TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Industrial Software Engineering-

Slides

03.2020

Prof.‘in Dr. Jessica Rubart & Prof. Dr. Robert Mertens
Industrial Software Engineering
Summer term 2020

Organizational Issues

Contact

Prof.in Dr. Jessica Rubart Business Information Systems	
Phone	+49 (0)5271 / 687-7870
E-Mail	jessica.rubart@hs-owl.de

Prof. Dr. Robert Mertens, HSW Anwendungsentwicklung und Medieninformatik	
Phone	+49 (0)5151 95-59-36
E-Mail	mertens@hsw-hameln.de

Organizational Issues

Course

Event	Day	Room	Time
Lecture	Friday	Virtual or 55.015	09.40 – 11.20
Exercises	Friday	Virtual or 55.015 / 1.344	11.35 – 13.15

- **Optional course**
- **5 ECTS credits**

- **“Ausarbeitung”**

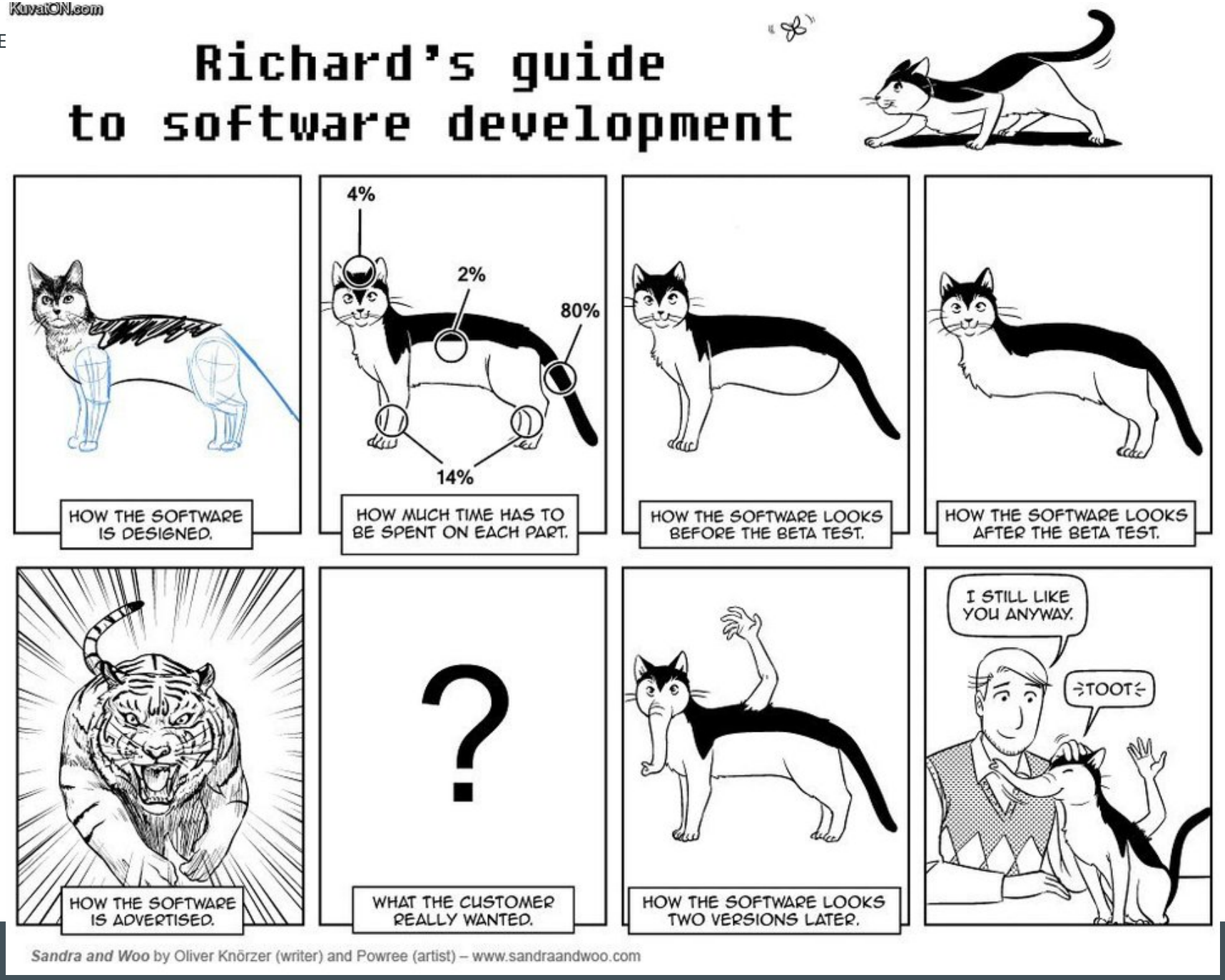
TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Introduction

Round of Introductions

Effort Estimation
Test-Driven Development
Process Models
System Analysis
Business Change Request
Scrum
Software Patterns
Software Architectures
Industrial Software Engineering
User Stories
MDA
Testing
Background
SIX SIGMA
Quality Management
SOA
RUP
Expectations
Service-Oriented Architecture
Project Management
Work Packages
Model-Driven Architecture
Unified Process
Use Cases
Component-Oriented Development
Agile Methods
Kanban
BPMN
Software Evolution
Customer
Project Plan
Requirements Management
Reuse
Prototyping
Business Analysis
Controlling

Motivation


Richard's guide
to software development
HOW THE SOFTWARE IS DESIGNED.
4%
2%
80%
14%
HOW MUCH TIME HAS TO BE SPENT ON EACH PART.
HOW THE SOFTWARE LOOKS BEFORE THE BETA TEST.
HOW THE SOFTWARE LOOKS AFTER THE BETA TEST.
HOW THE SOFTWARE IS ADVERTISED.
?
WHAT THE CUSTOMER REALLY WANTED.
HOW THE SOFTWARE LOOKS TWO VERSIONS LATER.
I STILL LIKE YOU ANYWAY.
TOOT
Sandra and Woo by Oliver Knörzer (writer) and Powree (artist) – www.sandraandwoo.com

Contents

Overview

TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Rational Unified Process

Rational Unified Process

RUP

- **Product of *Rational Software* (IBM)**
  - Originally established through the „three amigos“ Ivar Jacobson, Grady Booch and James Rumbaugh who also developed the UML (Unified Modeling Language)
  - Industry standard
  - Process framework for software development, which needs to be adapted to the concrete application setting
  - Iterative Development is one of the *Best Practices*
  - Main Characteristics:

    Iterative, Use-Case-driven, Architecture-centric, Risk-oriented

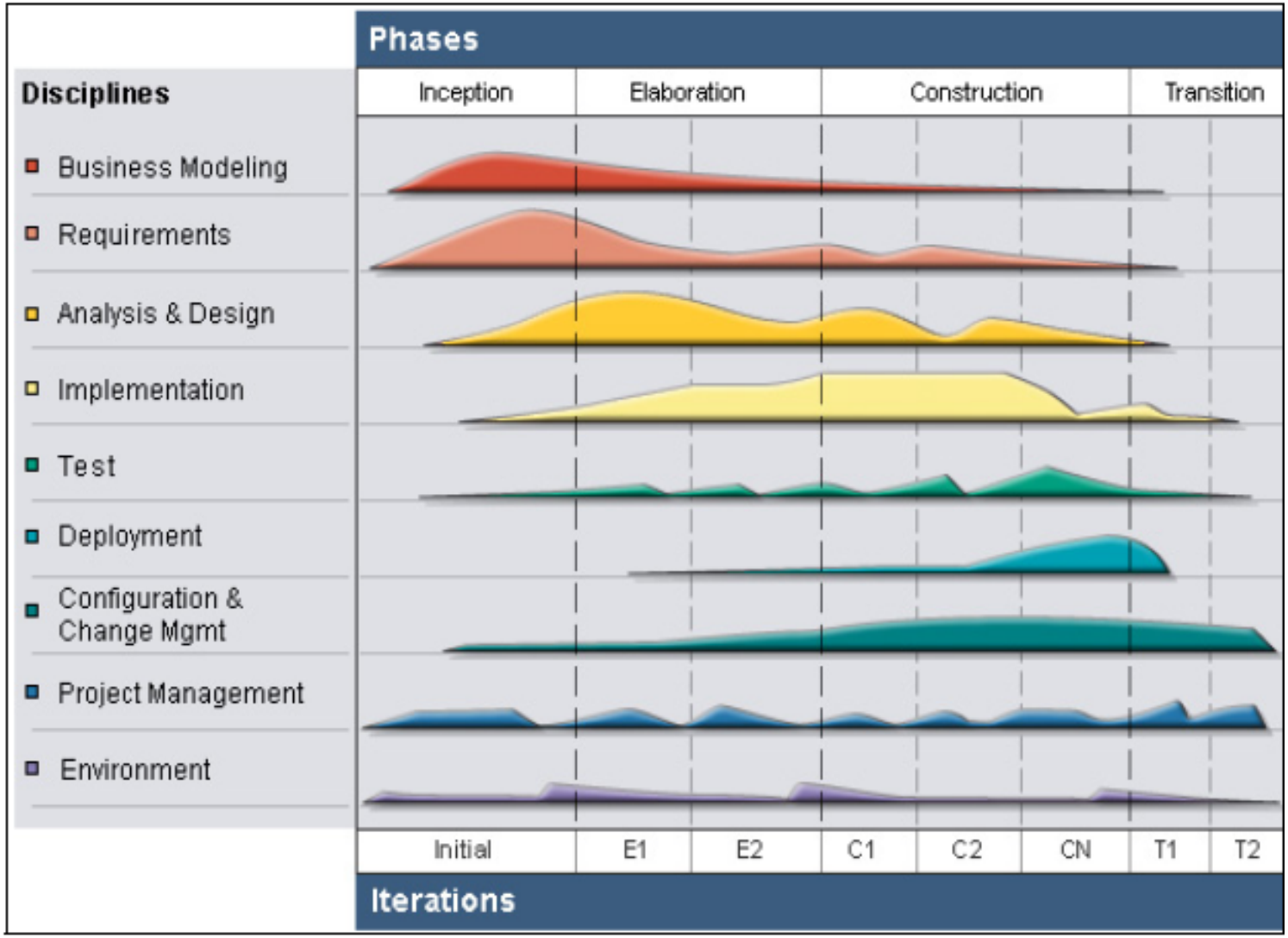
Rational Unified Process

Best Practices

- Develop Iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

Rational Unified Process

Iterative Development



© IBM

Rational Unified Process

Iterative Development

- **Primary Aims:**
  - Minimizing risks with respect to a shared understanding, technical feasibility, acceptance of the customer
  - Making changes more manageable
  - Higher level of reuse
  - The project team can learn along the way (improvement process)
  - Better overall quality

Rational Unified Process

Phases and Milestones

- **Inception Phase: Objectives**
  - Establish project scope and boundary conditions
  - Determine the use cases and primary scenarios
  - Demonstrate a candidate architecture against some of the scenarios
  - Identify potential risks
  - Estimate the overall cost and schedule
  - Prepare the supporting environment for the project

Rational Unified Process

Phases and Milestones

- **Inception Phase: Evaluation Criteria**
  - Stakeholder agreement on
    - Scope definition, cost/schedule estimates, set of requirements, shared understanding of these requirements, risk management, and development process
  - Milestone: *Lifecycle Objective*

Rational Unified Process

Phases and Milestones

- **Elaboration Phase: Objectives**
  - Detail the use case model with additional requirements
  - Identify use cases with high risk
  - Define, validate and baseline the architecture
  - Refine support environment
  - Baseline a detailed plan for the Construction phase
  - Demonstrate that the baseline architecture will support the project vision at a reasonable cost in a reasonable period of time

Rational Unified Process

Phases and Milestones

- **Elaboration Phase: Evaluation Criteria**
  - Project vision and requirements are stable
  - Architecture is stable
  - Major risks have been addressed and resolved
  - Iteration plans for Construction phase are of sufficient detail
  - Stakeholder agreement on
    - Current vision can be met in the context of the current architecture

  - Milestone: *Lifecycle Architecture*

Rational Unified Process

Phases and Milestones

- **Elaboration Phase: Addressing risks example**

  Which of the given use cases would you select in the following scenario to cover the given risks?

  A supermarket needs to analyze different kinds of data:

Use Cases
Analyze Effectiveness of Market Campaigns
Analyze Stock Turnover Rate in a District
Analyze Stock Turnover Rate in a Store
Archive Logistics Data After One Year
Identify Hourly Staffing Needs for a Store
Identify Sales Trends By District
Archive Store Data After Three Months

Risks to address:

- Volume of store data
- Quality of store data

Rational Unified Process

Phases and Milestones

- **Construction Phase: Objectives**
  - Complete the product for transition to production
  - Detailed development and testing of the system components
  - Minimize development costs by optimizing resources and avoiding unnecessary scrap and rework
  - Achieve adequate quality
  - Achieve useful versions (alpha, beta, test releases, …)

Rational Unified Process

Phases and Milestones

- **Construction Phase: Evaluation Criteria**
  - Is this product release stable and mature enough to be deployed in the user community?
  - Are all stakeholders ready for the product's transition?
  - Are planned resources available?

  - Milestone: *Initial Operational Capability*

Rational Unified Process

Phases and Milestones

- **Transition Phase: Objectives**
  - Achieve stakeholder agreement that deployment baselines are complete and consistent with the evaluation criteria of the vision
  - Integration tests with the customer
  - Training courses for the customer
  - Data conversion
  - Achieve user self-supportability
  - Achieve final product baseline

Rational Unified Process

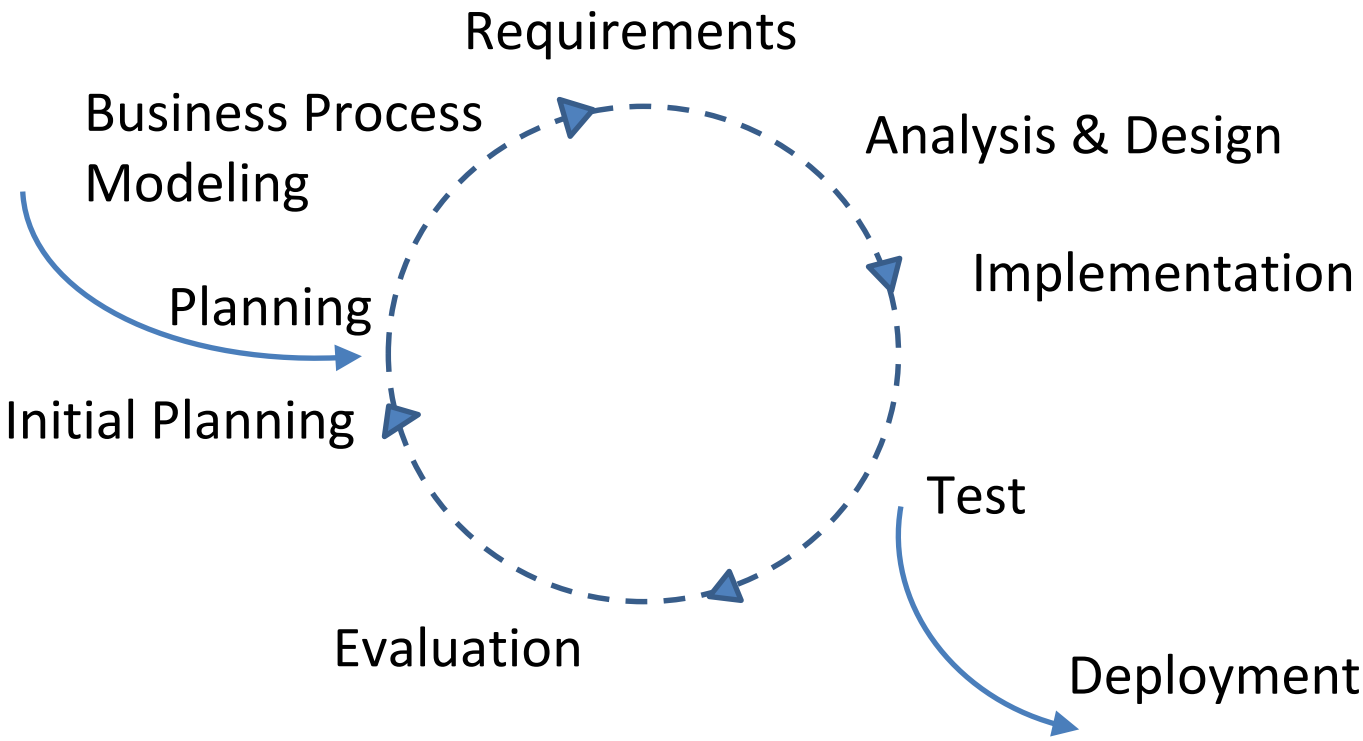
Phases and Milestones

- **Transition Phase: Evaluation Criteria**
  - Is the user satisfied?
  - Are the actual resource efforts acceptable?

  - Milestone: *Product release*

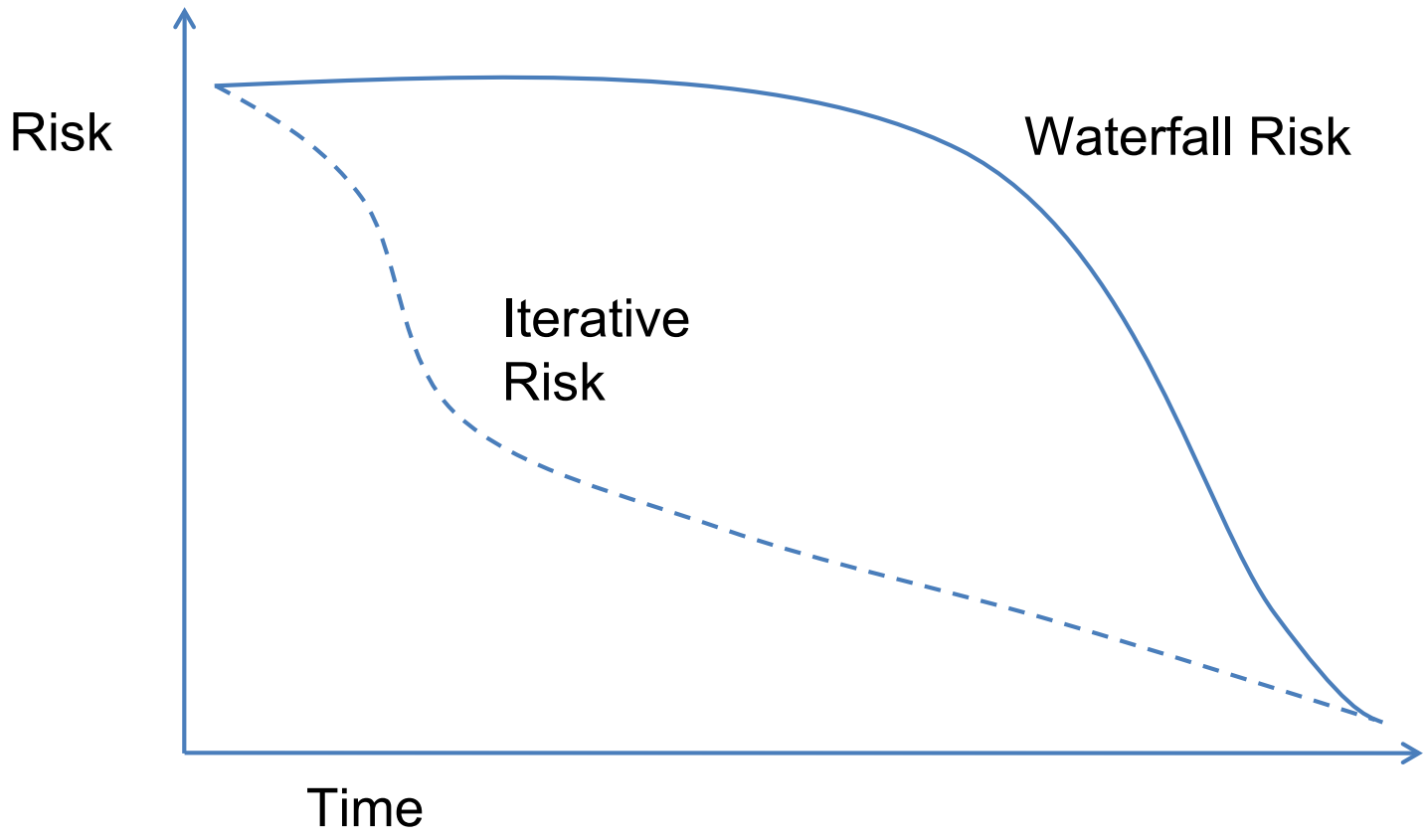
Rational Unified Process

An Iteration



Rational Unified Process

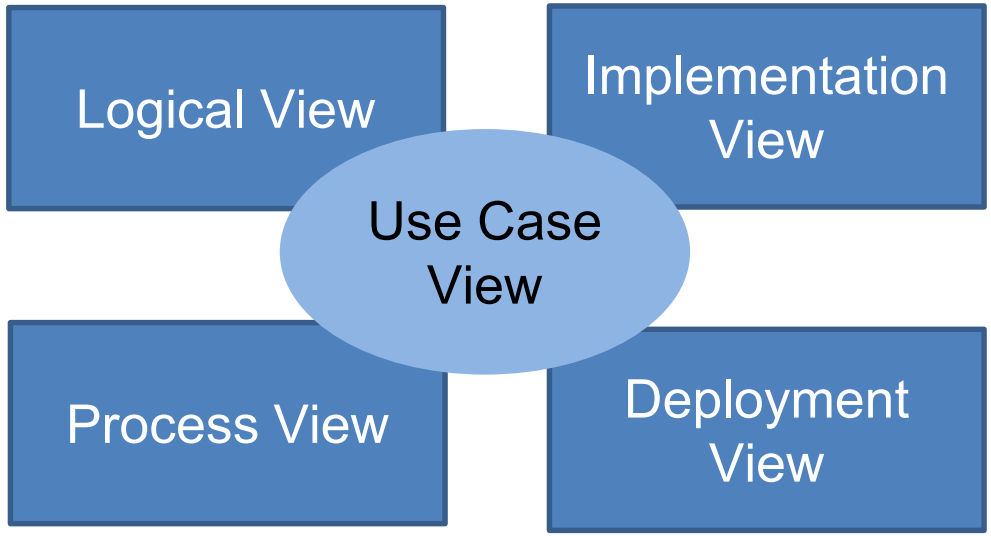
Risk Profiles



Rational Unified Process

4 + 1 Architecture Views

- Provide multiple views of the architecture to discuss with different stakeholders
- The use case view contains a few key use cases that are used to drive and validate the architecture



