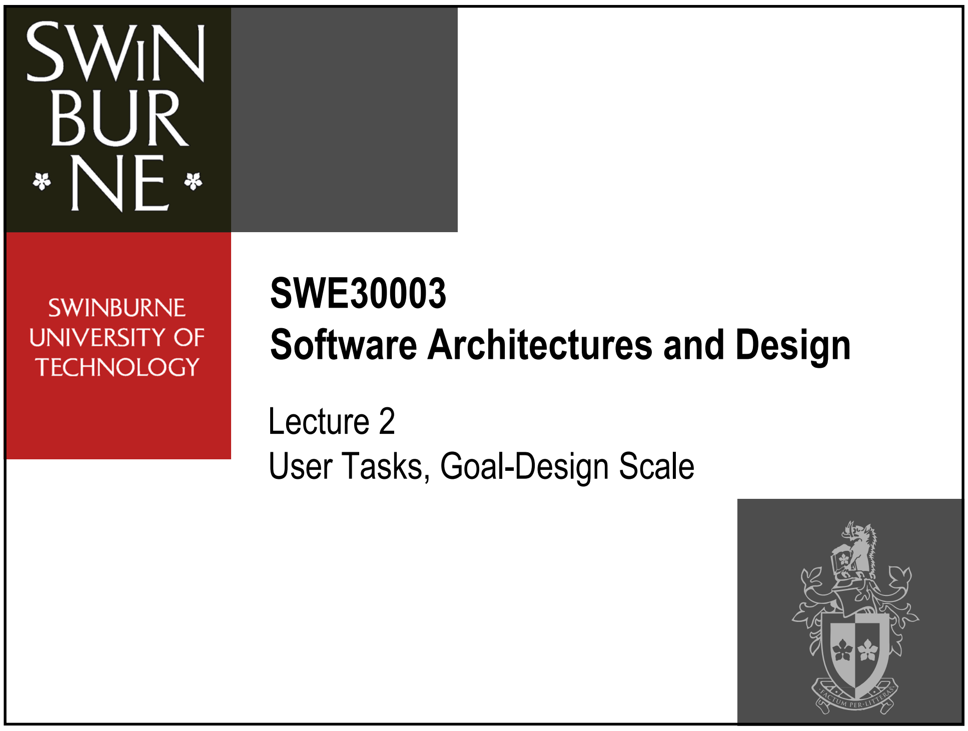
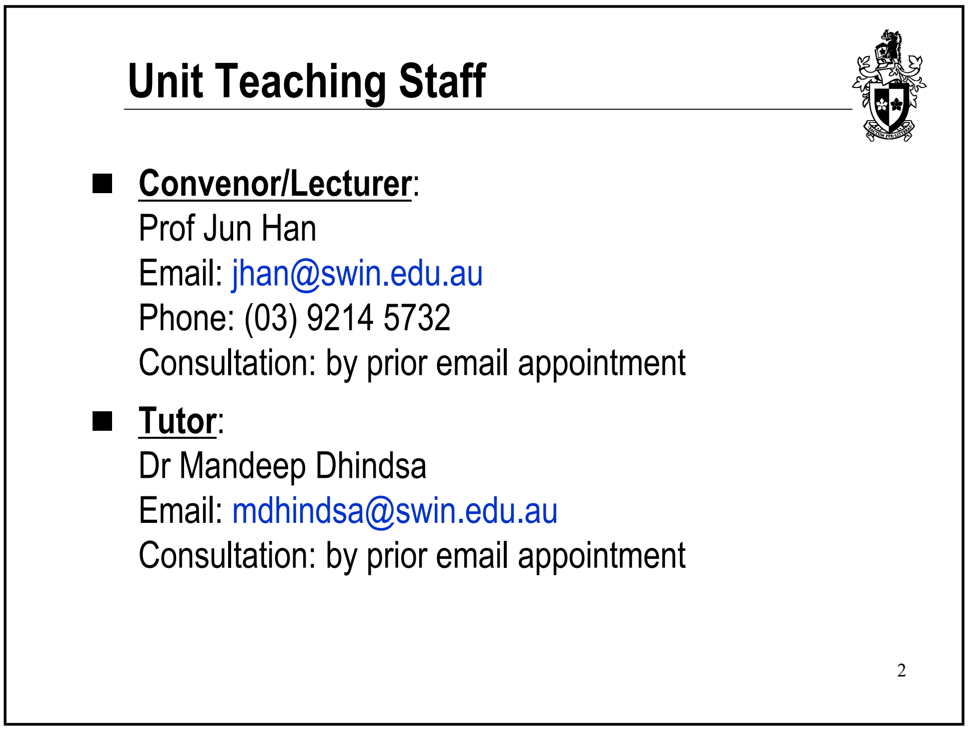


SWINBURNE UNIVERSITY OF TECHNOLOGY

SWE30003 Software Architectures and Design

Lecture 2
User Tasks, Goal-Design Scale

1

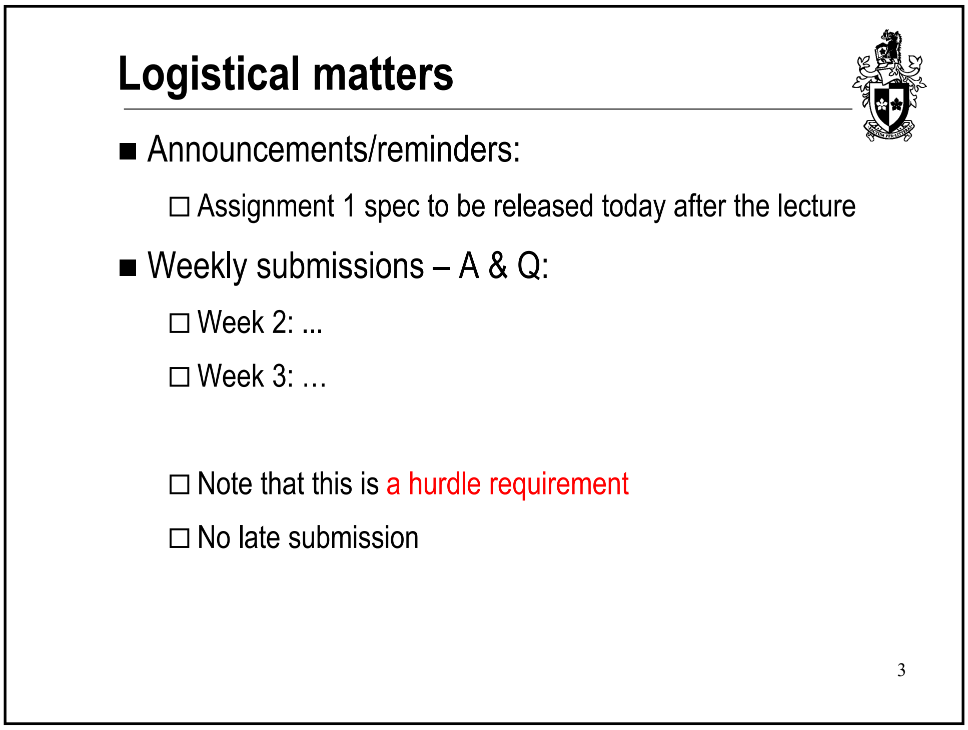


Unit Teaching Staff

- **Convenor/Lecturer**:
  Prof Jun Han
  Email: jhan@swin.edu.au
  Phone: (03) 9214 5732
  Consultation: by prior email appointment
- **Tutor**:
  Dr Mandeep Dhindsa
  Email: mdhindsa@swin.edu.au
  Consultation: by prior email appointment

2

2

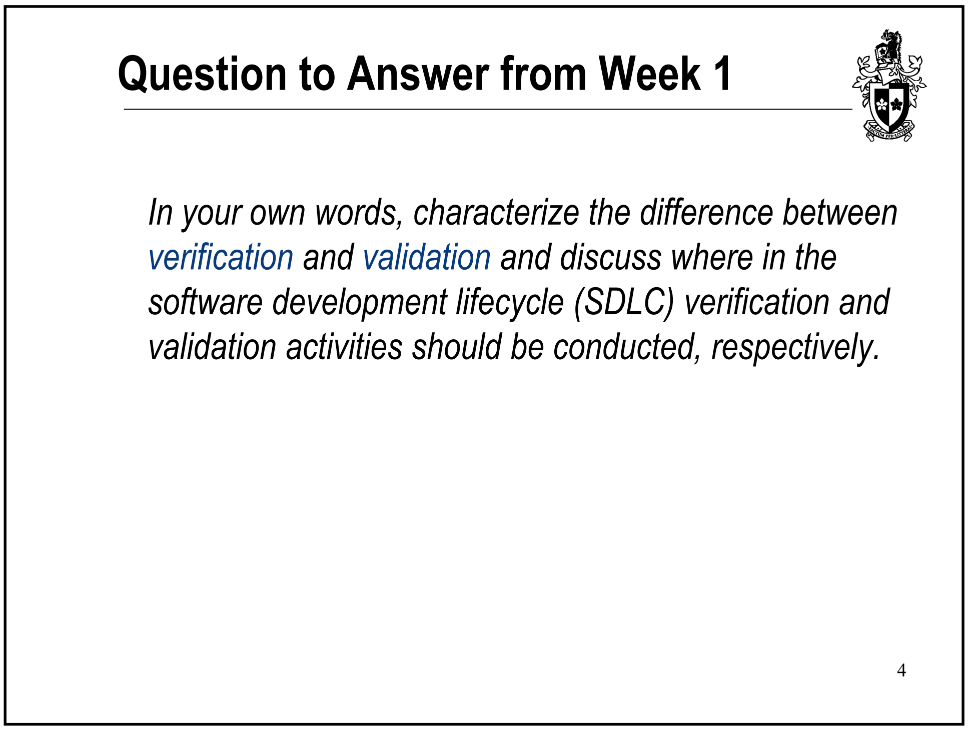


Logistical matters

- Announcements/reminders:
  - ☐ Assignment 1 spec to be released today after the lecture
- Weekly submissions – A & Q:
  - ☐ Week 2: ...
  - ☐ Week 3: …
  - ☐ Note that this is a hurdle requirement
  - ☐ No late submission

3

3

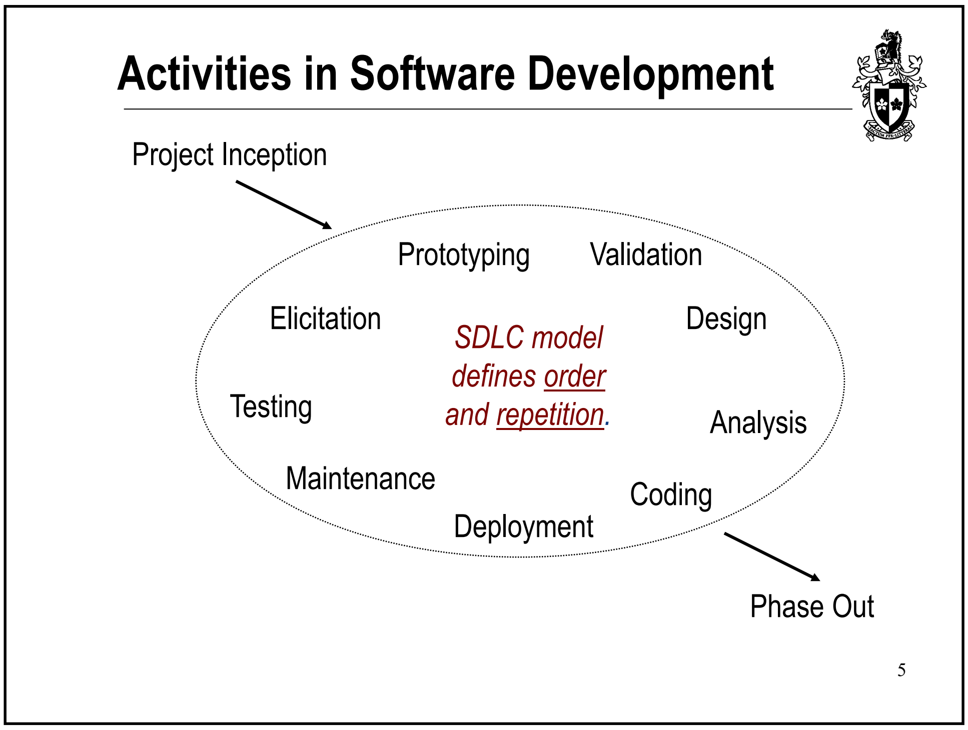


Question to Answer from Week 1

In your own words, characterize the difference between verification and validation and discuss where in the software development lifecycle (SDLC) verification and validation activities should be conducted, respectively.

