

Requirements Engineering

Software Engineering Theory

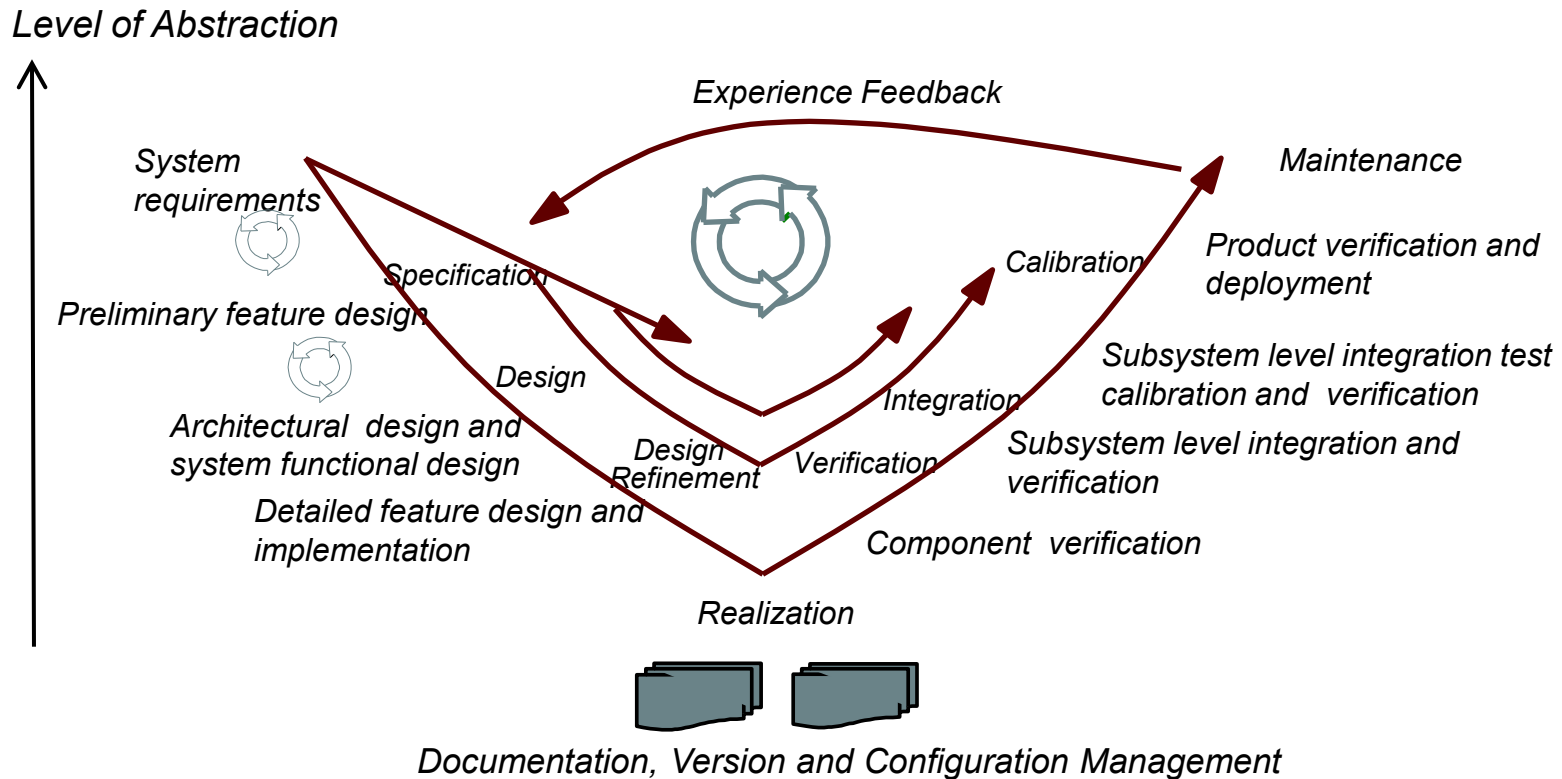
Daniel Varro / Lena Buffoni / Kristian Sandahl
Department of Computer and Information Science

Assignment #1: Requirements

- Solve the problems by reading and reflecting about the concepts introduced today and on next week.
- Submit before the deadline according to instructions on the homepage.
- You may work in groups of 1-2 people
- You may get 0-4 bonus credits for the written exam
- Make sure to avoid plagiarism,
run <http://noplaiat.bibl.liu.se/default.en.asp>

Requirement formalization in product life-cycle

Flaws in system requirements cause large cost increase in product development, or even completely failed projects



Definition and Categorization of Requirements

*What are requirements?
How to categorize them?*

What is a software requirement?

5

**Course
standard**

- Intuitively:
 - A property or behaviour we want from a system
- More formally:
 - *“Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problems.”*

(Kotonya and Sommerville, 2000)

- *“A requirement is something the product must do or a quality it must have.”*

(Suzanne & James Robertson, 2006)

Examples

Web shop:

- *When the user enters type of T-shirt, the system shall show a palette of available colors.*

LIU Intranet:

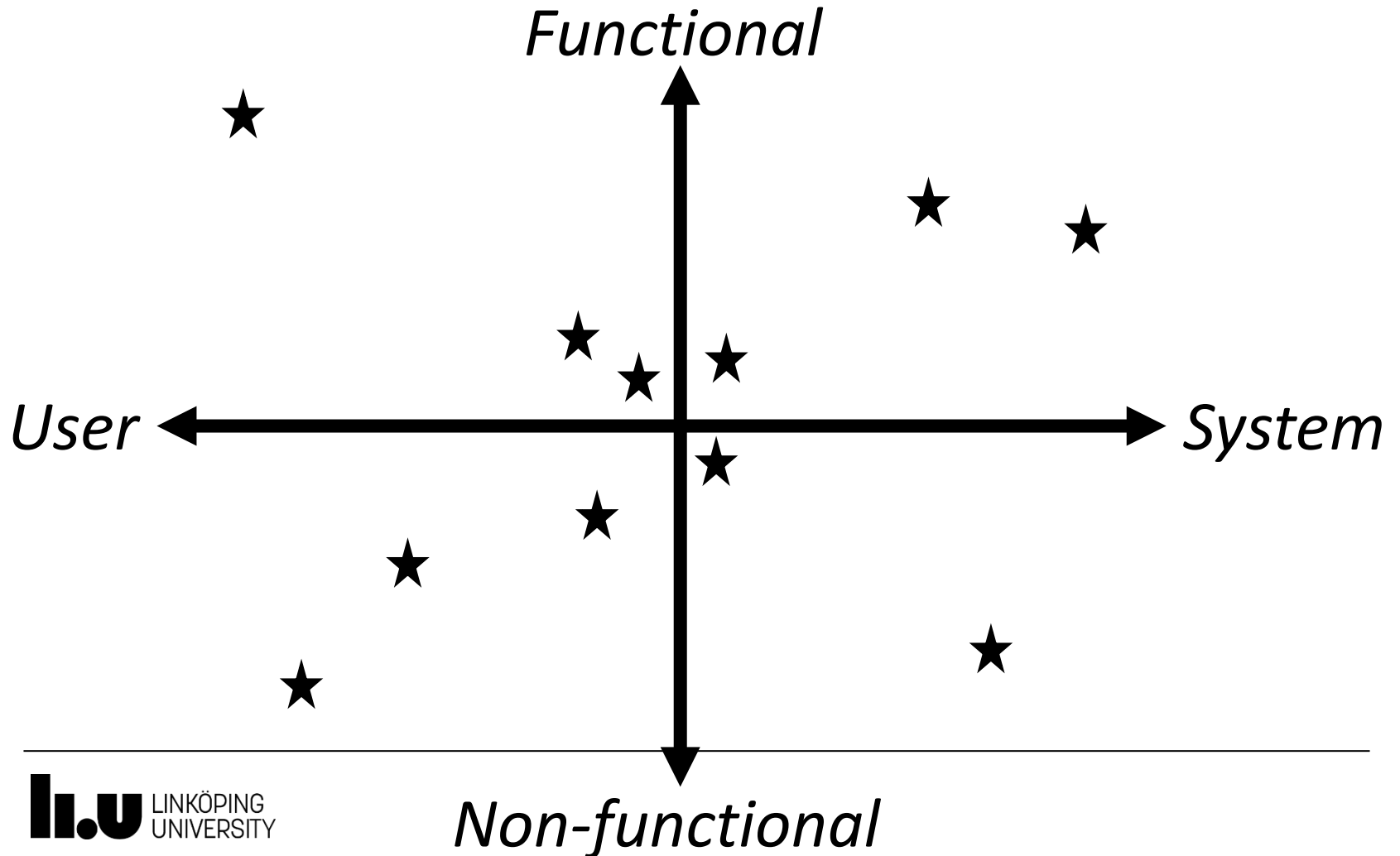
- *A password must contain a combination of literals and numerals and be at least 8 characters long*

Car simulator:

- *The feel of the brakes must be realistic*



The requirements plane





User vs. System Requirements

- **User requirements** (High-level requirements):
 - Describe the services the system is expected to provide in a language (no technical details)
 - Example: „*The system shall generate monthly management reports showing the cost of drugs prescribed by each clinic.*”
- **System requirements** (Low-level requirements):
 - Detailed descriptions of system functions and operational constraints
 - Example: „*On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.*”

Functional vs. Non-functional Requirements

Functional requirements

- Describe what the system should do
- Typically attributed to one component of the system
- Typical form: *“The system shall do ...”*

Non-functional requirements (aka. extra-functional)

- Specify properties of the system as a whole
- (no single component for safety)
- Difficult to check

Non-functional Requirements: Sample Properties and Metrics

Property	Measure
Scalability & performance	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Features

A distinguishing characteristic of a system item (includes both functional and nonfunctional).

(IEEE Std 829)

Higher level stuff, used in advertising:

“The system shall have an SMS delivery notification service.”

*R1: The user phone number shall be pretty-printed in the format:
07x – xxx xx xx*

R3: The user shall be notified at maximum 30 minutes after arrival to the pick-up point.

R2: The delivery tracking system shall interface the DHL system.

How to Write Good Requirements?

What vs. how:

- *Never describe how the system should deliver the requirement*
- *Focus on what the system should do*

Anatomy of a good requirement

*Defines the system
being discussed*

*Use modal verbs:
"shall"*

The online web shop system shall allow the user to access the current content of their cart in less than 5 seconds for 99.9% of attempts.