Compare Kruchten

Rational Unified Process

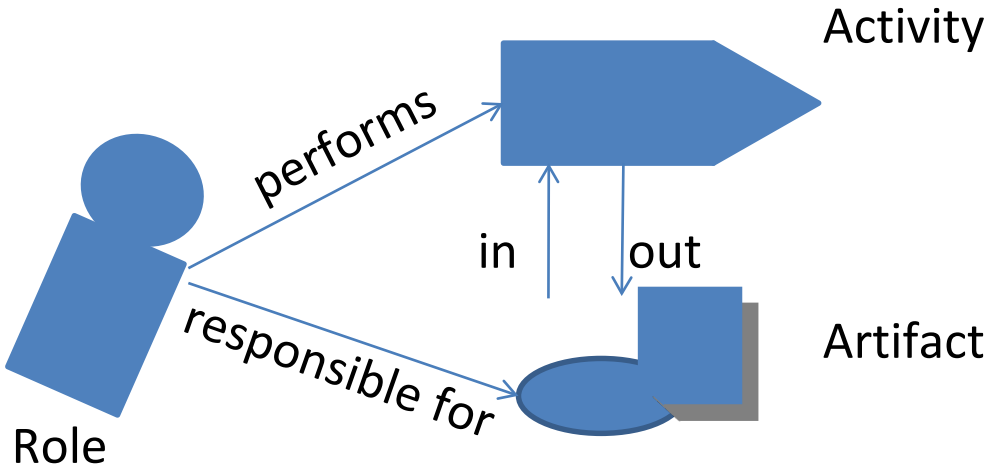
4 + 1 Architecture Views

- **Logical View:**
  - Addresses the conceptual structure of the system identifying major design packages, subsystems, and classes.
- **Implementation View:**
  - Describes the organization of software modules in the development environment, e.g. packaging, layering, configuration management.
- **Deployment View:**
  - Shows how the various executables and other run-time components are mapped onto the underlying computing nodes.
- **Process View:**
  - Addresses the concurrent aspects of the system at run-time, e.g. tasks, threads or processes, and their interactions.

Rational Unified Process

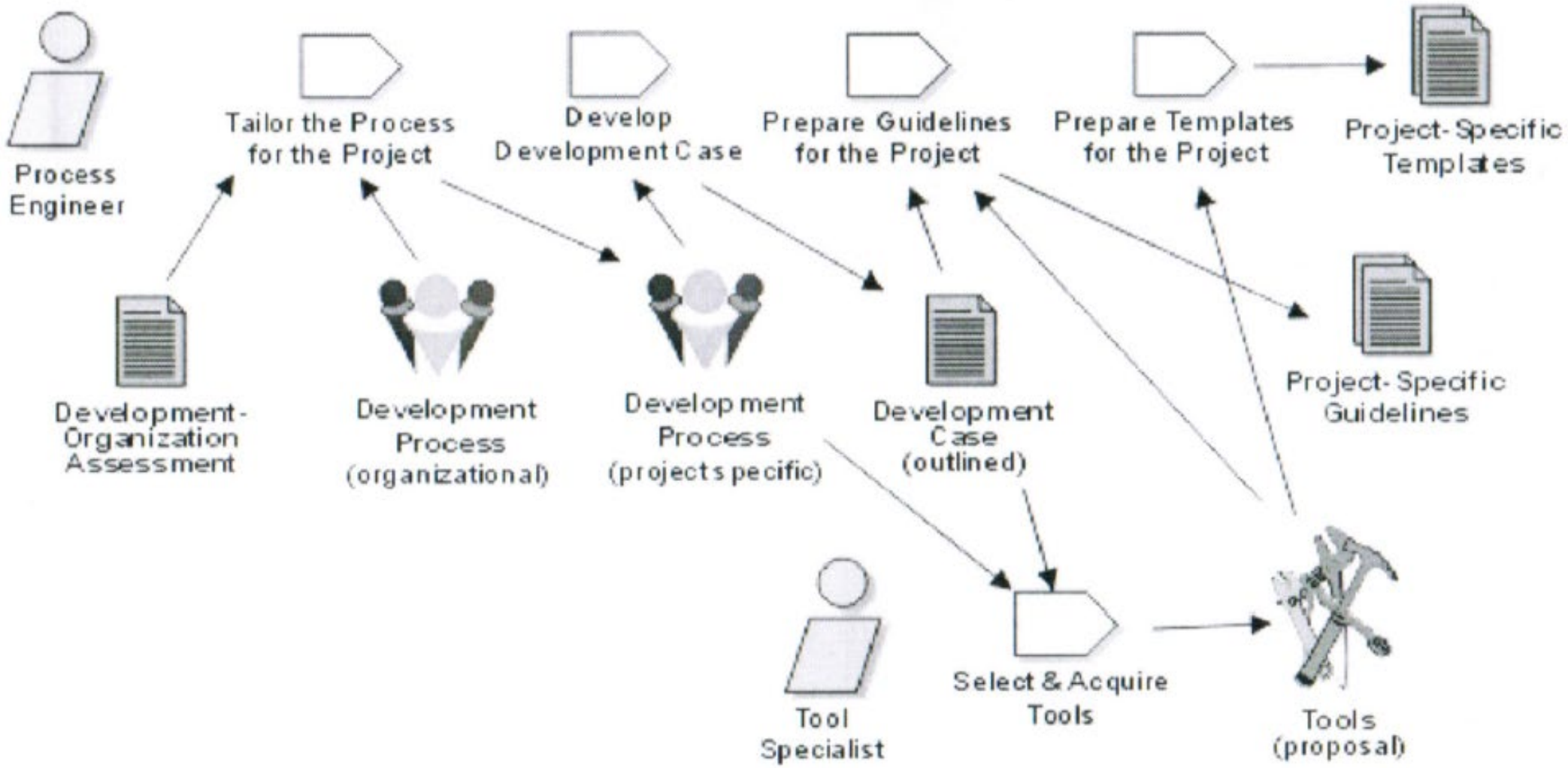
Workflow Descriptions

- The RUP is represented using four primary modeling elements:
  - Roles / Workers, the “who”
  - Activities, the “how”
  - Artifacts, the “what”
  - Workflows, the “when”



Rational Unified Process

Workflow Example



© IBM: Workflow *Prepare Environment for Project*

Rational Unified Process

Main Roles – Brainstorming

Rational Unified Process

Main Roles

- **Business Modeling:**
  - *Business Process Analyst*: Discovers all business use cases
  - *Business Designer*: Details a single set of business use cases
- **Requirements:**
  - *Systems Analyst*: Discovers all requirement/system use cases
  - *Requirements Specifier*: Details a single set of requirement use cases
- **Analysis and Design:**
  - *Software Architect*: Decides on technologies for the whole solution
  - *Designer*: Details the analysis and design for a single set of use cases
- **Implementation:**
  - *Integrator*: Owns the build plan that shows what classes will integrate with one another
  - *Implementer*: Codes a single set of classes or class operations

Rational Unified Process

Main Roles

- **Test:**
  - *Test Manager*: Ensures that testing is complete and conducted for the right motivators
  - *Test Analyst*: Selects what to test based on the motivators
  - *Test Designer*: Decides which tests should be automated or manual and implements automations
  - *Tester*: Runs a specific test
- **Deployment:**
  - *Deployment Manager*: Oversees deployment for all deployment units
  - *Tech Writer / Course Developer / Graphic Artist*: Create detailed materials to ensure a successful deployment

Rational Unified Process

Main Roles

- **Project Management:**
  - *Project Manager:* Creates the business case and the project plan, makes go / no-go decisions, plans, tracks, and manages each iteration
- **Environment:**
  - *Process Engineer*: Owns the process for the project
  - *Tool Specialist*: Creates guidelines for using a specific tool
- **Configuration and Change Management:**
  - *Configuration Manager*: Sets up the CM environment, policies, and plan, creates a deployment unit, reports on configuration status
  - *Change Control Manager*: Establishes a change control process, reviews and manages change requests

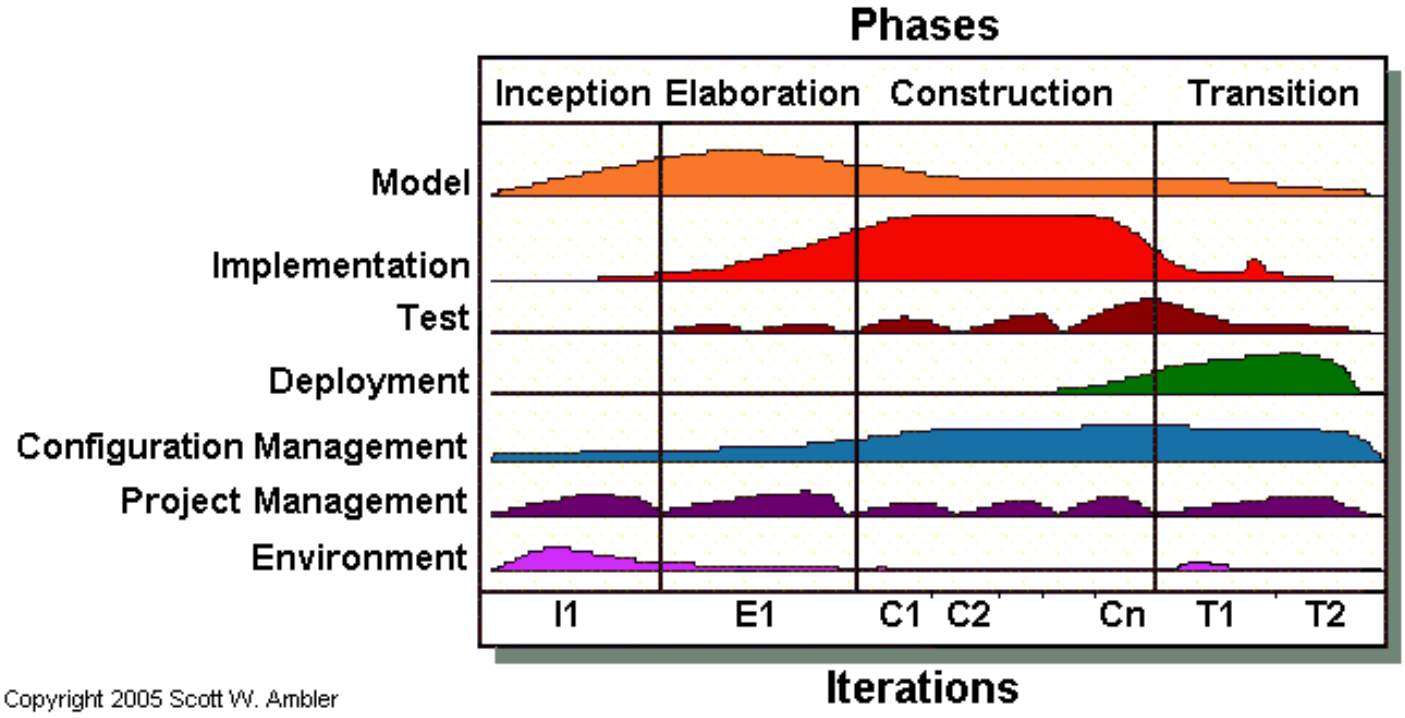
Rational Unified Process

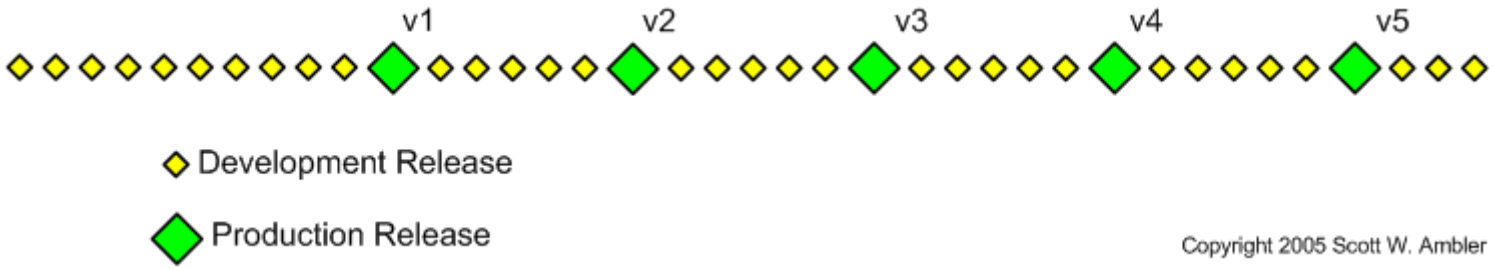
Exercises

- **Investigate in small groups about the following variations of the RUP and prepare a small talk about it, which you then present in the end of this exercise:**
  - *Agile Unified Process* (AUP), a lightweight variation developed by Scott W. Ambler: http://www.ambysoft.com/unifiedprocess/agileUP.html
  - *The Enterprise Unified Process* (EUP), an extension of the Rational Unified Process by Scott Ambler + Associates Services: http://www.enterpriseunifiedprocess.com/
    - *The Disciplined Agile (DA) process decision toolkit,* https://www.pmi.org/disciplined-agile/introduction-to-disciplined-agile
  - *Open Unified Process* (OpenUP), the Eclipse Process Framework software development process: http://www.eclipse.org/epf/general/getting_started.php
  - *Essential Practices* developed by Ivar Jacobsen: http://www.ivarjacobson.com/Practices/