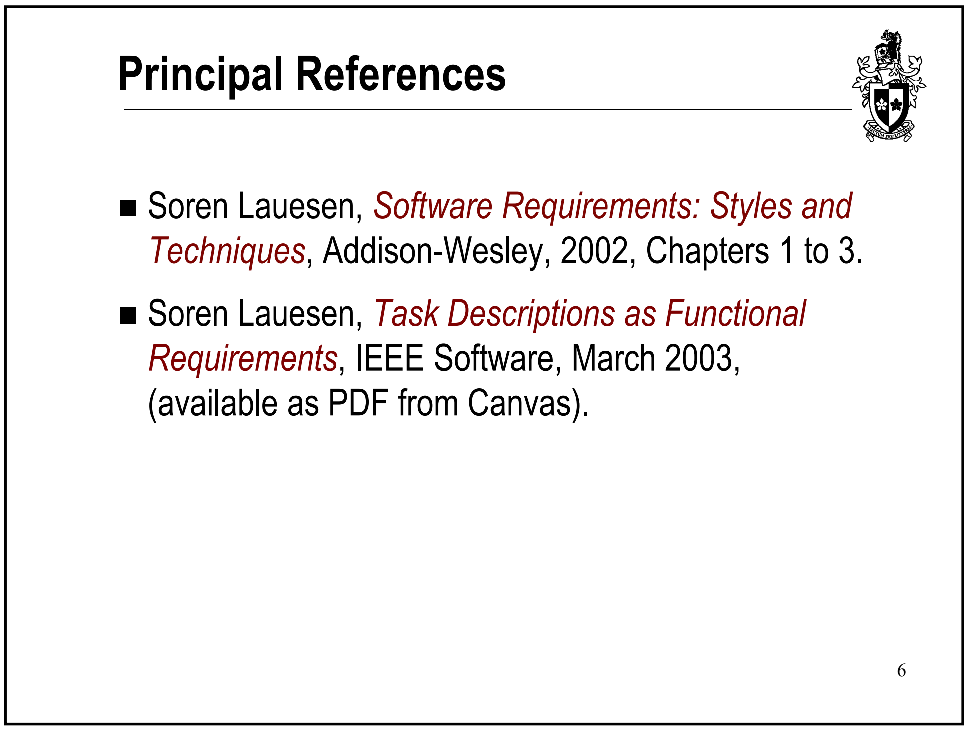
4

4



Activities in Software Development

5

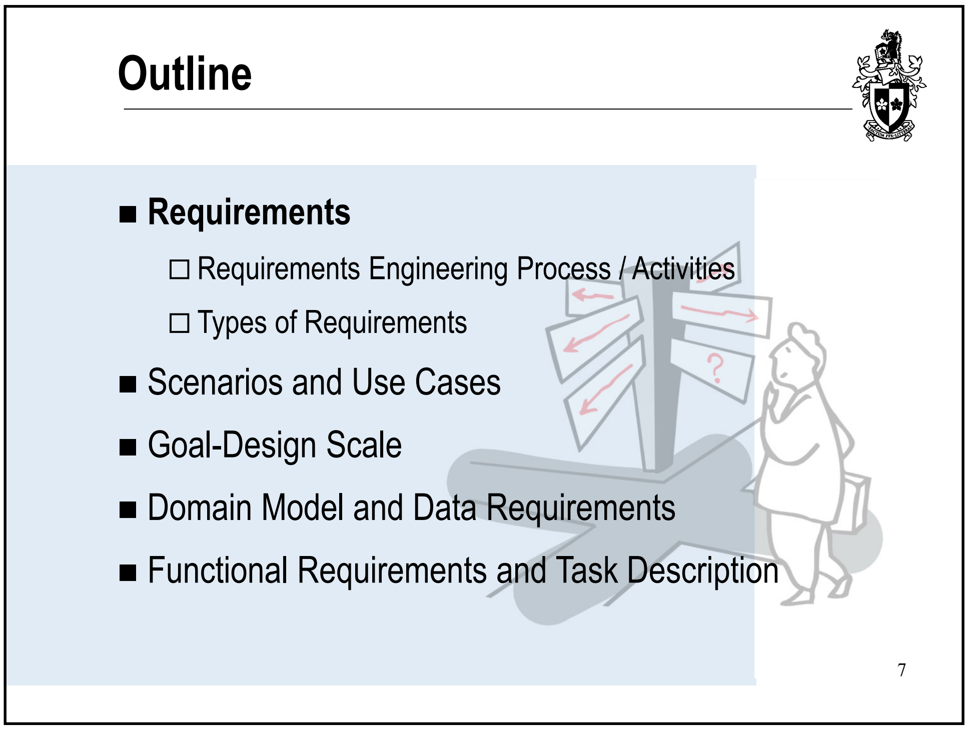


Principal References

- Soren Lauesen, *Software Requirements: Styles and Techniques*, Addison-Wesley, 2002, Chapters 1 to 3.
- Soren Lauesen, *Task Descriptions as Functional Requirements*, IEEE Software, March 2003, (available as PDF from Canvas).

6

6



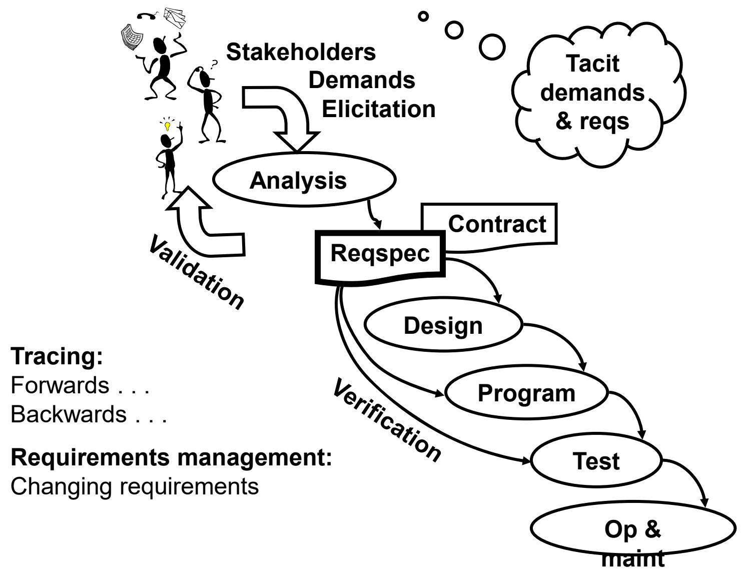
Outline

- **Requirements**
  - ☐ Requirements Engineering Process / Activities
  - ☐ Types of Requirements
- Scenarios and Use Cases
- Goal-Design Scale
- Domain Model and Data Requirements
- Functional Requirements and Task Description

7

7

The Role of Requirements



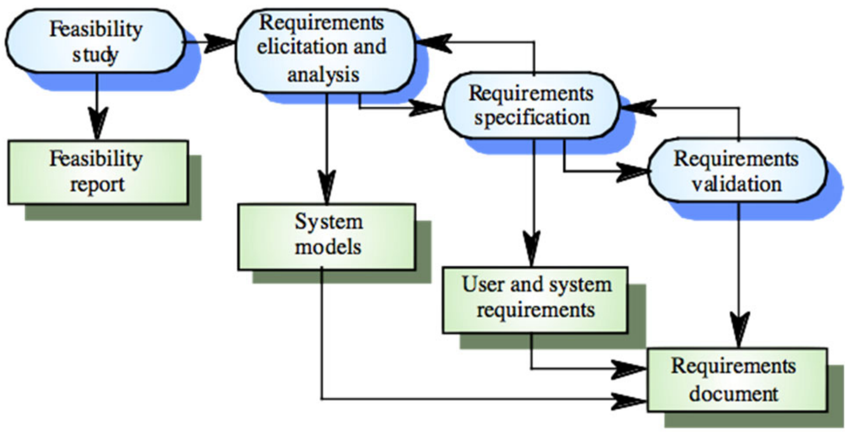
Tracing:
Forwards . . .
Backwards . . .

Requirements management:
Changing requirements

8

8

The Requirements Engineering Process



9

© Ian Sommerville 2000

9

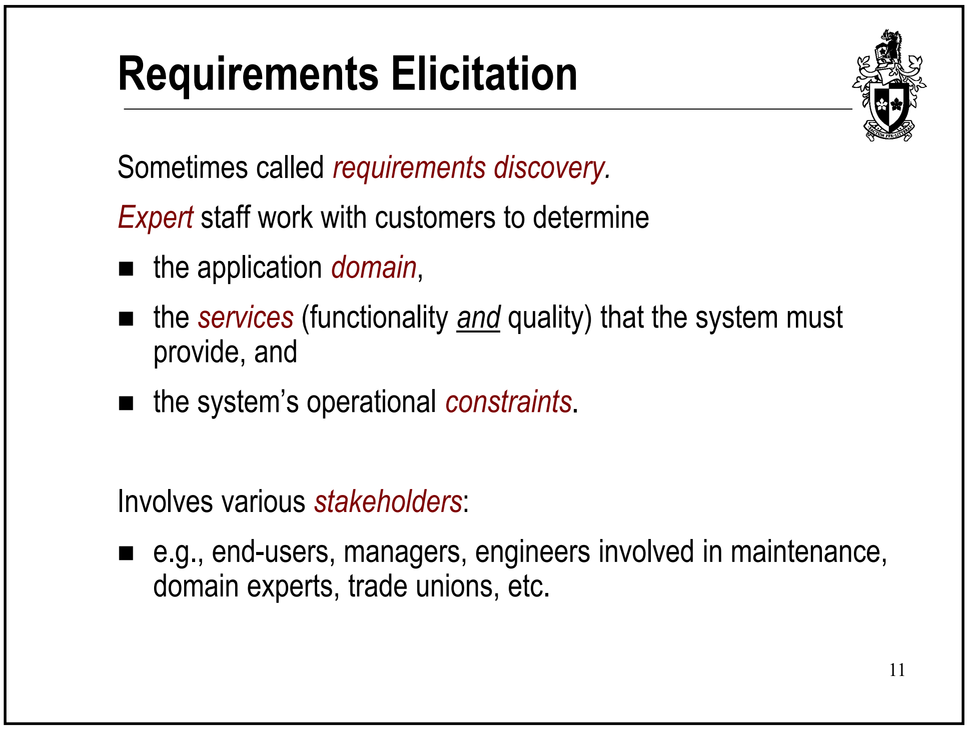
Requirements Engineering Activities

Feasibility study	Determine if the *user needs* can be *satisfied* with the *available technology* and *budget*.
Requirements elicitation	Find out *what system stakeholders require* from the system.
Requirements analysis	*Clarify/define/document the requirements* in a form understandable to the customer.
Requirements specification	Define the requirements in *detail*. (Written as a contract between client and contractor.)

"Requirements are for ***users****; specifications are for analysts and developers."*

10

10



Requirements Elicitation

Sometimes called *requirements discovery*.

