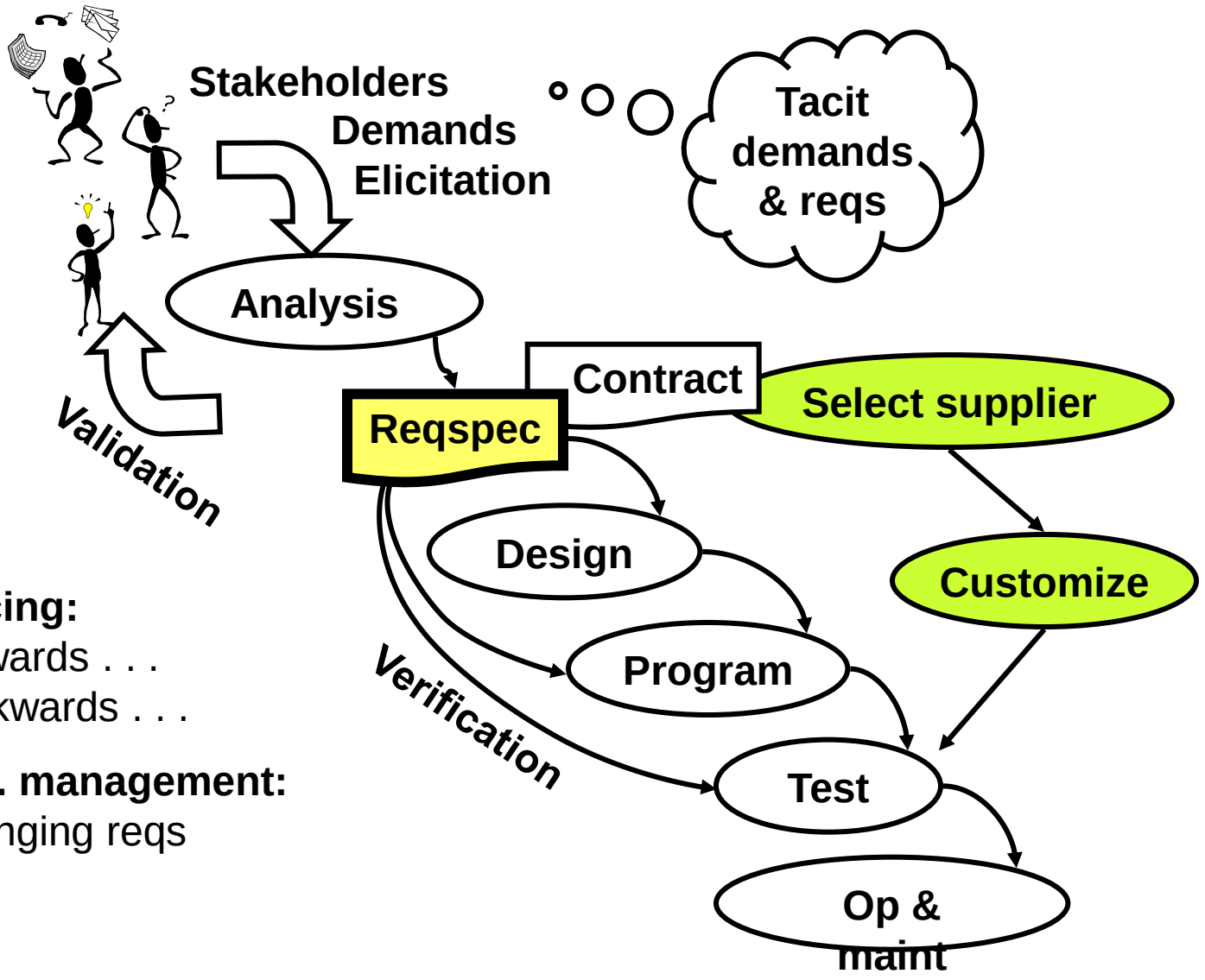
Requirements, use cases and tasks

2. The role of requirements



Tracing:
Forwards . . .
Backwards . . .

Req. management:
Changing reqs

From: Soren Lauesen: Software Requirements

3. The system border

Find room(dates)	
	the system selects the optimal room
The room is shown to user	
VS.	
Find room(dates)	
	the system finds a list of available rooms
Select room	
	the selected room is reserved

4. Traditional requirements - hospital roster planning

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

R144. The supplier must update the system according to new union agreements no later than a month after their release.

