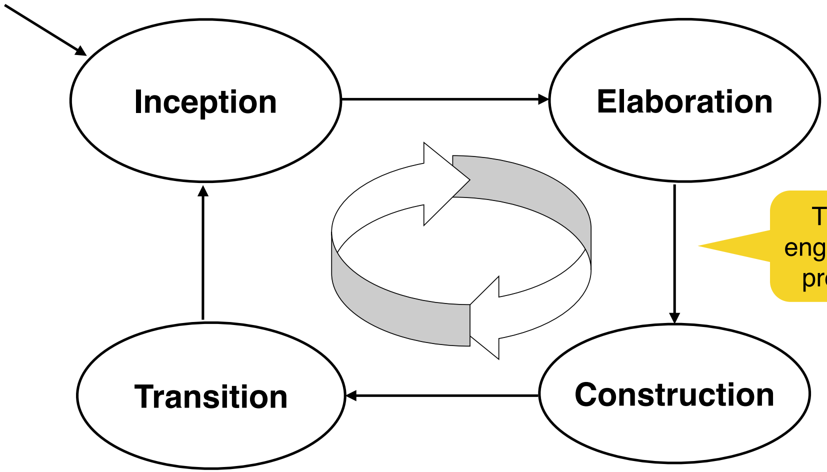
1. Engineering stage

- Less predictable but smaller teams, focusing on design and synthesis activities
- The engineering stage consists of 2 phases
  - Inception phase
  - Elaboration phase

2. Production stage

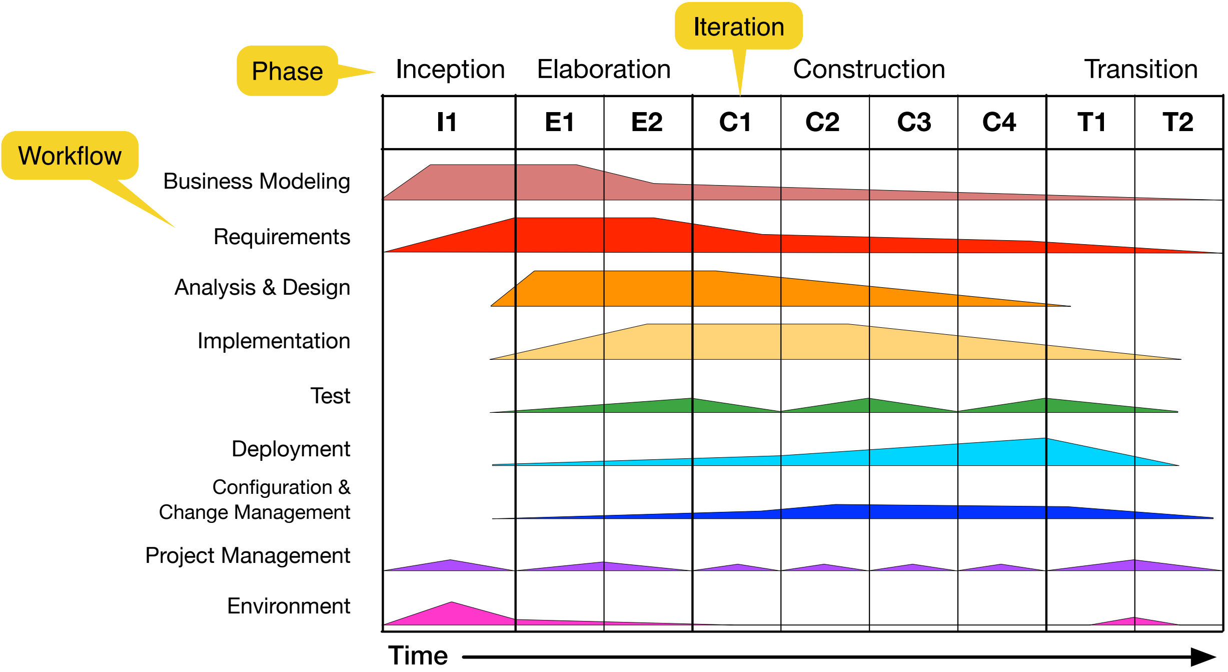
- More predictable but larger teams, focusing on construction, test and deployment activities
- The production stage also consists of 2 phases
  - Construction phase
  - Transition phase

Phases in the unified process



Transition from engineering stage to production stage

Workflow vs. phase



Outline

- Software development as application domain
- UML activity diagrams
- Waterfall model
- V-model
- Spiral model
- Unified process

➡ **Limitations of linear and iterative models**

- Extreme programming
- Scrum
- Kanban

Limitations of linear and iterative models

- Linear and iterative models do not deal well with **frequent change**
    - The Waterfall model assumes that once you are done with an activity, all issues covered in that activity are closed and cannot be reopened
    - The Spiral model can deal with changes between activities, but does not allow change within an activity
- What do you do if changes are happening **more frequently**?