Expert staff work with customers to determine

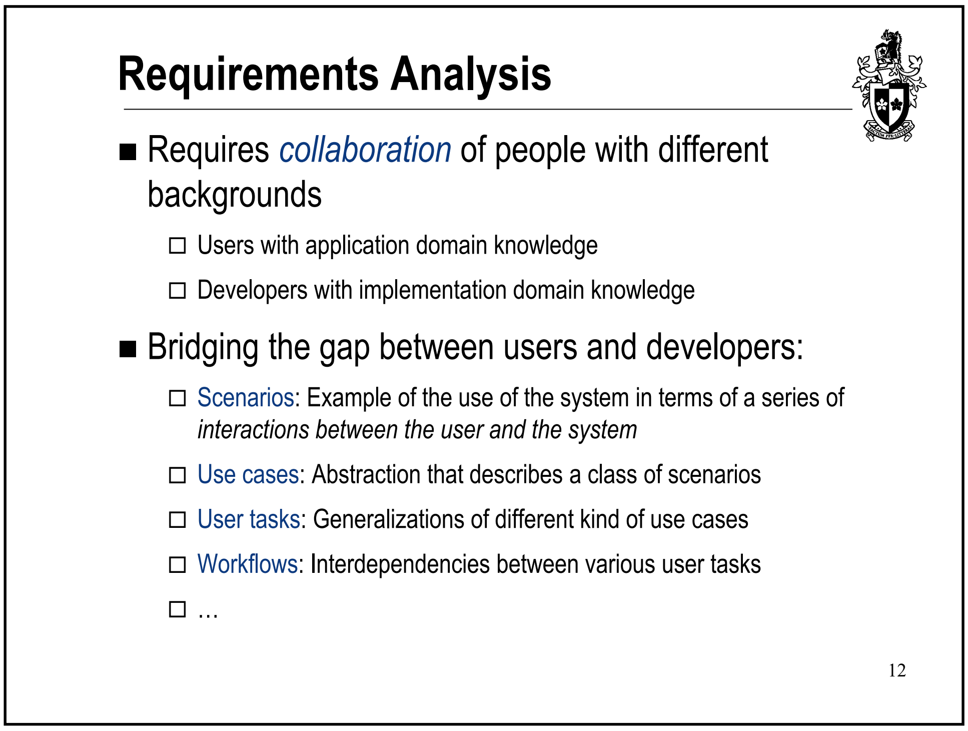
- the application *domain*,
- the *services* (functionality *and* quality) that the system must provide, and
- the system's operational *constraints*.

Involves various *stakeholders*:

- e.g., end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc.

11

11

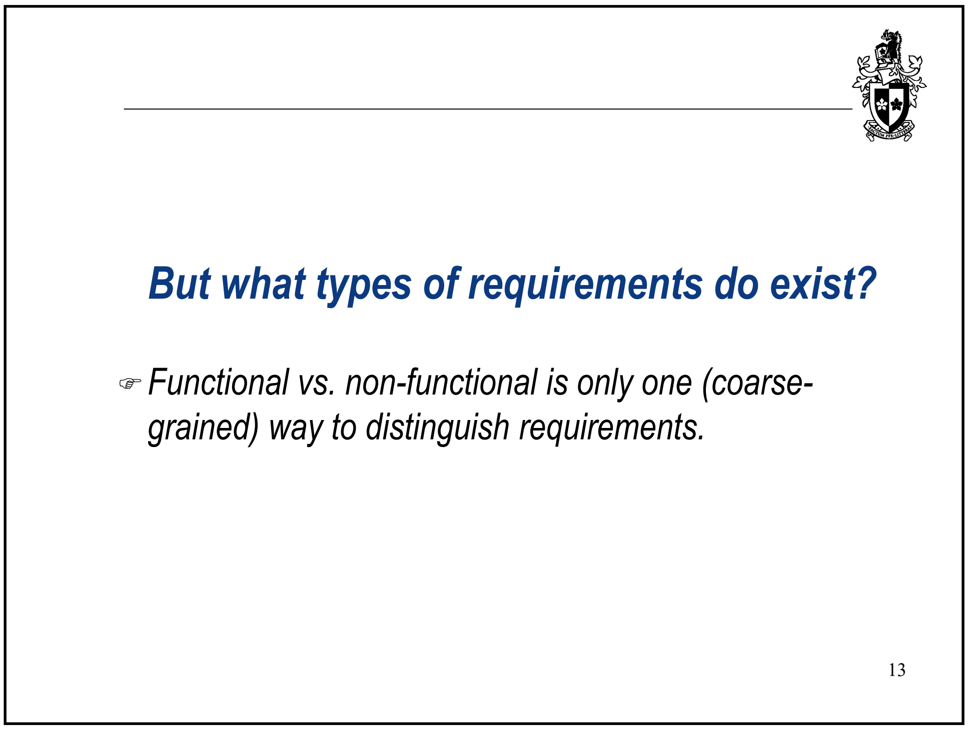


Requirements Analysis

- Requires *collaboration* of people with different backgrounds
  - Users with application domain knowledge
  - Developers with implementation domain knowledge
- Bridging the gap between users and developers:
  - Scenarios: Example of the use of the system in terms of a series of *interactions between the user and the system*
  - Use cases: Abstraction that describes a class of scenarios
  - User tasks: Generalizations of different kind of use cases
  - Workflows: Interdependencies between various user tasks
  - ...

12

12

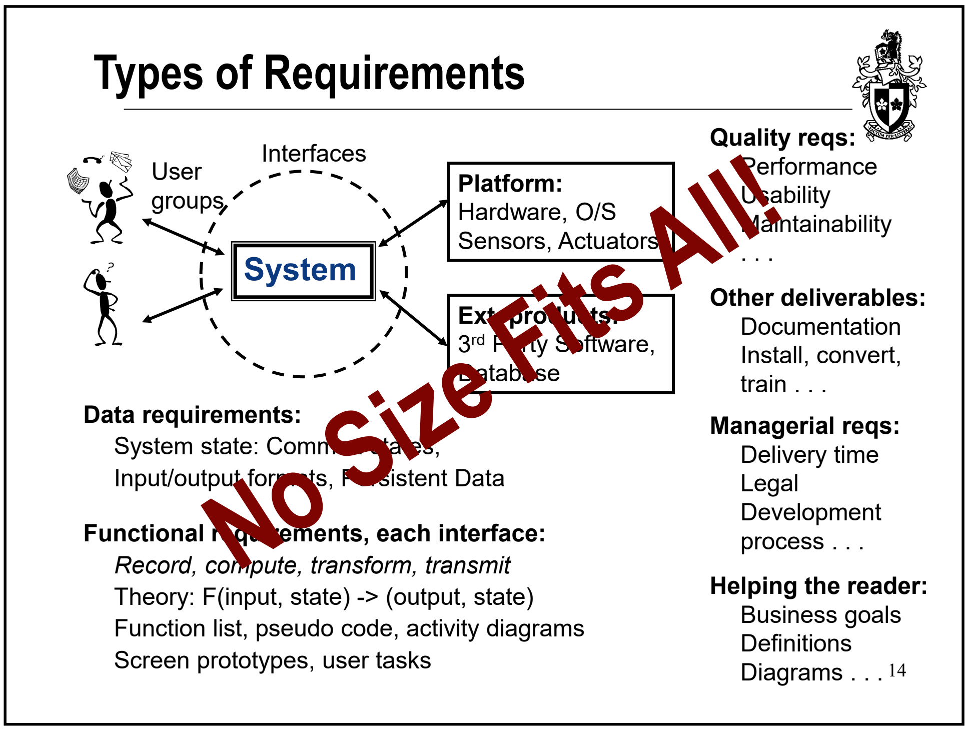


But what types of requirements do exist?

- *Functional vs. non-functional is only one (coarse-grained) way to distinguish requirements.*

13

13



Types of Requirements

Platform:
Hardware, O/S
Sensors, Actuators

Ext. products:
3rd Party Software,
Database

Quality reqs:
Performance
Usability
Maintainability
. . .

Data requirements:
System state: Commands, states,
Input/output formats, Persistent Data

Functional requirements, each interface:
Record, compute, transform, transmit
Theory: F(input, state) -> (output, state)
Function list, pseudo code, activity diagrams
Screen prototypes, user tasks

Other deliverables:
Documentation
Install, convert,
train . . .

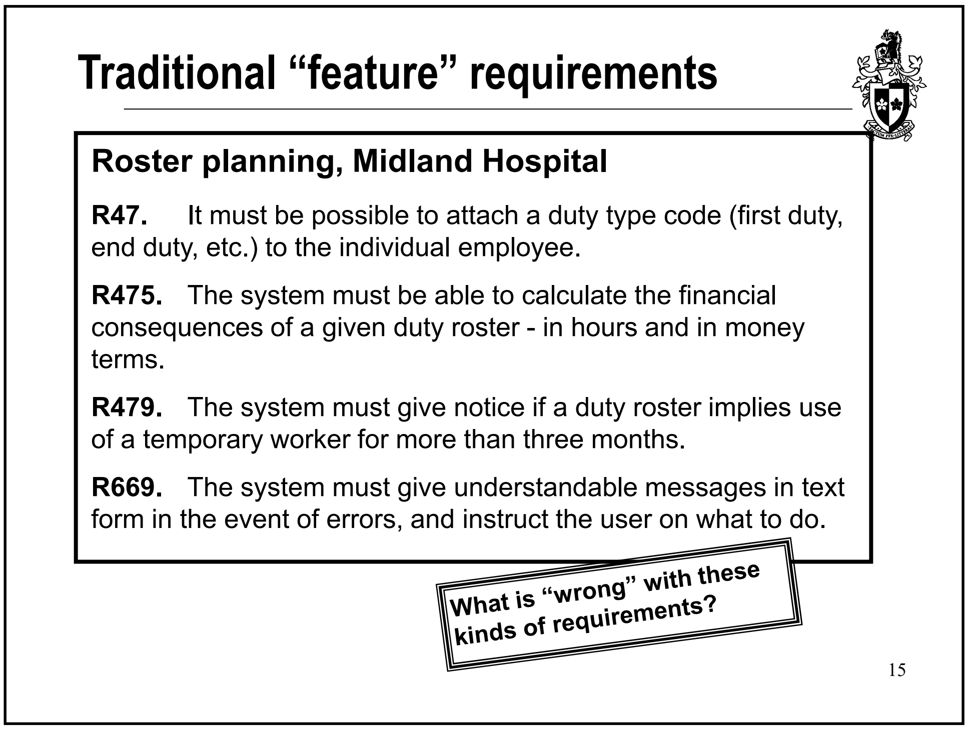
Managerial reqs:
Delivery time
Legal
Development
process . . .

Helping the reader:
Business goals
Definitions
Diagrams . . .

No Size Fits All!

14

14



Traditional "feature" requirements

Roster planning, Midland Hospital

R47. It must be possible to attach a duty type code (first duty, end duty, etc.) to the individual employee.

R475. The system must be able to calculate the financial consequences of a given duty roster - in hours and in money terms.

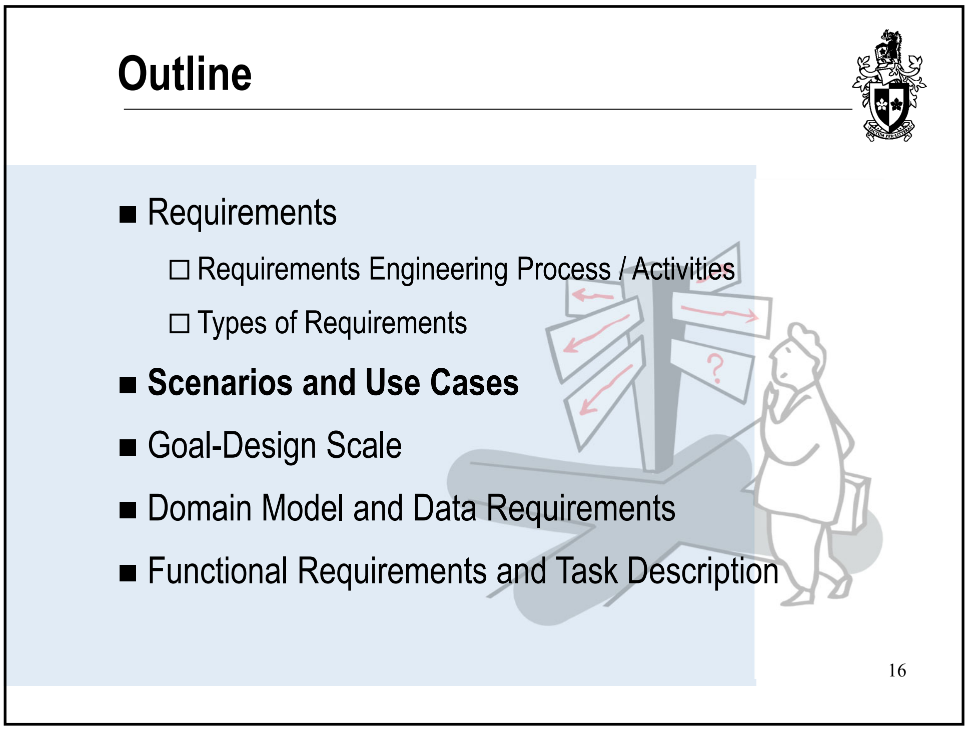
R479. The system must give notice if a duty roster implies use of a temporary worker for more than three months.