Rational Unified Process

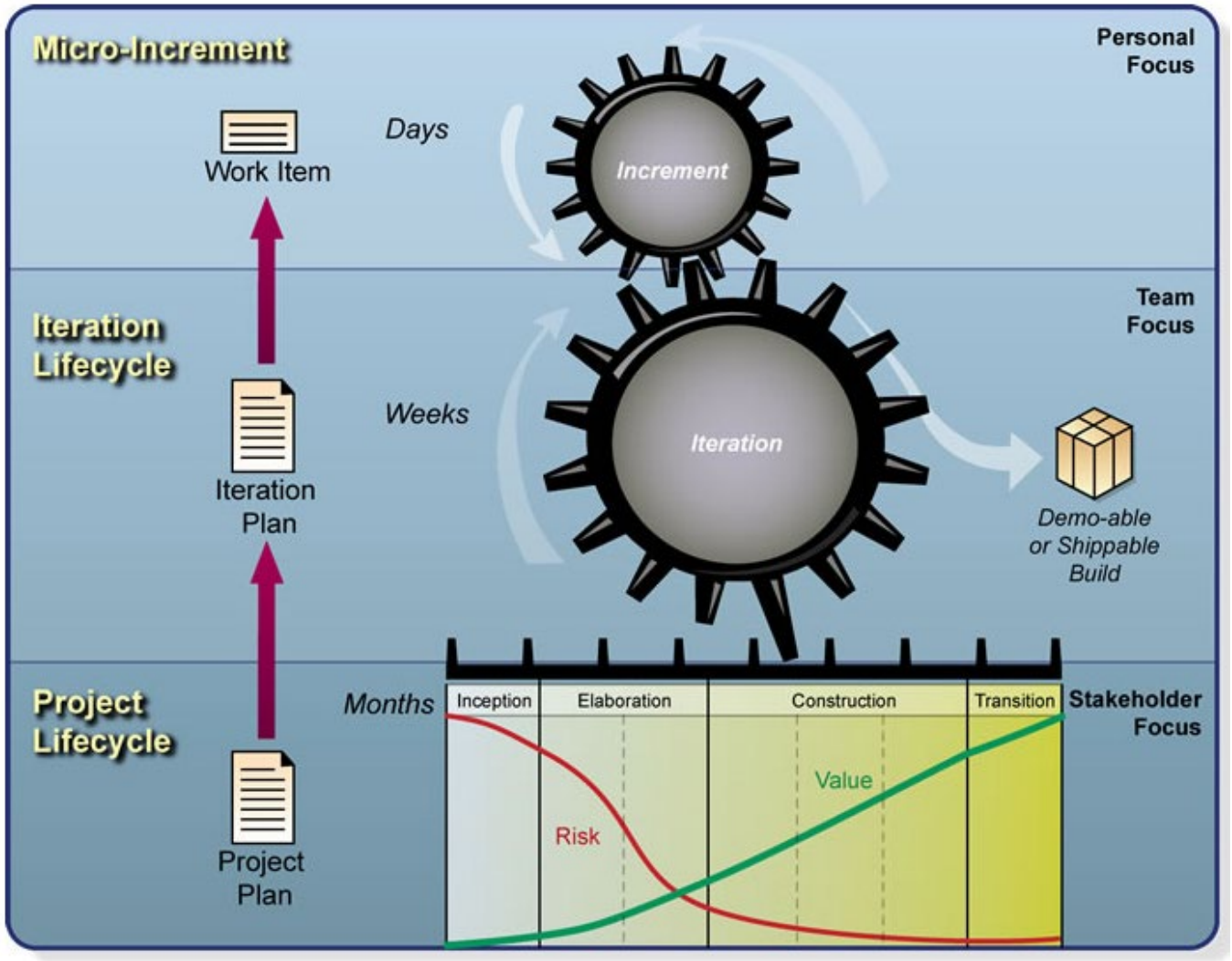
Exercises: *Agile UP*





Rational Unified Process

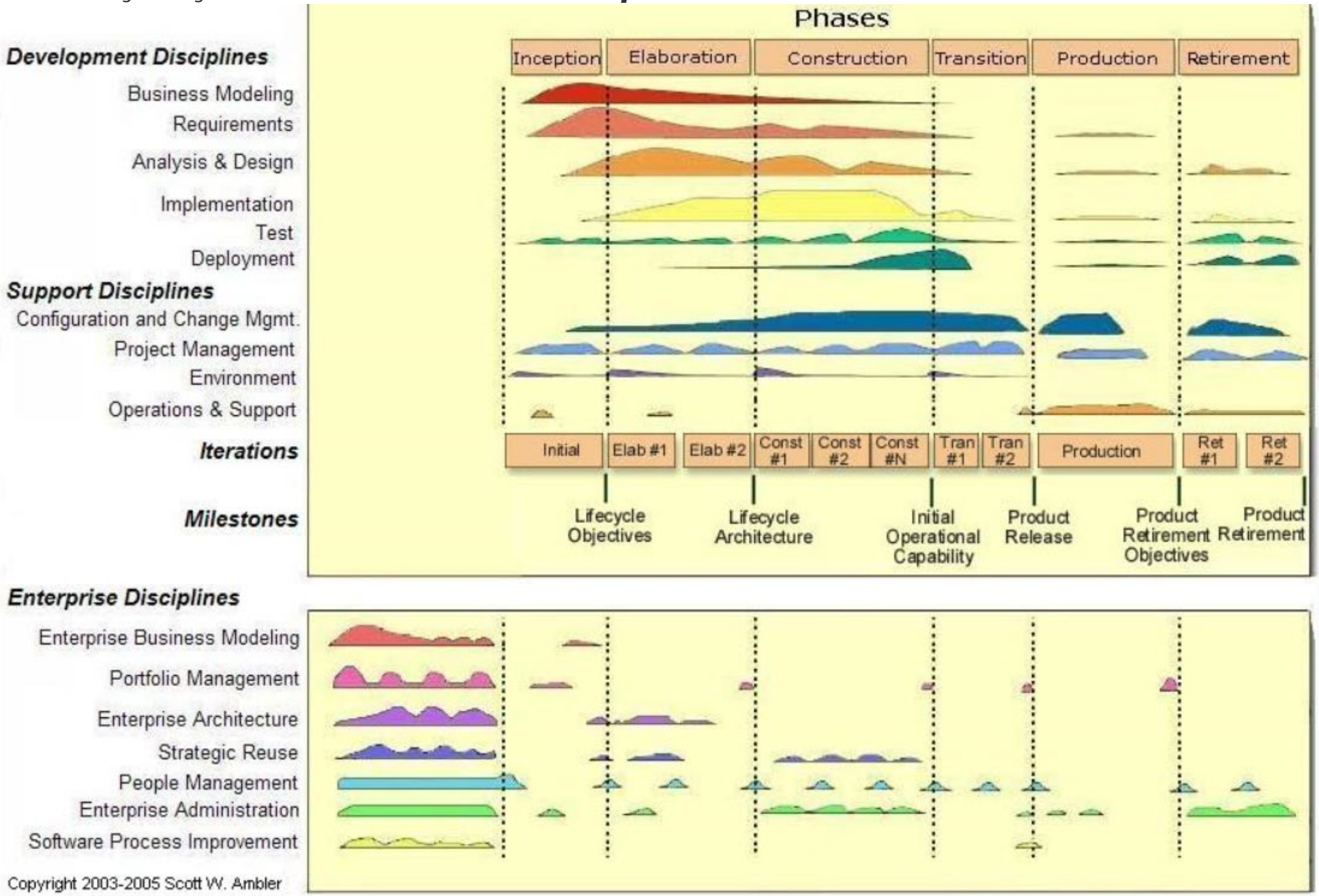
Exercises: *Open UP*



© Eclipse Process Framework

Rational Unified Process

Exercises: *Enterprise UP*



TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Requirements Engineering

Requirements Engineering

RUP Best Practice: Manage Requirements

Requirements Management means …

Making sure you

- Solve the right problem
- Build the right system

By taking a systematic approach to

- Eliciting
- Organizing
- Documenting
- Managing

The changing requirements of a software application

Requirements Engineering

RUP Best Practice: Manage Requirements

Aspects of Requirements Management …

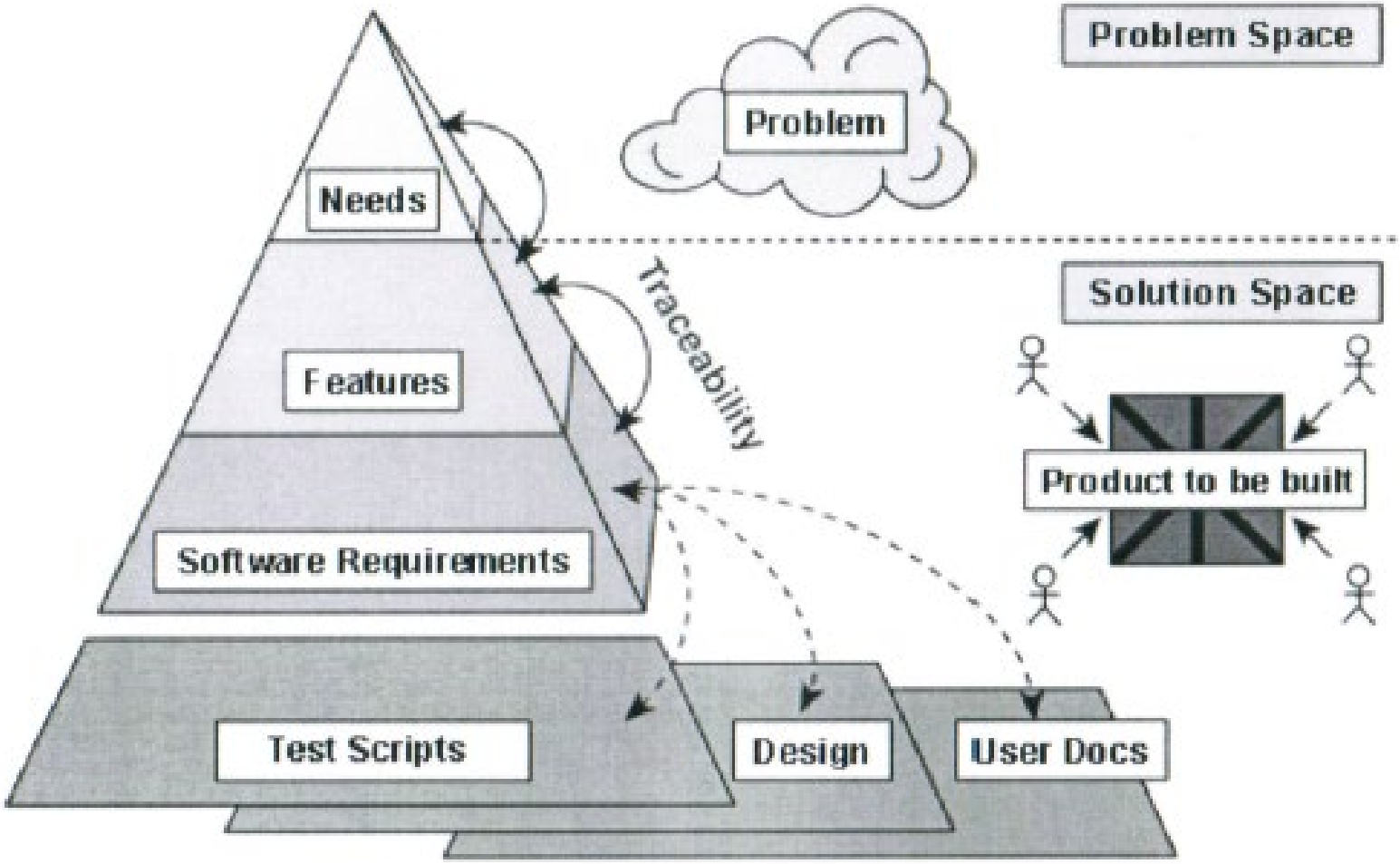
- Analyze the problem
- Understand user needs
- Define the system
- Manage scope
- Refine the system definition
- Manage changing requirements

- Requirements are driven by business processes

Requirements Engineering

RUP Best Practice: Manage Requirements

Traceability



© IBM

Requirements Engineering

RUP Best Practice: Manage Requirements

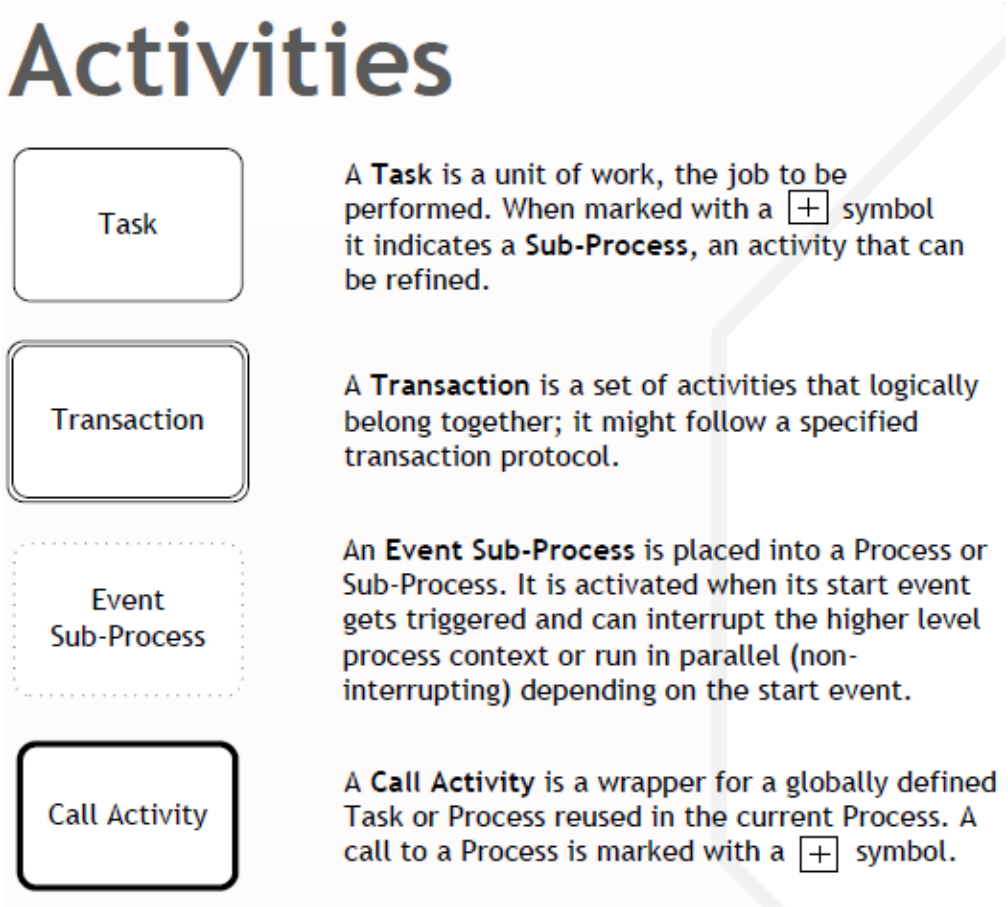
Traceability allows you to …

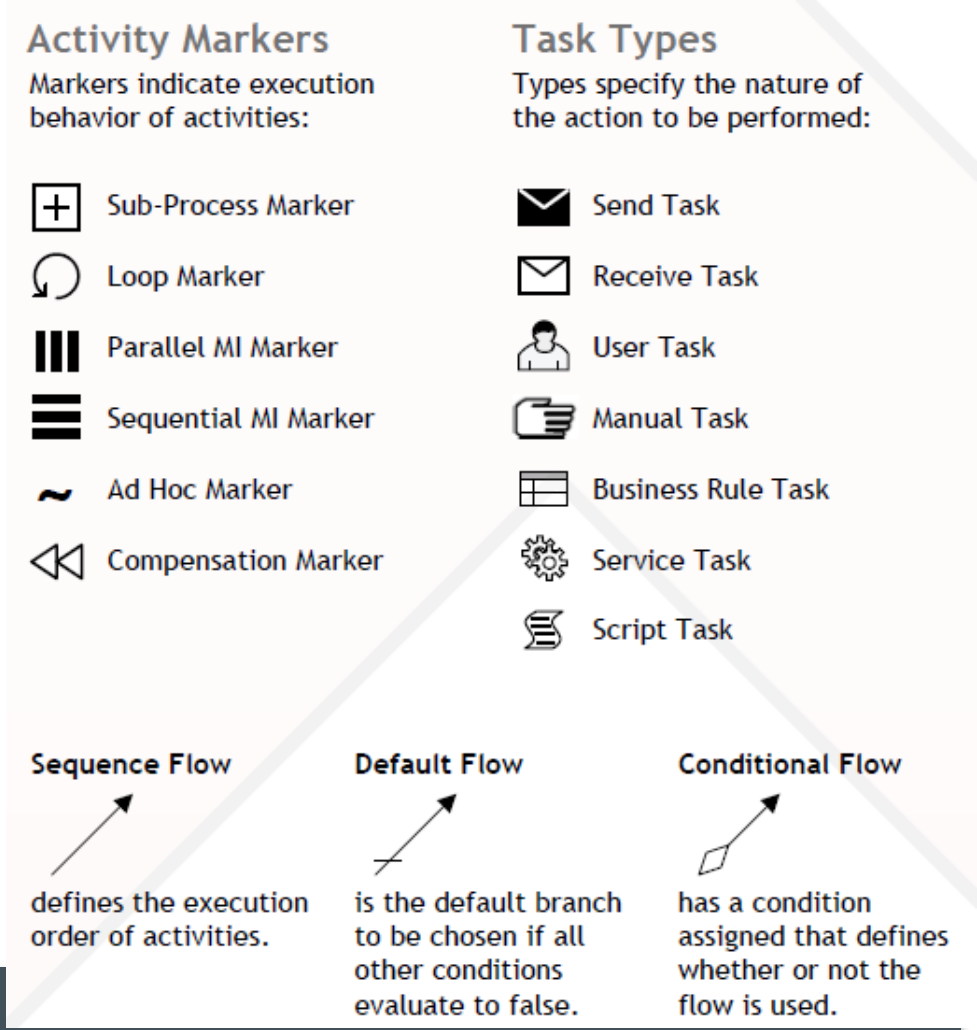
- Assess the project impact of a change in a requirement
- Assess the impact of a failure of a test on requirements
- Manage the scope of the project
- Verify that all requirements are fulfilled by the implementation
- Verify that the application does only what is was intended to do
- Manage change

Requirements Engineering

BPMN - Business Process Model and Notation

- A graphical notation to understand business processes
- Supports communication and understanding between stakeholders





BPM Offensive Berlin:
http://www.bpmb.de/index.php/BPMNPoster

Requirements Engineering

BPMN - Business Process Model and Notation

Gateways

Exclusive Gateway

When splitting, it routes the sequence flow to exactly one of the outgoing branches. When merging, it awaits one incoming branch to complete before triggering the outgoing flow.

Event-based Gateway

Is always followed by catching events or receive tasks. Sequence flow is routed to the subsequent event/task which happens first.

Parallel Gateway

When used to split the sequence flow, all outgoing branches are activated simultaneously. When merging parallel branches it waits for all incoming branches to complete before triggering the outgoing flow.

Inclusive Gateway
When splitting, one or more branches are activated. All active incoming branches must complete before merging.

Exclusive Event-based Gateway (instantiate)
Each occurrence of a subsequent event starts a new process instance.

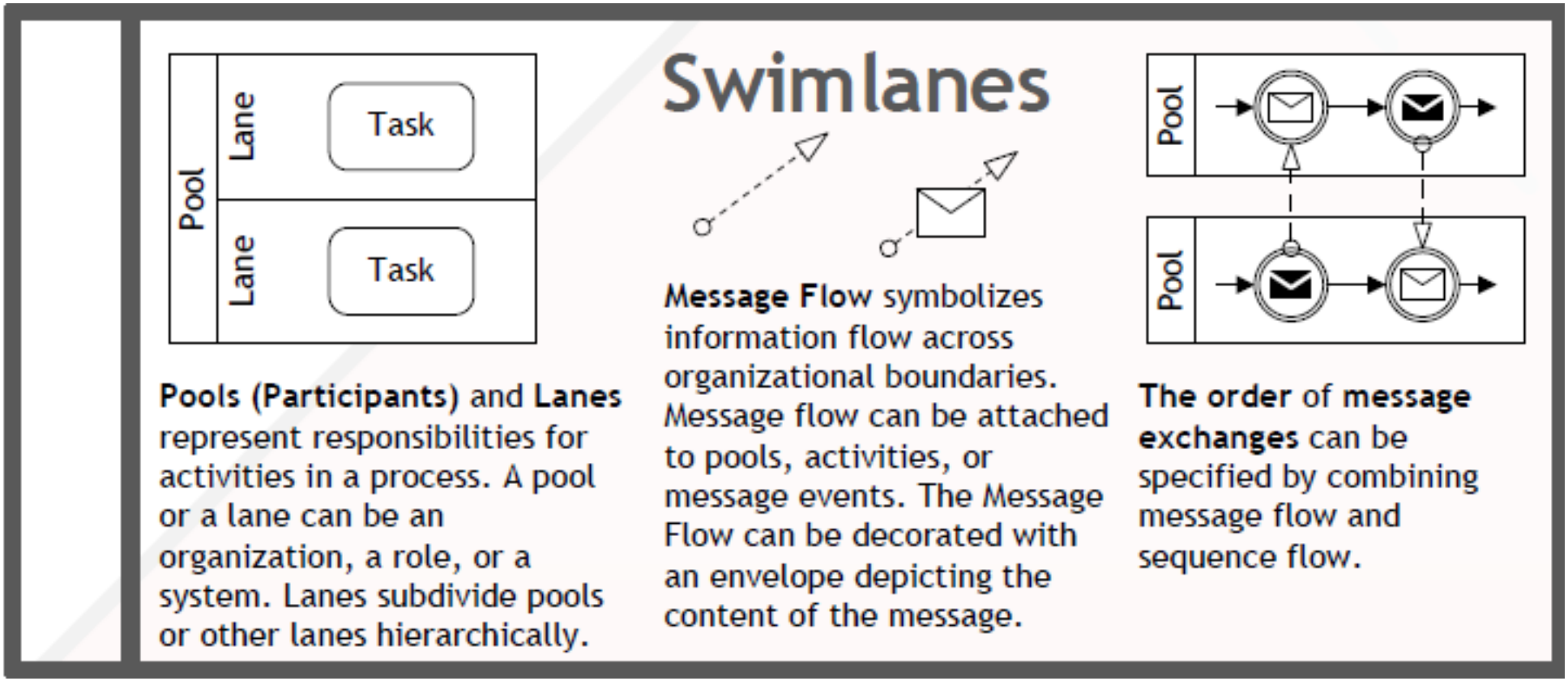
Complex Gateway
Complex merging and branching behavior that is not captured by other gateways.

Parallel Event-based Gateway (instantiate)
The occurrence of all subsequent events starts a new process instance.

BPM Offensive Berlin:
http://www.bpmb.de/index.php/BPMNPoster

Requirements Engineering

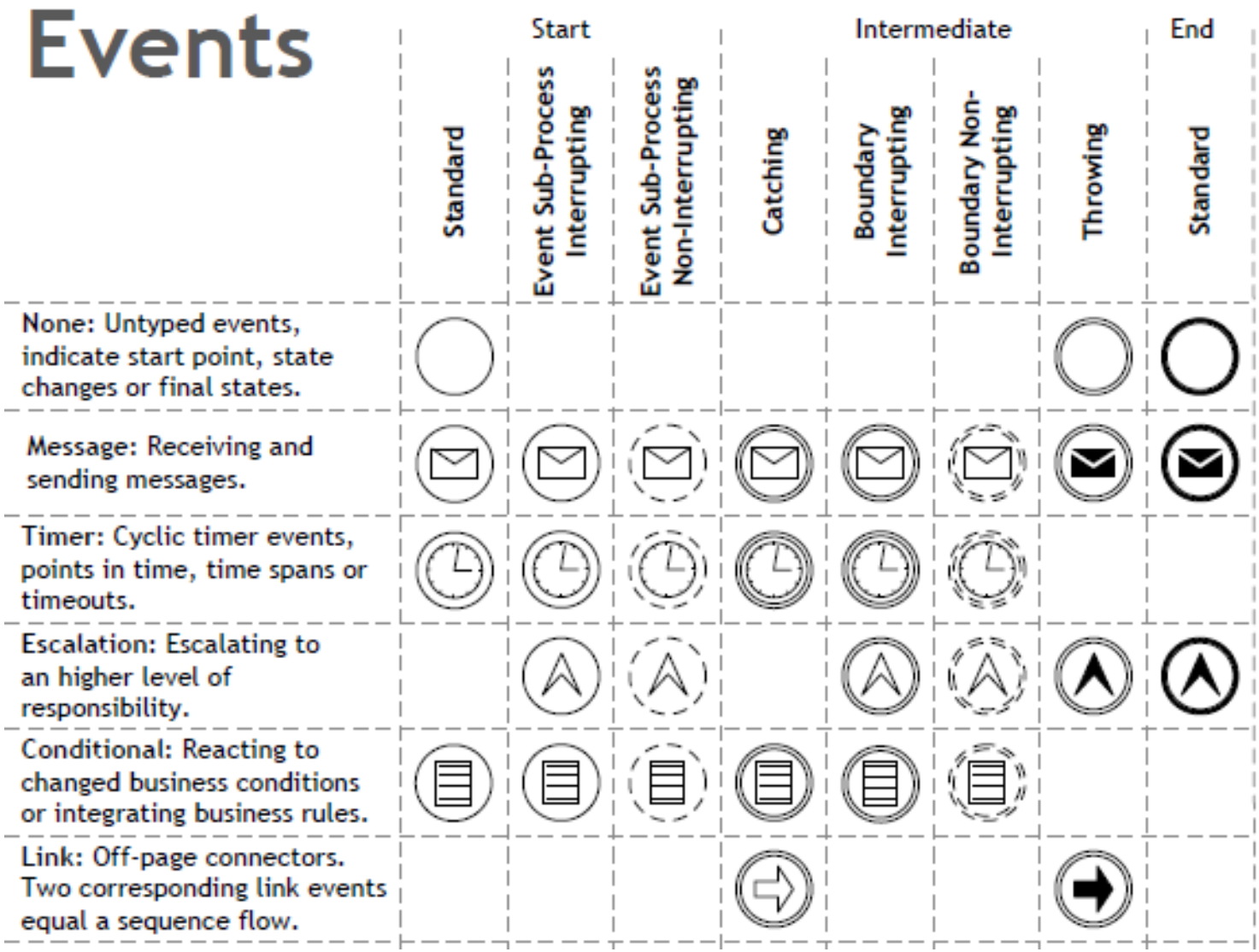
BPMN - Business Process Model and Notation