*Measurable quality
criteria is listed*

*Defines a positive
end result*

3 Characteristics of a Good Requirement

Feasibility

Can be implemented by the system

Necessity

Provides the rationale and the desired end goal

Testability

Contains quantifiable success criteria

Concerns for (functional) requirements

- **Completeness:**
Describe *all* the system features that are required
- **Consistency:**
Avoid conflicts or contradictions between requirements
- (End-to-end) **Traceability:**
Link requirements with design models, source code, test cases which are developed to satisfy them

How do you write a natural language requirement?

- To avoid misunderstandings, always use a **complete sentence**.
- A sentence expresses a complete thought and consists of a **subject** and a **predicate**.
- Use modal verbs: "**shall**", "**will**", and "**must**"
- Don't use: "should", "would", or "might"
- Be careful with quantifiers "all", "never", "each"
- "Timely review should be done as soon as possible, and shall not exceed two working weeks."
(TS/ISO 16949)

Functional requirements

**Course
standard**

- Describe the behaviour or features of a software.
- Can be tested by giving input and checking the output.
- Example: *“The user shall be able to add an item to the shopping basket.”*

Think of the mathematical definition: $f(x)=y$

a function is a relation between a set of inputs and a set of permissible outputs with the property that each input is (deterministically) related to exactly one output.

Software Quality Factors

ISO/IEC 25010 (2011)

See later in the course!

*“The quality of a system is the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value. Those stakeholders' needs (functionality, performance, security, maintainability, etc.) are precisely what is represented in the quality model, which categorizes the product quality into **characteristics** and **sub-characteristics**” (www.iso2500.com)*



+ **Safety** !

Non-functional requirements

- Design constraints, limiting the solution space
 - Example: “*The system shall transfer data in a JSON format.*”
- Quality requirements, possible to measure
 - Example: “*The minimum response time is at most 2.0 seconds in 99% for user requests.*”



Discuss: What is wrong with the following requirement?

The user quickly sees their current account balance on their laptop screen.

The system shall be easy-to-use and require minimal training except for professional mode.

User stories: A lightweight alternative for requirement specification

As a (role) I want (something) so that (benefit)

As a student I want to buy a parking card so that I can drive the car to school.

Priority: 3

Estimate: 4

**Course
standard**

Good for:

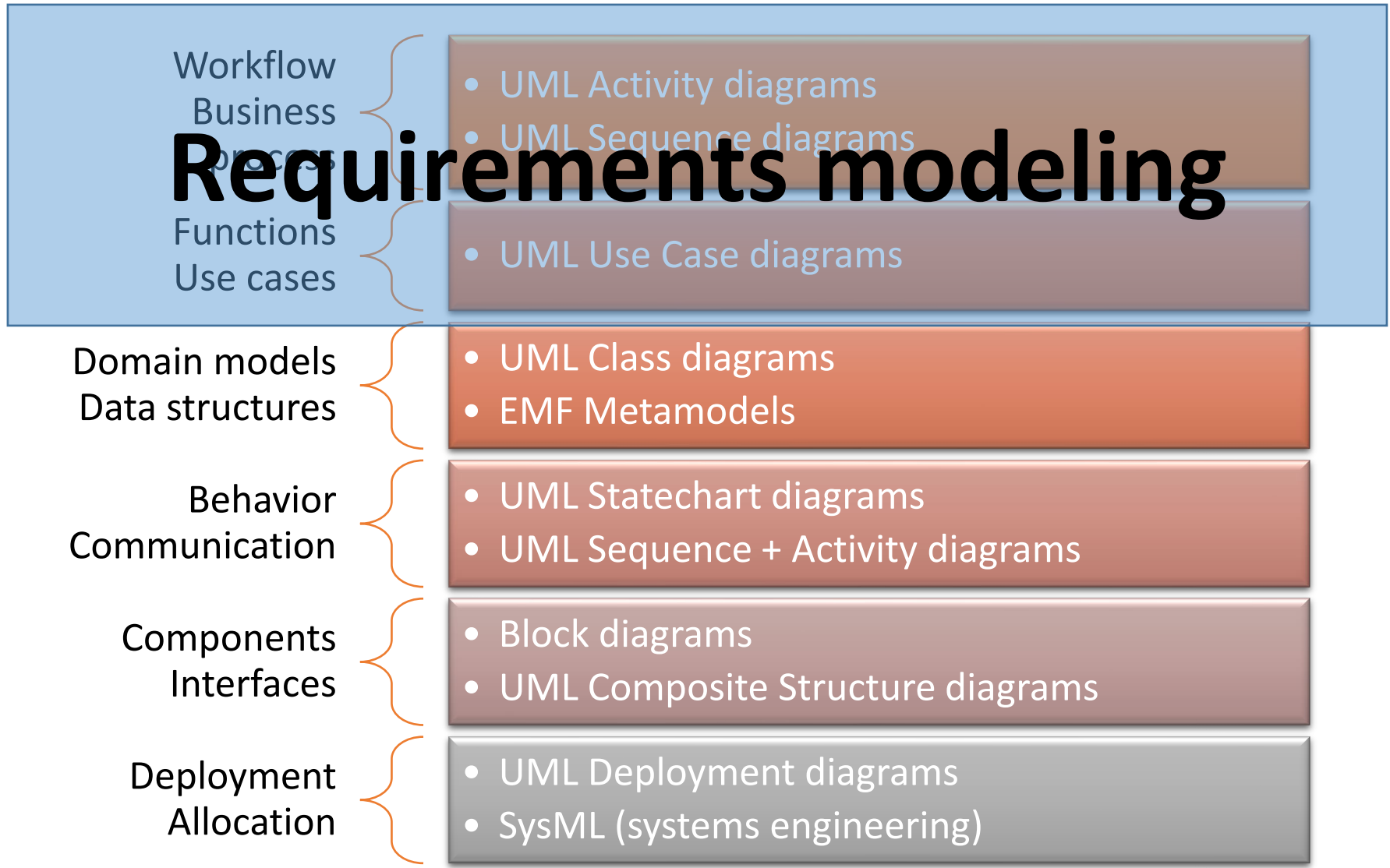
- *Smaller (sub-)projects*
- *Frequent releases*
- *User feed-back*
- *Prototyping*
- *Elicitation*
- *Functional requirements*

See more at:

<http://www.agilemodeling.com/artifacts/userStory.htm>

How to Model Requirements?

What to model?



Use Case Modeling

- *Capture main features / functions of the system*
- *Capture how end users interact with the system*
- *Using a graphical notation*

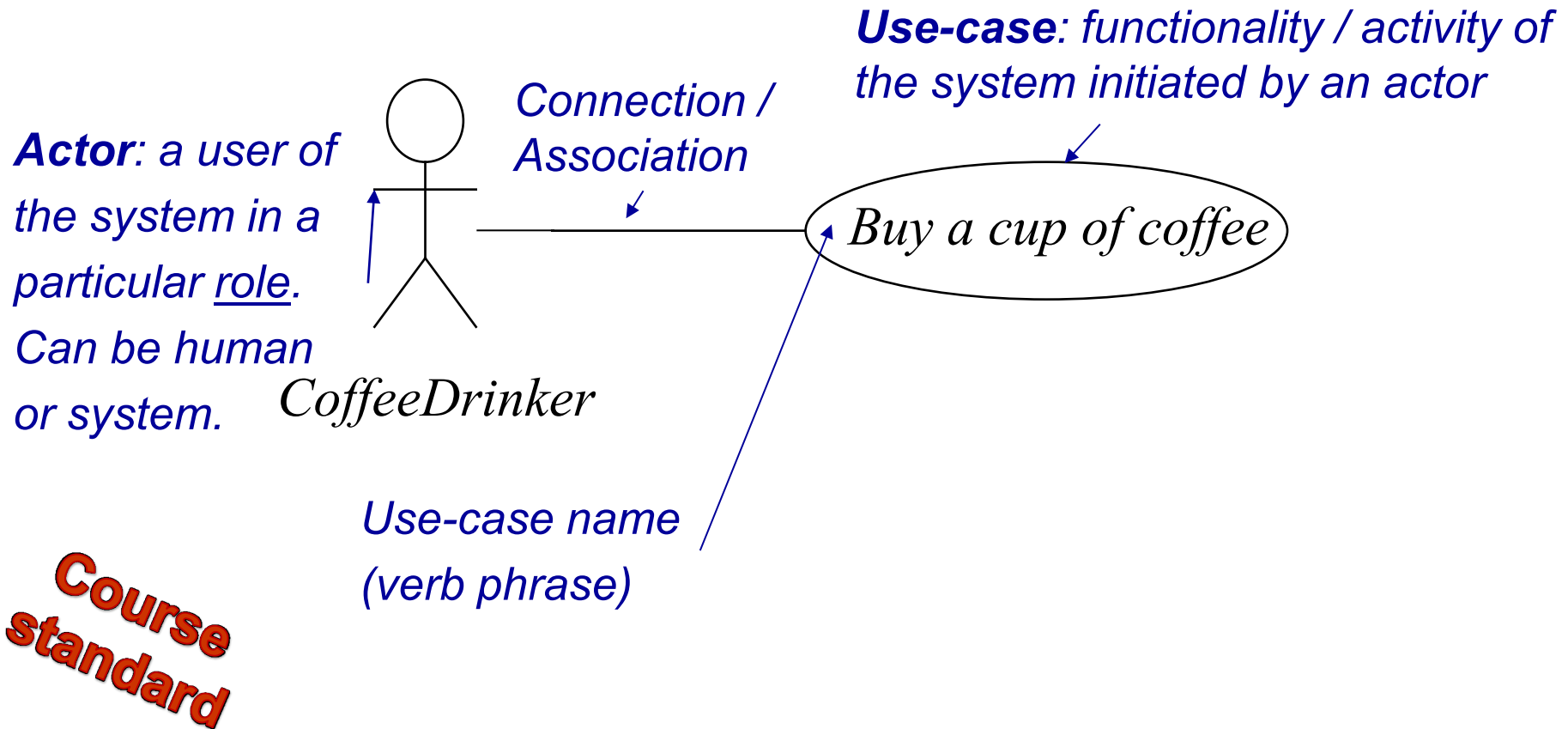
Use-case modelling

A use-case is:

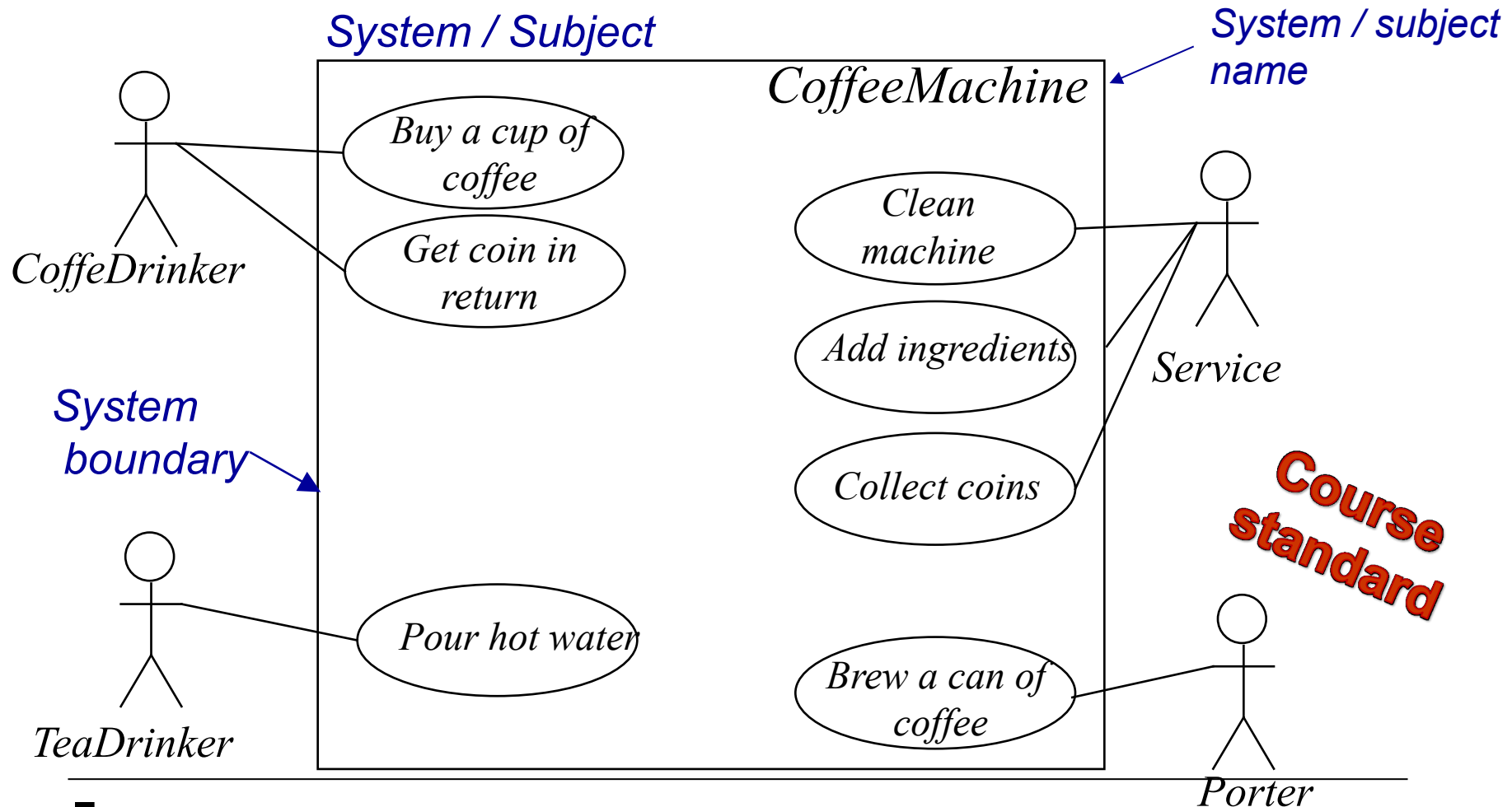
“... a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction or sequence of interrelated events.”

Jacobson, m fl 1992: Object-oriented software engineering. Addison-Wesley

Core Concepts in a UML Use Case diagram

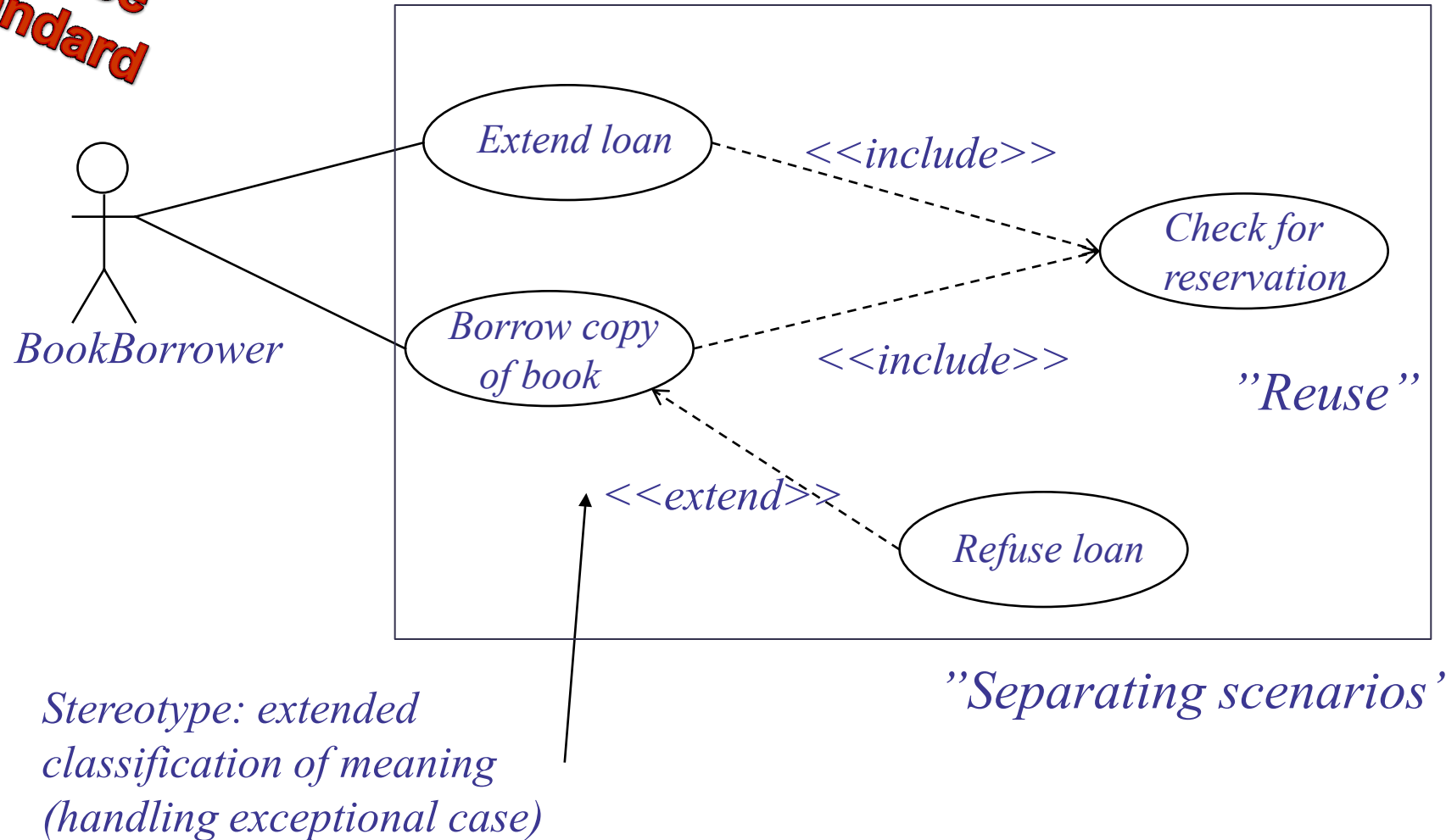


Use-case diagram for the coffee-machine



Relations between use-cases

**Course
standard**

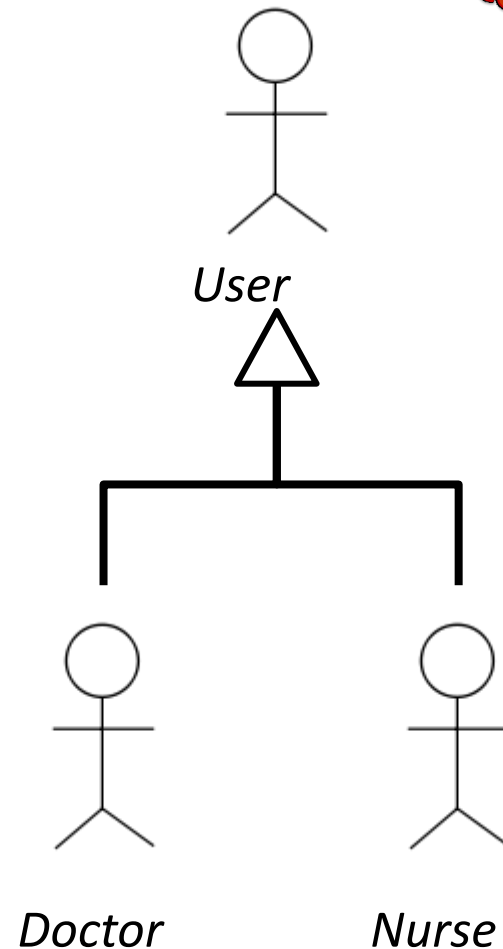


Generalization between Actors

**Course
standard**

Generalization:

- *The child can initiate each UC its parent can access*



Generalization may exist between UCs, but it is rare so it is not detailed further

Traceability of Use Cases to Other Artifacts

- Each high-level requirement needs to be addressed by at least one UC
- Each UC will be refined to at least one business method (in implementation)
- One UC → a set of interrelated scenarios with a single user's goal
- One UC → at least 1 (system-level) test case,
 - but more typically one for each scenario

Tips for writing good use cases

- Start by identifying the list of actors
 - remember systems can also be actors, e.g.: PayPal
 - be consistent with the breakdown
- Describe “sunny day” use cases (main success scenario)
- Use them to produce “rainy-day” use-cases
- Agile-workflow? Iterate on the use-cases

Best Practices for Use Case Diagrams

For diagram arrangement

- Put the most important Actors and UCs to the top-left corner
- Use several diagrams to avoid clutter (at most 5-7 UCs per diagram)
- Actors on the left / right, UCs in the middle
- You may have overview diagrams (e.g. all actors on a single diagram depicting generalization but no UCs)

Capturing Detailed Scenarios

Three alternatives to provide more detailed descriptions of use cases / requirements / user stories?



#1: Structured Scenarios for Detailing Use Cases

**Course
standard**

Scenario: Starts with a **user story** explaining

- Who is using the system
- What they are trying to accomplish?