R669. The system must give understandable messages in text form in the event of errors, and instruct the user on what to do.

What is "wrong" with these kinds of requirements?

15

15

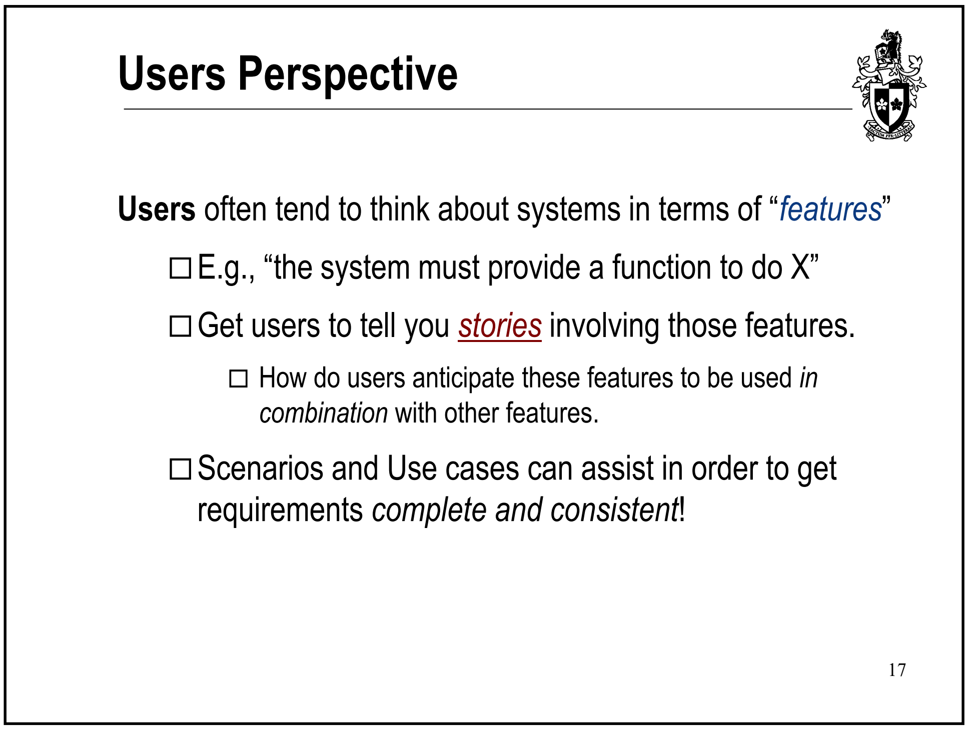


Outline

- ■ Requirements
  - ☐ Requirements Engineering Process / Activities
  - ☐ Types of Requirements
- ■ **Scenarios and Use Cases**
- ■ Goal-Design Scale
- ■ Domain Model and Data Requirements
- ■ Functional Requirements and Task Description

16

16



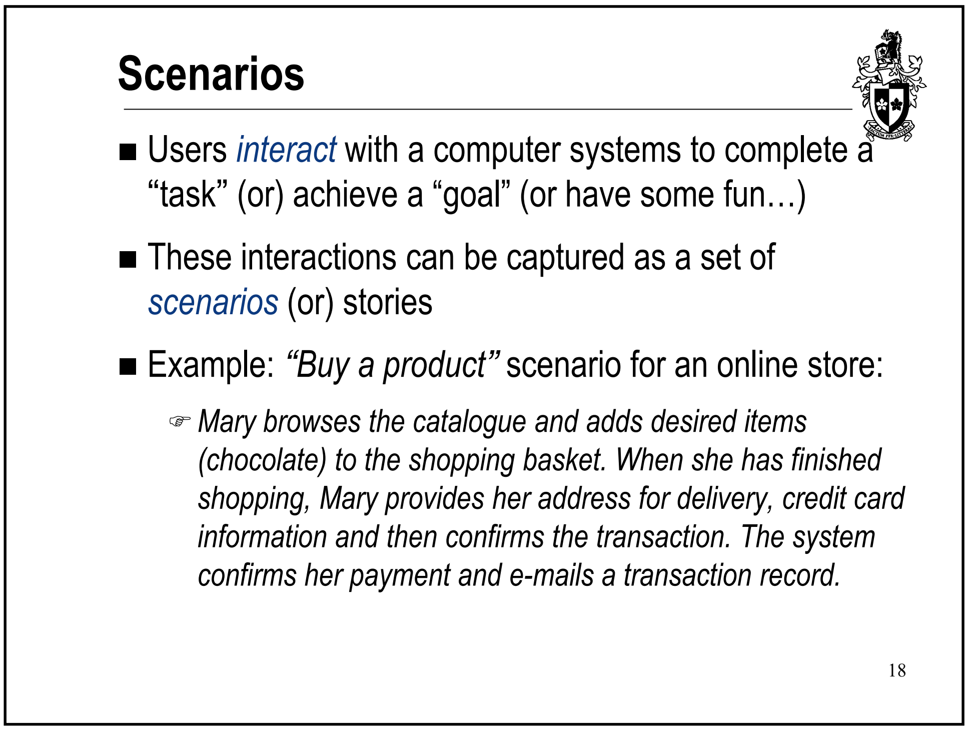
Users Perspective

Users often tend to think about systems in terms of "*features*"

- ☐ E.g., "the system must provide a function to do X"
- ☐ Get users to tell you *stories* involving those features.
  - ☐ How do users anticipate these features to be used *in combination* with other features.
- ☐ Scenarios and Use cases can assist in order to get requirements *complete and consistent*!

17

17



Scenarios

- ■ Users *interact* with a computer systems to complete a "task" (or) achieve a "goal" (or have some fun…)
- ■ These interactions can be captured as a set of *scenarios* (or) stories
- ■ Example: *"Buy a product"* scenario for an online store:
  - ☞ *Mary browses the catalogue and adds desired items (chocolate) to the shopping basket. When she has finished shopping, Mary provides her address for delivery, credit card information and then confirms the transaction. The system confirms her payment and e-mails a transaction record.*

18

18

Scenarios (cont.)

- In the "*Buy a product*" scenario we covered the normal *ideal interaction* between a user and "the system". But
  - ☐ What happens if the credit-card payment fails?
  - ☐ What if the customer is a repeat customer, so shipping details are already in the system?
- The key scenario would hold, but to answer the above questions we have to add *alternative* scenarios.
- Scenarios can also be written down as a *series of steps*
- Identify different scenarios that can be written up

19

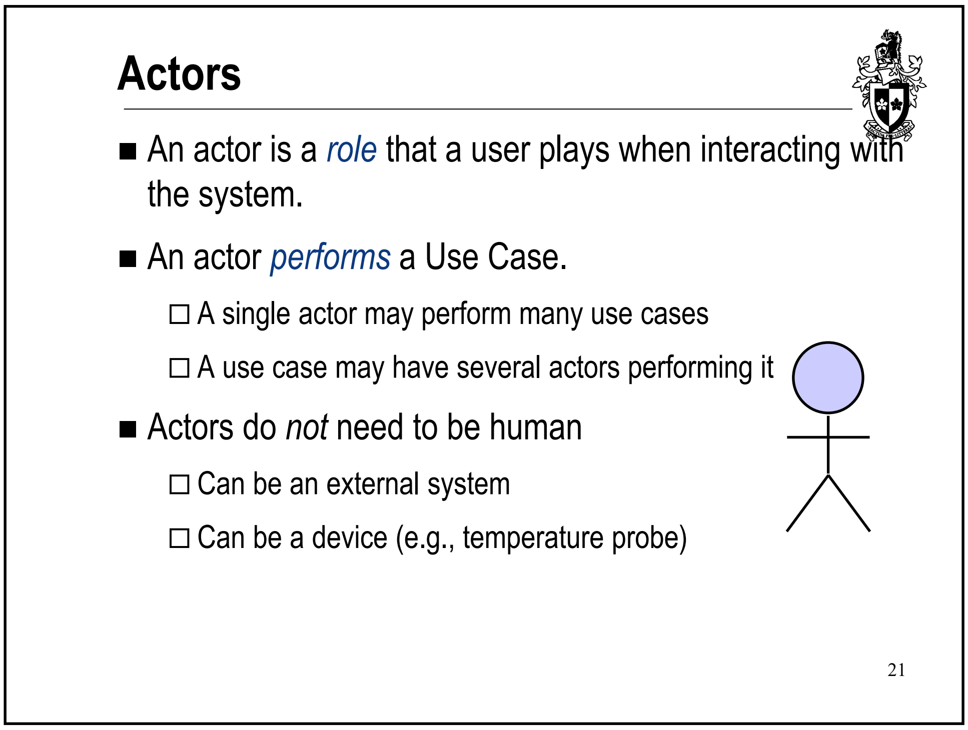
19

Use Case

- If we structure scenarios, they can represent the behavioural requirements of the system – from a *user's view point*.
- *"A use case is a set of scenarios tied together by a common user goal"* – Martin Fowler
  - ☐ Every use case must be tied to a *specific goal*
  - ☐ A use case is directly or indirectly invoked by an *actor*
- "A *use case* is the *specification* of a *sequence of actions*, including *variants*, that a system (or other entity) can perform, *interacting with actors* of the system."

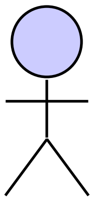
20

20



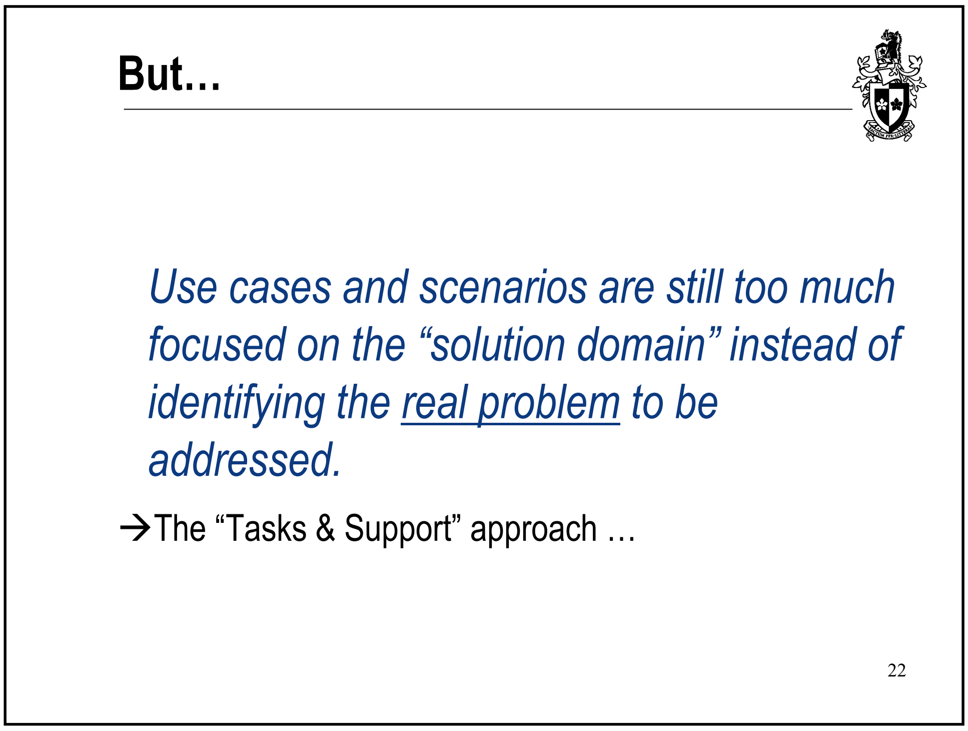
Actors

- An actor is a *role* that a user plays when interacting with the system.
- An actor *performs* a Use Case.
  - ☐ A single actor may perform many use cases
  - ☐ A use case may have several actors performing it
- Actors do *not* need to be human
  - ☐ Can be an external system
  - ☐ Can be a device (e.g., temperature probe)