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

IEEE 830

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.

Experiences

Requirements are met, but the user tasks are supported badly.
The business goals are not met.
Too expensive - no freedom for the developer.

5. Write it as use cases?

When is it used and for what?

Use case 475: Calculate the financial consequences of a roster.
Trigger: The user wants to calculate the consequences.
Precondition: The user is logged on.

1. The system shows a list of rosters.
2. The user selects a roster
3. The user selects "Calculate consequences"
4. The system calculates the consequences.
5. The system shows the consequences.

Exception: No rosters in the list.

Invented dialog. Would be harmful here.

Trivial details - seduced by the template. No real value added.

6. Better requirements: Support tasks C1, C2 . . .

C2: Make roster
Frequency: Every 14 day. In some departments . . .
Difficult: Vacation periods.

Customer: Help - we bought the wrong system

Subtasks and variants:	**Example solutions:**
1. Create new roster.	Automatically from last plan . . .
2. Record leave. Two kinds . . . **Present problem:** Recorded on stickers many months ahead.	System checks the vacation rules. System can record several years ahead.
3. Allocate staff. Ensure competence level, leave, union agreements. Avoid extra pay. **Pres. problem:** Difficult manually. Errors and too much extra pay. 3a. Substitute not yet in the system. 3b. Get staff from other department.	System suggests staffing of unstaffed duties. Warns in case violated rules and excess pay. Supports the "jigsaw puzzle" with Undo and several trial versions. Shows free staff from other depts.
4. Distribute plan for comments.	A print of the roster suffices.
5. Park the plan or release it.	