Step:

- Numbered discrete steps
- Performed by either the System or the User

Main Success Scenario (MSS): Basic Path

- Describes the sequence of steps when everything goes as expected

Alternate / Exception Path

- Alternative sequence of steps when one (or more) step in the MSS fails to complete

Main success scenario

1. Call Handler (CH) responds to call
2. CH encodes incident details
3. CH validates incident location with map gazetteer
4. CH terminates call
5. CAD software assigns incident to relevant Allocator

Alternative scenarios

2a. CH determines calls does not require intervention

2b. Caller or Phone network terminates call

2b.1 CH determines calls does not require intervention

2b.1a CH determines call could require intervention

#2: Scenario Description in Gherkin (BDD)

35

- **Feature**: a requirement captured in the form of a **user story**
 - Example:
As a retail customer,
I want to return an electronically purchase merchandise in 14 days,
so that the refund will be processed
- **Scenario**: Acceptance criteria for each user story exemplified by one or more scenarios

- **Given** a certain scenario

Precondition:

initial context / conditions that must hold to run the scenario

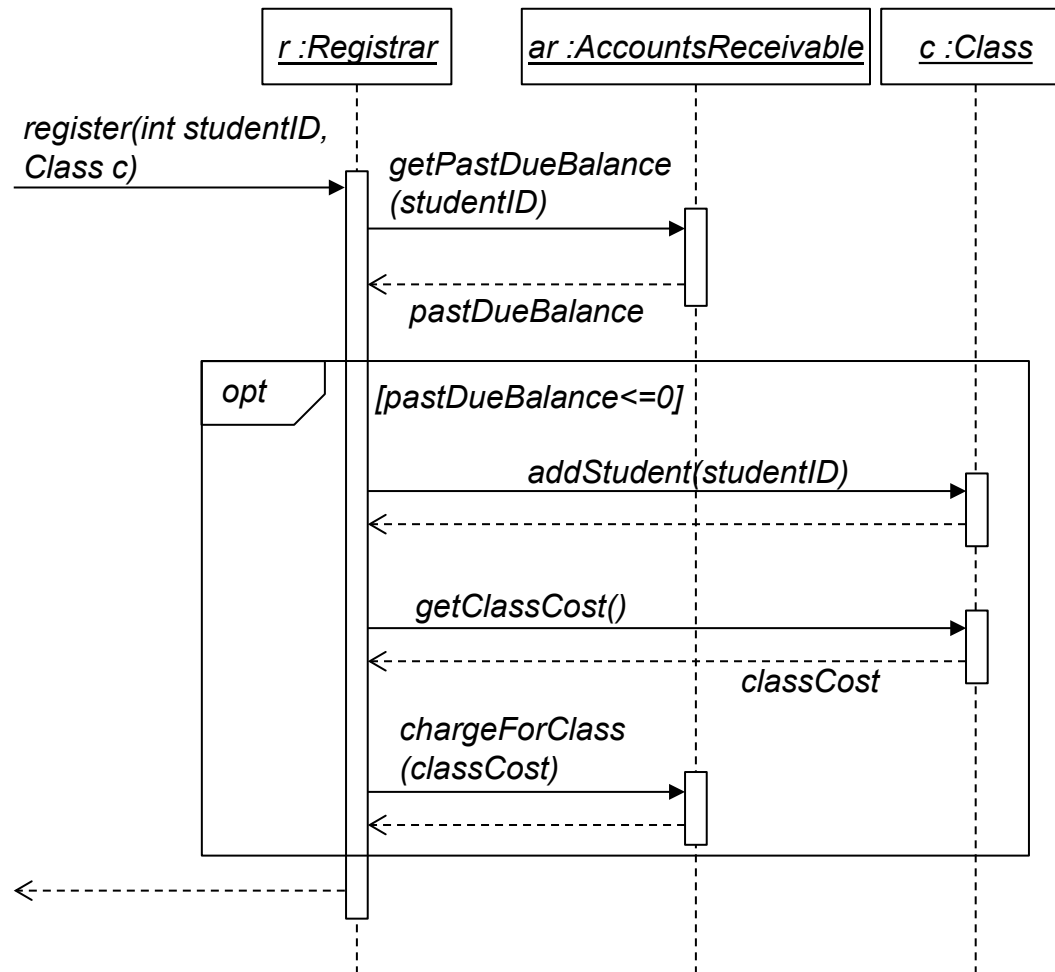
When an action takes place

Action: event that triggers the execution of the scenario (e.g. user action)

Then this should be the outcome

Postcondition: expected outcome to be observed after scenario execution

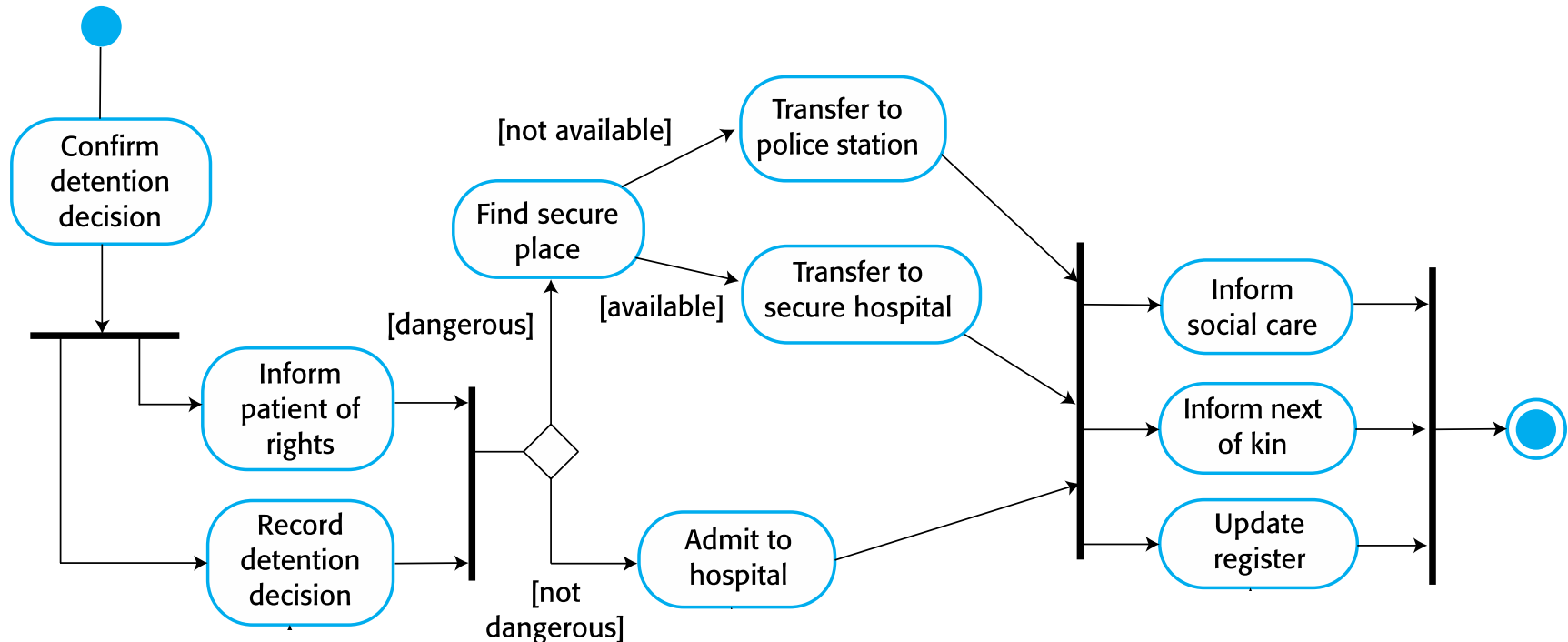
#3: UML Sequence Diagrams (next lecture)³⁶



Workflow Modeling

*Describes how the system fits into the business context
(e.g. the business process operates)*