BPM Offensive Berlin:
http://www.bpmb.de/index.php/BPMNPoster

Requirements Engineering

BPMN - Business Process Model and Notation



BPM Offensive Berlin:
http://www.bpmb.de/index.php/BPMNPoster

Requirements Engineering

BPMN - Business Process Model and Notation

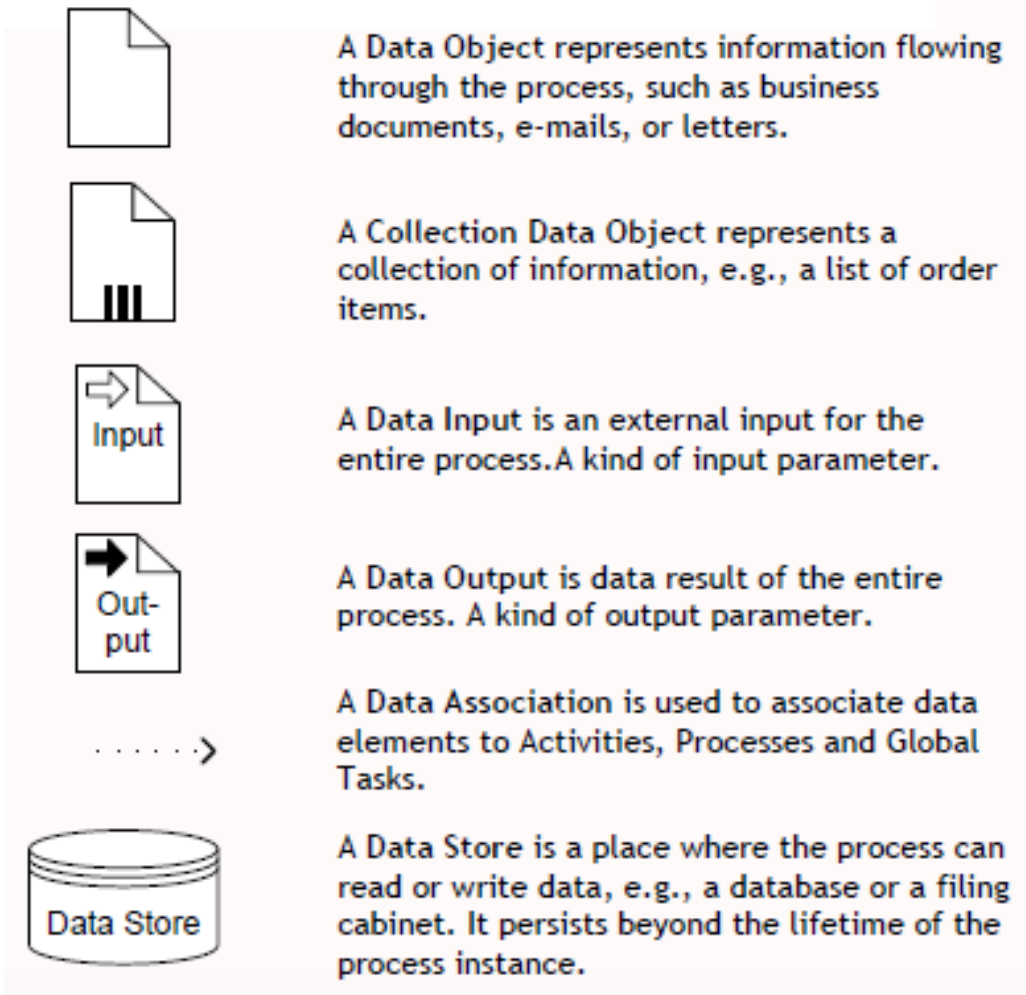
Error: Catching or throwing named errors.								
Cancel: Reacting to cancelled transactions or triggering cancellation.								
Compensation: Handling or triggering compensation.								
Signal: Signalling across different processes. A signal thrown can be caught multiple times.								
Multiple: Catching one out of a set of events. Throwing all events defined								
Parallel Multiple: Catching all out of a set of parallel events.								
Terminate: Triggering the immediate termination of a process.								

BPM Offensive Berlin:
http://www.bpmb.de/index.php/BPMNPoster

Requirements Engineering

BPMN - Business Process Model and Notation

Data



A Data Object represents information flowing through the process, such as business documents, e-mails, or letters.

A Collection Data Object represents a collection of information, e.g., a list of order items.

A Data Input is an external input for the entire process.A kind of input parameter.

A Data Output is data result of the entire process. A kind of output parameter.

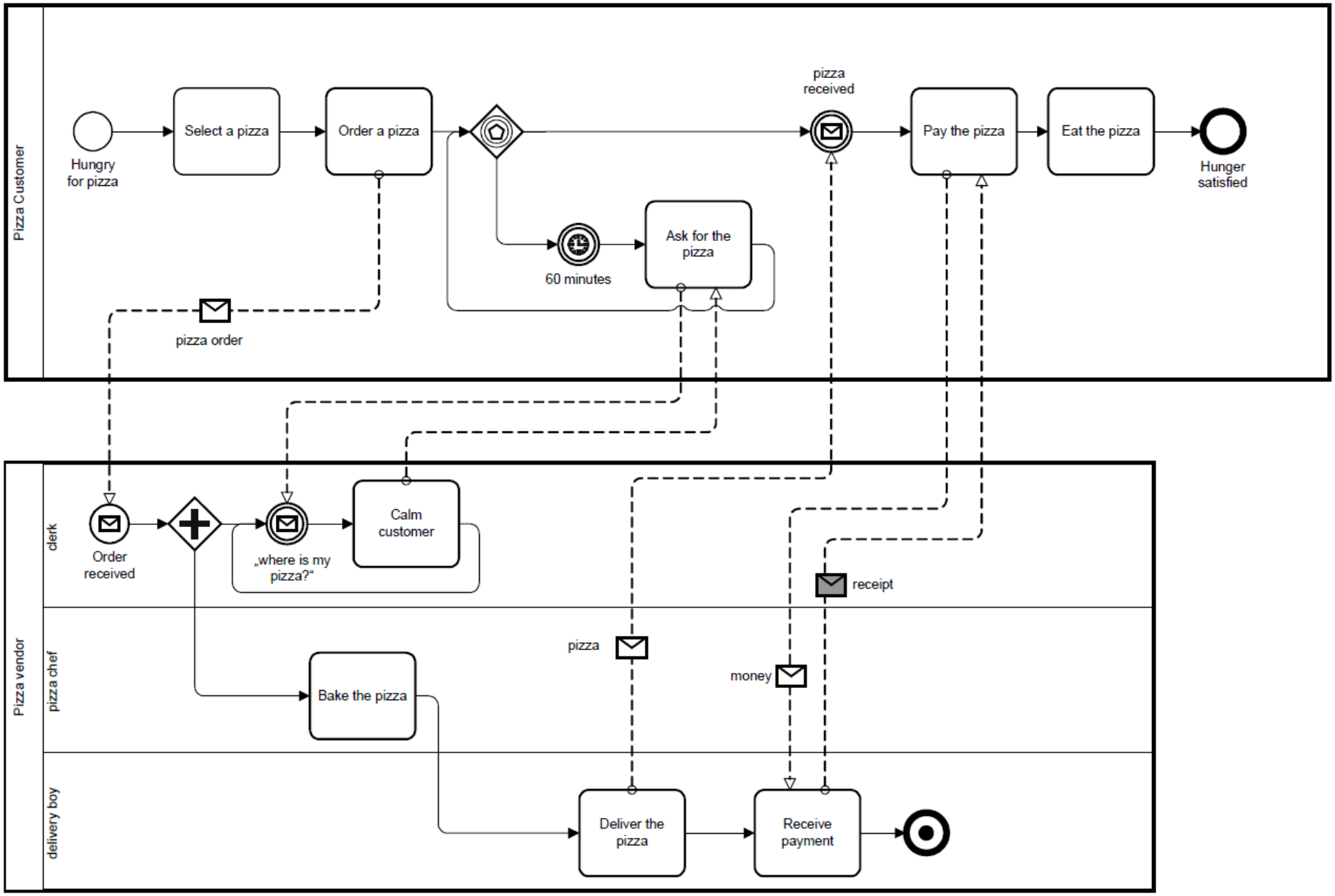
A Data Association is used to associate data elements to Activities, Processes and Global Tasks.

A Data Store is a place where the process can read or write data, e.g., a database or a filing cabinet. It persists beyond the lifetime of the process instance.

BPM Offensive Berlin:
http://www.bpmb.de/index.php/BPMNPoster

Requirements Engineering

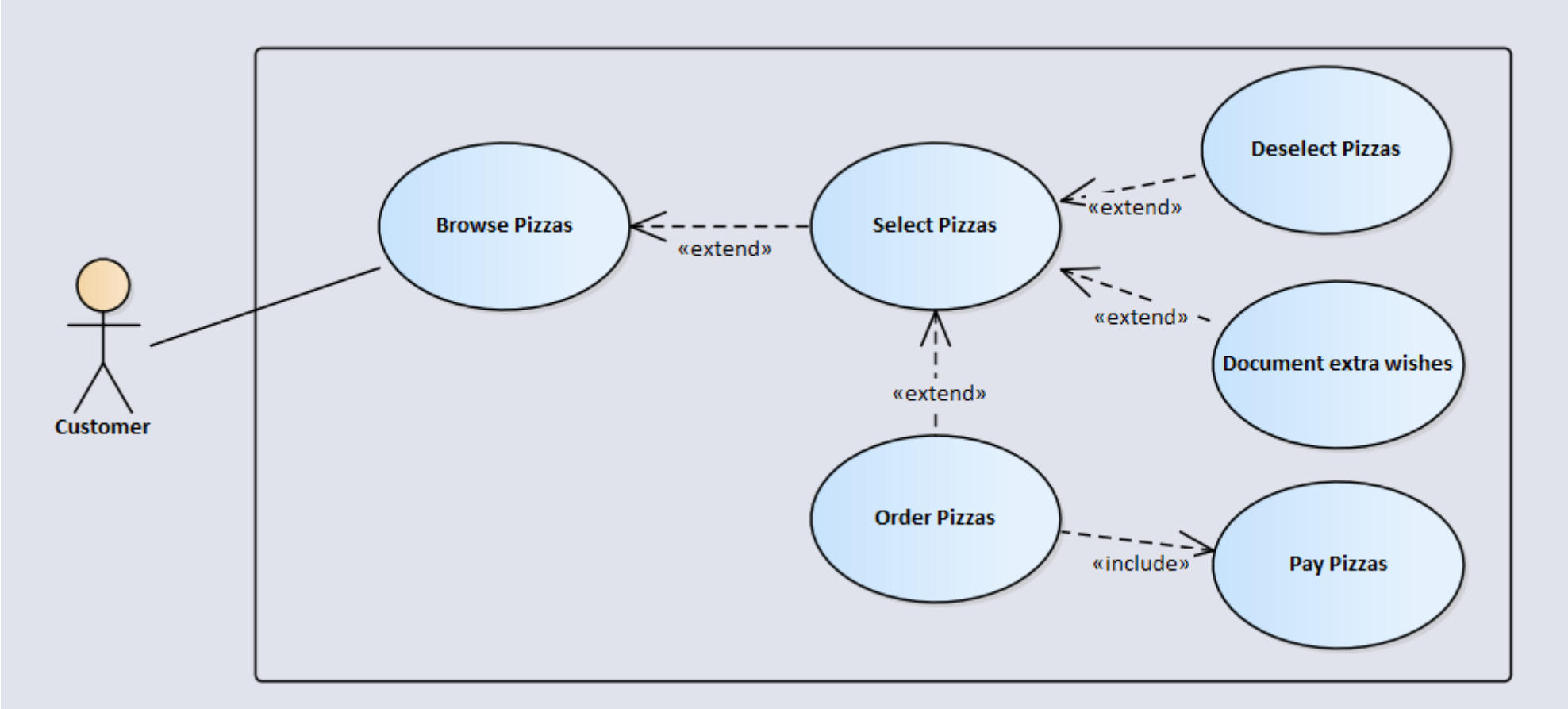
BPMN - Business Process Model and Notation



© OMG:
http://www.bpmn.org/

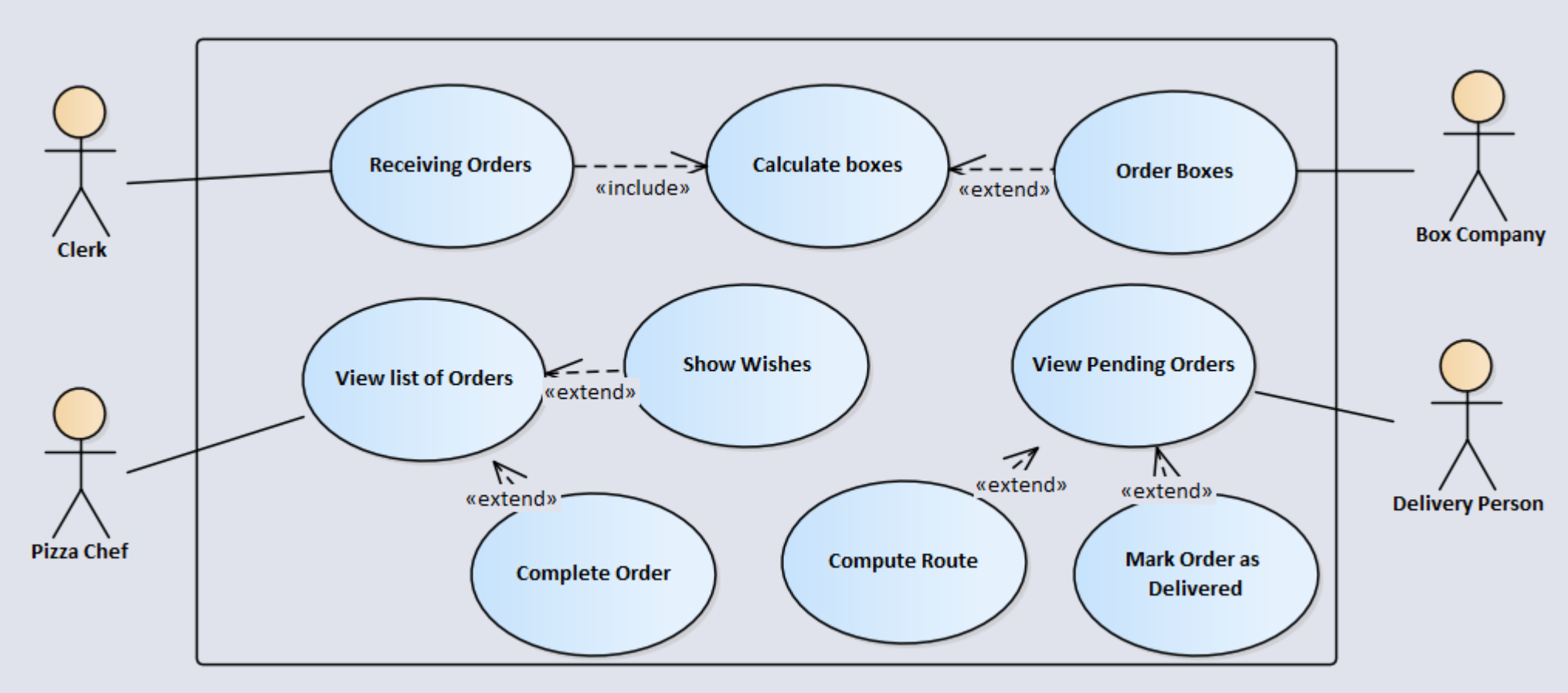
Requirements Engineering

System Use Cases



Requirements Engineering

System Use Cases



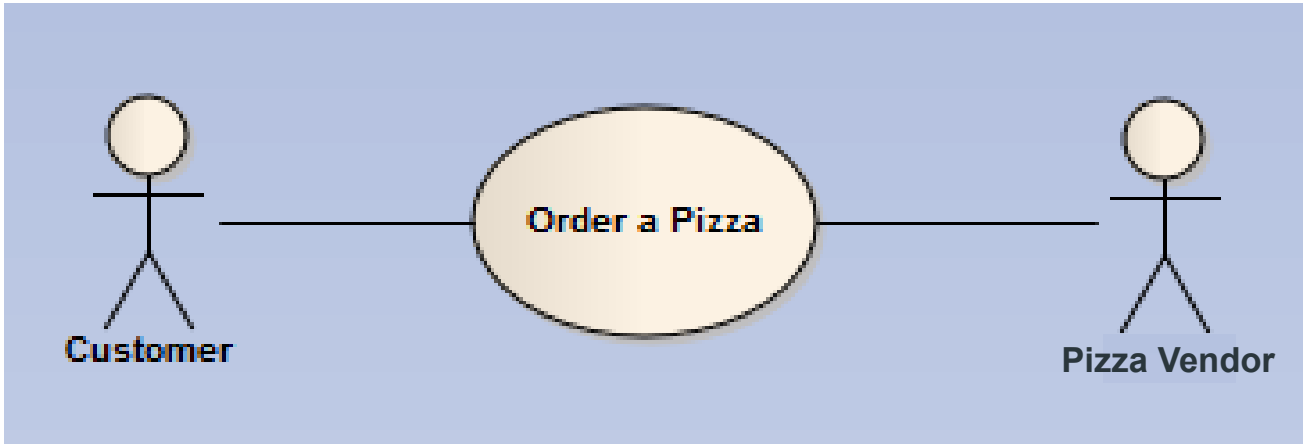
Requirements Engineering

Business Use Cases

A Business Use Case

- Describes a business process
- Is triggered by a business event
- Ends with a result, which represents a business value

A Business Use Case can be detailed with a text description and a business process model



Requirements Engineering

Business Use Cases – Text Description

- ***Name:*** Order a Pizza
- ***Brief Description:*** In this Use Case, a Customer places a new pizza order for delivery. The pizza vendor bakes the pizza, delivers it to the customer, and receives the payment.
- ***Principal Actor:*** Customer
- ***Precondition:*** Customer must reside in the area of the business to qualify for delivery.
- ***Trigger:*** The Customer is hungry
- ***Postcondition:*** The pizza has been baked and delivered. The customer has paid.

Requirements Engineering

Business Use Cases – Text Description

Main Flow

Step Name	Description
Select pizza and place order	The customer selects the pizza from the product catalogue, optionally offers online payment, and advices delivery address.
Authorize Payment	If online payment, the finance partner confirms that the offered payment is valid, and authorizes the payment.
Bake pizza	The pizza chef bakes the pizza.
Deliver pizza	The pizza vendor delivers the pizza.
Receive payment	The delivery person receives the payment in case the customer did not pay online.

Requirements Engineering

System Use Cases

A System Use Case describes

- The behavior of the system for specific actors (interacting humans or system components)
- A functional requirement
- The process supported by a system

System Use Cases can be derived from business use cases or business process models.