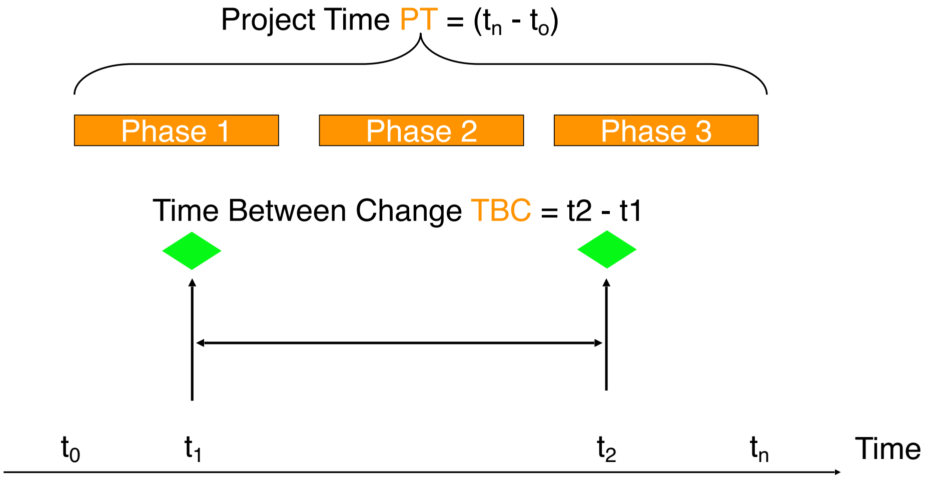
→ Agile methods

Frequency of change

3 definitions:

- **PT** = Project Time
- **TBC** = Time Between Change
- **MTBC** = Mean Time Between Change

◆ Change occurred



Project Time PT = $(t_n - t_o)$

Phase 1

Phase 2

Phase 3

Time Between Change TBC = t2 - t1

t_0 t_1 t_2 t_n Time

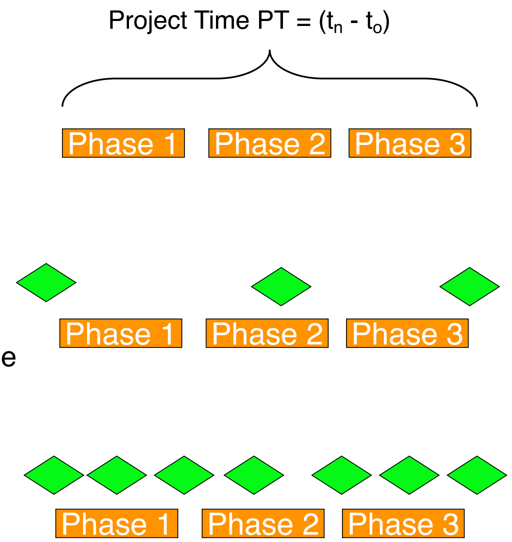
Change influences the choice of the lifecycle model

PT = Project Time

MTBC = Mean Time Between Change

◆ Change occurred

- **Change rarely occurs** (MTBC >> PT)
    - **Sequential** model: Waterfall model, V-model
    - Open issues are closed before moving to next phase
- **Change occurs sometimes** (MTBC ≈ PT)
    - **Iterative** model: Spiral model, Unified process
    - Change during a phase may lead to the iteration of a previous phase or cancellation of the project
- **Change is frequent** (MTBC << PT)
    - **Agile** model: Scrum, Kanban, Extreme programming (XP)
    - Change during a phase can lead to reengineering the requirements or the design



Definition: incremental vs. iterative vs. adaptive

- Incremental means to "**add onto something**"

  ➡ **Incremental** development helps you improve your **process:** various parts of the system are developed at different times or rates and integrated as they are completed

- Iterative means to "**redo something**" or to "**rework something**"

  ➡ **Iterative** development helps you to improve your **product:** time is set aside to revise and improve parts of the system

- Adaptive means to "**react to changing requirements**"

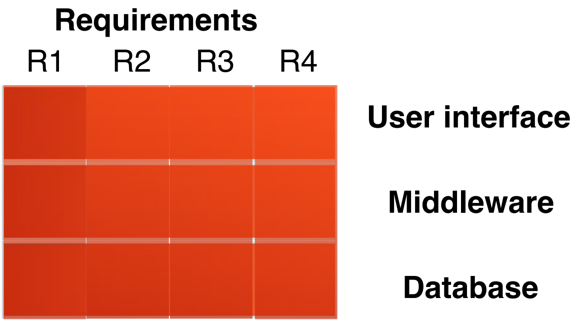
  ➡ **Adaptive** development improves the reaction to **changing** customer needs

Incremental vs. iterative vs. adaptive development

Incremental means to “**add onto something**”

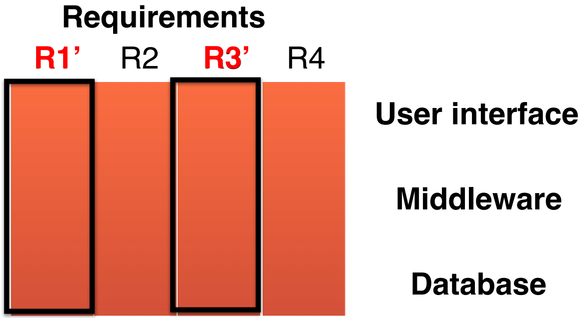
- Incremental development helps to improve the process

Bottom-up integration
Top-down integration
Vertical integration



Incremental vs. iterative vs. adaptive development

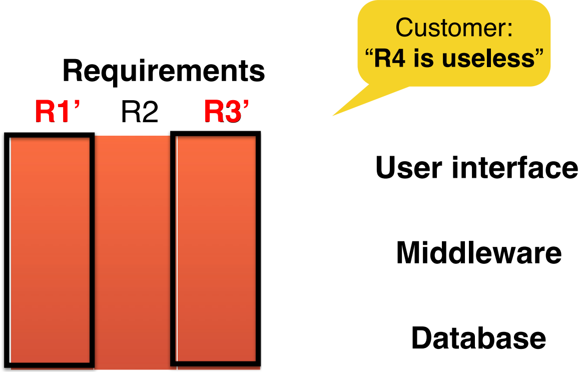
- **Incremental** means to “**add onto something**”
  - Incremental development helps to improve the process
- **Iterative** means to “**redo something**” or to “**rework something**”
  - Iterative development helps to improve the product



Incremental vs. iterative vs. adaptive development

- **Incremental** means to “**add onto something**”
  - Incremental development helps to improve the process
- **Iterative** means to “**redo something**” or to “**rework something**”
  - Iterative development helps to improve the 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.