Activity diagrams for workflow modeling



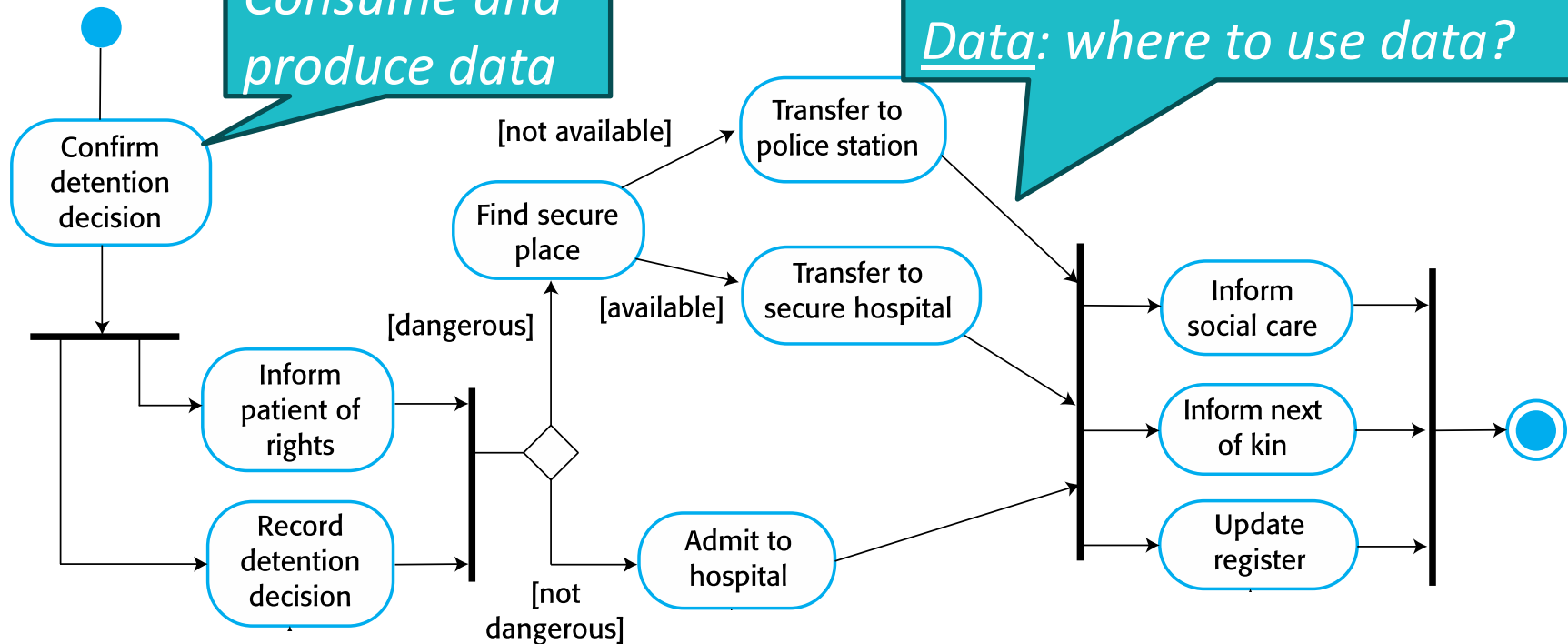
Activity diagrams: Basic elements

Activity:
Consume and produce data

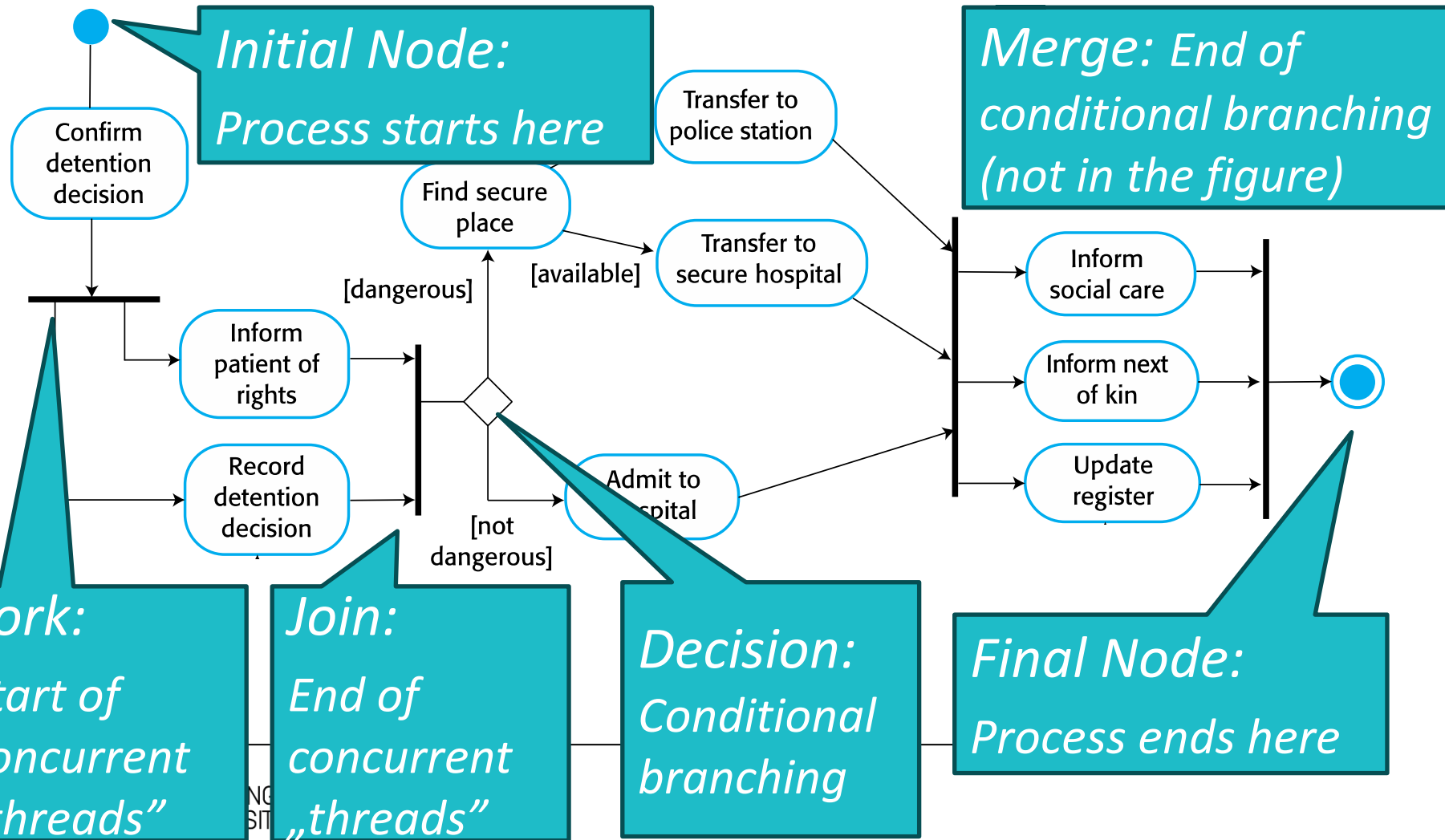
Flow:

Control: what comes next?

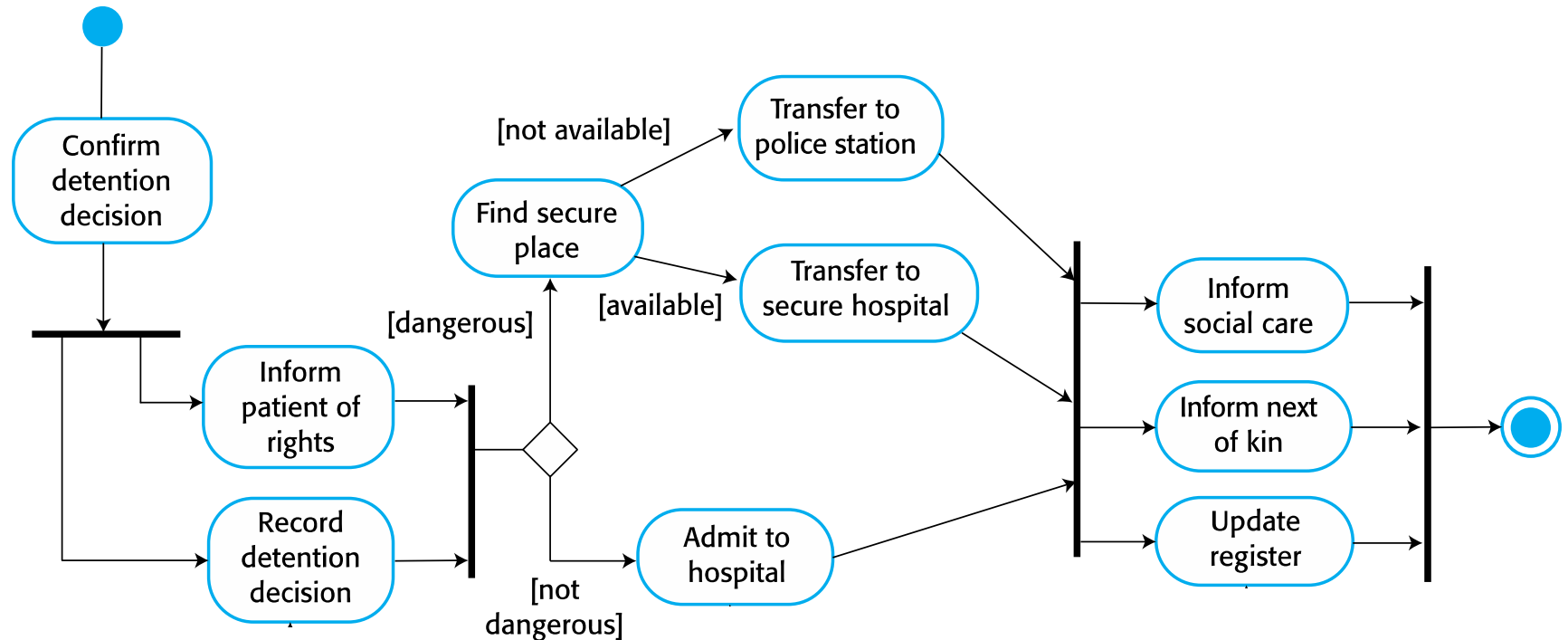
Data: where to use data?



Activity diagrams: Control nodes



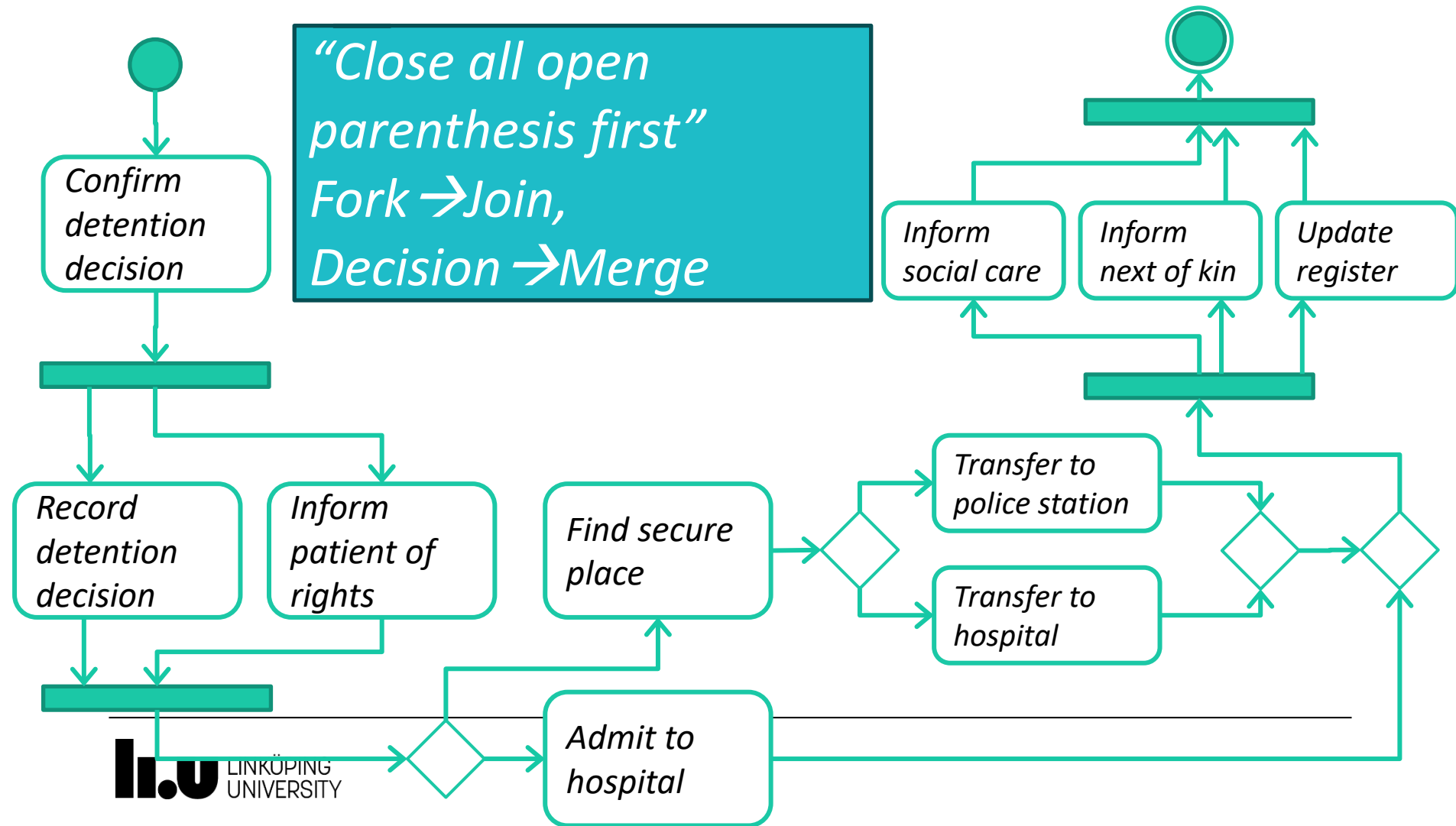
Activity diagrams



Find a semantic inconsistency in the diagram!