Done by human plus computer

Example of computer's part - not requirements

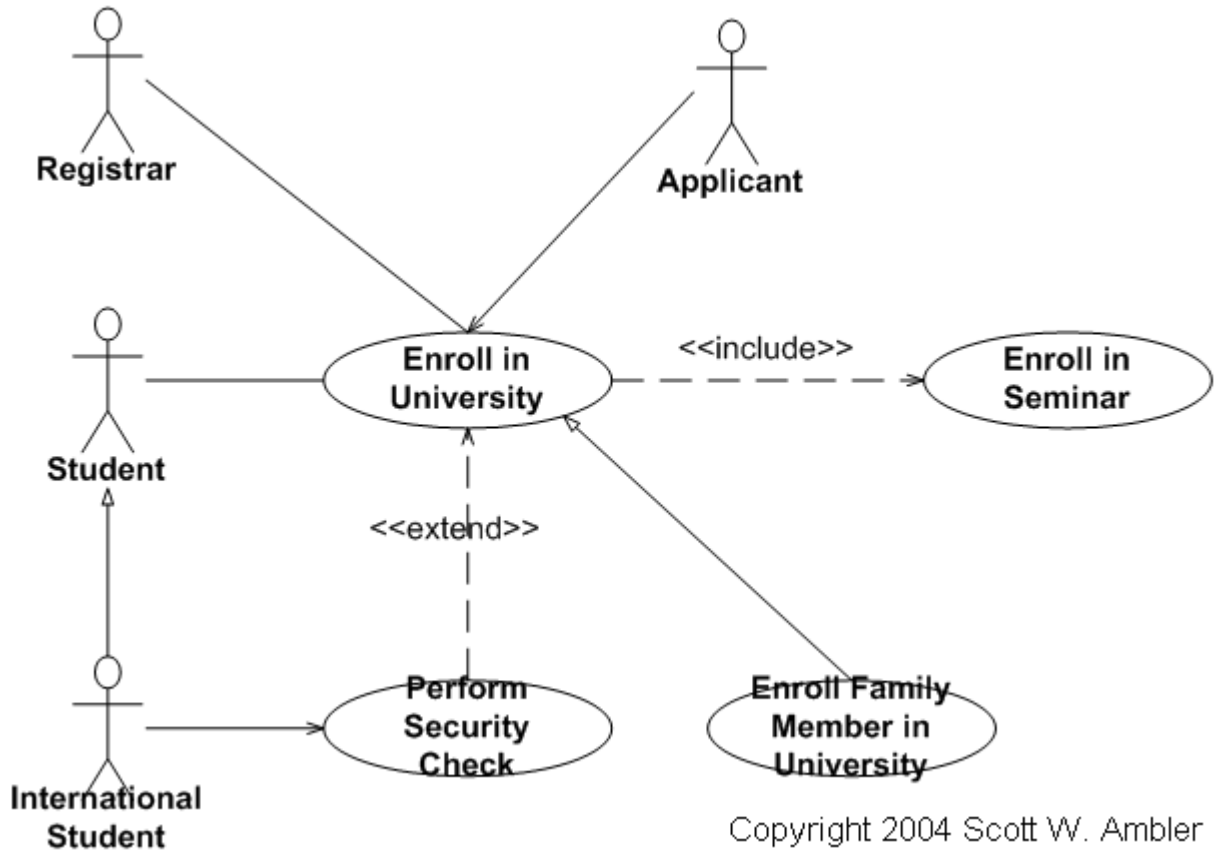
7. Use cases vs. tasks

A Use case is:

A task is:

8. UML Use case structure

What is the use for the following?:



9. Business goals and how to meet them

Business goals \ User tasks	**User: Planner in department** C1. Monthly report to personnel dept.	C2. Make roster	**User: Staff in department** C10. Record actual work hours	C11. Swop duties	C12. Staff illness	**User: Personnel department** C20. Check rosters	C21. Payroll amendments	C22. Record new employees
Personnel department: Automate some tasks	●					●	●	
Remove error sources						●	●	
Observe the 120 day rule	●					●	●	
Less trivial work and stress						●	●	
Hospital department: Reduce over-time pay etc.		●						
Faster roster planning		●						
Improve roster quality		●			●			

10. Requirements template SL-07 (EHR example)

A. Background, vision, guide . . .

B. High-level demands — 20% reuse
B1. Business goals
B2. Early proof of concept

C. Tasks to support
C1. Admit patient — 1% reuse
C2. Clinical session . . .

D. Data to record
D1. Diagnoses — 1% reuse
D2. Diagnosis types . . .
D10. Data in existing systems

E. Other functional requirements
E1. Complex calculations and rules
E2. Reports — 30% reuse
E3. Expansion of the system

F. Integration with external systems

G. Technical IT architecture
G1. Use of existing HW and SW
G2. New hardware and software

H. Security — 50% reuse
H1. Access rights for users
H2. Security management
H3. Protection against data loss
H4. Protection against unintended . . .
H5. Protection against threats

I. Usability and design — 80%
I1. Ease-of-learning and task efficiency
I2. Accessibility and Look-and-Feel

J. Other requirements and deliverables
J1. Other standards to follow
J2. User training
J3. Documentation — 80% reuse
J4. Data conversion
J5. Installation

K. The customer's deliverables

L. Operation, support, and maintenance
L1. Response times
L2. Availability — 90% reuse
L3. Data storage
L4. Support
L5. Maintenance

11. SL-07 example: Demand at left, solution at right

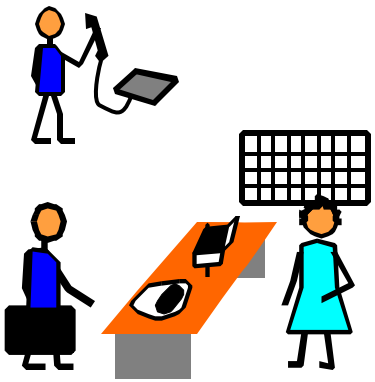
H3. Protection against data loss

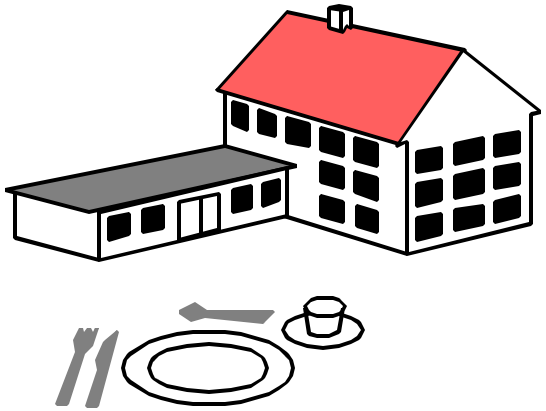
Data may unintentionally be lost or misinterpreted.