Outline

- Software development as application domain
- UML activity diagrams
- Waterfall model
- V-model
- Spiral model
- Unified process
- Limitations of linear and iterative models
- ➡ **Extreme programming**
- Scrum
- Kanban

History of agile models

- 1995: Jeff Sutherland and Ken Schwaber analyzed common software development processes
  - “Linear and iterative processes are not suitable for unpredictable and non-repeatable situations”
- 1996: Introduction of **Scrum** at OOPSLA
- 1996: Kent Beck formulates the software engineering methodology **Extreme programming** (XP)
- 2001: Publication “Agile Software Development with Scrum” by Ken Schwaber & Mike Beedle
- ➡ 2001: Manifesto for Agile Software Development
- 2003: Lean software development: translate principles of Toyota production systems to software development: **Kanban**

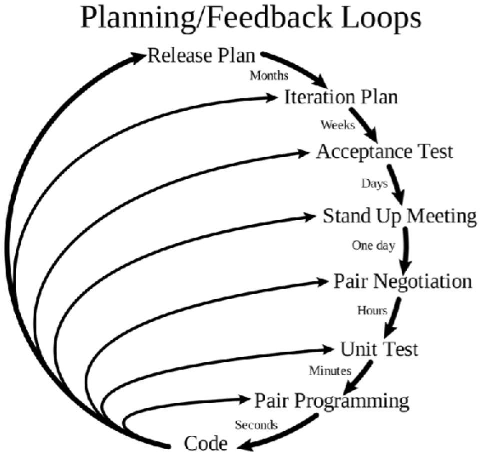
Manifesto for Agile Software Development

- **Individuals and interactions** are more important than processes & tools
- **Working software** is more important than comprehensive documentation
- **Customer collaboration** is more important than contract negotiation
- **Responding to change** is more important than following a plan

http://www.agilemanifesto.org

Extreme programming (XP)

- **Original contributors:** Kent Beck, Ron Jeffries, Ward Cunningham
- Main goals
  - Avoid over-planning
  - Improve software quality
  - Improve responsiveness to changing customer requirements
- **Terminology:** iteration, deliverable, release
- **5 fundamental principles:** rapid feedback, assume simplicity, incremental change, embracing change, quality work
- **4 roles:** developer, customer, manager, coach
- **12 practices:** how to approach the development process



Extreme programming: 12 practices grouped in 4 areas

1. Rapid, fine feedback

1) Test driven development

2) On-site customer

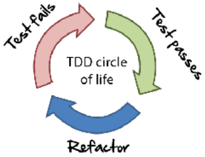
3) Pair programming

2. Continuous process

4) Continuous integration

5) Refactoring

6) Short releases



More about **continuous integration** in **L10 Software Configuration Management**

3. Shared understanding

7) The planning game

8) Simple design

9) Metaphor

10) Collective ownership

11) Coding standards

4. Developer welfare

12) 40 hour Week

Outline

- Software development as application domain
- UML activity diagrams
- Waterfall model
- V-model
- Spiral model
- Unified process
- Limitations of linear and iterative models
- Extreme programming

➡ **Scrum**

- Kanban

Scrum

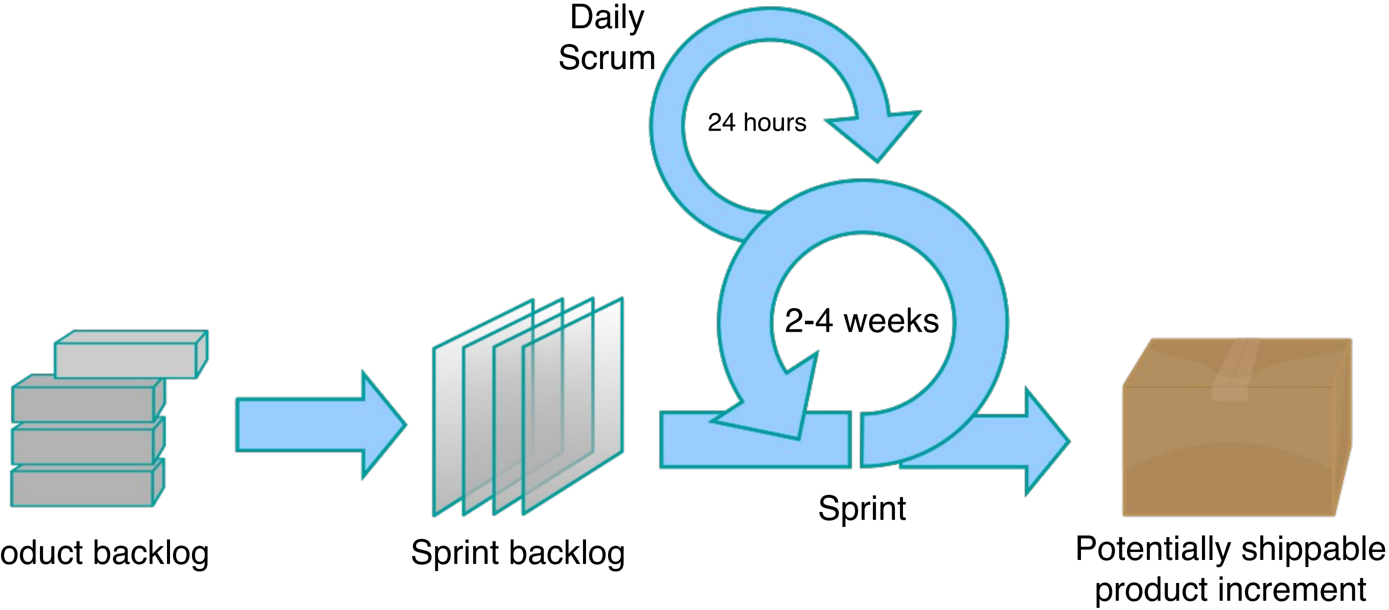
- Original definition (from Rugby): a Scrum is a way to restart the game after an interruption
  - The forwards of each side come together in a tight formation and try to get the ball when it is tossed in among them
- Definition in agile processes: Scrum is a technique that deals with interruptions (change)
  - Manages and controls software and product development with rapidly changing requirements
  - Improves risk management by improved communication, cooperation and the delivery of product increments

Why Scrum?