Correctly:

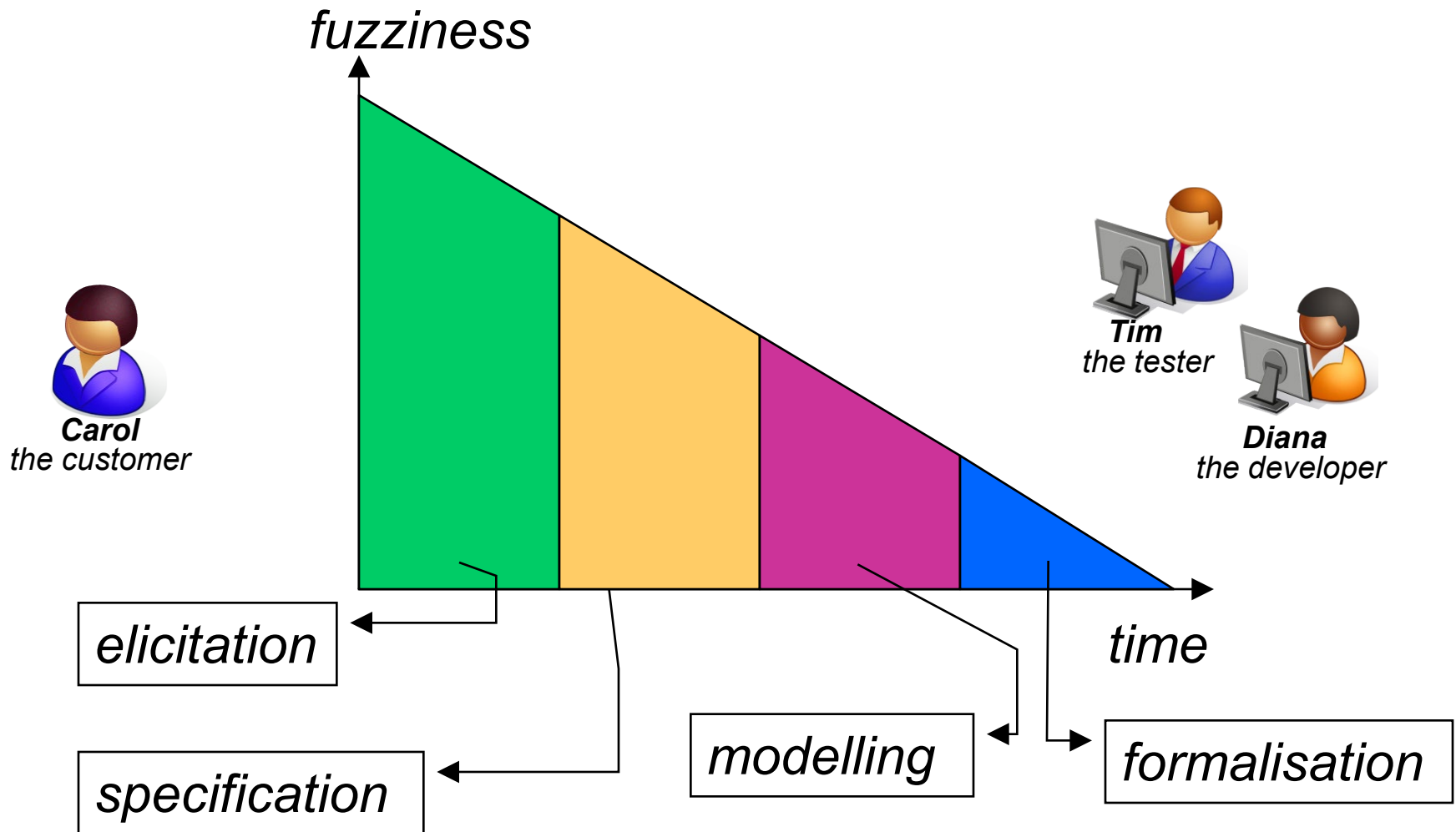
*“Close all open parenthesis first”
Fork \rightarrow Join,
Decision \rightarrow Merge*



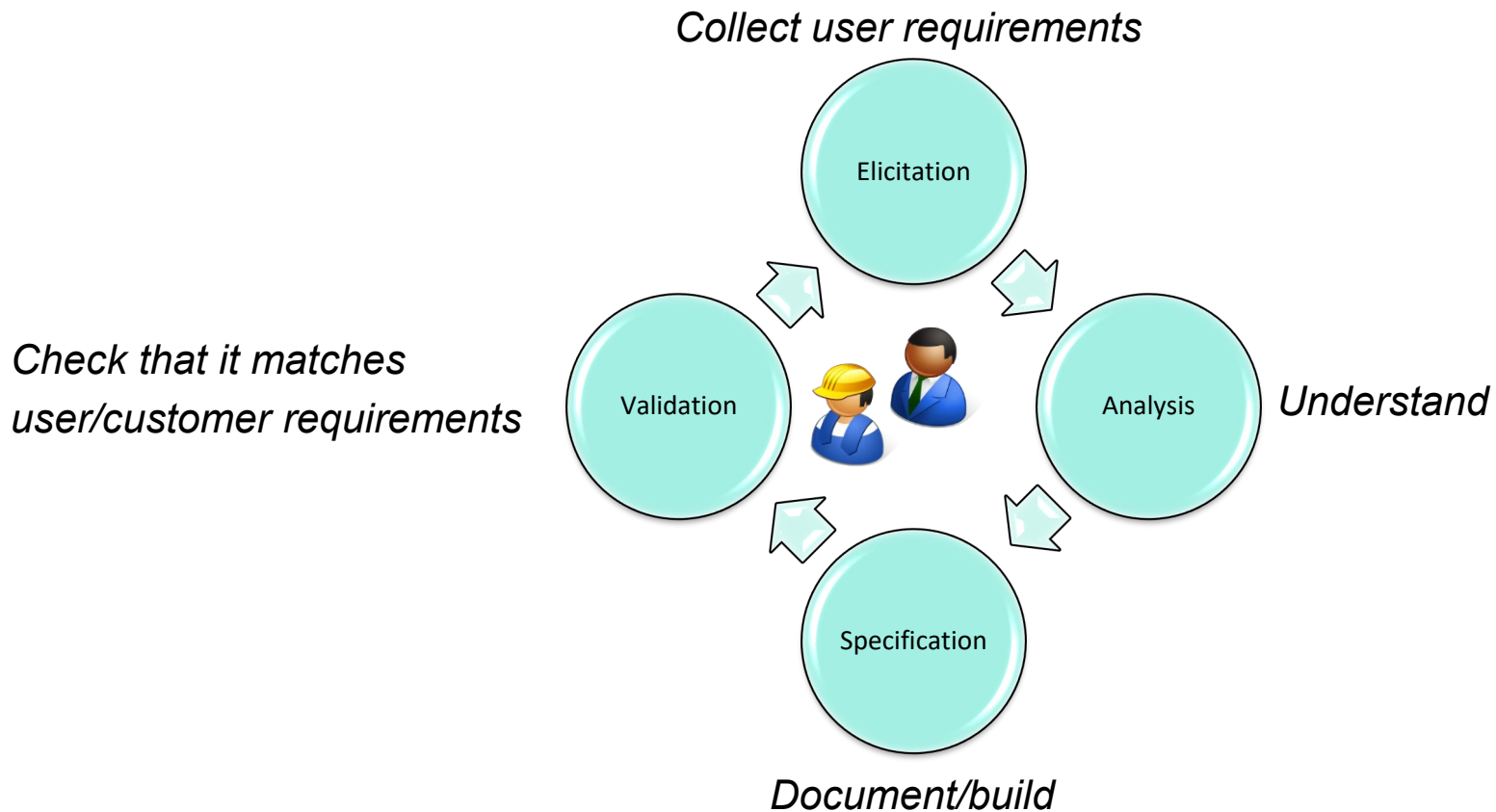
Requirements Elicitation Process

The role of requirements in the life-cycle

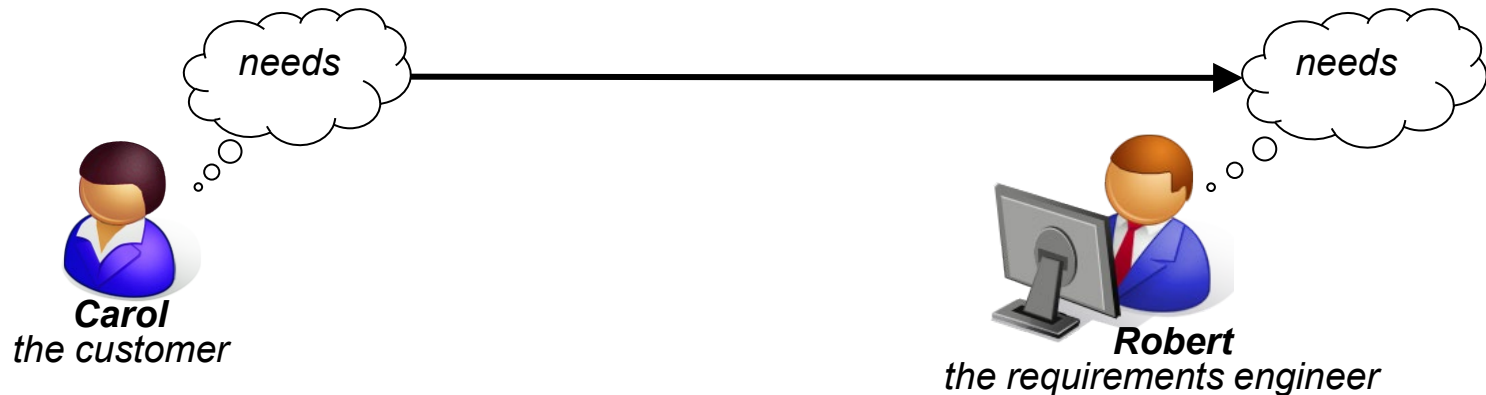
44



Iterative process



Elicitation



Purpose:

- Understand the **true** needs of the customer
- Trace future implementation to needs

Sources:

- Goals
- Domain knowledge
- Stakeholders
- Environment

Techniques:

- *Interviews*
- *Scenarios*
- *Prototypes*
- *Facilitated meetings*
- *Observation*

Interviews

Process:

- Start
- Q & A
- Summary teach-back
- Thank you!
- What's next

Kinds:

- Structured
- Unstructured

Tips:

- Be **2** interviewers – shift roles
- Plan the interview
- Don't stick to the plan – use feelings
- Let the customer talk
- Alternate between open-ended and yes/no questions
- Prepare ice-breakers
- Probe thinking
- Look for body language
- Think of human bias
- Why do you get the answers you get?