The system must protect against:	Example solutions:	Code:
1. Loss or replication of data transferred between two systems, e.g. because one or both systems close down.		
2. Concurrency problems, e.g. that user A makes a decision about medication, but before the system has recorded the decision, user B has prescribed a drug that interacts. Neither A nor B will notice the conflict.		
3. Disk crash	Periodic backup or RAID disks.	
4. Fire	Remote backup . . .	

Visions and task descriptions

13. Business goals and reqs for a hotel system





Data model
D1. Guests
D2. Rooms
D3. Services

Business goals:
- Catch small-hotel market.
- Much easier to use and install than existing hotel systems.
- Interface to existing Web-booking systems.

Verifiable?

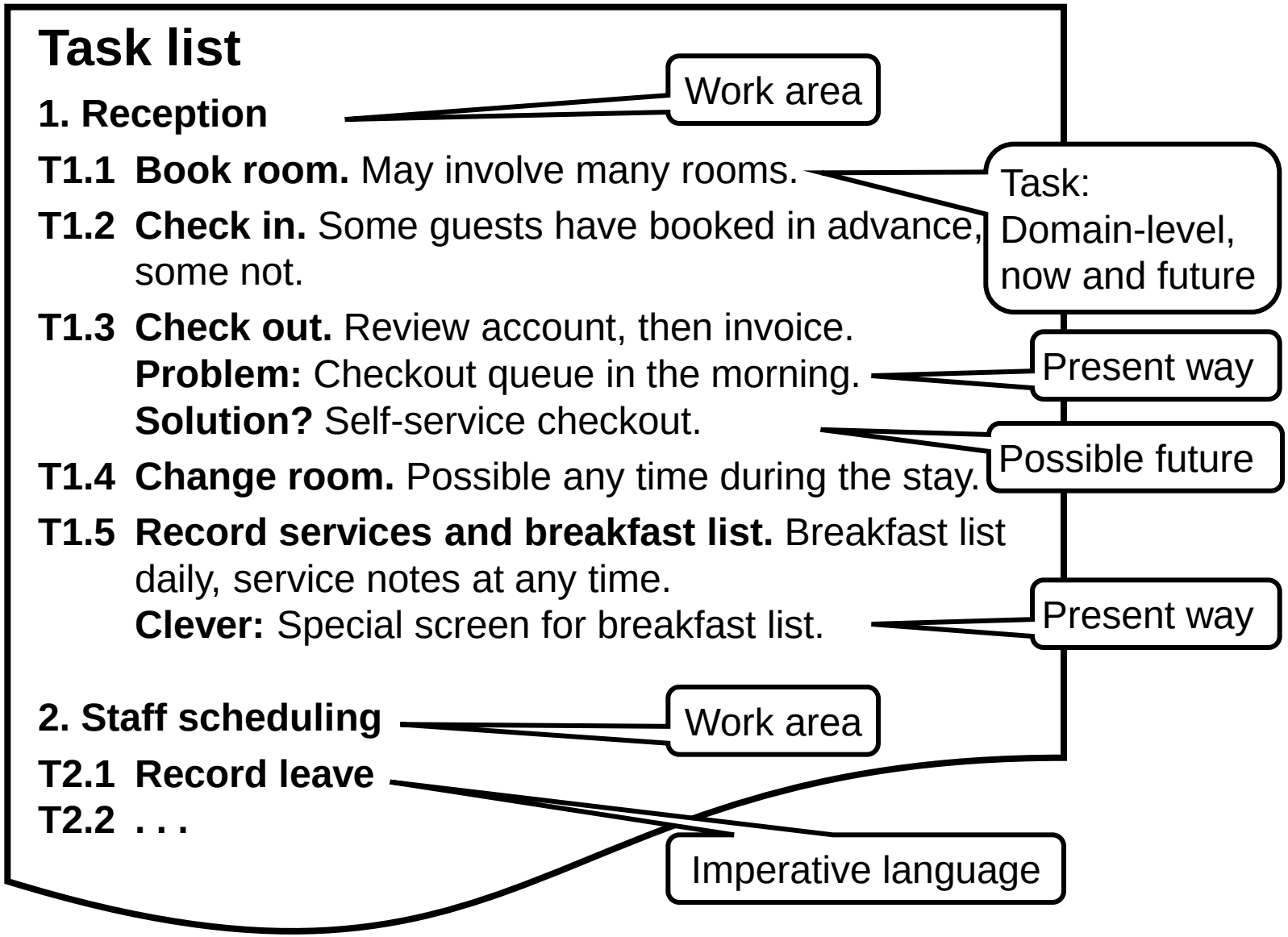
Requirements:
R1: Support tasks T1 to T34.
R2: Store data according to data model.
. . .
R7: Usable with 10 min of instruction.