Linear processes are like relay races (work is performed sequentially)

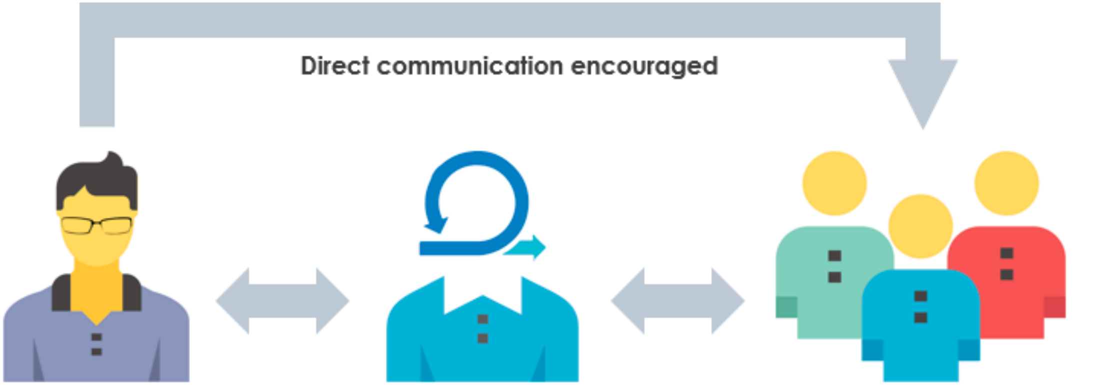
Agile processes are like rugby (work is performed in parallel)

Review: empirical process control model with Scrum



Source: https://commons.wikimedia.org/wiki/File:Scrum_process.svg

Scrum team roles



Direct communication encouraged

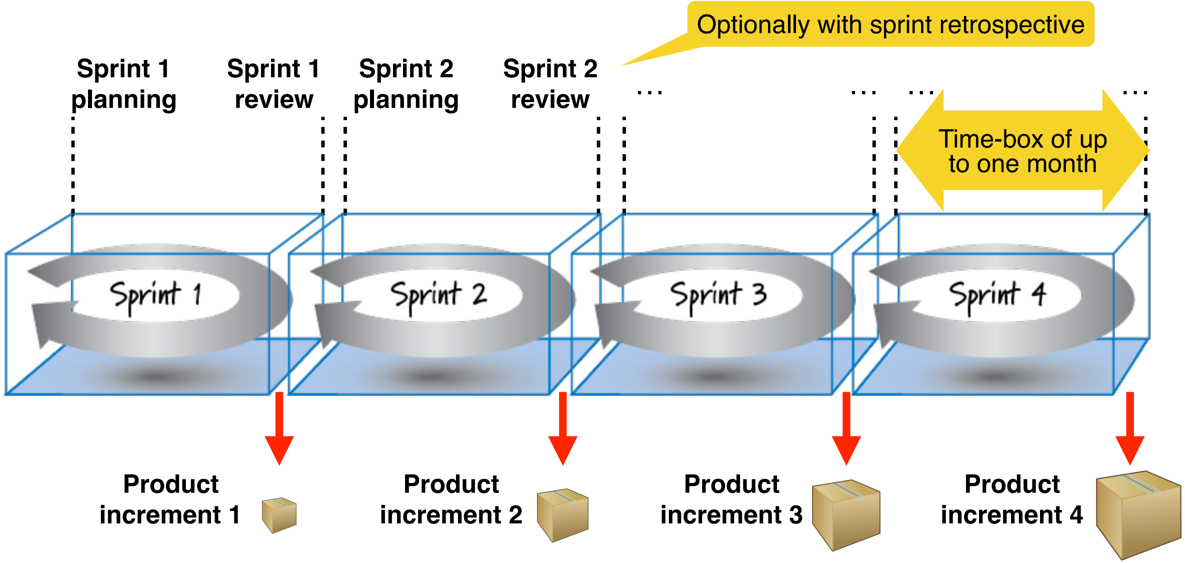
Product Owner:
Owns "what" is desired
And "why" it's desired

Scrum Master:
Keeper of Scrum
Process, facilitator

Scrum Team Members:
Owns "how" and "how quickly"
work is delivered

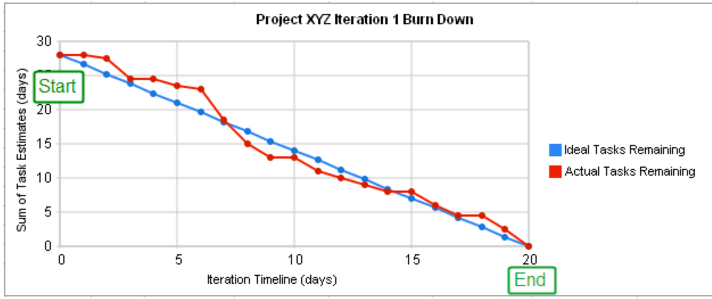
Sprint

- Time-box (up to one month) during which the **team** creates a potentially shippable product increment
- Sprints have **constant durations** throughout a project
- A new sprint starts immediately after the conclusion of the previous sprint