Requirements Engineering

Business vs. System Use Case

Aspect	Business Use Case	System Use Case
Primary Actor	Business actor, e. g. customer, maybe internal or external	System actor, e. g. a human user or another system initiating system behavior
Purpose	Something the actor wants to get done by using the business	Something the actor wants to get done by using the system
Who / what else may be involved?	External business parties	Other systems
Duration	Varying duration	Typically quite short

Requirements Engineering

System Use Cases – Text Description

Possible Template

UC Name	**Submit loan request**
Unique UC ID:	UC-100
Primary actor(s):	Applicant
Secondary actor(s):	Loan assistant, credit bureau, accounts management system
Preconditions:	The system presents a loan request.
Flow of events:	// compare printout
Postconditions:	The loan request is ready to be evaluated.
Priority:	High

Requirements Engineering

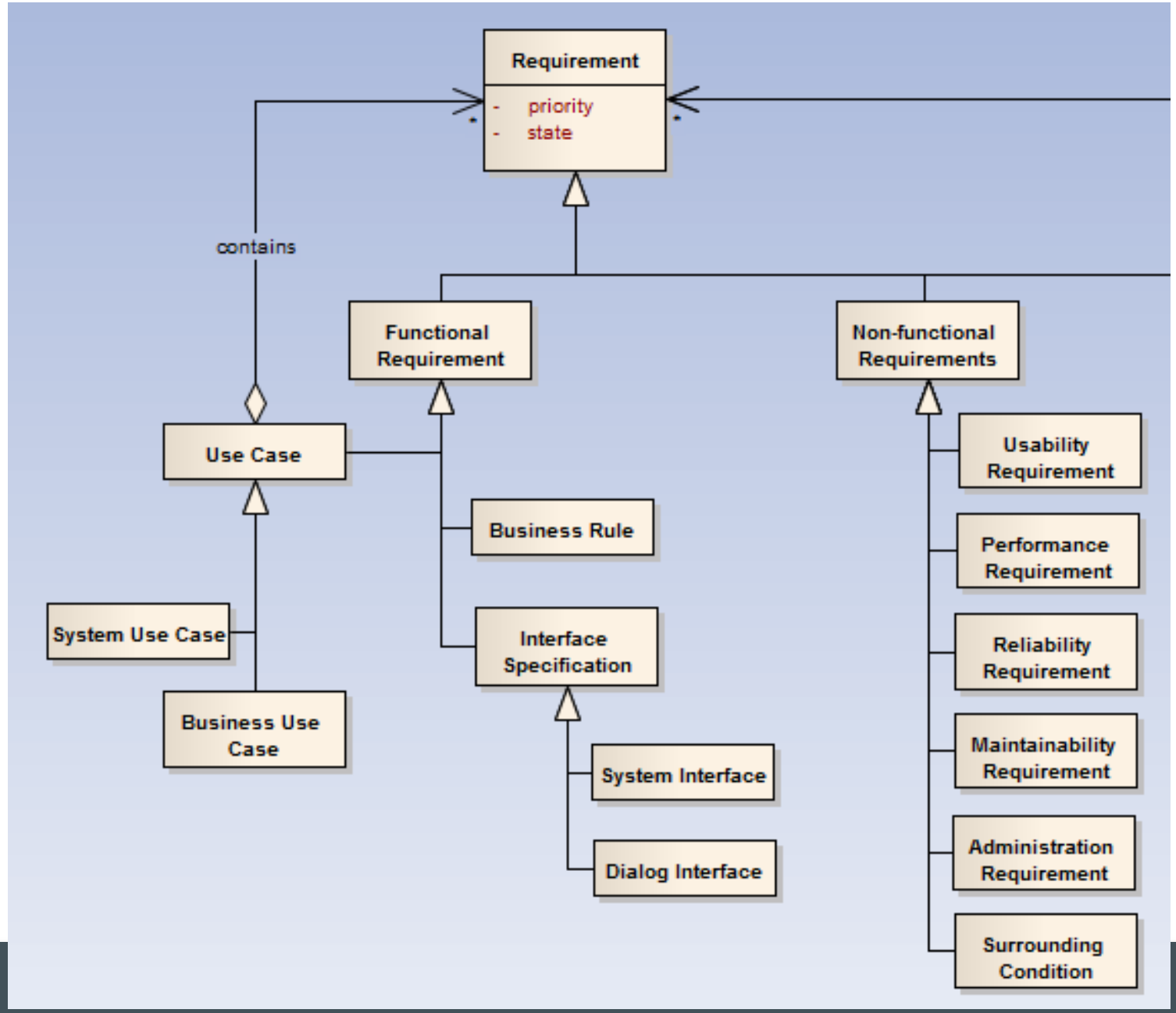
System Use Cases – Text Description

Possible Template

Alternative flows and exceptions:	// compare printout [Armour and Miller]
Nonbehavioral (Non-functional) requirements:	// compare printout
Assumptions:	// compare printout
Issues:	// compare printout
Design Remarks:	// Sometimes used by the software architect when reviewing the specification to note remarks for the software developer
Source:	Requirements workshop …

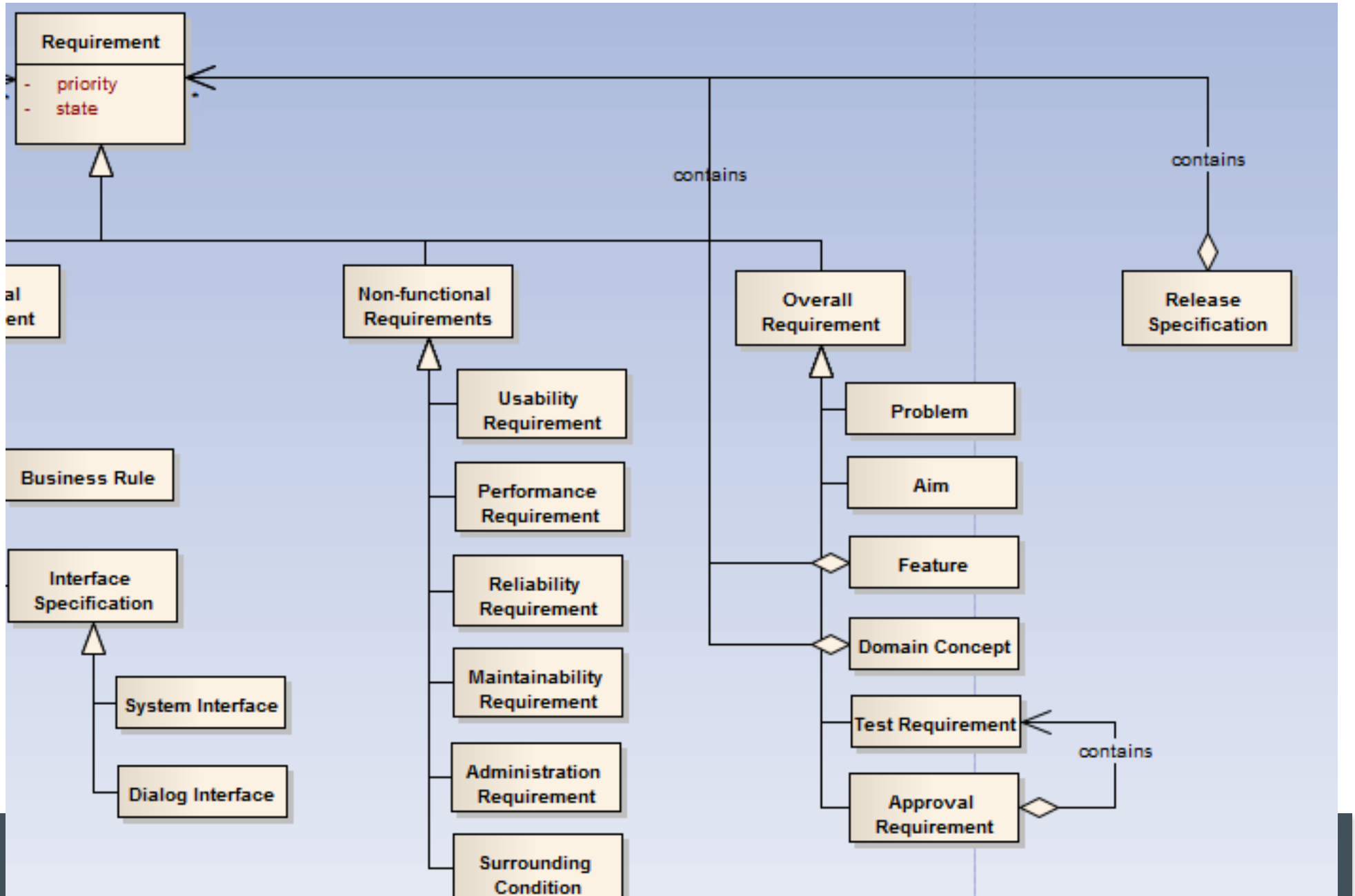
Requirements Engineering

Types of Requirements – Metamodel



Requirements Engineering

Types of Requirements – Metamodel



Requirements Engineering

Prototyping

A prototype of a system is used to

- Demonstrate concepts
- Try out design options
- Find out more about the problem and possible solutions
- Let stakeholders experiment early in the software process
- Get feedback from the customer
- Find areas of strength and weakness
- Involve end users
- …

Requirements Engineering

Prototyping

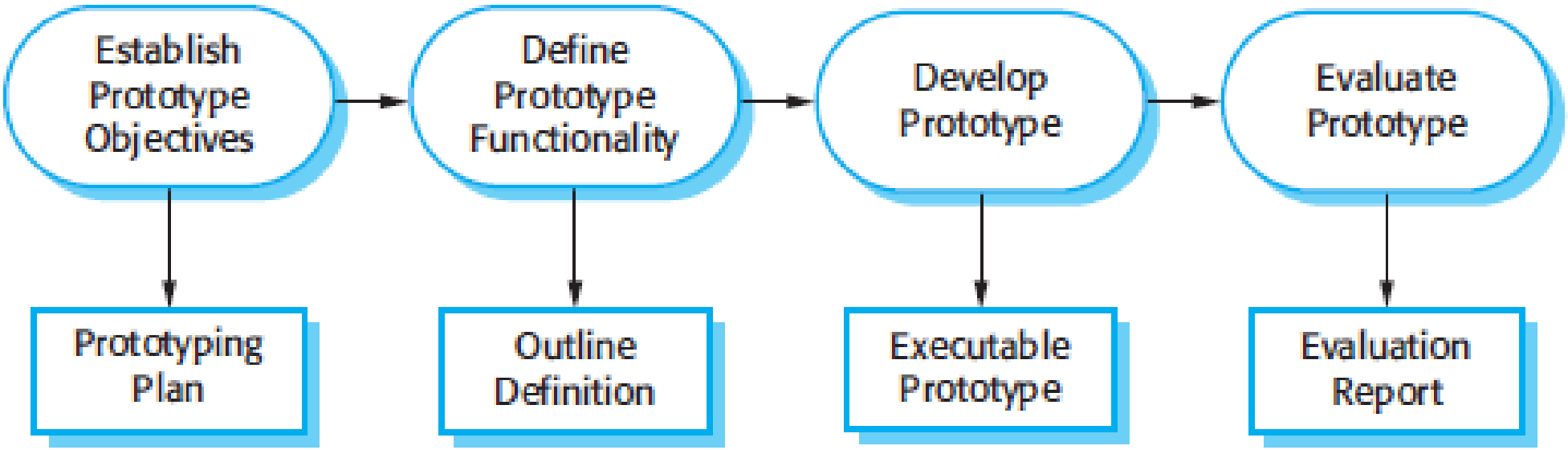
Problems with prototyping:

- The objectives of the prototype might be unclear (e. g. prototype the user interface or validate functional systems requirements).
- Additional costs might be produced.
- Non-functional requirements, such as performance, might have been ignored.
- Testers of the prototype may not be typical users.
- “Throwaway” prototypes might be undocumented.
- Paper-based mock-ups of the user interface are cheap, but maybe not accepted.
- …

Requirements Engineering

Prototyping

Process of prototype development

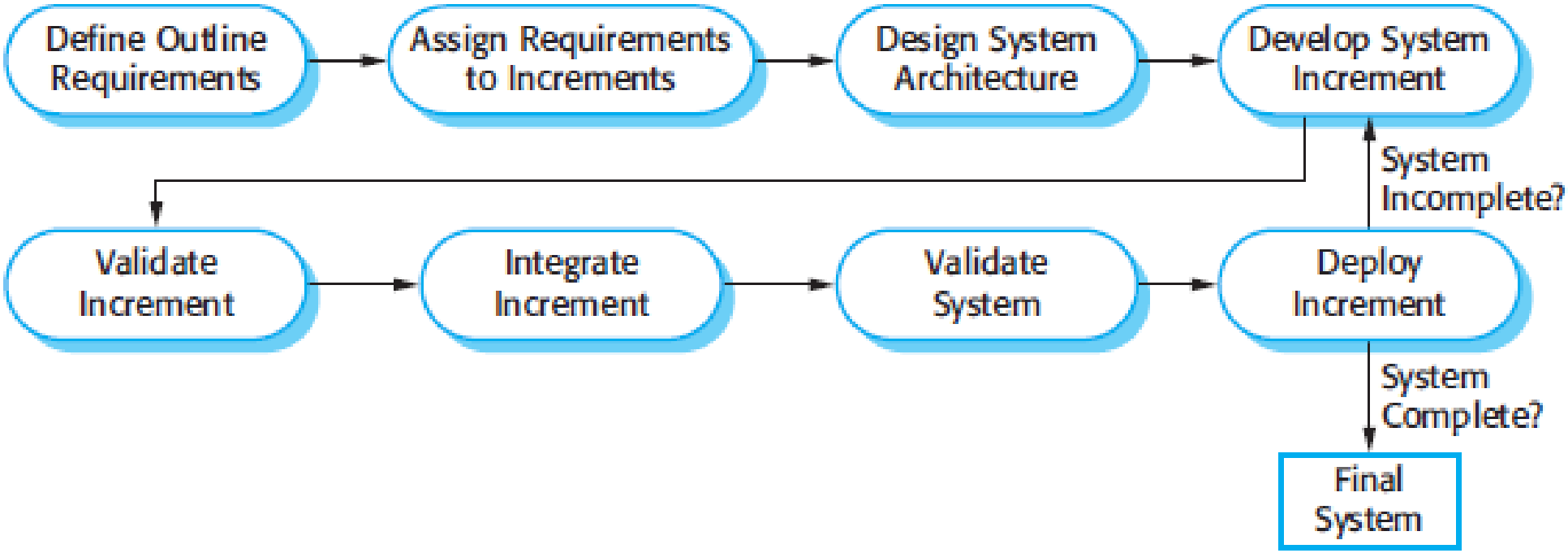


Sommerville, p. 45

Requirements Engineering

Prototyping

Incremental delivery



Sommerville, p. 47

TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Agile Planning

Agile Project Management

Manifesto for agile software development

- **"*Individuals and interactions* over processes and tools**
- ***Working software* over comprehensive documentation**
- ***Customer collaboration* over contract negotiation**
- ***Responding to change* over following a plan"**

(compare http://agilemanifesto.org/)

⇨ **The focus is on**

- Lightweight processes (less bureaucracy)
- Human aspects

Agile Project Management

Seven Principles of Lean Software Development

- **Eliminate Waste**
  - Waste is anything that does not add value, e. g. partially done work, too long development cycles, extra features that are rarely used
- **Build Quality In**
  - Build quality into the code from the start, not test it in later
- **Create Knowledge**
  - Systematic learning throughout the development cycle
- **Defer Commitment**
  - Most decisions should be made reversible, so they can be made and then easily changed

Poppendieck, pp. 23-43

Agile Project Management

Seven Principles of Lean Software Development

- **Deliver Fast**
  - Cost advantage, elimination of waste
  - Continually improve processes
- **Respect People**
  - Development of good leaders and technical expertise
  - Responsibility-Based Planning and Control: Self organizing teams
- **Optimize the Whole**
  - Optimize the entire value stream

Poppendieck, pp. 23-43

Agile Project Management

Background: Lean Production

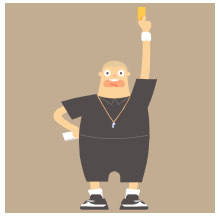
- Just-in-Time production
- Pull principle
  - In Scrum: The team controls the amount of work that it can handle in a respective sprint

cf. Gloger

Scrum

Overview

Roles



Scrum Master

Product Owner

Team

Communication

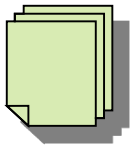
Sprints

Continuous Improvement

Artifacts

Vision

Product Backlog



Selected Product Backlog

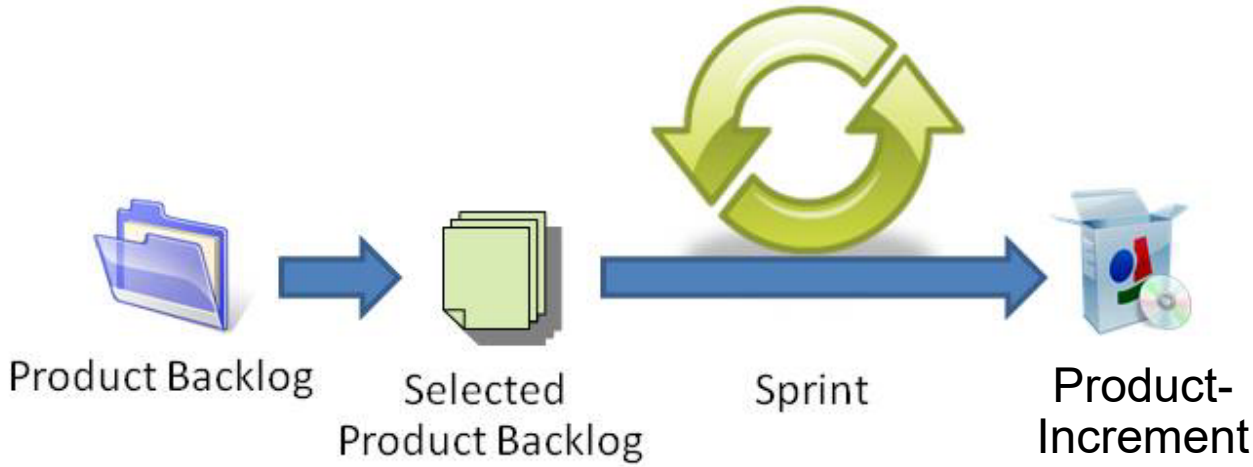
Sprint Backlog

Product-Increment

Scrum

Overview

- **Change management approach**
  - Frequent short iterations
  - Sprint planning based on a prioritized product backlog (prioritized according to *business value*)
  - Self organizing team
  - Simultaneous assignment of one and the same resource in several projects might be critical



Scrum

User Stories

- **User stories originated with *Extreme Programming* (XP)**
- **Scrum planning uses product backlog items, which can be user stories, use cases, features, etc.**

- **User stories**
  - are written by or for business users as a primary way to influence the functionality of the system
  - it is written down on a note card with a name and a brief description
  - "As a <*role*>, I want <*goal/desire*> so that <*benefit*>“
  - E. g. “As a student I want to purchase a parking pass so that I can drive to school” (cf. Ambler)

Scrum

User Stories

Front of Card

173

As a student I want to purchase a parking pass so that I can drive to school

Priority: Should
Estimate: 4

Back of Card

Confirmations:

~~The student must pay the correct amt~~

One pass for one month is issued at a time

The student will not recieve a pass if the payment isn't sufficient

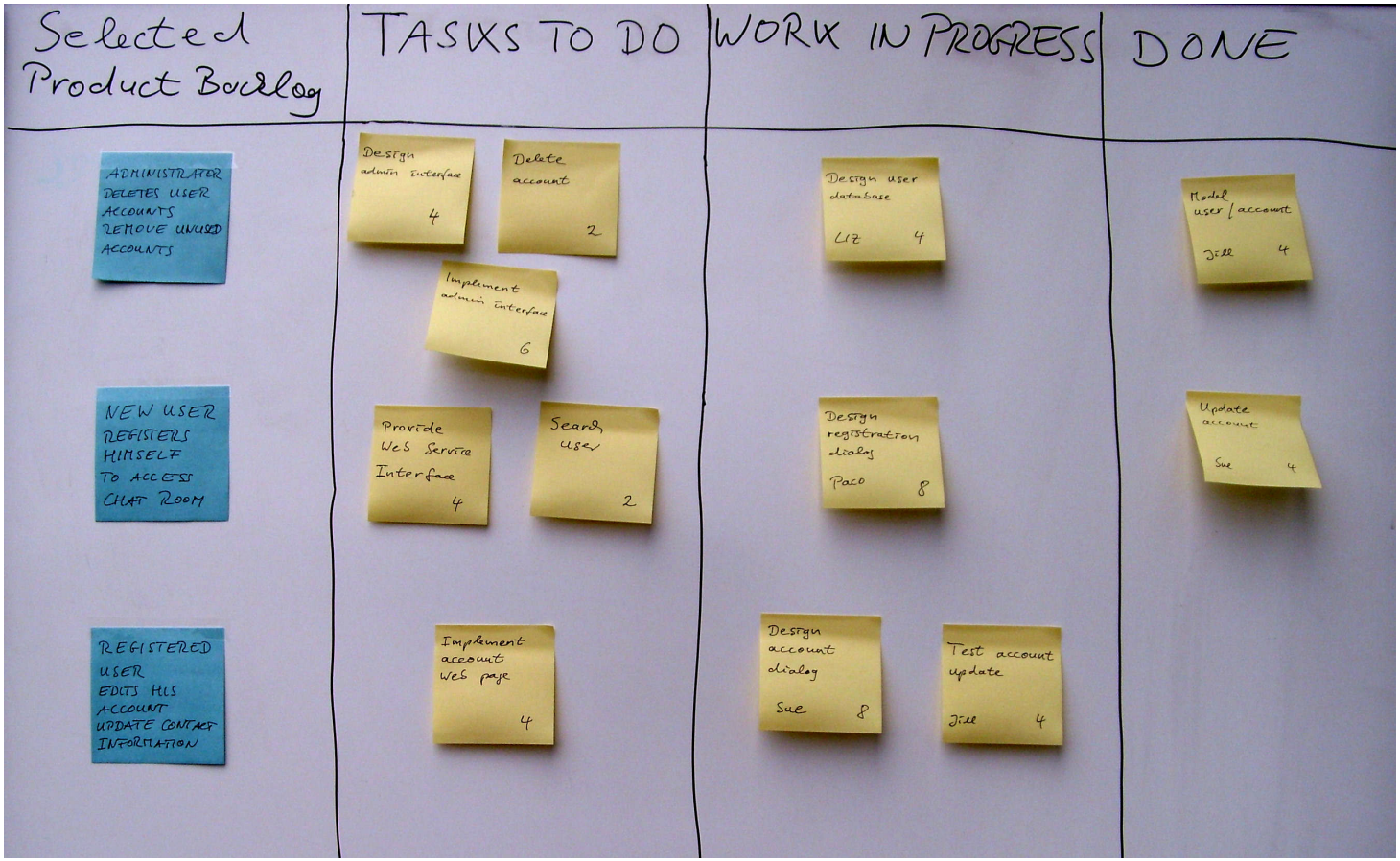
The person buying the pass must be a currently enrolled student.