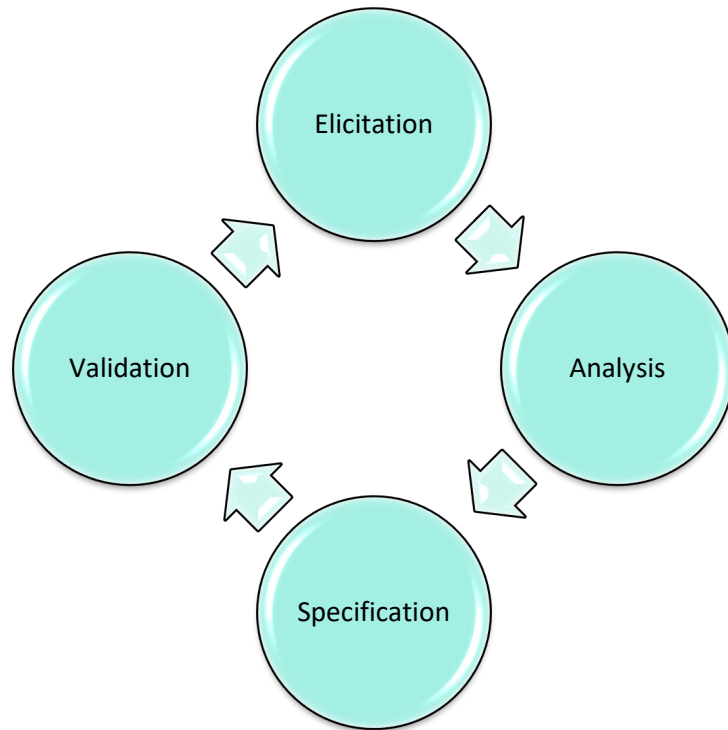
If Henry Ford asked his customers what they wanted, they'd have said a faster horse.

Prototyping

- Consistent with requirement specification
- Clear scope
- Provide and execute a test scenario
- Minimize throwaway code



Requirements analysis goals



- Detect and resolve conflicts between requirements
- Discover bounds of software
- Define interaction with the environment
- Elaborate high-level requirements to derive detailed requirements
- Classify requirements for more effective management

Often accomplished with requirements modelling

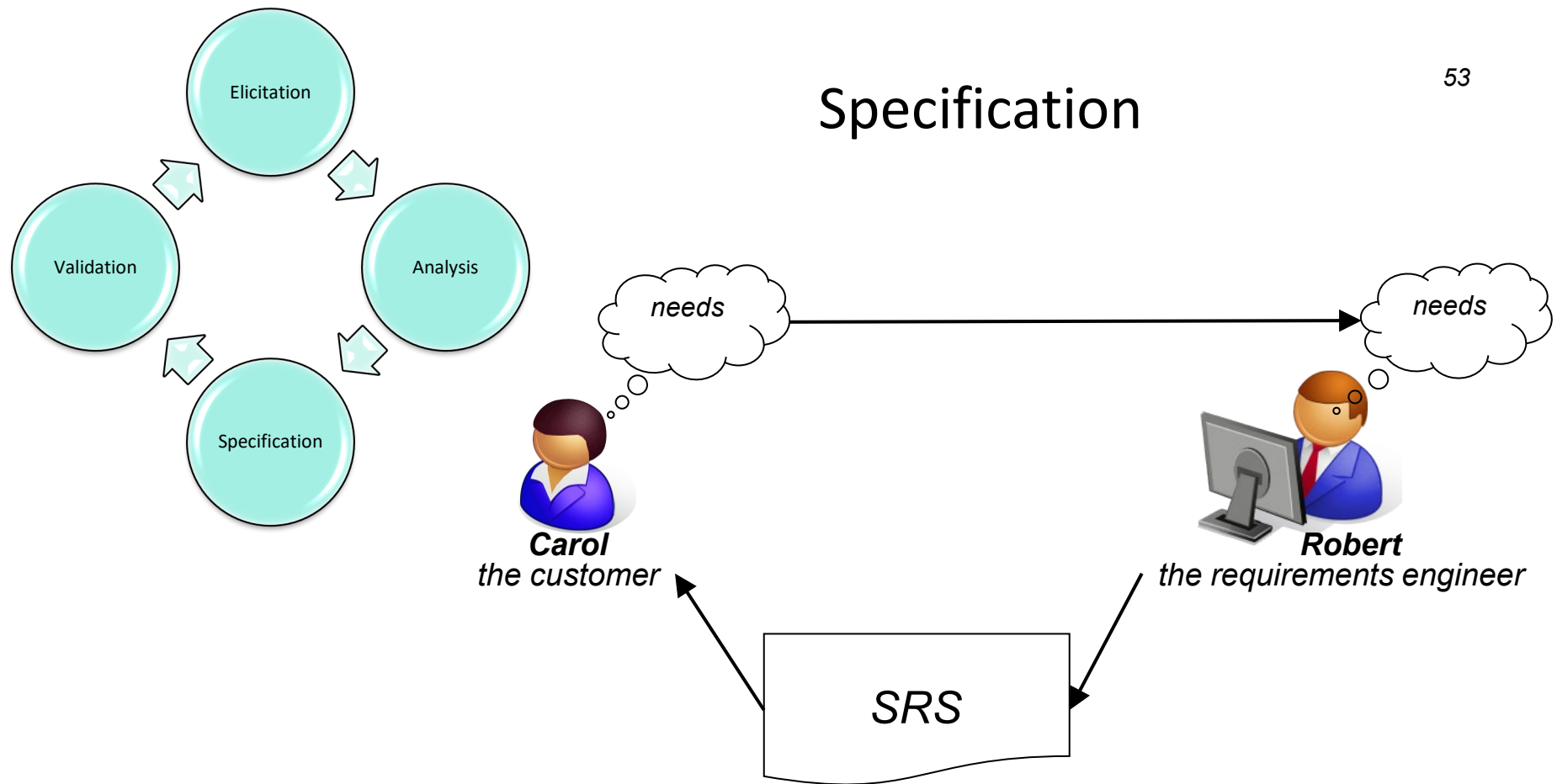
Requirements classification

- Functional vs non-functional requirements
- Source
- Product vs process requirements
- Priority
- Scope in terms of affected components
- Volatility vs stability

Requirement prioritization

- Different prioritization strategies
(numerical, categories, 100-dollar method, ...)
- Identify and order priorities for trade-off requirements
Eg: *I want a fast web-site with high resolution graphics*
- Good ratio between different priorities – all requirements cannot be critical
- Cost, technical-risk, time frames are all considerations in prioritizing
- Low priority requirements are not a maybe/someday wish-list!

Specification

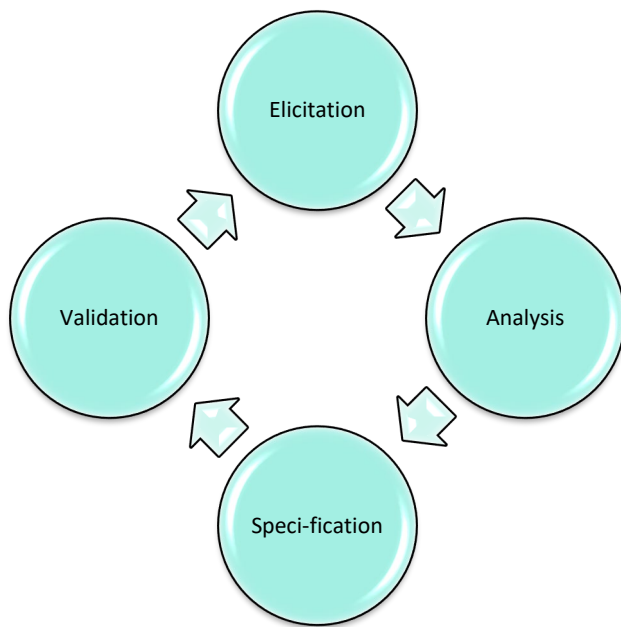


- There is no perfect specification, but you can write a good one
- The Requirement Specification (RS), or SRS (Software RS) avoids many misunderstandings
- The RS is of special importance in outsourcing programming

Iterative process

- The requirement specification is a living document
- Priorities change
- Technical issues can lead to modifications
- Some requirements may need additional elicitation