Sprint activities

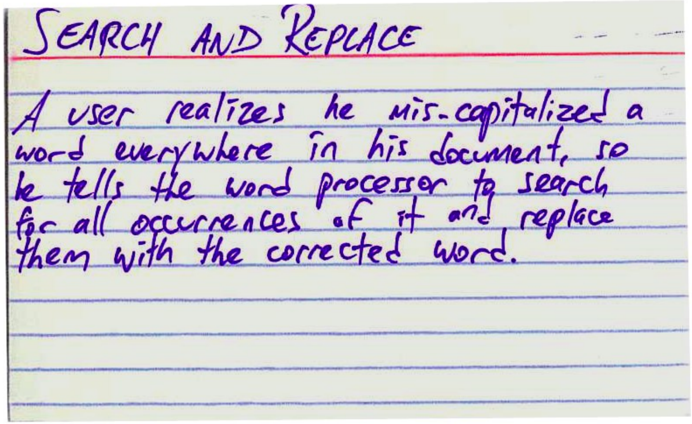
Items	Todo	In	In review	Done
User Story 1	Task	Task	Task	Task
User Interface Design	Task	Task	Task	Task
Scenario 1	Task	Task		



- Development team
  - Realizes the backlog items in the sprint backlog
  - Uses a **task board** to visualize the status of these items
- Scrum master
  - Visualizes the progress in a **burn-down chart**
  - Identifies deviations from the plan and resolves them

User story

- Includes a sentence that describes what the user does or needs
- Often written on a card
- Users (roles) are the actual users of the system



SEARCH AND REPLACE

A user realizes he mis-capitalized a word everywhere in his document, so he tells the word processor to search for all occurrences of it and replace them with the corrected word.

Properties of a good user story: INVEST

- **Independent** - avoid overlapping user stories
- **Negotiable** - a user story is not a contract, but a basis for discussion between development team and product owner
- **Valuable** - for the user and the business; and **vertical**: plan and develop features, not layers
- **Estimable** - the stories in the product backlog represent the basis of the project plan
- **Small** - too large user stories must be partitioned into smaller ones to avoid an over-proportional increase of complexity
- **Testable** - if a user story is not testable, it might not be of real value for the product → this also implies realizability

Product backlog

Problem statement
Project kickoff meeting

Prioritized product backlog

ID	Name	Priority
1	□ Backlog Item 1	Critical
2	□ Backlog Item 2	Critical
3	□ Backlog Item 3	Major
4	□ Backlog Item 4	Minor
5	□ Backlog Item 5	Minor

- Collection of items (typically requirements, e.g. user stories, scenarios, etc.) **prioritized by the product owner**
- The product backlog can always be changed and reprioritized during the projects
- Created based on the problem statement during the **project kickoff meeting** or in the phase before the actual project starts

Sprint planning meeting

Prioritized and estimated product backlog

ID	Name	Priority	Difficulty
1	□ Search for available Pedelecs	Critical	Medium
2	□ Check working radius	Critical	Large
3	□ Reserve available Pedelec	Critical	Small
4	□ Return Pedelec	Major	Small
5	□ Contact colleague	Minor	Medium
6	□ Pass reservation	Minor	Medium
7	□ Report damage	Minor	Large
8	□ Unlock Pedelec	Major	Small

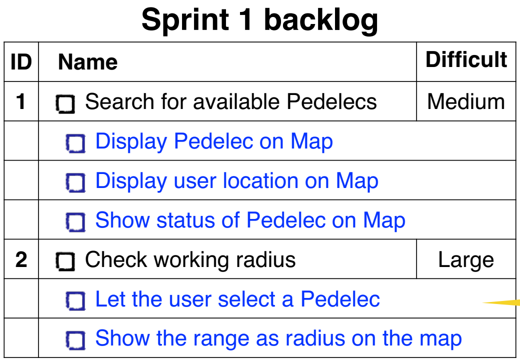
Sprint planning meeting

Sprint 1 backlog

ID	Name	Difficult
1	□ Search for available Pedelecs	Medium
2	□ Check working radius	Large

- The development team **estimates** the difficulty for the most important items in the product backlog
- Development team and product owner select product backlog items that can be realized in the sprint
  - The development team negotiates with the product owner how many items it can realize in the Sprint
  - The product owner defines when an item is accepted
  - The sprint backlog **should not be changed** within the Sprint to protect the team from too many changes

Refinement of the sprint backlog



Sprint 1 backlog

ID	Name	Difficult
1	□ Search for available Pedelecs	Medium
	□ Display Pedelec on Map	
	□ Display user location on Map	
	□ Show status of Pedelec on Map	
2	□ Check working radius	Large
	□ Let the user select a Pedelec	
	□ Show the range as radius on the map	

Backlog item

Task

- The development team needs to discuss the **concrete implementation work** for each backlog item
- The development team creates tasks to realize the backlog items

Sprint review meeting

Sprint 1 backlog

ID	Name	Difficult
1	□ Search for available Pedelecs	Medium
	☑ Display Pedelec on Map	
	□ Display user location on Map	
	□ Show status of Pedelec on Map	
2	☑ Check working radius	Large
	☑ Let the user select a Pedelec	
	☑ Show the range as radius on the map	

Unfinished

Finished

- The development team delivers a **product increment** including realized sprint backlog items
- It demonstrates the product increment to the product owner and key stakeholders
- Product owner and stakeholders provide feedback and decide if the items are realized completely

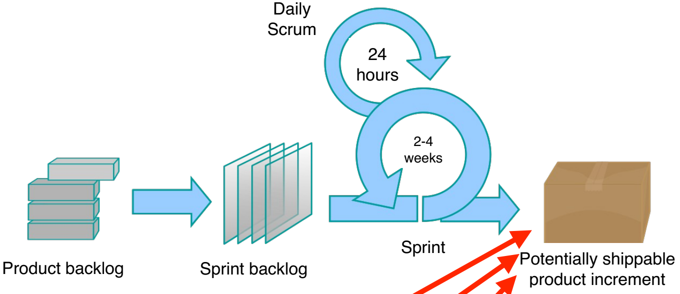
→ Only **completely finished backlog** items count towards this sprint