The student may only buy one pass per month.

Copyright 2005-2009 Scott W. Ambler

Scrum

Sprint backlog



Scrum

Reporting

- **Sprint Burn-Down Chart**
  - Shows the remaining hours for a sprint in the *daily meetings* (therefore, usually hours and days are the dimensions of the graph)

- **Sprint Product Burn-Down Chart**
  - Shows the remaining *Backlog Items* for a sprint in the *daily meetings* (bar chart)

Scrum

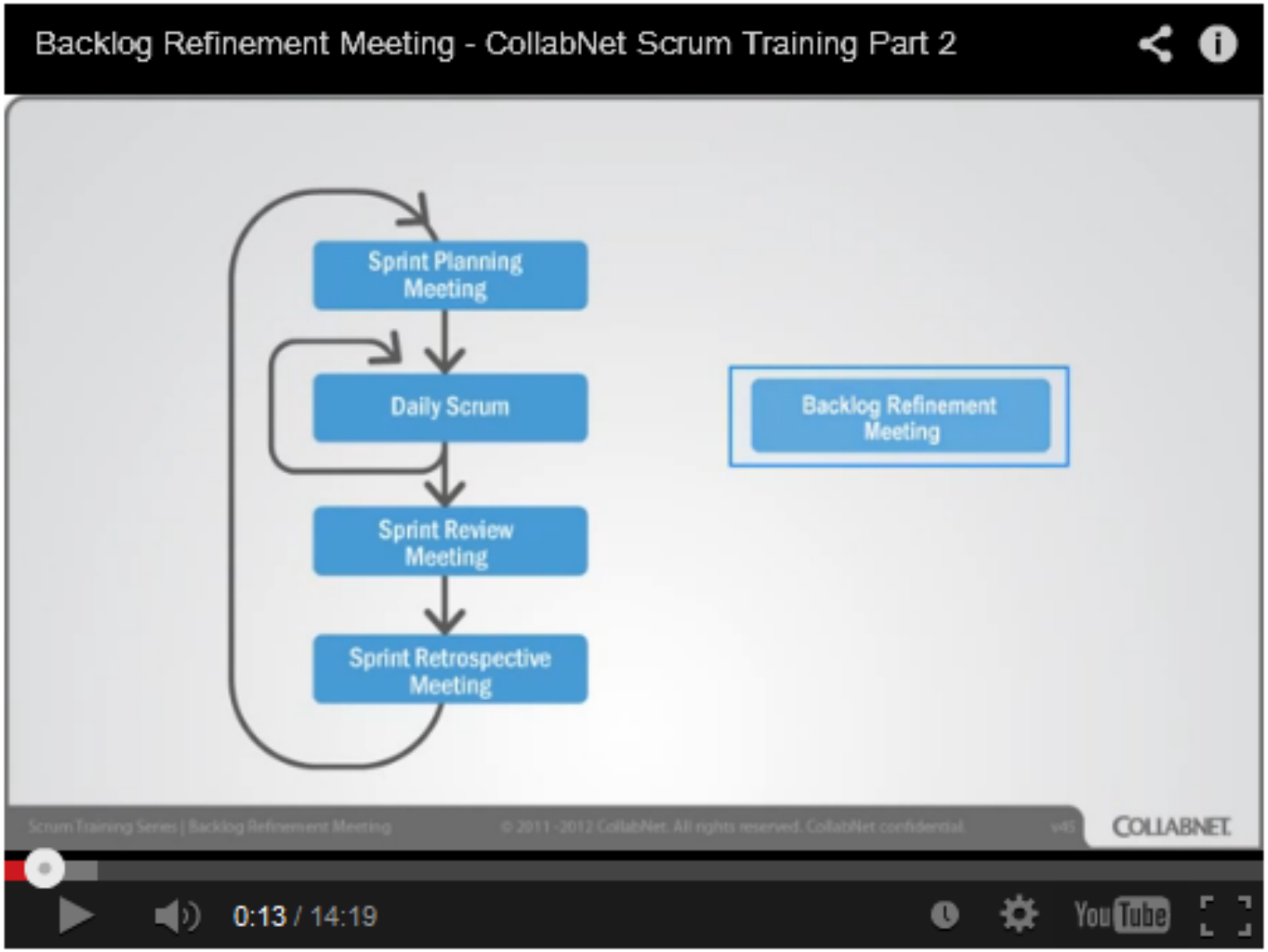
Retrospective

- **Similar to *lessons learned* at the end of a sprint**
  - Learning from experiences
  - Identification of room for improvement

- **What went well during the sprint?**
- **What could be improved in the next sprint?**
  - Which issues can be improved by the team itself? (→ *Product Backlog*)
  - Which issues need to be addressed by the organization, the management, or the customer? (→ *Scrum Master*)

Scrum

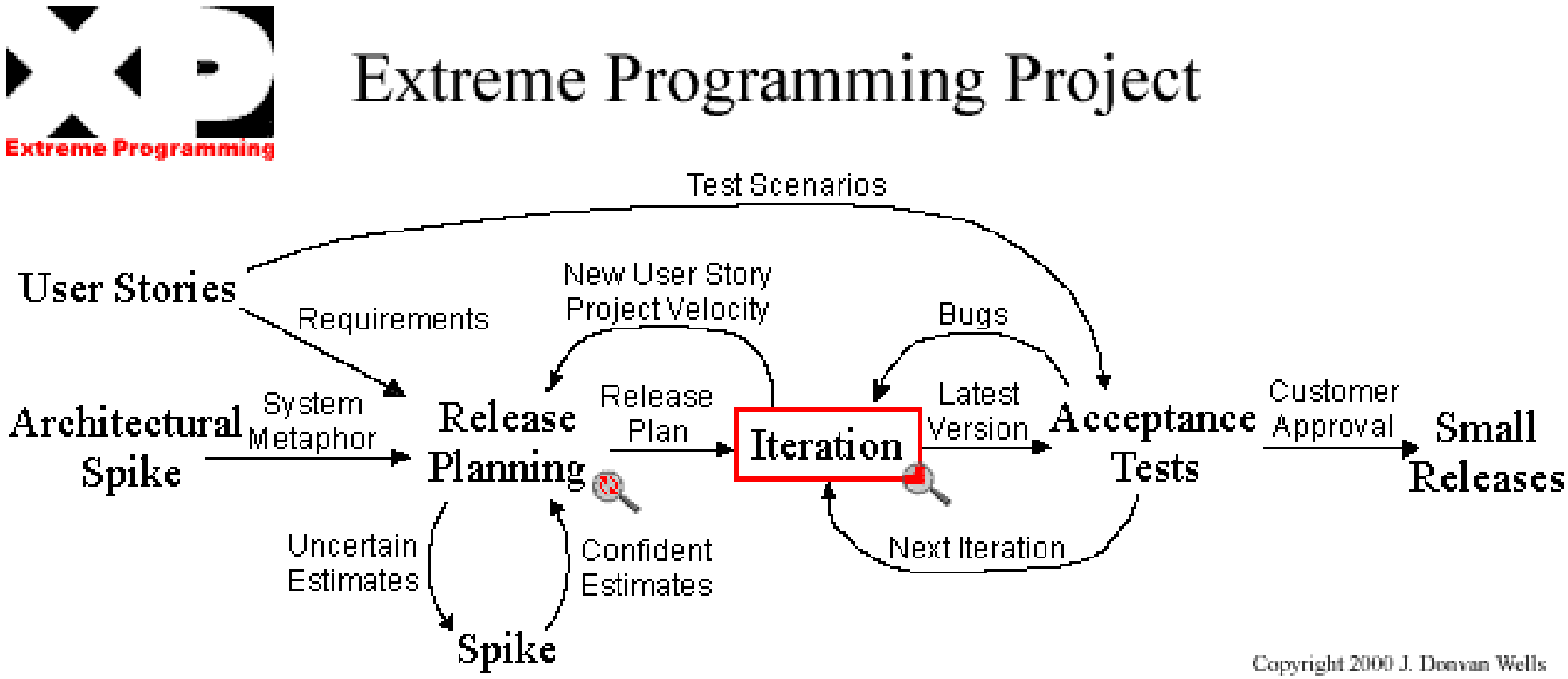
Effort Estimation



http://www.youtube.com/watch?v=b_WeHcZcx1w

Extreme Programming

Overview



Cf. http://www.extremeprogramming.org/rules.html

Scrum

Exercises

- **Investigate in small groups about the following Scrum tools and prepare a small talk about it, which you then present in the end of this exercise:**
  - *Axosoft:* http://www.axosoft.com/
  - *Scrumwise:* http://www.scrumwise.com/
  - *VersionOne:* http://www.versionone.com/
  - *Agilefant (open source):* http://agilefant.com/
  - *ScrumDo*: https://www.scrumdo.com/

TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Software Architecture & Reuse

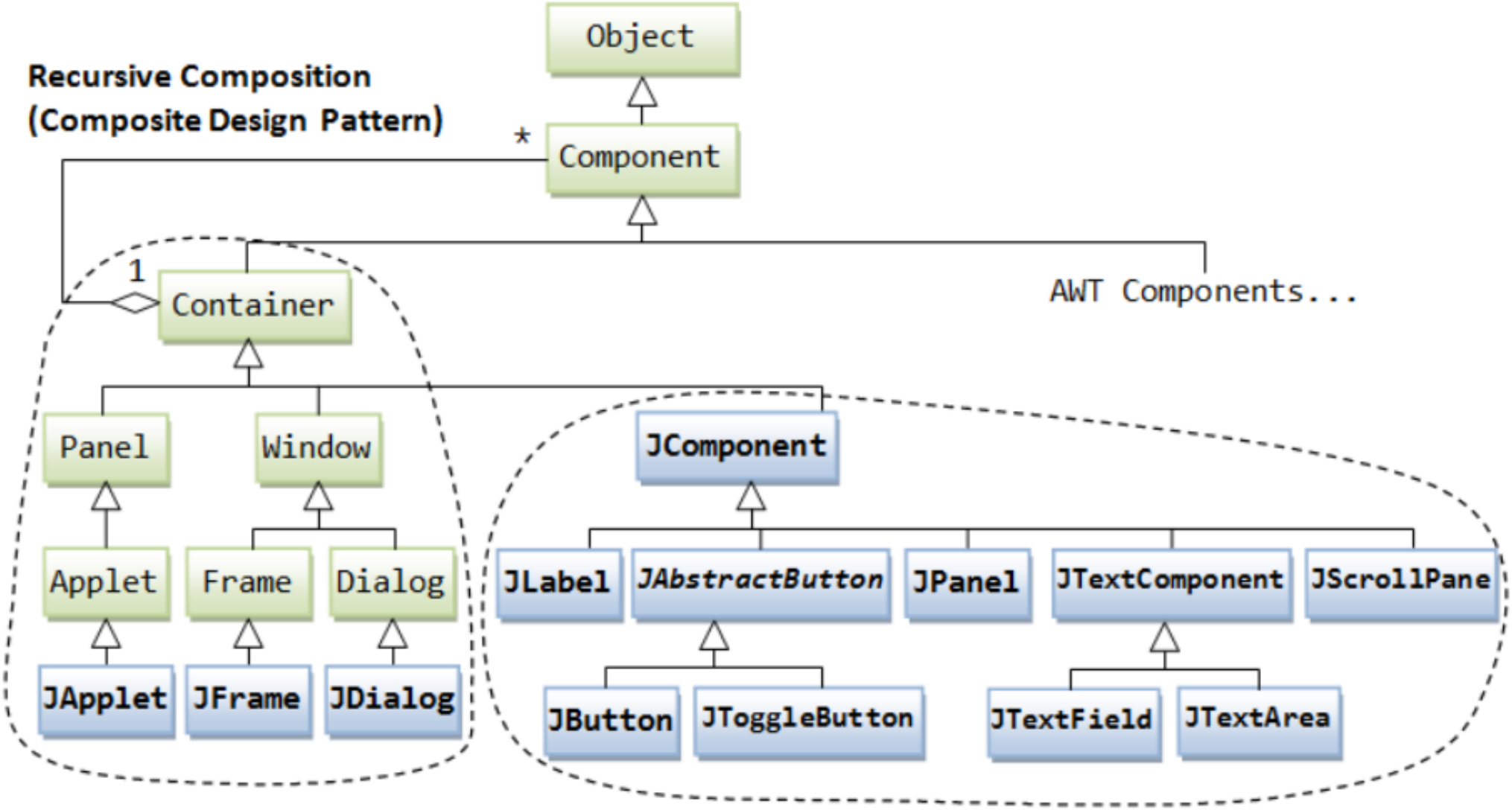
Software Architecture

"Software component"

- ***Philippe Krutchen, Rational Software*:**
  - "A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces."
- ***Clemens Szyperski in "Component Software":***
  - "A software component is a unit of composition with contextually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition."
- ***Grady Booch, Jim Rumbaugh, Ivar Jacobson in "The UML User Guide":***
  - "A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. A component typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations."

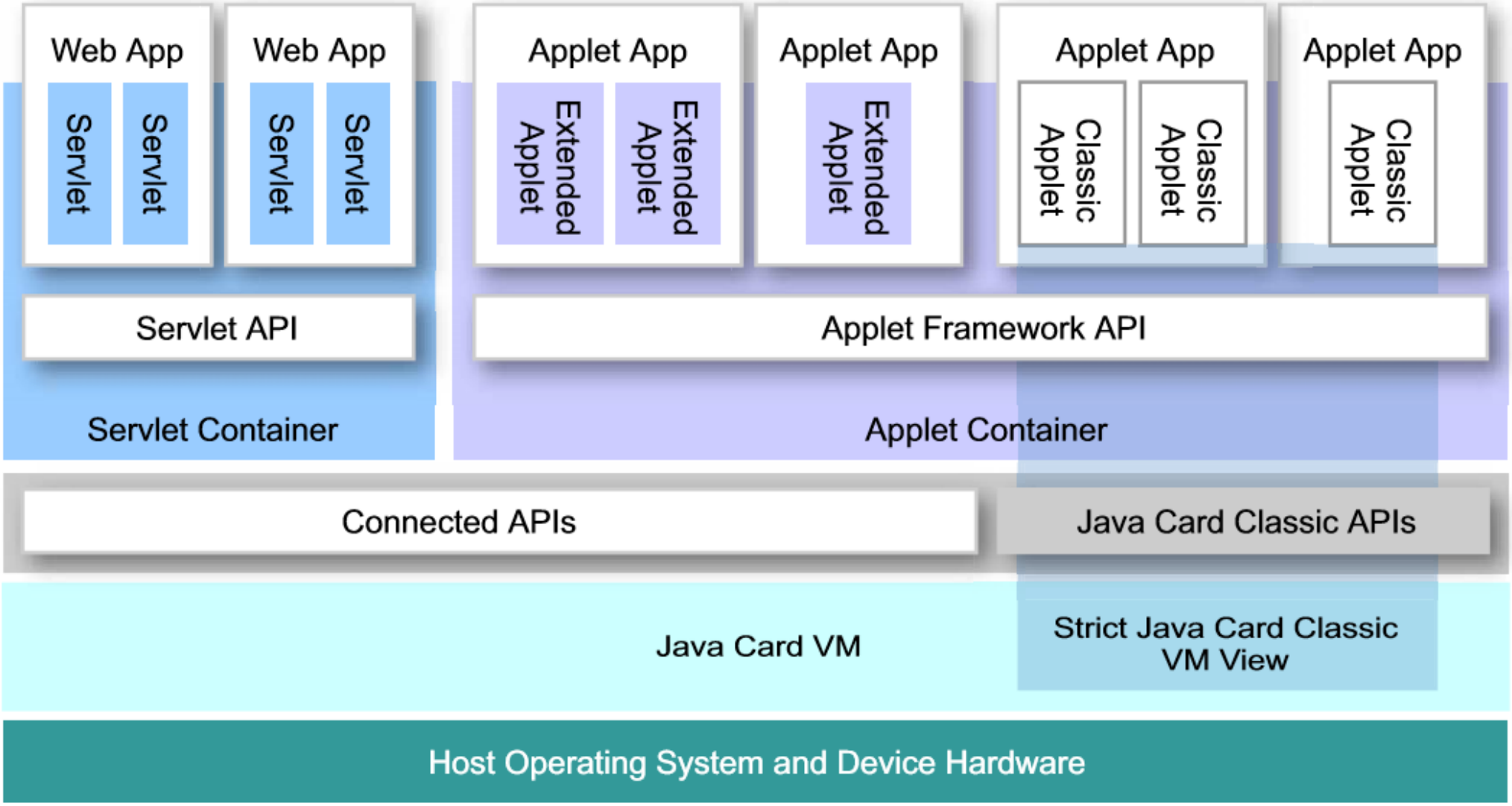
Software Architecture

Example: Java Swing Component Framework



Software Architecture

Example: JavaCard 3 Connected Edition



© Sun Microsystems: Development Kit User‘s Guide

Software Architecture

UML Component Diagrams

https://www.youtube.com/watch?v=KQUGFFN4M90

Software Architecture

"Design Patterns"

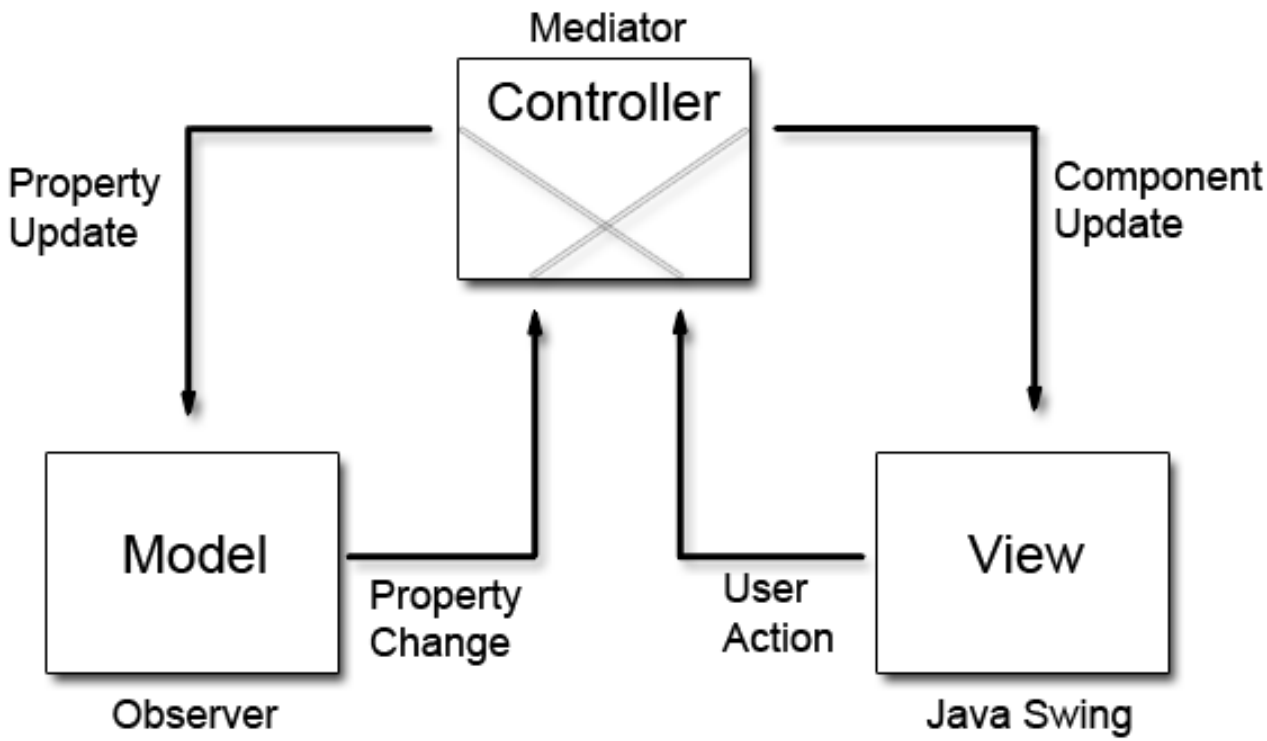
- ***Christopher Alexander about patterns in buildings and towns (in „A Pattern Language", Oxford University Press, 1977):***
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."