21

21



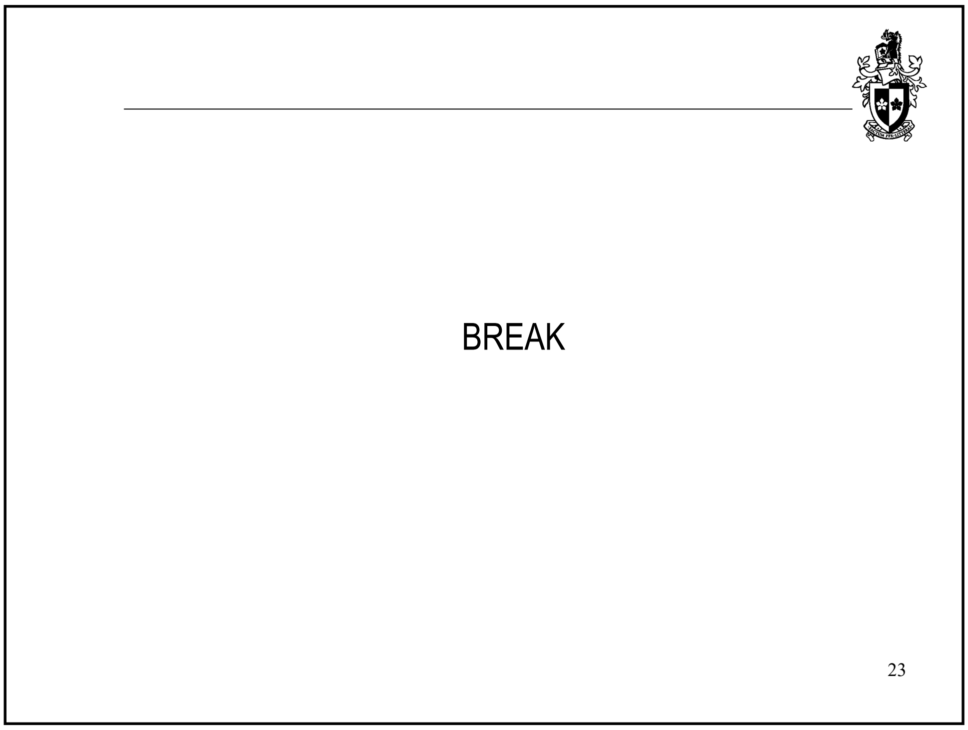
But…

Use cases and scenarios are still too much focused on the “solution domain” instead of identifying the real problem to be addressed.

→The “Tasks & Support” approach …

22

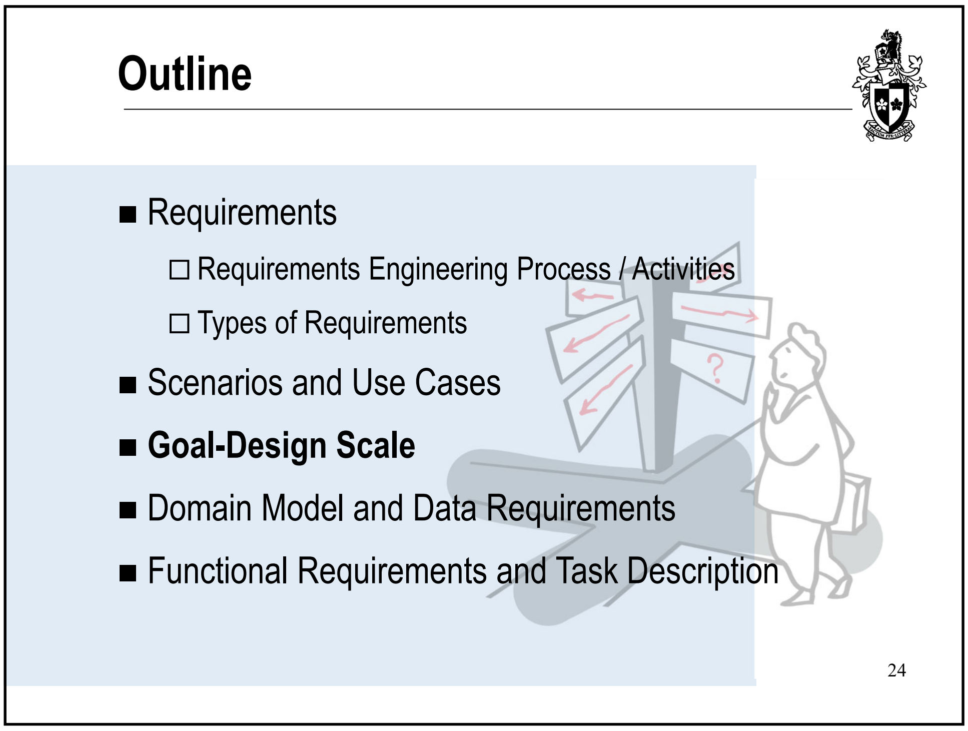
22



BREAK

23

23



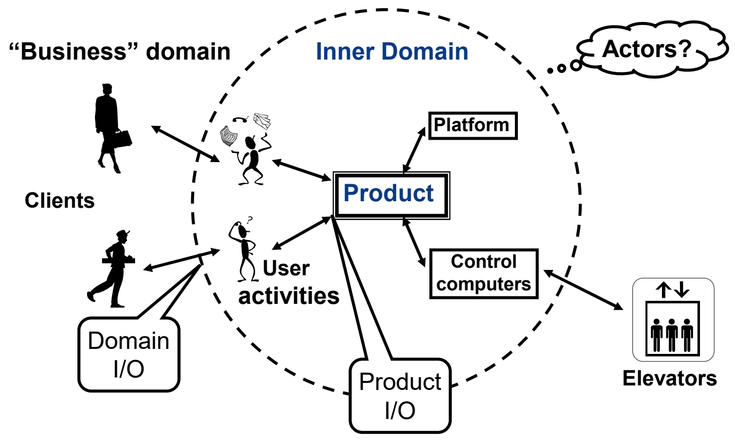
Outline

- Requirements
  - ☐ Requirements Engineering Process / Activities
  - ☐ Types of Requirements
- Scenarios and Use Cases
- **Goal-Design Scale**
- Domain Model and Data Requirements
- Functional Requirements and Task Description

24

24

Domain and Product Levels



Domain-level requirements:
The product *shall support*
the following user activities: . . .

Product-level requirements:
The product shall accept
the following input: . . .

25

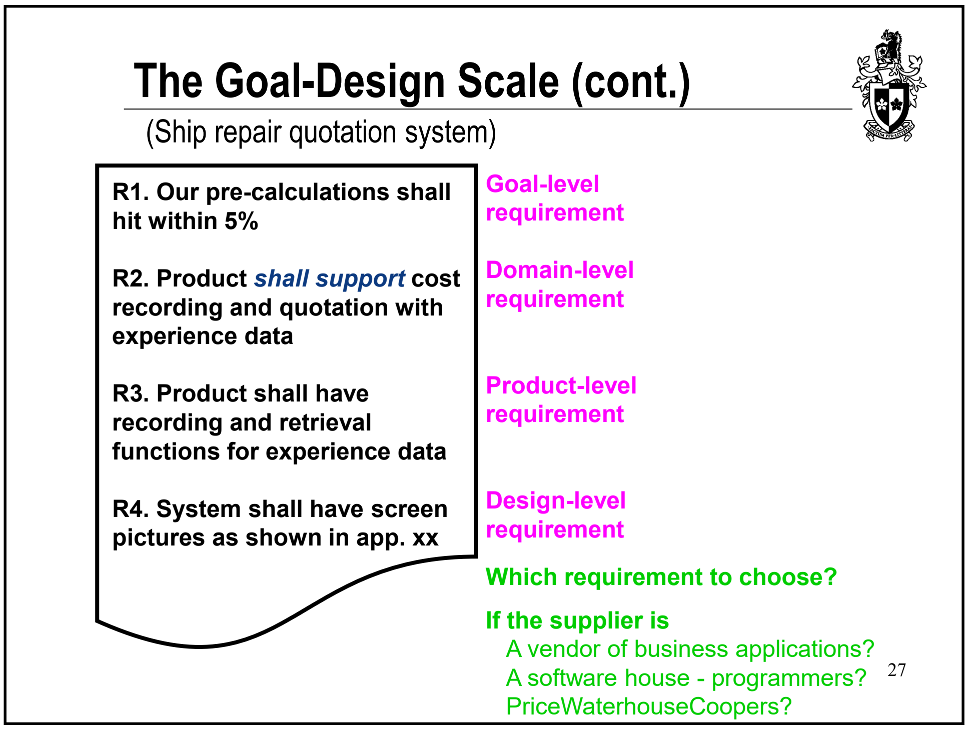
25

The Goal-Design Scale

- *Goal level* requirements specify business goals.
- *Domain level* requirements specify the activities that go on outside the product (i.e. in the domain).
- *Product level* requirements specify what should come in and out of the product.
- *Design level* requirements specify user-interface details and internals of the system.

26

26



The Goal-Design Scale (cont.)

(Ship repair quotation system)

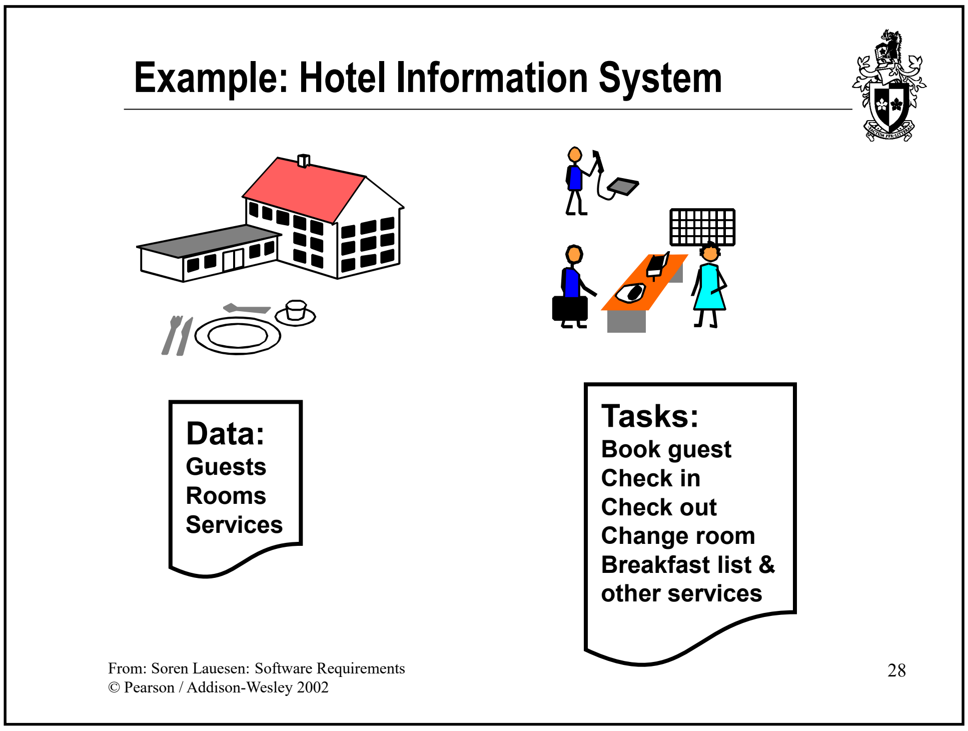
R1. Our pre-calculations shall hit within 5%	**Goal-level requirement**
R2. Product *shall support* cost recording and quotation with experience data	**Domain-level requirement**
R3. Product shall have recording and retrieval functions for experience data	**Product-level requirement**
R4. System shall have screen pictures as shown in app. xx	**Design-level requirement**

Which requirement to choose?