→ Realized items are ticked off in the sprint backlog

Potentially shippable product increment

- Each sprint focuses on the **incremental** creation of a working system
- The product increment can be thrown away (“falsified”) or delivered

→ The product owner decides



Daily Scrum

24 hours

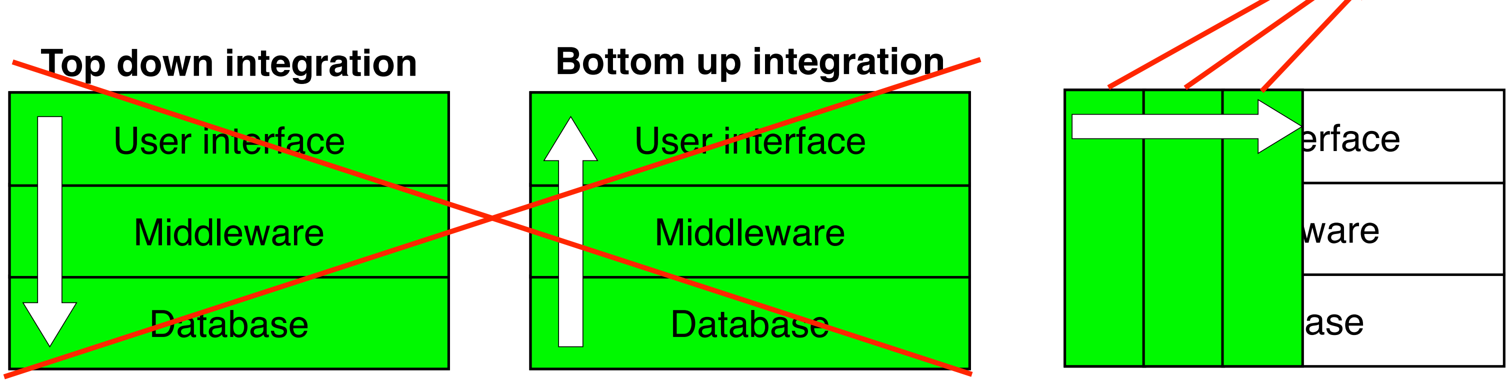
2-4 weeks

Product backlog

Sprint backlog

Sprint

Potentially shippable product increment



Top down integration

User interface

Middleware

Database

Bottom up integration

User interface

Middleware

Database

erface

ware

ase

Horizontal integration (traditional)

Vertical integration (agile)

Sprint review meeting

Sprint 1 backlog

ID	Name	Difficult
1	□ Search for available Pedelecs	Medium
	☑ Display Pedelec on Map	
	□ Display user location on Map	
	□ Show status of Pedelec on Map	
2	☑ Check working radius	Large
	☑ Let the user select a Pedelec	
	☑ Show the range as radius on the map	

Sprint review meeting

Prioritized and estimated product backlog

ID	Name	Priority	Difficulty
1	□ Search for available Pedelecs	Critical	Medium
2	☑ Check working radius	Critical	Large
3	□ Reserve available Pedelec	Critical	Small
4	□ Return Pedelec	Major	Small
5	□ Contact colleague	Minor	Medium
6	□ Pass reservation	Minor	Medium
7	□ Report damage	Minor	Large
8	□ Unlock Pedelec	Major	Small
9	□ Notification after Repair	Minor	?

- **Unrealized** sprint backlog items move back to the product backlog
  - These are candidates for the next sprint
  - Feedback from the product owner (or other stakeholders) is collected
- The product owner can add **new requirements** to the product backlog or change existing ones
- The sprint review meeting can include a **sprint retrospective** (can also be a separate meeting)

Sprint retrospective meeting

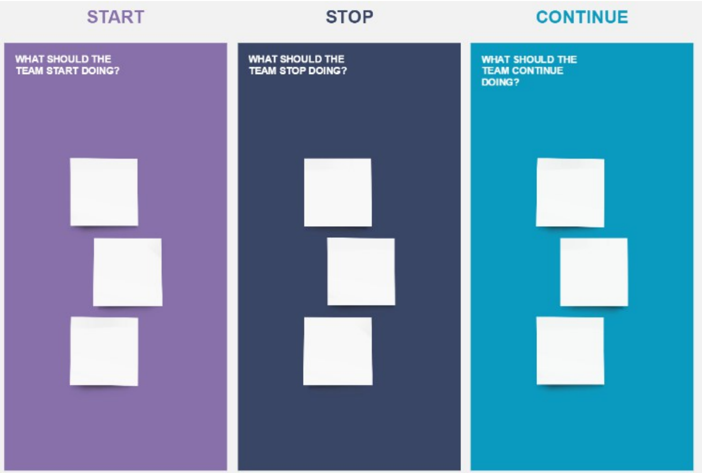
- Scrum master and development team meet to discuss how the previous sprint worked out

  → The product owner can participate

- There are different **retrospective** techniques

  → It is more effective to brainstorm about things that did **not** work well

- Each team member is asked to identify specific things that the team should
  - Start doing
  - Stop doing
  - Continue doing