- ***Gamma et al. about design patterns (in "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995):***
  - "Each design pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems."

Software Architecture

Design Patterns: MVC

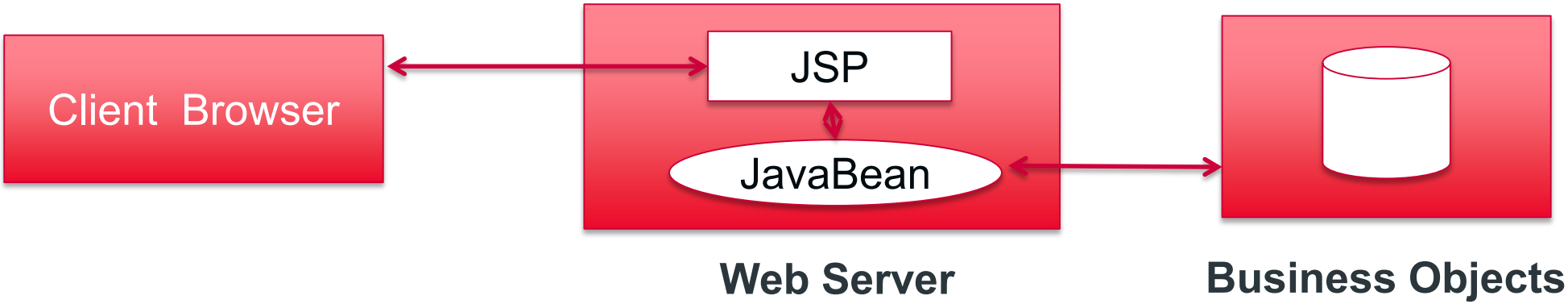
- **Model View Controller paradigm**
  - Originated in Smalltalk
- **Several design patterns are included, e. g.:**
  - “Observer” (for notifications)
  - “Composite” (composite view objects)
  - “Strategy” (assignment view-controller)



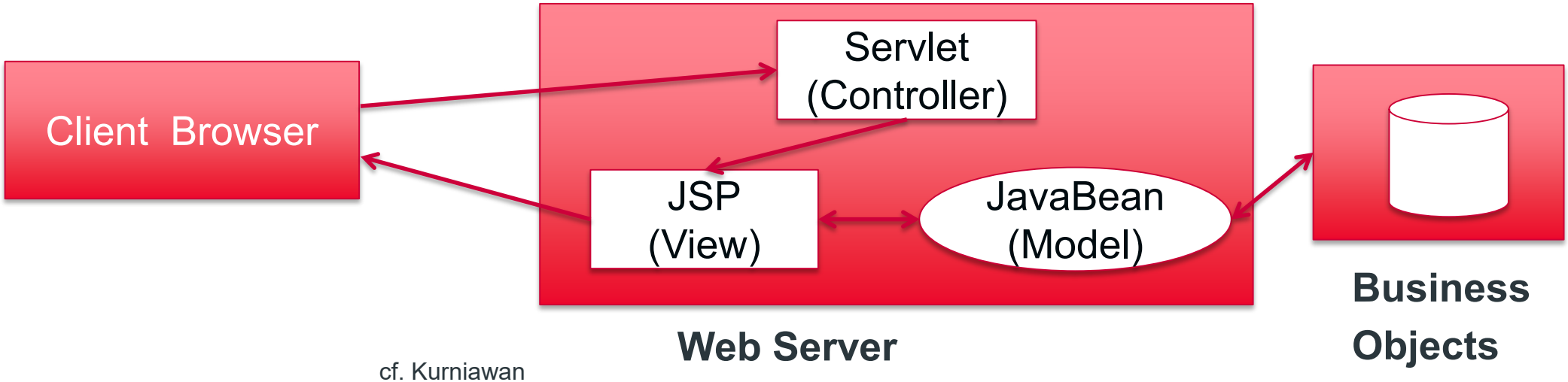
Software Architecture

Design Patterns: MVC & Java Web Applications

Model 1 Architecture:



Model 2 Architecture:

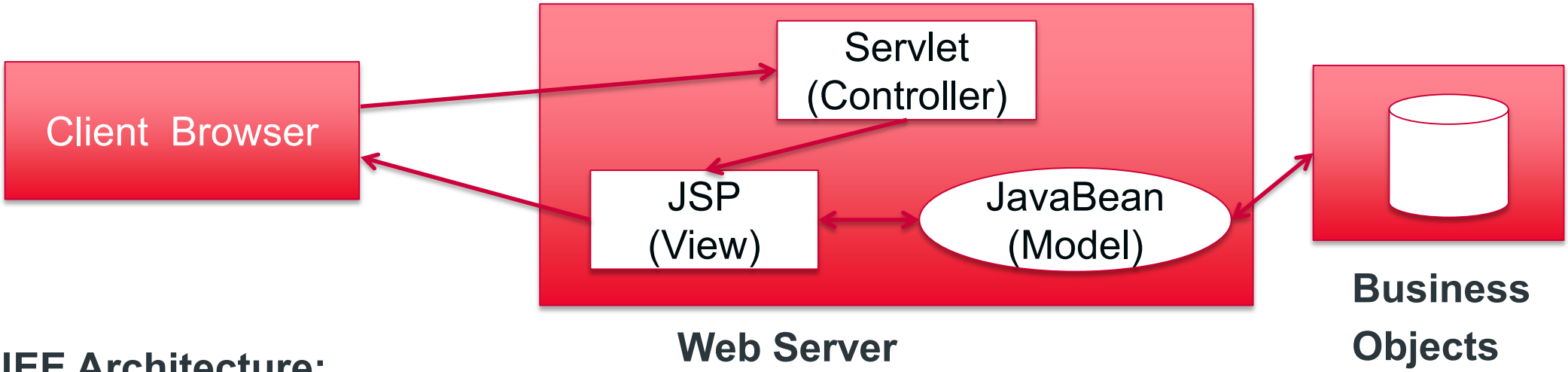


cf. Kurniawan

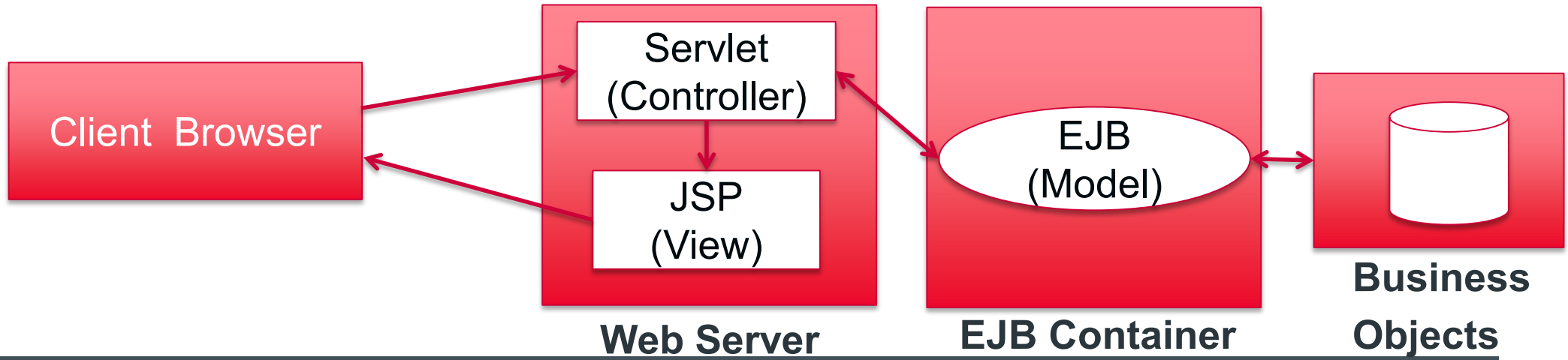
Software Architecture

Design Patterns: MVC & Java Web Applications

Model 2 Architecture:



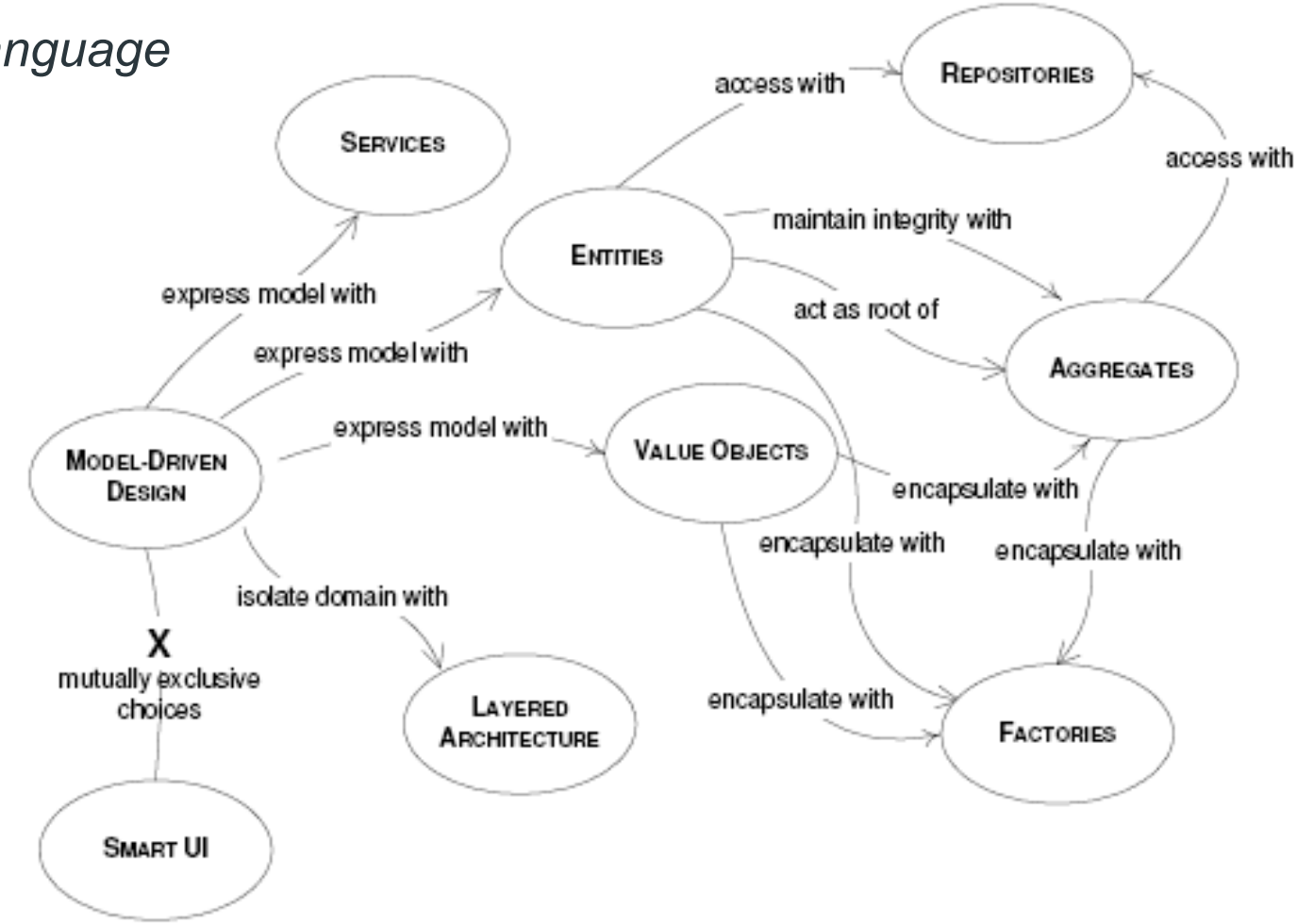
JEE Architecture:



Software Architecture

Design Patterns: Domain-Driven Design (DDD)

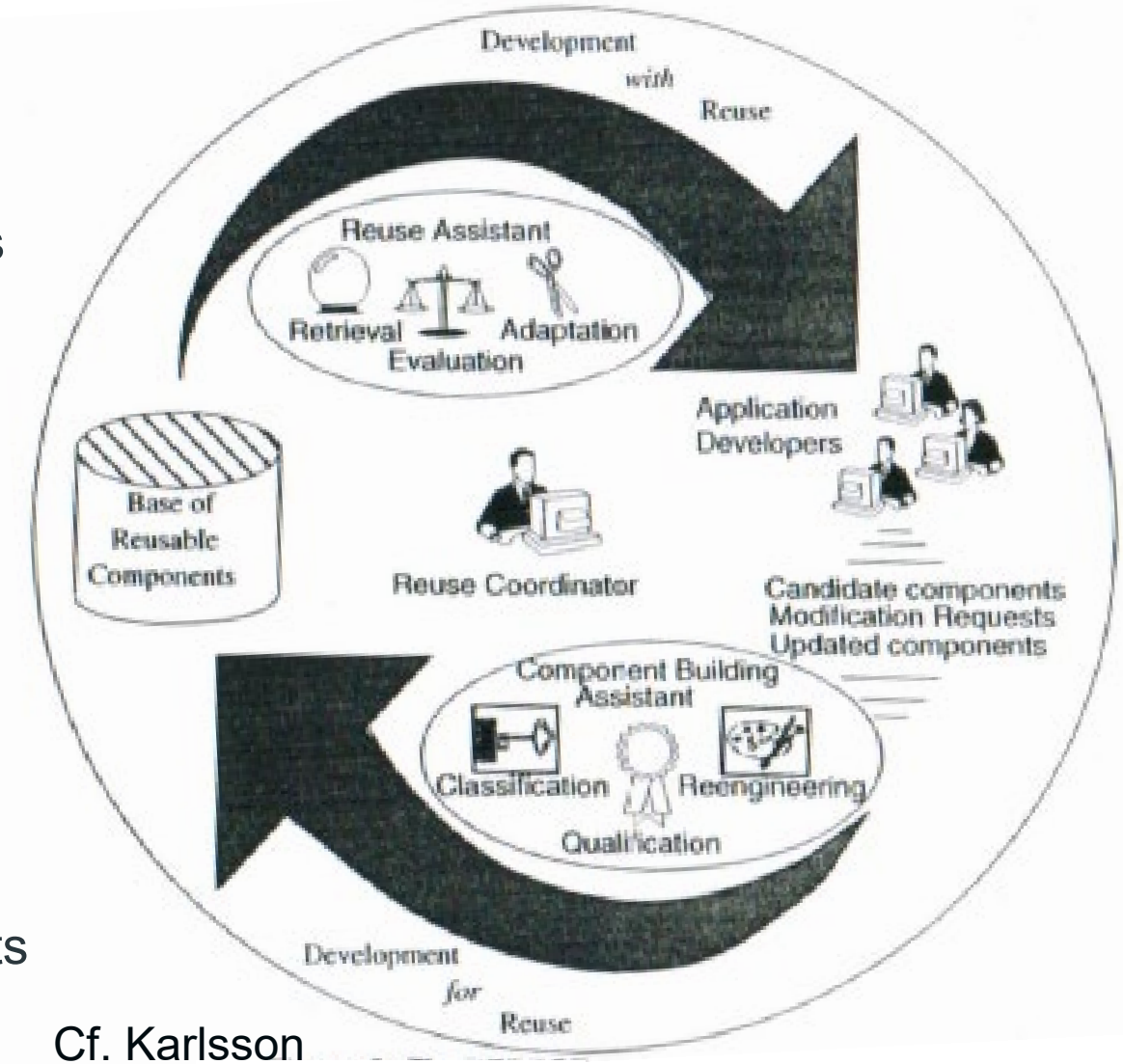
- **By Eric Evans**
  - *Ubiquitous language*
  - Patterns:



Software Architecture

Reuse

- **Development for reuse**
  - Analyze variability in requirements
  - Analyze costs and benefits
  - Design the components
  - Re-engineer existing components
- **Development with reuse**
  - Search for candidate components
  - Evaluate candidate components
  - Adapt selected components



Cf. Karlsson

Software Architecture

Component-Based Architecture

- **Basis for reuse**
  - Component reuse
  - Architecture reuse
- **Basis for project management**
  - Planning
  - Staffing
  - Delivery
- **Intellectual control**
  - Manage complexity
  - Maintain integrity

Software Architecture

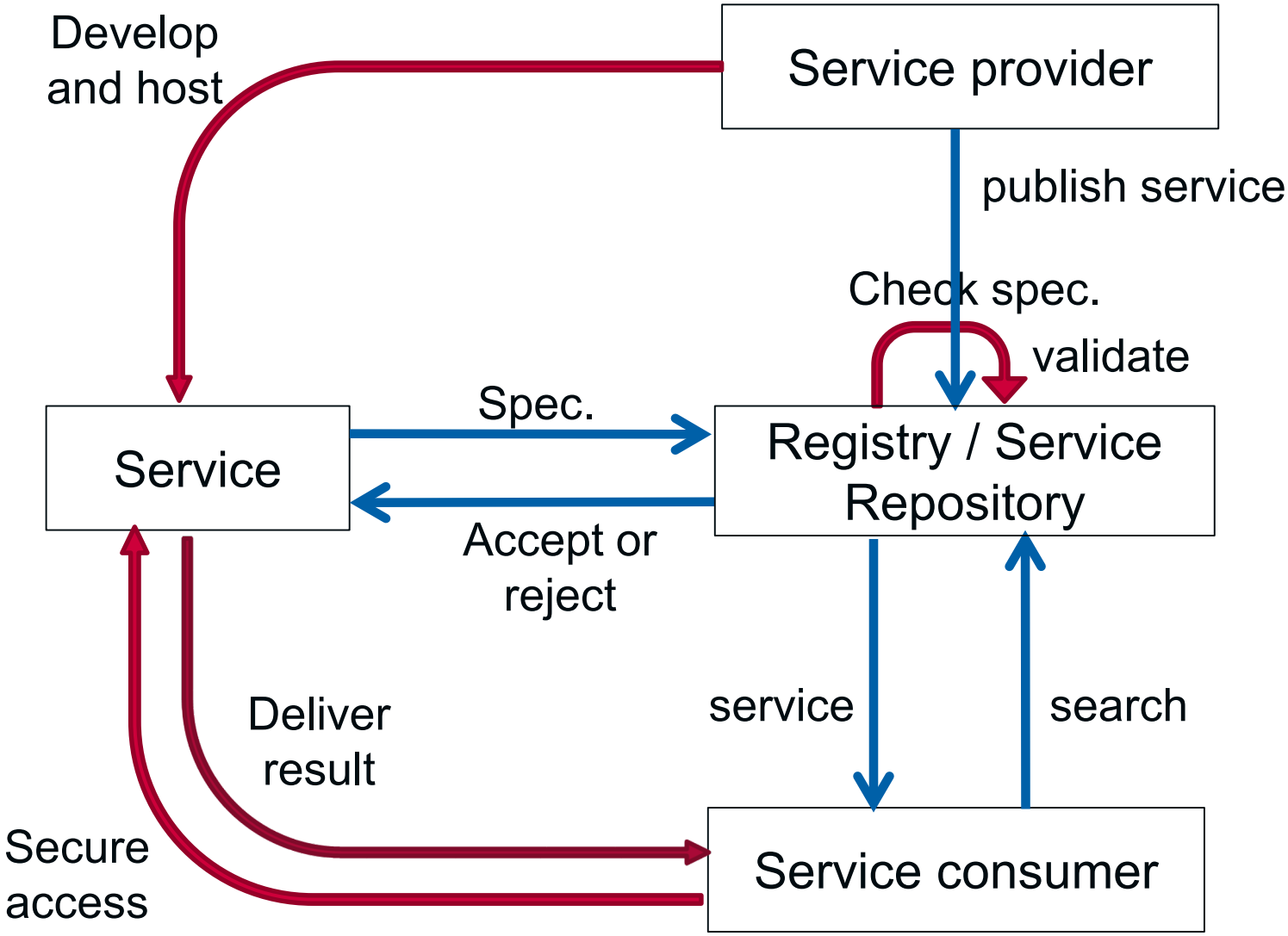
"Service-Oriented Architecture"

- ***OASIS, Organization for the Advancement of Structured Information Standards, 2006):***
  - "A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."