If the supplier is
- A vendor of business applications?
- A software house - programmers?
- PriceWaterhouseCoopers?

27

27



Example: Hotel Information System

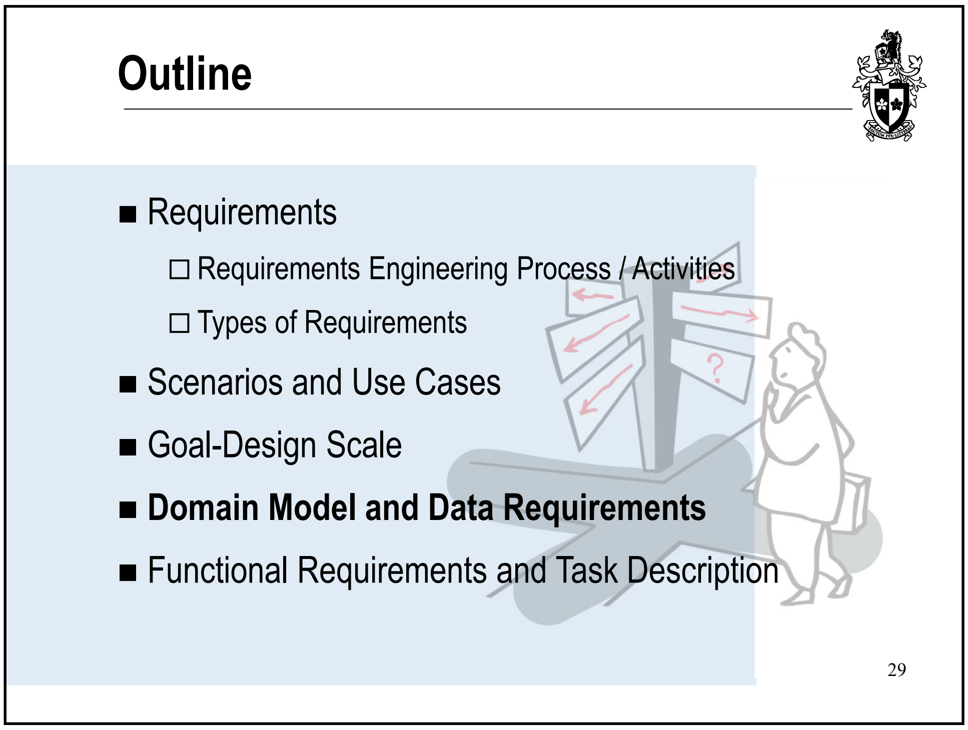
Data:
Guests
Rooms
Services

Tasks:
Book guest
Check in
Check out
Change room
Breakfast list & other services

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002

28

28

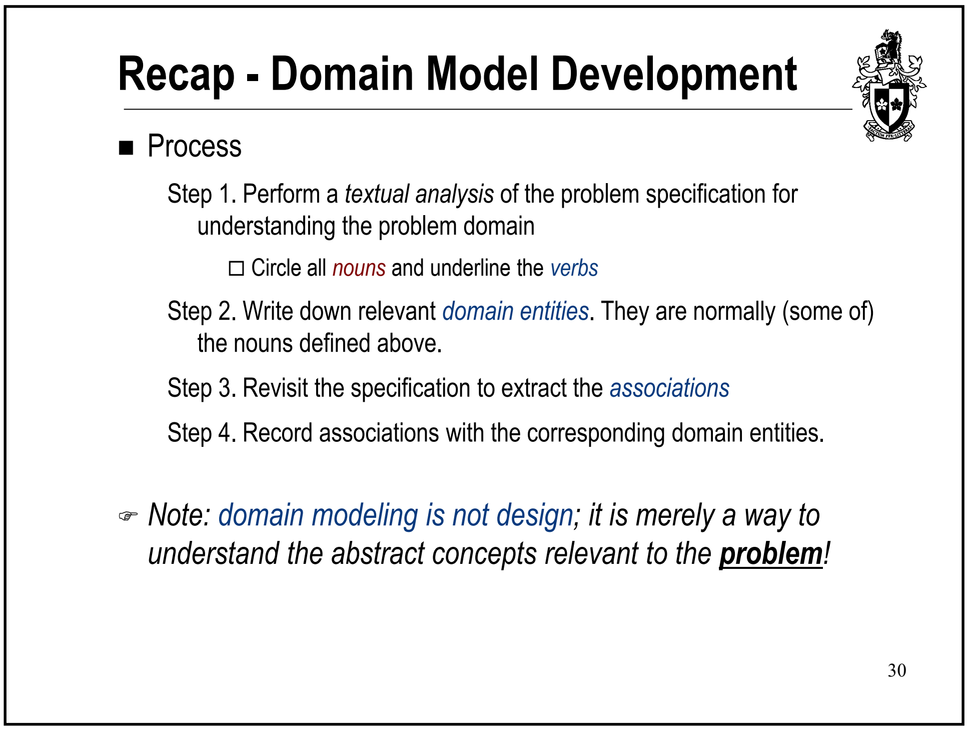


Outline

- Requirements
  - ☐ Requirements Engineering Process / Activities
  - ☐ Types of Requirements
- Scenarios and Use Cases
- Goal-Design Scale
- **Domain Model and Data Requirements**
- Functional Requirements and Task Description

29

29



Recap - Domain Model Development

- Process

  Step 1. Perform a *textual analysis* of the problem specification for understanding the problem domain

  - ☐ Circle all *nouns* and underline the *verbs*

  Step 2. Write down relevant *domain entities*. They are normally (some of) the nouns defined above.

  Step 3. Revisit the specification to extract the *associations*

  Step 4. Record associations with the corresponding domain entities.

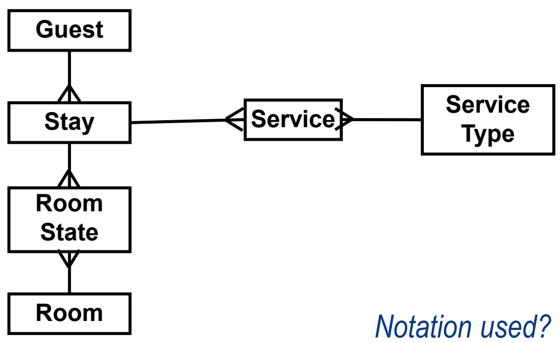
☞ *Note: domain modeling is not design; it is merely a way to understand the abstract concepts relevant to the **problem**!*

30

30

Hotel Information System – Domain Model

The system shall store the following ***domain entities*** ***(NOT database entities)***:



Notation used?

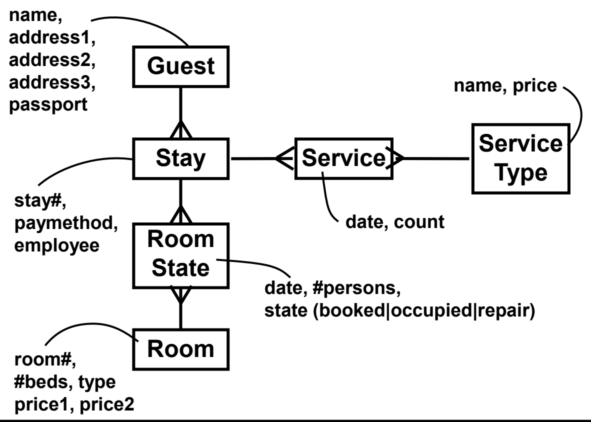
From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002

31

31

Beware of solution specifications! ... too far

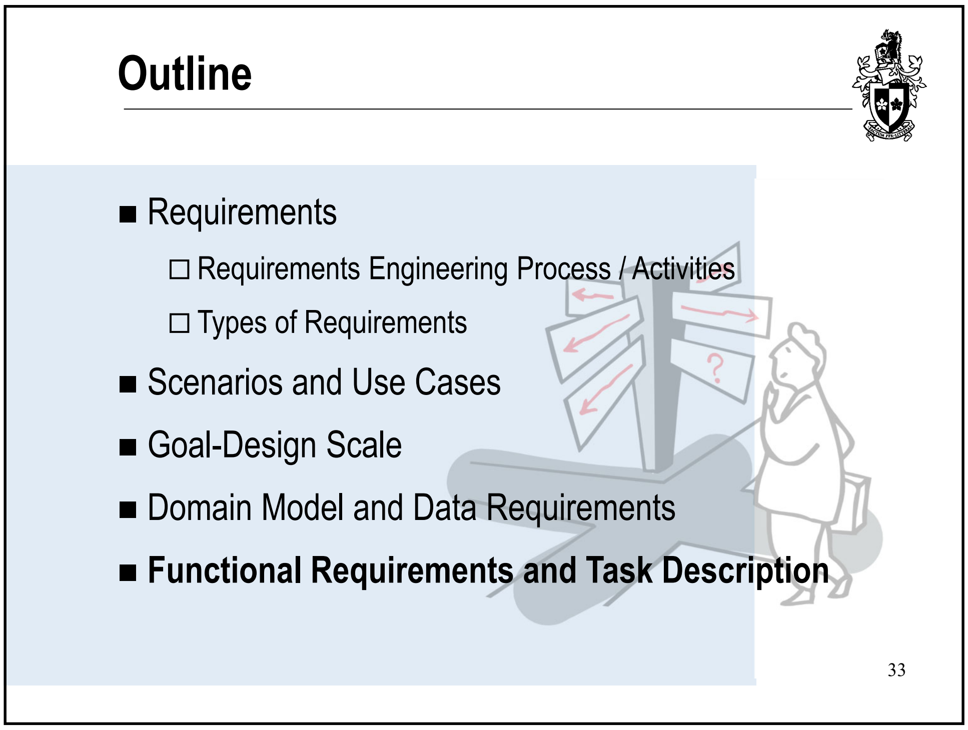
The system shall store the following *domain entities*:


name, address1, address2, address3, passport
Guest
name, price
Stay
Service
Service Type
stay#, paymethod, employee
date, count
Room State
date, #persons, state (booked|occupied|repair)
Room
room#, #beds, type price1, price2

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002

32

32

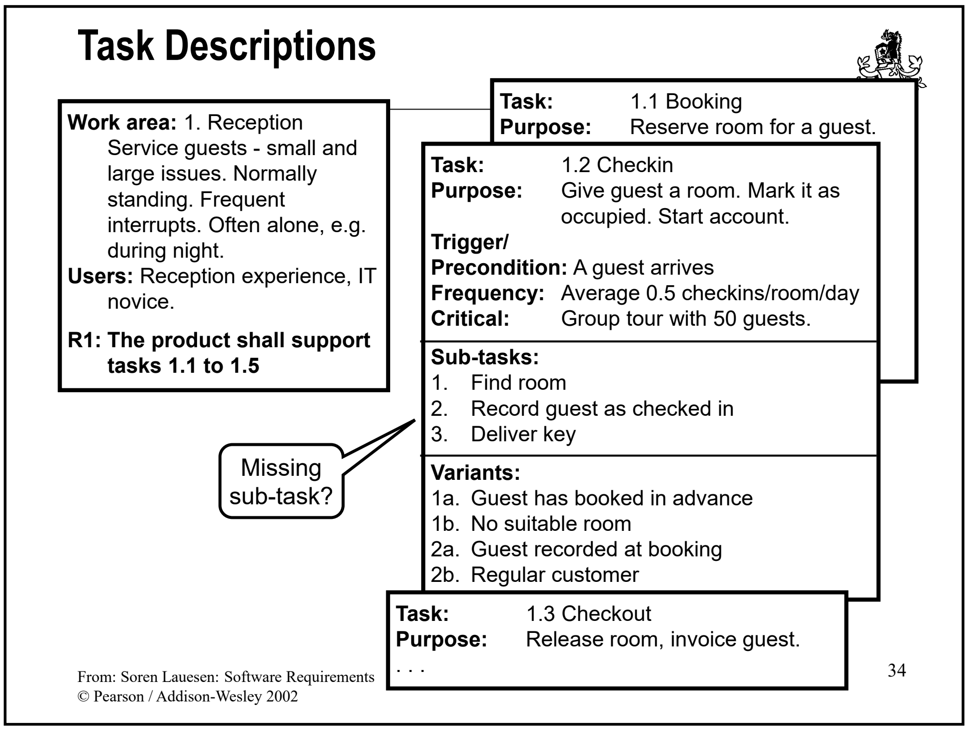


Outline

- Requirements
  - Requirements Engineering Process / Activities
  - Types of Requirements
- Scenarios and Use Cases
- Goal-Design Scale
- Domain Model and Data Requirements
- **Functional Requirements and Task Description**

33

33



Task Descriptions

Work area: 1. Reception
Service guests - small and large issues. Normally standing. Frequent interrupts. Often alone, e.g. during night.
Users: Reception experience, IT novice.