Tasks to support
T1. Book room
T2. Check in
- May have booked
- Neighbor rooms

T3. Check out
T4. Change room
T5. Record services

T10. Staff scheduling

14. Annotated task list for the hotel system



15. Task description - Hotel system

C2: Check in
Start: A guest arrives.
End: The guest has got rooms. Accounting started.
Frequency: Total: Around 0.5 check-ins per room per night. Per user: 60 ...
Difficult: A bus with 60 guests arrive.
Users: Novices with little domain knowledge. Expert receptionists.

Not requirements but assumptions behind the requirements

Subtasks and variants:	**Example solutions:**
1. Find free room. **Problem:** Guest wants neighbour rooms. Wants to bargain.	System shows free rooms on floor map. System shows bargain prices, time and capacity dependent.
1a. Guest has booked in advance. **Problem:** Find guest in the records.	Soundex and closest match.
1b. No suitable rooms.	
2. Record guest data.	
3. Record guest as checked in.	
4. Deliver the key. **Problem:** Guest forgets to return key. Wants two keys.	System prints electronic keys. New key for each customer

Validation: Something missing !

Past: Problems

Domain level: Hide who does what

Future: What computer does

16. Good or bad tasks?

Good tasks:
- Closed: From trigger to "coffee break"
- Session task: Small, related tasks bundled into one task
- Imperative: Hide who does what
- Don’t program - "if the customer has booked then ... "
- No precondition: Part of the task is to check the condition

Examples:

1 Manage rooms?
2 Enter guest name?
3 Check in a guest?
4 Check in a bus of tourists?

5 Change the guest’s address etc?
6 Change the booking dates?
7 Cancel entire booking?

8 A stay at the hotel from booking to check out?

Optional subtasks in "Change booking"

Many coffee breaks. A task flow.

17. Task flow or high-level task

Flow 1: A stay at the hotel
User: The guest
Start: . . .

High-level steps:	**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.

Hierarchical decomposition?
Only in simple cases

18. Complex tasks - not hierarchical

Flow 2: Patient treatment

Start: The patient is referred to the hospital from a practitioner or arrives in emergency.

End: The patient is cured or . . .

High-level step:	**Tasks:**
1. Admit the patient	C1: Admit before arrival C2: Immediate admission
2. Make a diagnosis	C10: Clinical session
3. Plan the treatment	C10: Clinical session
4. Carry out the treatment	C10: Clinical session
5. Assess the result	C10: Clinical session
6. Discharge the patient	C3: Discharge patient . . .
7. Follow-up at home	?

19. The true work task

C10: **Perform a clinical session**

Start: Contact with the patient, e.g. ward round or emergency ward.
Or conference about the patient.

End: When we cannot do more for the patient now.

Data needs: See the data description in . . .

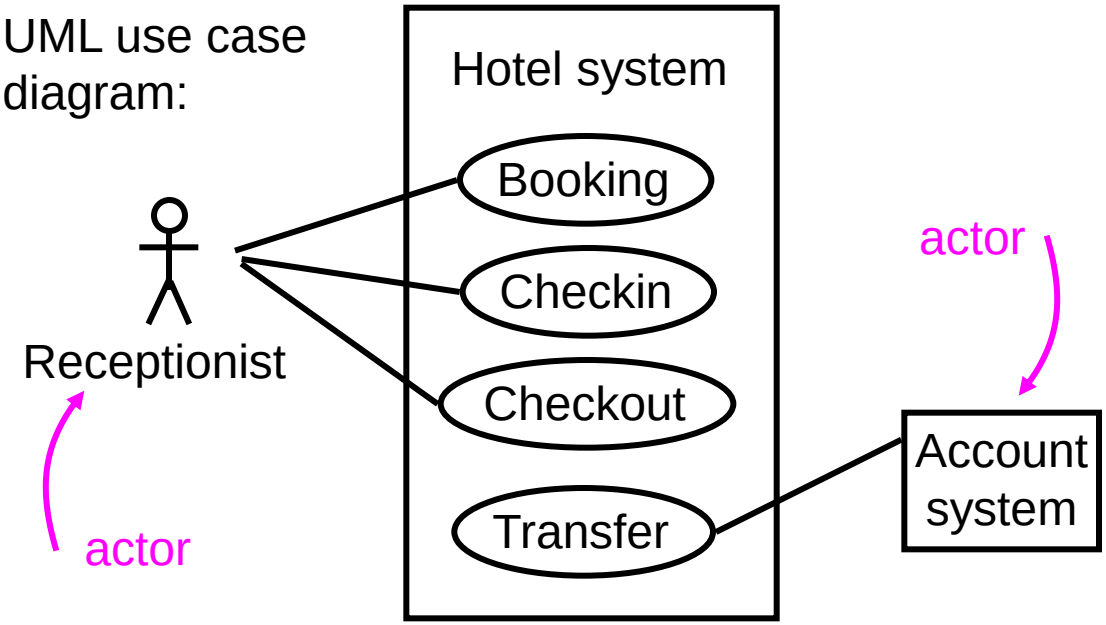
12 pages. Cover all tasks.