- ***Microsoft:***
  - "Service-oriented architecture (SOA) is a software design and software architecture design pattern based on discrete pieces of software providing application functionality as services to other applications. This is known as service-orientation. It is independent of any vendor, product or technology."

Software Architecture

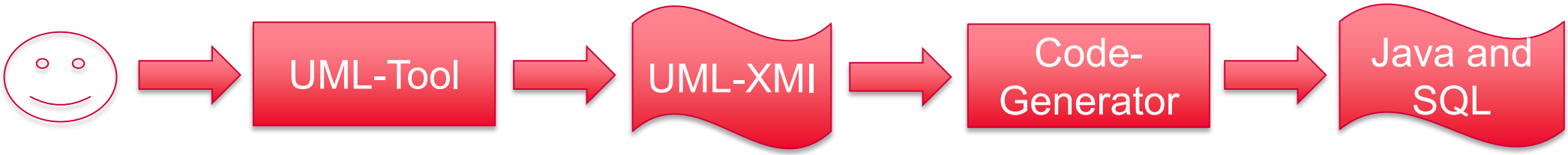
Service-Oriented Architecture



Software Architecture

Model-Driven Development

- **Automatic generation of source code, e. g. Java or SQL for relational database systems**
  - Usually code generation frameworks are used
  - UML models need to be enhanced with marks, such as stereotypes or tagged values

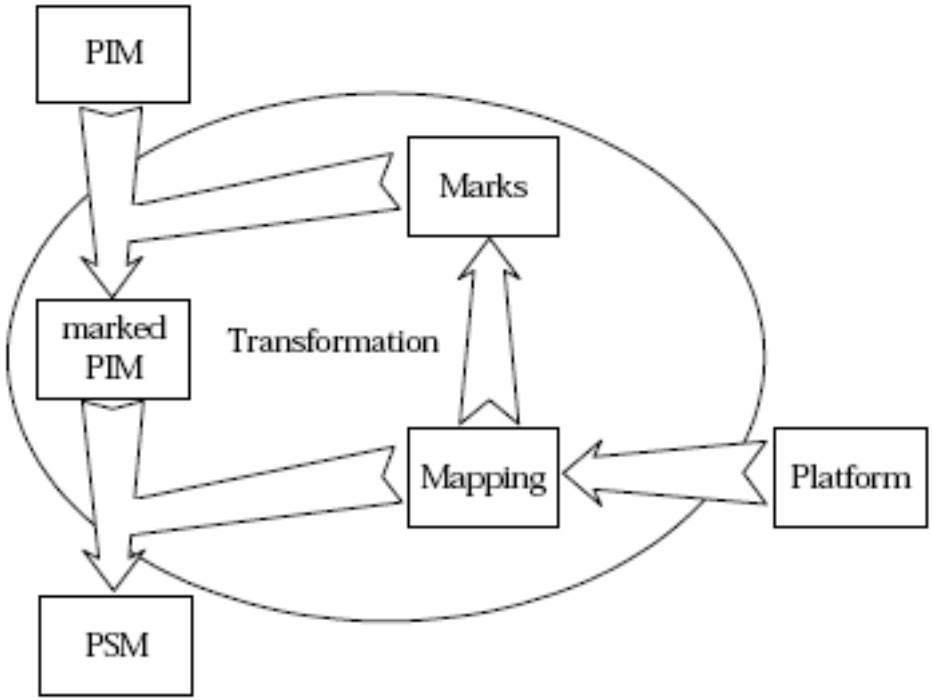


  - *Reverse Engineering*: Create a more abstract representation from a concrete one, e. g. create a UML model from source code

Software Architecture

Model-Driven Development

- **MDA – Model Driven Architecture**
  - PIM – Platform-Independent Model (business- and application logic)
  - PSM – Platform-Specific Model (modeling aspects concerning a technical platform, e. g. a concrete database system)



Cf. OMG

TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Test-Driven Development & Software Evolution

Test-Driven Development

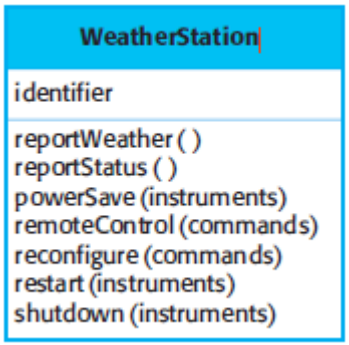
Testing

- **Development Testing**
  - Unit Testing
  - Component Testing
  - System Testing

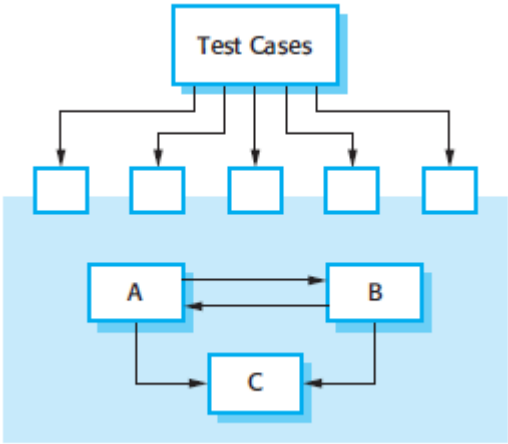
Release Testing

User Testing

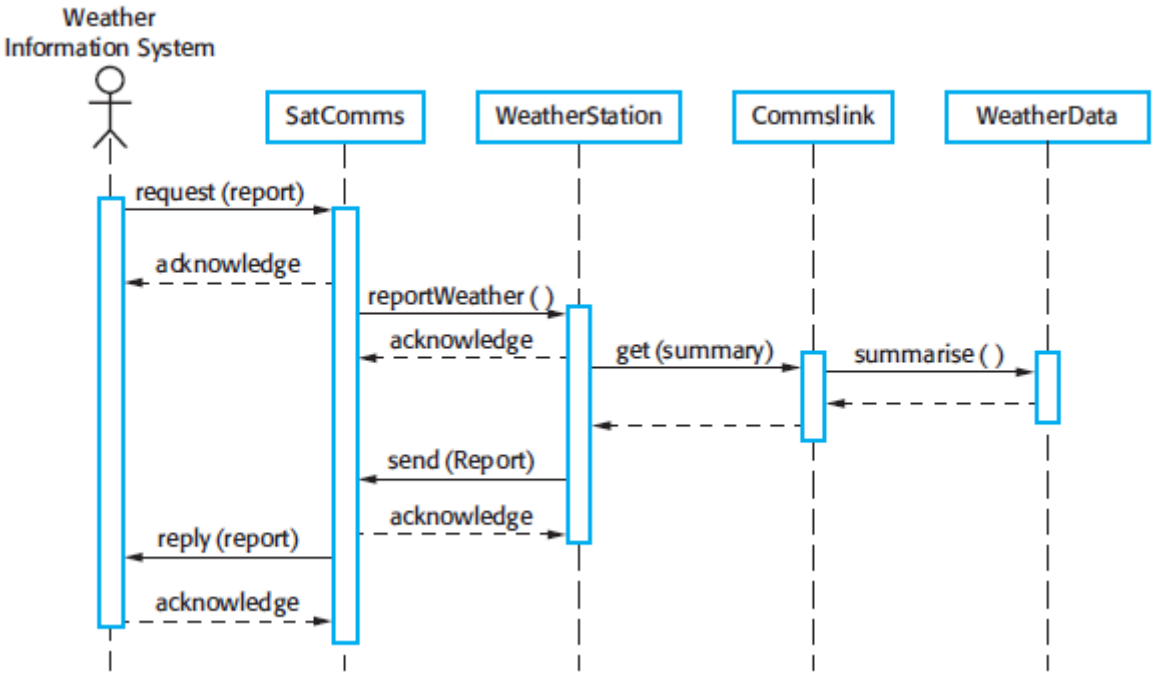
Unit



Component



System

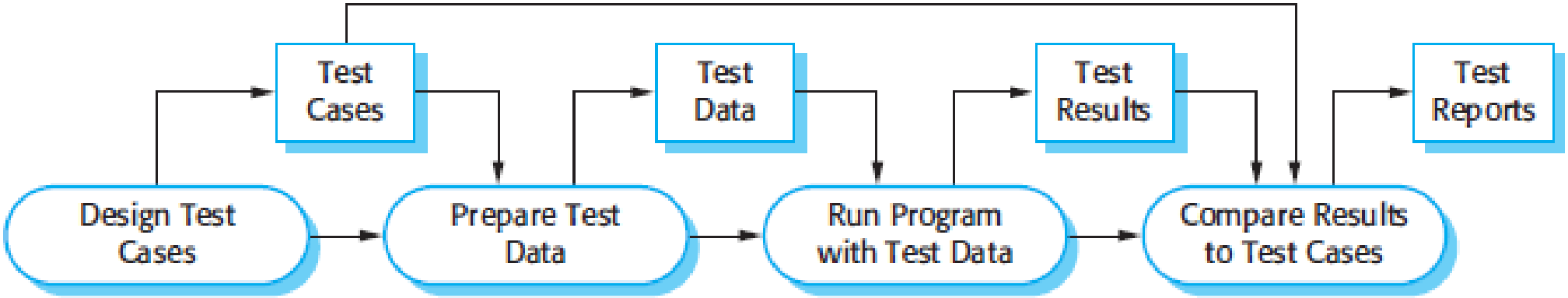


Cf. Sommerville, Chapter 8

Test-Driven Development

Plan-Driven Development

Traditional testing process in “plan-driven development”:

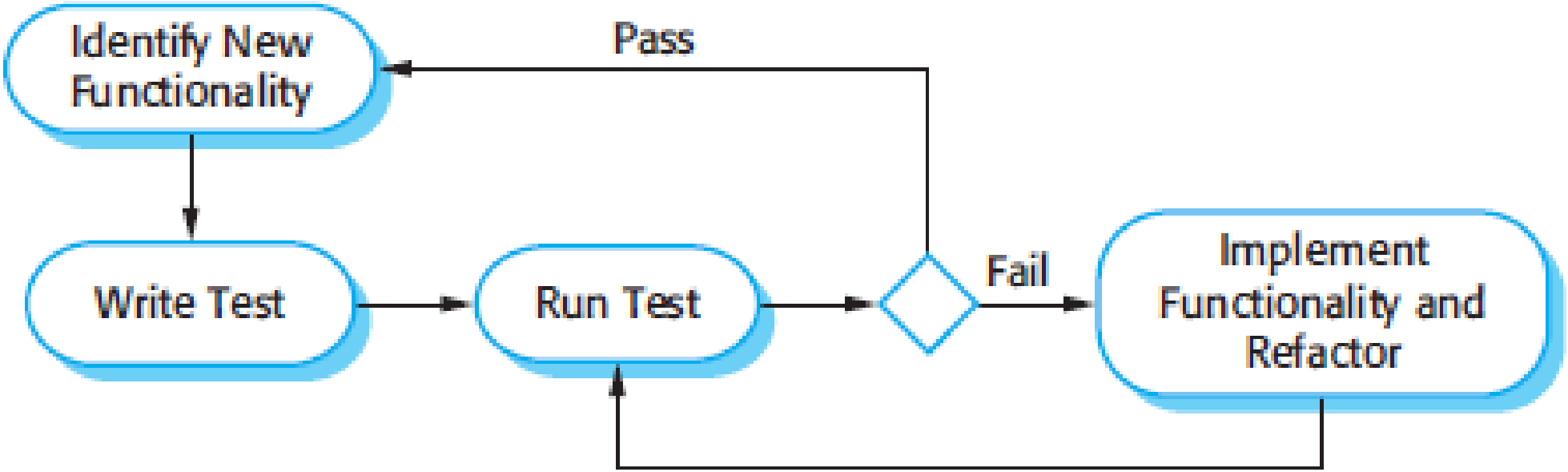


Cf. Sommerville, p. 210

Test-Driven Development

Test-Driven Development

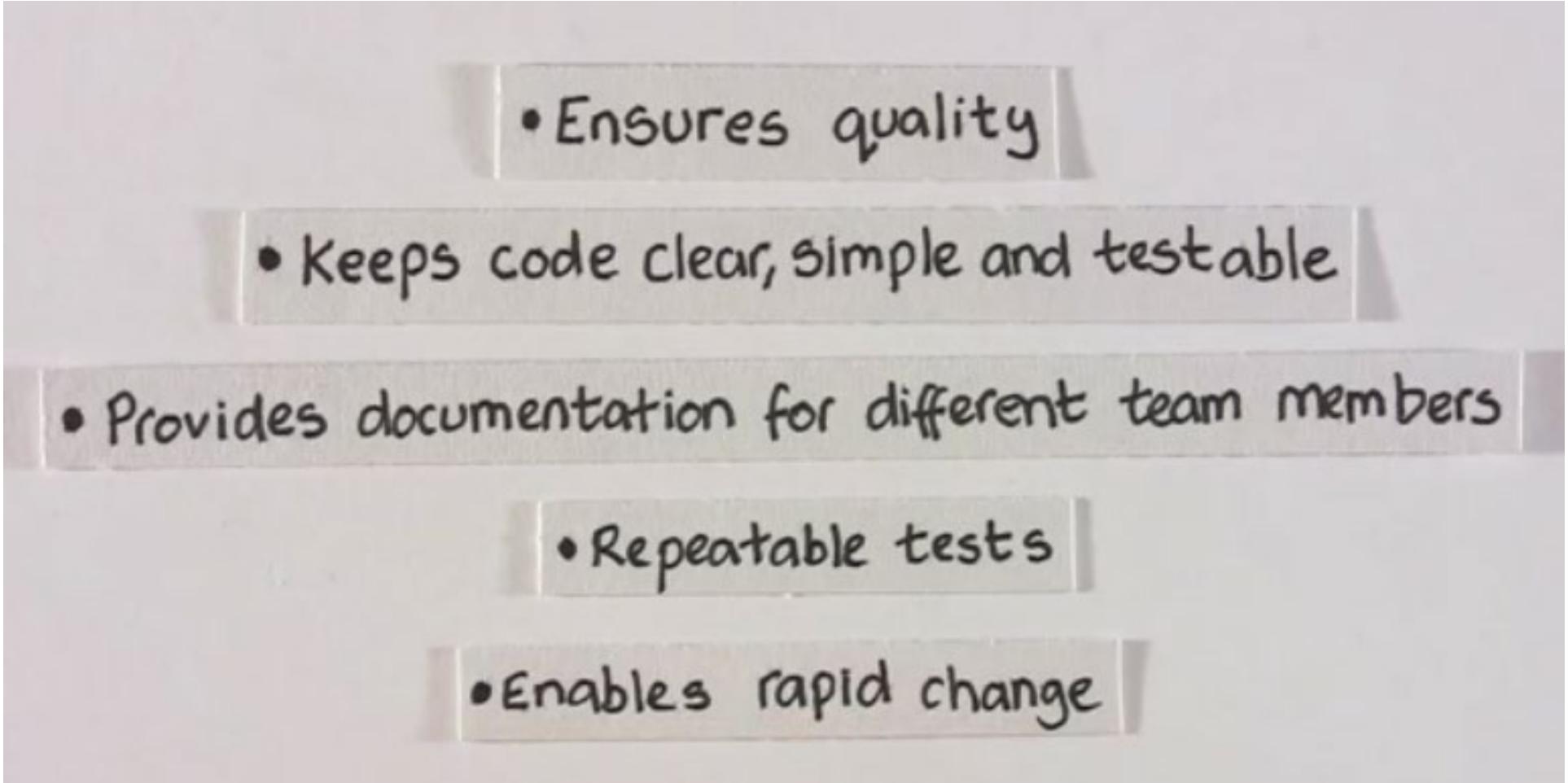
Was introduced as part of agile methods



Cf. Sommerville, p. 222

Test-Driven Development

Test-Driven Development



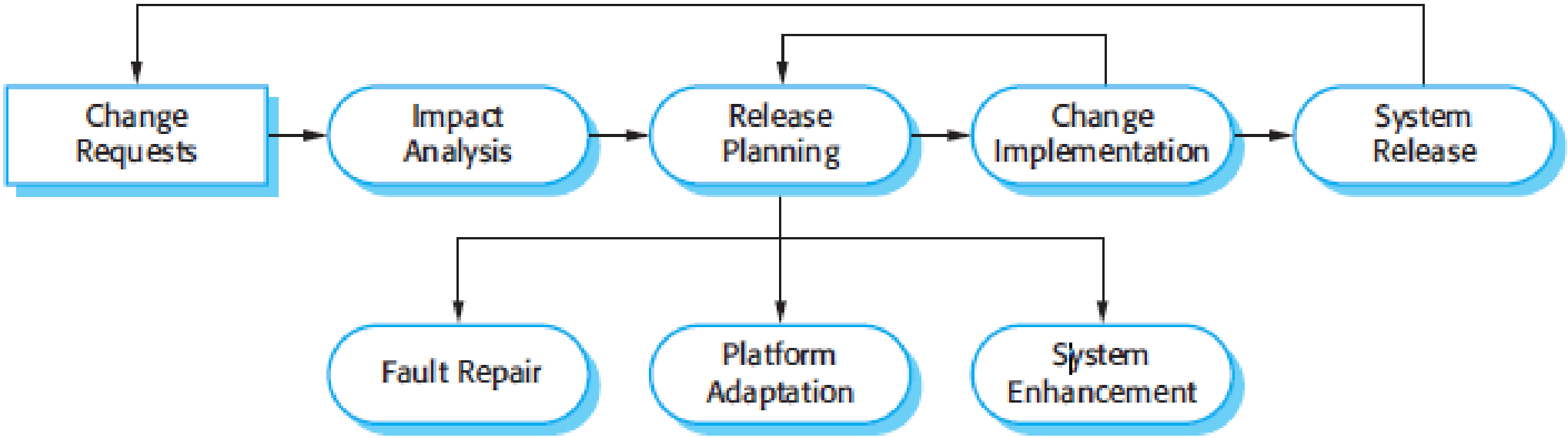
http://www.youtube.com/watch?v=uGaNkTahrIw

Software Evolution

Change

Change cannot be avoided if a system shall remain useful.

Overview of the evolution process:



Cf. Sommerville, p. 238

Literature

- Armour, Frank, and Miller, Granville: Advanced Use Case Modeling, Addison-Wesley, 2001.
- Dahm, Markus H./ Mohos, Csaba: Lean Six Sigma in IT Management: Enhancing Quality and Productivity, Erich Schmidt Verlag, 2012
- Evans, Eric: Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2003.
- Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John: Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.
- Herzum, P. and Sims, O.: Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise. OMG Press, John Wiley & Sons, 2000.
- Karlsson, E.-A. (Editor): Software Reuse. A Holistic Approach. John Wiley & Sons, 1995.
- Kruchten, Philippe: *The Rational Unified Process—An Introduction (3rd Edition), Reading, MA:* Addison-Wesley, 2003.
- Kurniawan, B.: Java for the Web with Servlets, JSP, and EJB, New Riders Publishing, Chapter 17, 2002.
- OMG: BPMN 2.0 by Example, http://www.bpmn.org/, 2010.

Literature

- Poppendieck, Mary and Tom: Implementing Lean Software Development, Addision-Wesley, 2007.
- Pyzdek, Thomas and Keller, Paul A.: The Six Sigma Handbook, Third Edition. New York, NY: McGraw-Hill, 2009.
- Reinfurt, Lukas; Falkenthal, Michael; Breitenbücher, Uwe und Leymann, Frank: *Applying IoT Patterns to Smart Factory Systems*. In: Proceedings of Advanced Summer School on Service Oriented Computing, IBM Research Division, 2017, S. 1-10.
- Schwaber, Ken: Agile Project Management with Scrum. Microsoft Press, 2004.
- Slama, Drik/ Puhlmann, Frank/ Morrish, Jim/ Bhatnagar, Rishi: Enterprise IoT: Strategies and Best Practices for Connected Products and Services, O'Reilly, 2015.
- Sommerville, Ian: Software Engineering. Ninth Edition, Pearson, 2010.
- Voulgaris, Zacharias: Data Scientist: The Definitive Guide to Becoming a Data Scientist, Technics Publications, 2014
- Yacoub, Sherif M. and Ammar, Hany H.: Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems, 2003.

TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Thanks for your attention!

Prof.‘in Dr. Jessica Rubart
jessica.rubart@th-owl.de

DEPT 8 – Environmental Engineering and
Applied Computer Sciences

Business Information Systems

Prof. Dr. Robert Mertens
mertens@hsw-hameln.de

Lecturer of DEPT 5
(Electrical Engineering and Computer Science)

Anwendungsentwicklung und Medieninformatik