R1: The product shall support tasks 1.1 to 1.5

Task: 1.1 Booking
Purpose: Reserve room for a guest.

Task: 1.2 Checkin
Purpose: Give guest a room. Mark it as occupied. Start account.
Trigger/ Precondition: A guest arrives
Frequency: Average 0.5 checkins/room/day
Critical: Group tour with 50 guests.

Sub-tasks:
1. Find room
2. Record guest as checked in
3. Deliver key

Variants:
1a. Guest has booked in advance
1b. No suitable room
2a. Guest recorded at booking
2b. Regular customer

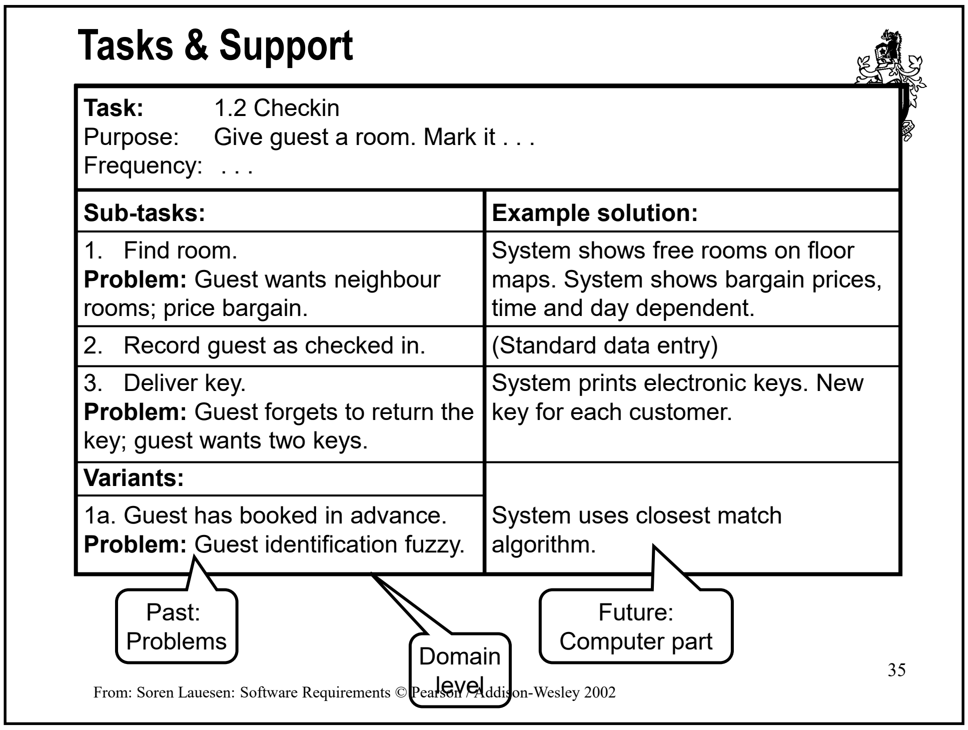
Task: 1.3 Checkout
Purpose: Release room, invoice guest.
. . .

Missing sub-task?

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002

34

34



Tasks & Support

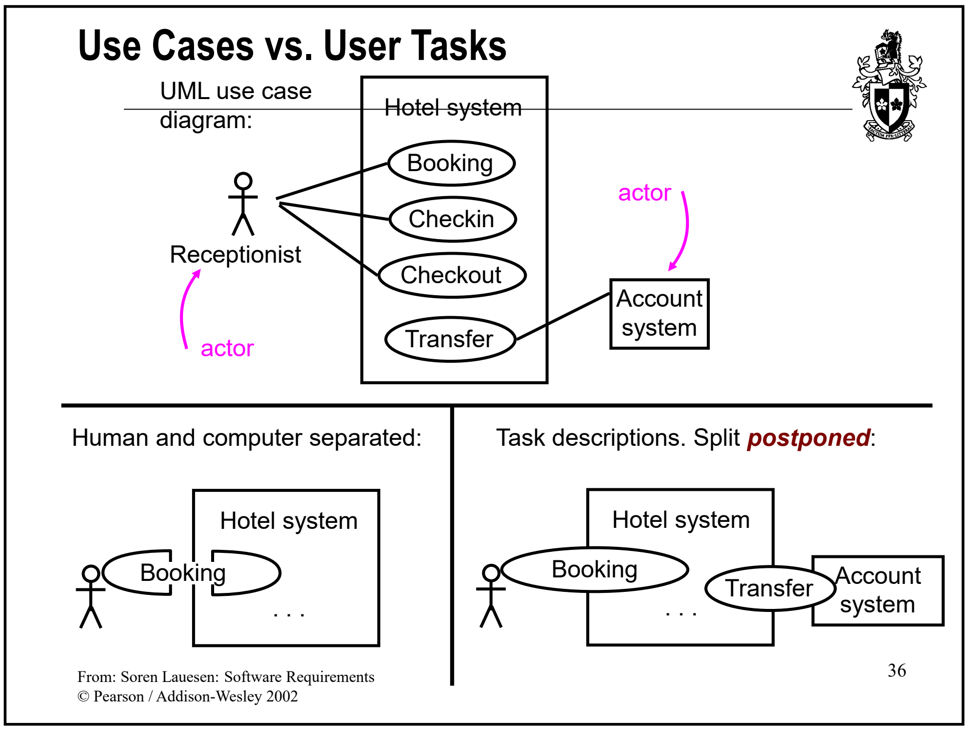
Task: 1.2 Checkin
Purpose: Give guest a room. Mark it . . .
Frequency: . . .

Sub-tasks:	**Example solution:**
1. Find room. **Problem:** Guest wants neighbour rooms; price bargain.	System shows free rooms on floor maps. System shows bargain prices, time and day dependent.
2. Record guest as checked in.	(Standard data entry)
3. Deliver key. **Problem:** Guest forgets to return the key; guest wants two keys.	System prints electronic keys. New key for each customer.
Variants:	
1a. Guest has booked in advance. **Problem:** Guest identification fuzzy.	System uses closest match algorithm.

From: Soren Lauesen: Software Requirements © Pearson / Addison-Wesley 2002

35

35



Use Cases vs. User Tasks

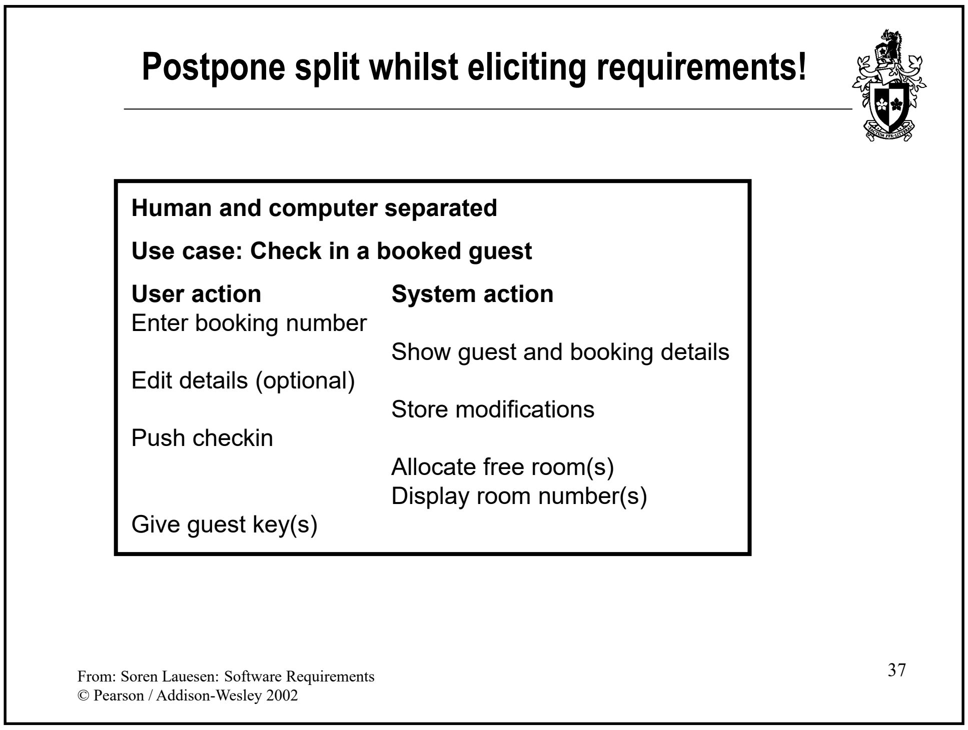
Human and computer separated:

Task descriptions. Split ***postponed***:

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002

36

36



Postpone split whilst eliciting requirements!

Human and computer separated

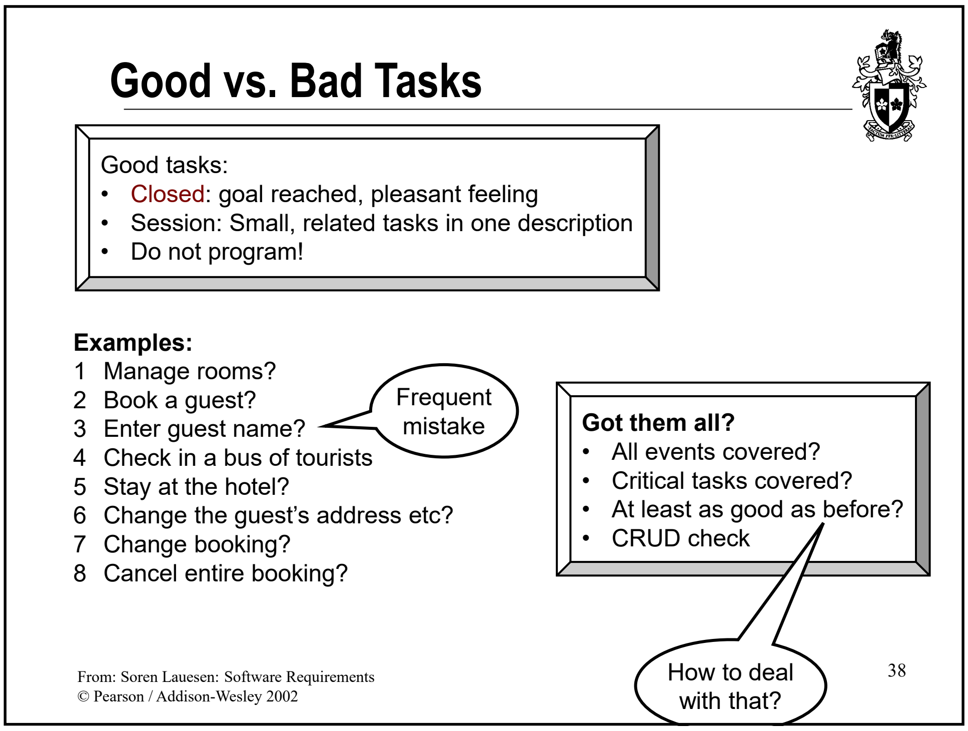
Use case: Check in a booked guest

User action	**System action**
Enter booking number	
	Show guest and booking details
Edit details (optional)	
	Store modifications
Push checkin	
	Allocate free room(s) Display room number(s)
Give guest key(s)	

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002

37

37



Good vs. Bad Tasks

Good tasks:
- Closed: goal reached, pleasant feeling
- Session: Small, related tasks in one description
- Do not program!

Examples:
1 Manage rooms?
2 Book a guest?
3 Enter guest name?
4 Check in a bus of tourists
5 Stay at the hotel?
6 Change the guest's address etc?
7 Change booking?
8 Cancel entire booking?