START

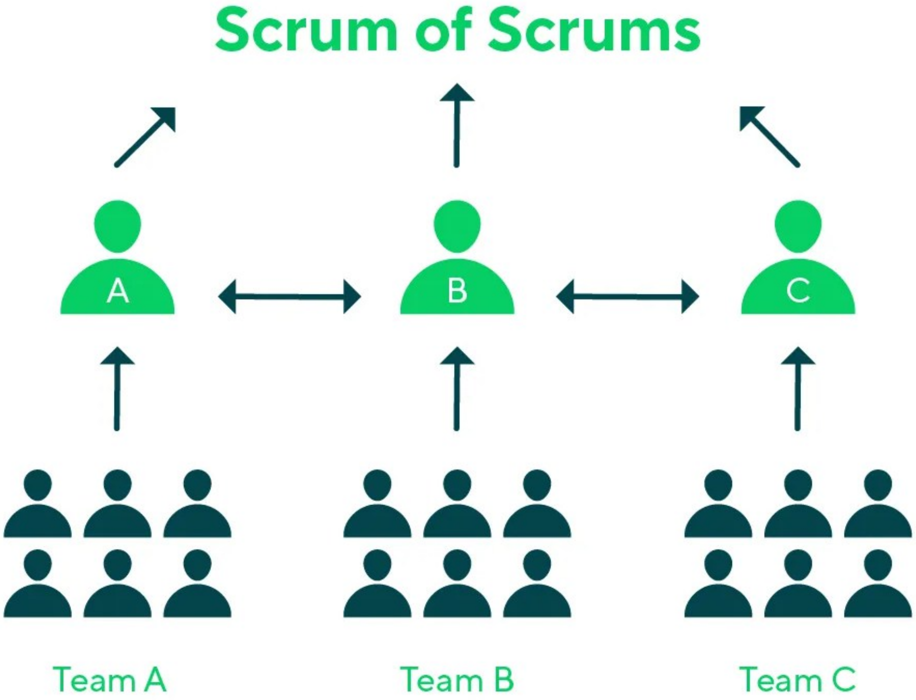
WHAT SHOULD THE TEAM START DOING?

STOP

WHAT SHOULD THE TEAM STOP DOING?

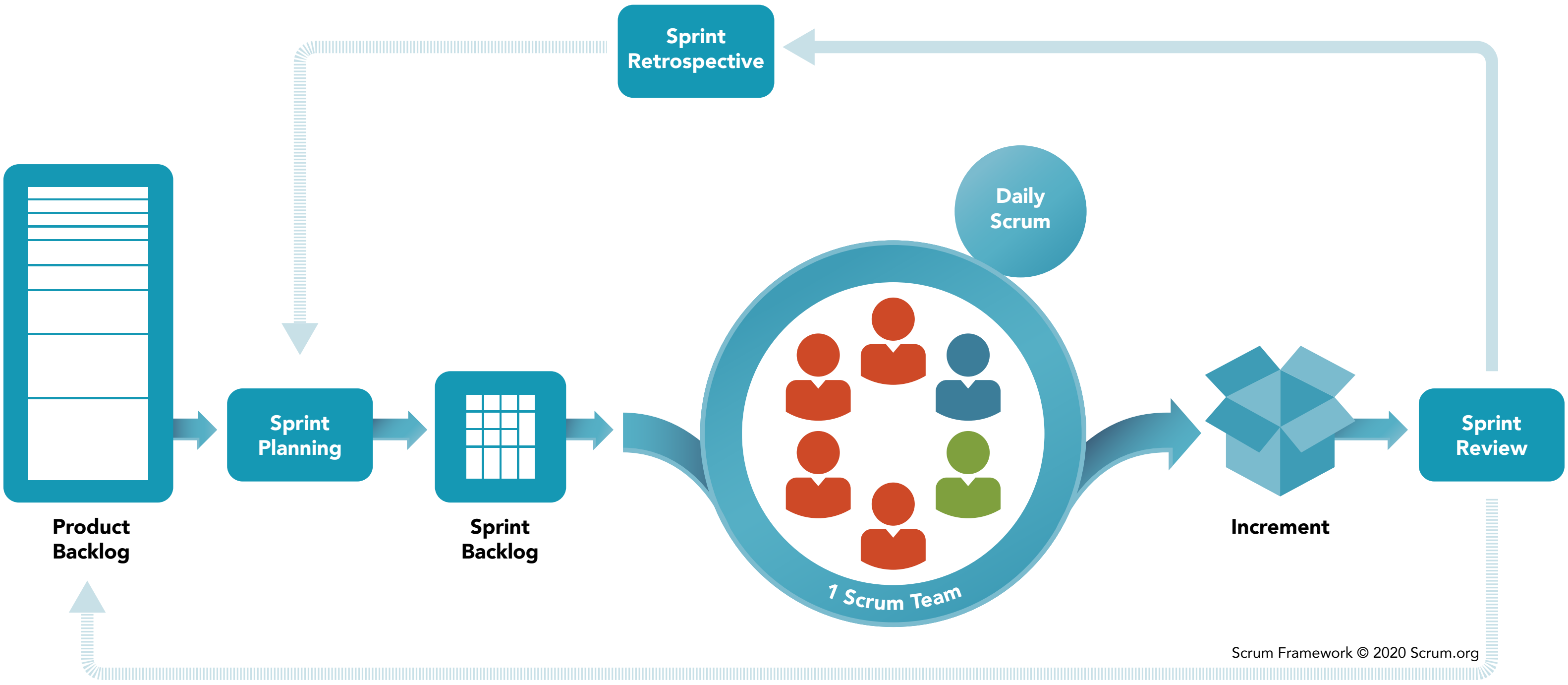
CONTINUE

WHAT SHOULD THE TEAM CONTINUE DOING?