(All subtasks are optional and repeatable. The sequence is arbitrary)

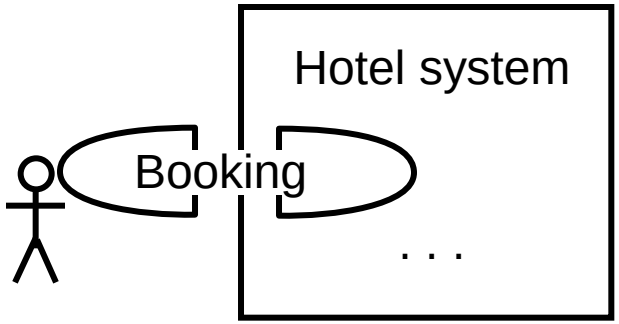
Subtasks and variants:	**Example solution:**
1. Review the state of the patient **Problem:** Overview of data	Overview of diagnoses and results
2. Provide services on the spot	Record results on the spot
3. Follow up on planned services	Overview of requested services
4. Adjust diagnoses	Record on the spot
5. Plan new services	Check with everybody's calendar ...
6. Maybe discharge the patient	Request transport, message to ...

20. Use cases versus tasks

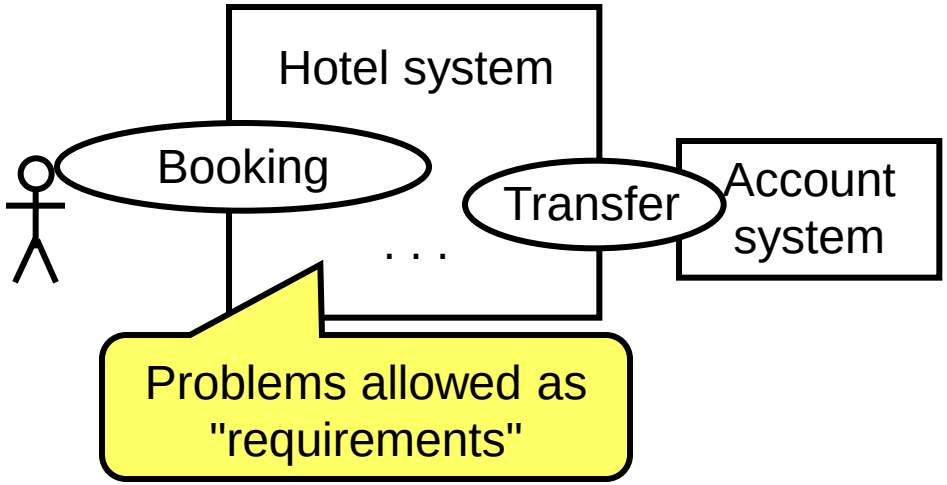
UML use case diagram:



Human and computer separated:



Task descriptions. Split postponed:



21. Eight use case solutions vs. seven task solutions

Use case approach	Task approach
Ignores problems without an easy solution. Hides very important business needs.	Records a "problem requirement". Makes it visible whether the supplier has a solution.
Invents a dialog and restricts the solution space. Often bad dialogs.	Describes what user and system do together. Supplier defines dialog.
Not suited for comparing solutions.	For each solution, customer tries out the tasks. Assesses goodness of support and problem resolution.
Template causes wrong requirements: preconditions, business rules, exceptions.	Dealt with in other requirements sections: Security, data requirements, special functional requirements.