Frequent mistake

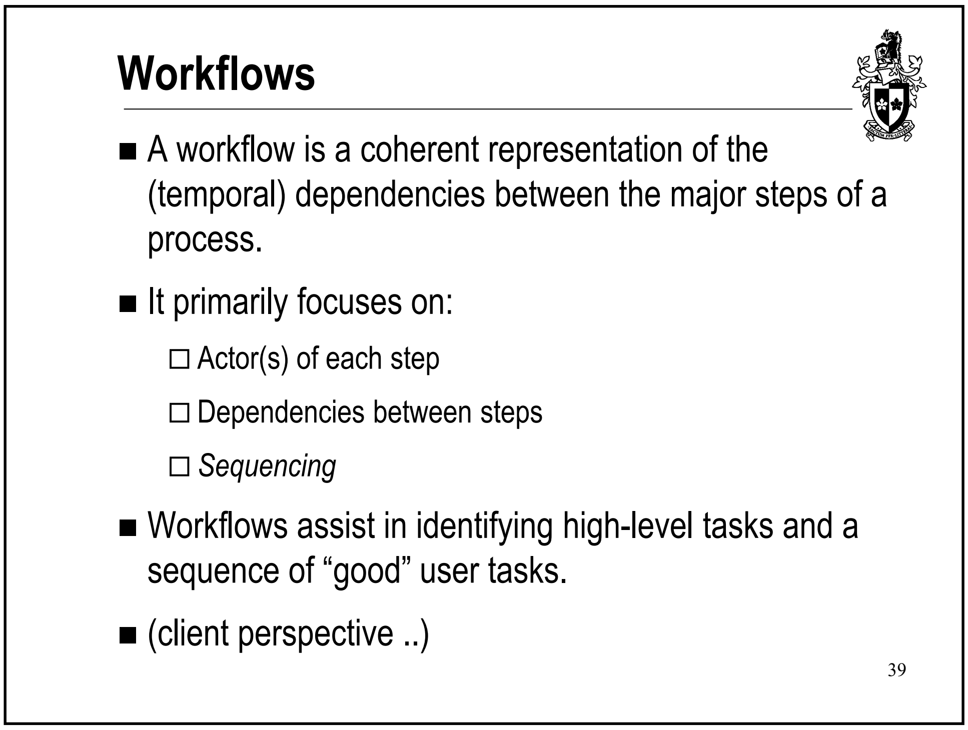
Got them all?
- All events covered?
- Critical tasks covered?
- At least as good as before?
- CRUD check

How to deal with that?

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002

38

38

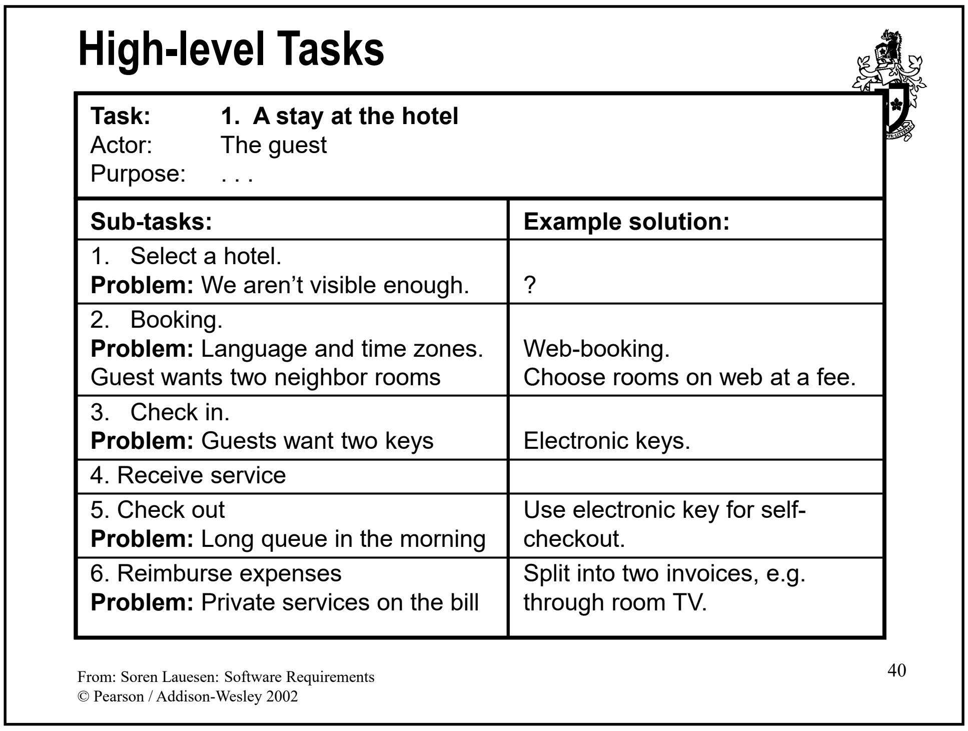


Workflows

- A workflow is a coherent representation of the (temporal) dependencies between the major steps of a process.
- It primarily focuses on:
  - ☐ Actor(s) of each step
  - ☐ Dependencies between steps
  - ☐ *Sequencing*
- Workflows assist in identifying high-level tasks and a sequence of “good” user tasks.
- (client perspective ..)

39

39



High-level Tasks

Task: **1. A stay at the hotel**
Actor: The guest
Purpose: . . .

Sub-tasks:	**Example solution:**
1. Select a hotel. **Problem:** We aren’t visible enough.	?
2. Booking. **Problem:** Language and time zones. Guest wants two neighbor rooms	Web-booking. Choose rooms on web at a fee.
3. Check in. **Problem:** Guests want two keys	Electronic keys.
4. Receive service	
5. Check out **Problem:** Long queue in the morning	Use electronic key for self-checkout.
6. Reimburse expenses **Problem:** Private services on the bill	Split into two invoices, e.g. through room TV.

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002

40

40

Questions for Consideration

1. What is a drawback of using Task Descriptions over other styles of functional requirements? How could this be managed?
2. What is the difference between subtasks and variants when designing a Task Description?
3. Within a Task Description, what is the difference between a Trigger and a Precondition?
4. Task descriptions don’t cover quality requirements such as response time and usability, but they point out where quality is crucial. How does a task description point out quality requirements?
5. How do you relate the different task descriptions to different actors when using the Task and Support method? How is this relation shown when the same tasks are applicable to different users of the system?

41

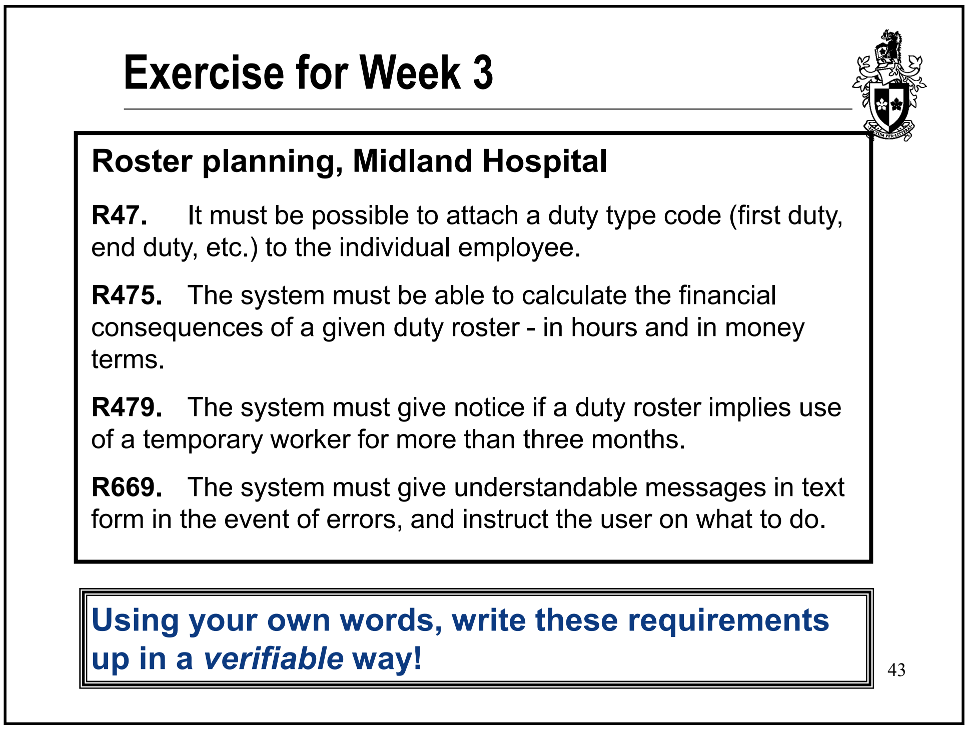
41

Question to Answer - Week 2

Describe the difference between Use Case diagrams and Task Descriptions. What are the advantages of using Task Descriptions as opposed to User Cases? Provide a situation where Task Descriptions are more suitable for use than Use Case Diagrams.

42

42



Exercise for Week 3

Roster planning, Midland Hospital

R47. It must be possible to attach a duty type code (first duty, end duty, etc.) to the individual employee.

R475. The system must be able to calculate the financial consequences of a given duty roster - in hours and in money terms.

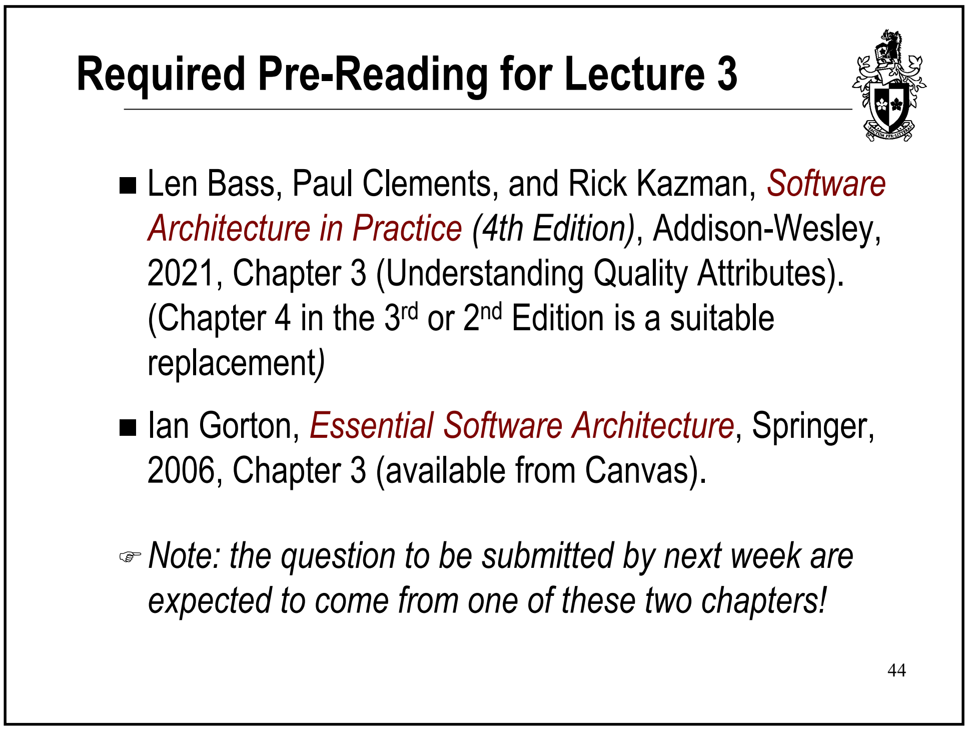
R479. The system must give notice if a duty roster implies use of a temporary worker for more than three months.

R669. The system must give understandable messages in text form in the event of errors, and instruct the user on what to do.

Using your own words, write these requirements up in a *verifiable* way!

43

43



Required Pre-Reading for Lecture 3

- Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice (4th Edition)*, Addison-Wesley, 2021, Chapter 3 (Understanding Quality Attributes). (Chapter 4 in the 3rd or 2nd Edition is a suitable replacement*)*
- Ian Gorton, *Essential Software Architecture*, Springer, 2006, Chapter 3 (available from Canvas).

☞ *Note: the question to be submitted by next week are expected to come from one of these two chapters!*

44

44