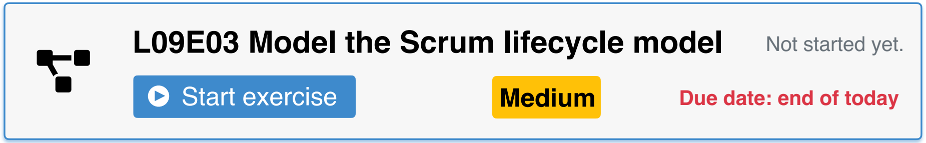


- Cross-team collaboration
- Maximizes the spread of information
- Encourages consensus
- Problem-solving platform
- **Important:** team members should also communicate directly

Scrum summary



Scrum Framework © 2020 Scrum.org



L09E03 Model the Scrum lifecycle model

Not started yet.

Due date: end of today

10 min

6 pts

- **Problem Statement:** Model the Scrum meetings as actions and the Scrum artifacts as objects using a UML activity diagram. Explain your solution!
  - **Meetings:** project kickoff meeting, sprint planning meeting, daily Scrum meeting, sprint review meeting, sprint retrospective
  - **Artifacts:** product backlog, sprint backlog, potentially shippable product increment
- **Hints**
  - Make sure to include a start and an end node and decision nodes where necessary
  - Make sure to model the iterative and incremental nature of Scrum

Outline

- Software development as application domain
- UML activity diagrams
- Waterfall model
- V-model
- Spiral model
- Unified process
- Limitations of linear and iterative models
- Extreme programming
- Scrum

➡ **Kanban**

Kanban

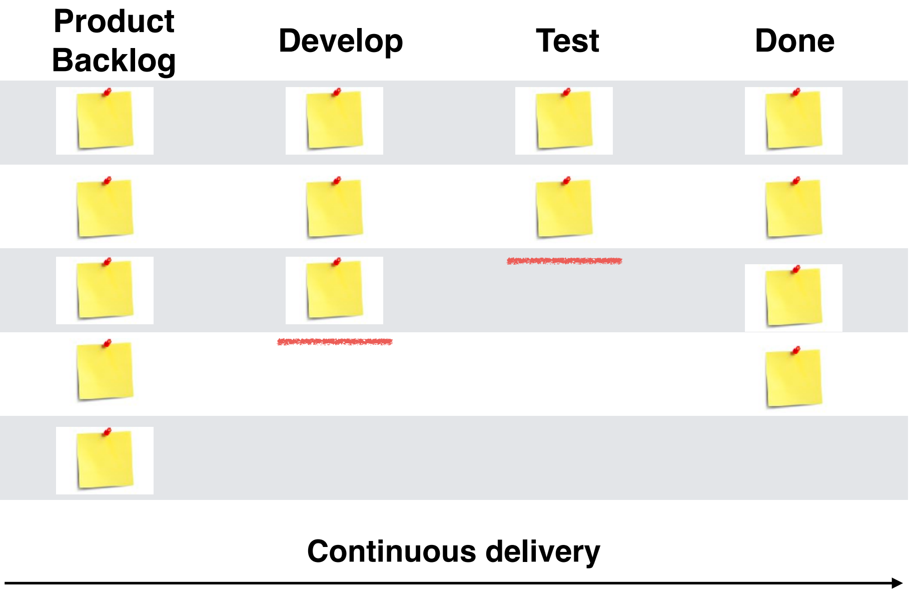
- An agile model for software development

Four basic principles

1) Start with **existing** process
2) Agree to pursue **incremental**, **evolutionary** change
3) **Respect** the current process, roles, responsibilities and titles
4) **Leadership** at all levels

Kanban: pull-based evolutionary agile methodology

- Continuous pull-based workflow
- Visualize your work
- Kanban board
  - Kanban cards
  - One card per work item
  - Put the card in the appropriate stage
  - The cards move from left to right
  - Limit WIP (work in progress) at each stage



Scrum vs. Kanban

	Scrum	Kanban
Ideology	Learn through experiences, self-organize and prioritize, and reflect to improve	Use visuals to improve work-in-progress
Cadence	Regular, fixed-length sprints (i.e. two weeks)	Continuous flow
Practices	Sprint planning, sprint, daily scrum, sprint review, sprint retrospective	Visualize the flow of work, limit work-in-progress, manage flo
Roles	Product owner, scrum master, development team	No required roles

Homework

- **H09E01** Advantages and Disadvantages of Scrum (text exercise)
- **H09E02** Model a UML Activity Diagram (modeling exercise)
- Read more about the **Kanban** on https://kanban.university/kanban-guide

→Due until 1h before the **next lecture**

Summary

- Software development lifecycle: set of activities and their relationships to each other to support the development of a software system
  - Software development lifecycles can be modeled as complex systems
- Software development lifecycle model: an abstraction representing the development of software for understanding, monitoring and controlling it
- Different types of models: linear, iterative, agile, activity oriented and entity oriented lifecycle models
- Choice of lifecycle model: influenced by the frequency of change
- Process control model: distinction between a defined process control model and an empirical process control model

Literature

- IEEE 1074-2006: Standard for Developing a Software Project Lifecycle Process, 2006
- W. Humphrey, Managing the Software Process, SEI Series in Software Engineering, Addison Wesley, 1989
- W. Royce, Managing the Development of Large Software Systems, Proceedings of the IEEE WESCON, August 1970, pp. 1-9
- Jenson & Tonies, Software Engineering, Prentice Hall, May 1979
- B. Boehm, A Spiral Model of Software Development and Enhancement, ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24, 1986
- I. Jacobson, J. Rumbaugh and G. Booch, The unified software development process, Addison-Wesley, 1999.
- Ken Schwaber and Mike Beedle - Agile Software Development with Scrum, Prentice Hall, October 2001
- Agile Manifesto - http://agilemanifesto.org
- https://kanban.university/kanban-guide
- The Scrum guide: https://scrumguides.org/scrum-guide.html
- Scrum methodology: https://www.wrike.com/scrum-guide/scrum-methodology