Reference: Lauesen, S., Kuhail, M.: Use cases versus task descriptions. Proceedings of REFSQ 2011, Springer Verlag
http://www.itu.dk/~slauesen/Papers/HotlineRefsq11.pdf

Usability requirements

23. Usability problems

Examples:

The system works as intended by the programmer, but the user:

P1. Cannot figure out how to start the search. Finally finds out to use F10.

P2. Believes he has completed the task, but forgot to click Update.

P3. Sees the discount code field, but cannot figure out which code to use.

P4. Says it is crazy to use six screens to fill in ten fields.

P5. Wants to print a list of discount codes, but the system cannot do it.

Severity classes:

1 Missing functionality (not really *usability*)

2 Task failure

3 Annoying, cumbersome

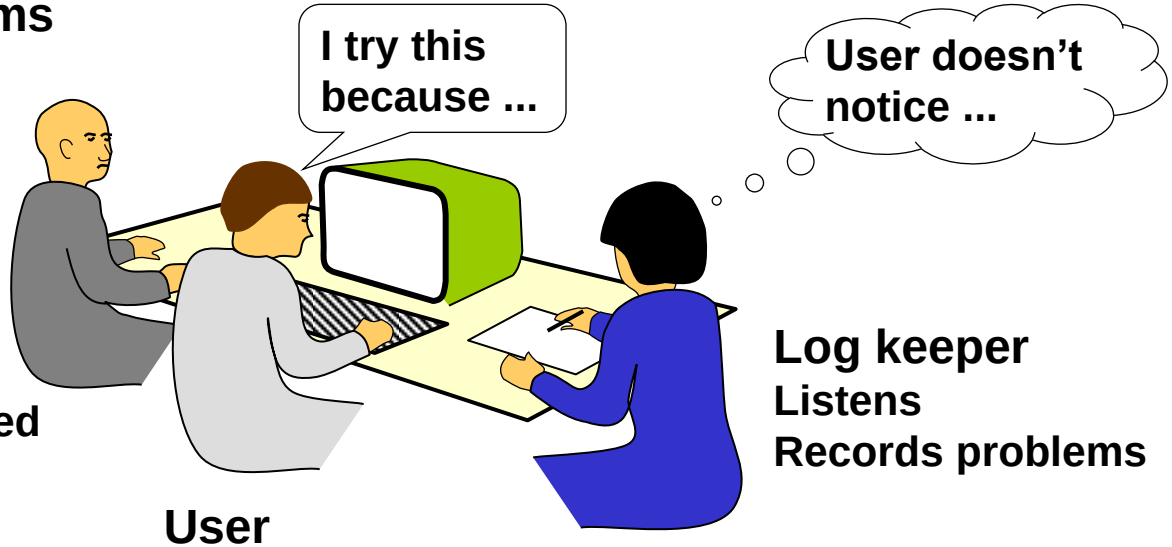
4 Medium problem (succeeds after a long time)

5 Minor problem (succeeds after a short time)

Critical problem =
Missing functionality, task failure, or annoying
+ For many users

24. Usability test - think aloud

Purpose:
Find usability problems



Facilitator
Listens
Asks as needed

Log keeper
Listens
Records problems

User
Performs tasks
Thinks aloud

Must be done before programming - Use mockups

25. Usability requirements

Details: A novice has got 5 min instruction ...
Task Q: John Simpson calls to reserve a room ...

	Risk: Cust.	Risk: Suppl.
Problem counts R1: At most 1 of 5 novices shall encounter critical problems during tasks Q and R. At most 5 medium problems on list.		
Task time R2: Novice users shall perform tasks Q and R in 15 minutes. Experienced users tasks Q, R, S in 2 minutes.		
Keystroke counts R3: Recording breakfast shall be possible with 5 keystrokes per guest. No mouse.		
Opinion poll R4: 80% of users shall find system easy to learn. 60% shall recommend system to others.		
Score for understanding R5: Show 5 users 10 common error mesages, e.g. *Amount too large.* Ask for the cause. 80% of the answers shall be correct.		

From: Soren Lauesen: Software Requirements