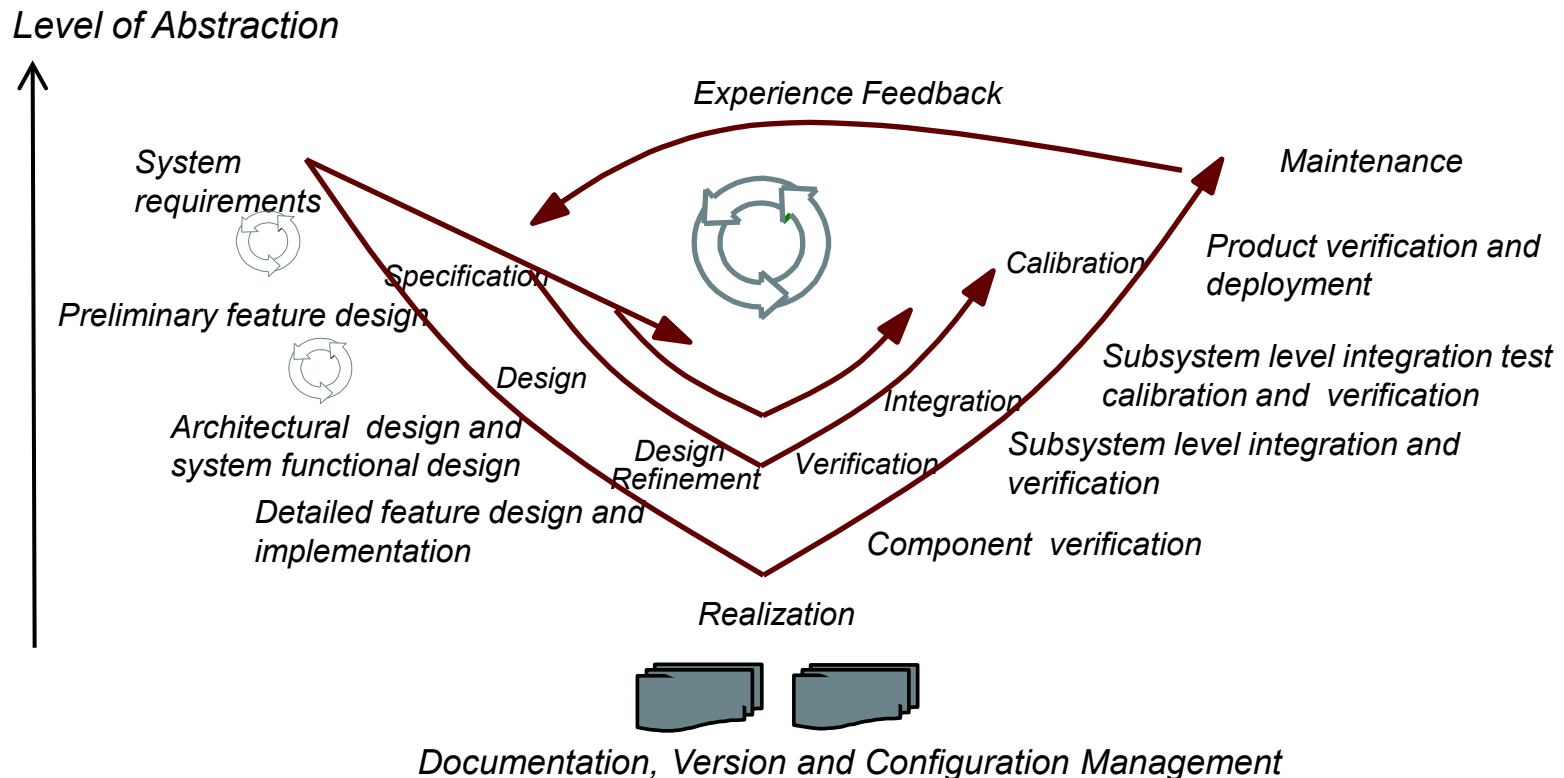
Means of validation

- Prototyping
- Simulation
- Software Reviews
- Model checking
- Formal proofs
- Acceptance testing

Agile Requirements Engineering

What about requirements + agile development?

Moving from iteration in a single stage to iterating over the whole process



Agile requirement modelling

No single standardized definition of the process
BUT

Several workflow models emerged, including:

- **Behavior Driven Development**
- Scaled Agile Framework (SAFe) Requirements Model
- Specification by example

General concepts of agile RM:

- Frequently based on **user stories**
- Requirements are elicited at a general level and refined with each iteration
- Often relies on classic techniques for initial elicitation
- Stakeholders need to be tightly involved
- The notion of done is important
- Good support for change management is key
- Can be integrated with automated acceptance testing (behavior-driven development)

User stories – lightweight alternative for requirement specification

As a (role) I want (something) so that (benefit)

As a student I want to buy a parking card so that I can drive the car to school.

Priority: 3

Estimate: 4

Good for:

- *Smaller (sub-)projects*
- *Frequent releases*
- *User feed-back*
- *Prototyping*
- *Elicitation*
- *Functional requirements*

See more at:

<http://www.agilemodeling.com/artifacts/userStory.htm>

Problem: Notion of „Done”

As a student I want to buy a parking card so that I can drive the car to school.

Priority: 3

Estimate: 4

Compare with:

A student should be able to complete an online form, indicating contact information and payment method and once it is processed a parking card is sent in the mail.

When is the requirement completed?

User stories might need to be completed by more formal specifications, use-case diagrams and communication with stakeholders

Behavior Driven Development

What is Behavior-Driven Development? 65

- An agile software development process
- Emerged from test-driven development
- Uses a domain-specific language
- Relies on test/release automation tools

Agile process:

- Requirements and solutions co-evolve
- Adaptive project planning
- Flexible change management

Test-driven development:

- Develop your unit tests before developing your functionality
- Repetitions of very short development cycles

Domain-specific language:

- Computer language specific to a target application domain
- Natively uses concepts from the target domain

Test automation:

- Test cases are automatically executed for each commit
- Prevents integration of inconsistent source code

Goals of BDD

- Should be **business-readable** to improve communication
 - Provide a language to help communication between stakeholders: engineers, business analysts, product owners, testers, etc.
 - Active participation of all stakeholders in requirements specification
- Provide a **domain-specific language** to describe
 - Features and their acceptance criteria
 - **System behavior**: What the system should do?
 - **Without implementation details**: NOT how the system should do?
- Promote **test automation for acceptance tests**:
 - Integrated with automated build systems (Gradle, Maven)
 - Executable specification (even when incomplete)
 - Integrated with test automation tools (e.g. Selenium)

Behavior-Driven vs. Test-Driven Development ⁶⁷

- BDD enforces a TDD-like workflow
 1. define a test set for checking the expected behavior;
 2. make the tests fail / pending
 3. then implement behavior;
 4. finally verify that the implemented behavior makes the tests succeed
- **Difference: TDD vs BDD**
 - TDD:
 - Focus on unit testing (class, component)
 - Tests written in a programming language
 - By developer
 - BDD:
 - Focus on acceptance testing (product as a whole)
 - Tests written in a domain-specific language (DSL)
 - Written by business analyst, implemented by developer

Popular Tools for TDD

- Cucumber: General BDD framework
 - Specification language: Gherkin
 - Supports 30 spoken languages
 - Java, Javascript, Ruby, etc.
- Other related tools:
 - JBehave (Java)
 - Behat (Php)
 - SpecFlow (.NET)

How to Organize Requirements?

Organize the Software Requirement Specification

Example from IEEE Std 830-1998

3.2 Functional requirements

3.2.1 Show colors

When the user enters type of T-shirt, the system shall show a palette of available colours.

3.2.2 Add to shopping basket

The user shall be able to add an item to the shopping basket.

3.3 Performance requirements

3.3.1 Response time

The minimum response time is 2.0 seconds.

3.4 Design constraints

The system shall be implemented in PHP.

If a more thorough description is needed

71

3.2.1 Show colors

3.2.1.1 Input variables

Type of T-shirt

3.2.1.2 Output variables

Set of color codes

3.2.1.3 Processing

1. Query database for available colors of Type of T-shirt.
2. Create a set of color codes.
3. Send color codes to palette viewer

3.2 System features

3.2.1 SMS notification

3.2.1.1 Purpose of SMS notification

The system shall have an SMS delivery notification service so customers can collect their delivery as fast as possible.

3.2.1.2 Stimulus/response sequence

When the delivery is has arrived to the pick-up point, notify the user.

3.2.1.3 Associated functional requirements

3.2.1.3.1 Number format

The user phone number shall be pretty-printed in the format:

07x – xxx xx xx

3.2.1.3.2 Change number

.....



SRS contents IEEE Std 830-1998

1 Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms and abbreviations
- 1.4 References
- 1.5 Overview

2 Overall description

- 2.1 *Product perspective*
- 2.2 *Product functions*
- 2.3 *User characteristics*
- 2.4 *General constraints*
- 2.5 *Assumptions and dependencies*
- 2.6 *Lower ambition levels*

3 Specific requirements

3.1 Interface requirements

- 3.1.1 *User interfaces*
- 3.1.2 *Hardware interfaces*
- 3.1.3 *Software interfaces*
- 3.1.4 *Communication interfaces*

3.2 Functional requirements

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements

4 Supporting information

- 4.1 Index
- 4.2 Appendices



SRS contents IEEE Std 830-1998

74

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, acronyms and abbreviations

1.4 References

1.5 Overview

- *Describe purpose of this SRS*
- *Describe intended audience*

- *Identify the software product*
- *Enumerate what the system will and will not do*
- *Describe user classes and benefits for each*

- *Define the vocabulary of the SRS (may reference appendix)*

- *List all referenced documents including sources (e.g., Use Case Model and Problem Statement; Experts in the field)*

- *Describe the content of the rest of the SRS*
- *Describe how the SRS is organized*

2 Overall description

2.1 Product perspective

2.2 Product functions

2.3 User characteristics

2.4 General constraints

2.5 Assumptions and dependencies

2.6 Lower ambition levels



SRS contents IEEE Std 830-1998

1 Introduction

1.1 Purpose

- ***Present the business case and operational concept of the system***
- ***Describe how the proposed system fits into the business context***
- ***Describe external interfaces: system, user, hardware, software, communication***
- ***Describe constraints: memory, operational, site adaptation***

1.2 Scope

1.3 Definitions, acronyms and abbreviations

1.4 References

1.5 Overview

- ***Summarize the major functional capabilities***
- ***Include the Use Case Diagram and supporting narrative (identify actors and use cases)***
- ***Include Data Flow Diagram if appropriate***

2 Overall description

2.1 Product perspective

2.2 Product functions

2.3 User characteristics

2.4 General constraints

2.5 Assumptions and dependencies

2.6 Lower ambition levels

- ***Describe and justify technical skills and capabilities of each user class***

- ***Describe other constraints that will limit developer's options; e.g., regulatory policies; target platform, database, network software and protocols, development standards requirements***



SRS contents IEEE Std 830-1998

1 Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms and abbreviations
- 1.4 References
- 1.5 Overview

2 Overall description

- 2.1 *Product perspective*
- 2.2 *Product functions*
- 2.3 *User characteristics*
- 2.4 *General constraints*
- 2.5 *Assumptions and dependencies*
- 2.6 *Lower ambition levels*

3 Specific requirements

3.1 Interface requirements

- 3.1.1 *User interfaces*
- 3.1.2 *Hardware interfaces*
- 3.1.3 *Software interfaces*
- 3.1.4 *Communication interfaces*

3.2 Functional requirements

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements

4 Supporting information

- 4.1 Index
- 4.2 Appendices

Pros and cons of IEEE 830

Cons

- Takes some time to read and understand
- Very general, needs to be tailored
- Is no guarantee for a good SRS


Pros

- Good checklist
- Many ways to adapt organization
- Many ways to detail requirements
- You don't need to have everything in

Requirements specification

Requirements in a good SRS are:

- Numbered
 - Inspected
 - Prioritised
 - Unambiguous
 - Testable
 - Complete
 - Consistent
- Traceable
 - Feasible
 - Modifiable
 - Useful for:
 - operation
 - maintenance
 - customer
 - developer
 -




R51: A car that feels more comfortable than the previous version

What is “comfortable”?
How much is “more”?

Who is the reader?

Summary

- Definition and categorization of requirements (functional vs. non-functional, user vs. system)
 - Best practices
 - Modeling requirements
 - Use cases
 - Workflows
-  *UML diagrams*
- The iterative process of RE:
 - Elicitation
 - Analysis
 - Specification
 - Validation
 - Agile Requirement models
 - Quality factors
 - Requirement Specification Documents (using IEEE Std 830-1998)



Linköping University

expanding reality

www.liu.se