ETS170, TFRG55 (2025)
Requirements Engineering

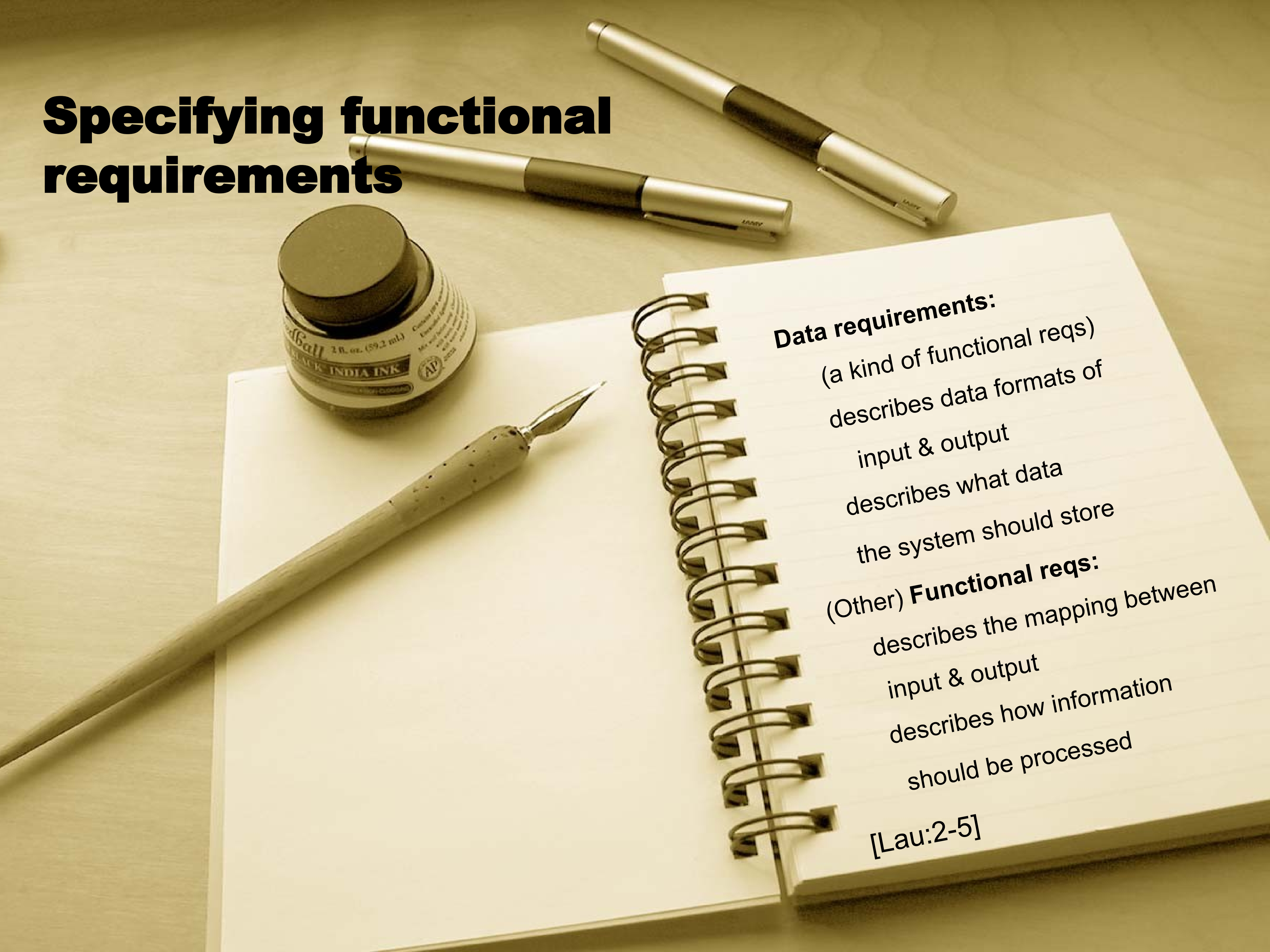
Week 2, Lectures 3+4:

Specification of functionality:
Data reqts: Lau:2,
Funtional reqts part 1: Lau:3.1-3.5
Lau:3.6 – 3.16, 4

Tutorial on a tool called reqT used at Lab1 & Lab2.

Björn Regnell
http://cs.lth.se/krav

Specifying functional requirements



Data requirements:

(a kind of functional reqs)

- describes data formats of input & output
- describes what data the system should store

(Other) **Functional reqs:**

- describes the mapping between input & output
- describes how information should be processed

[Lau:2-5]

Overview of techniques for functional requirements (Swedish terms)

Datakravstilar:

- Datamodell (=E/R-diagr.)
- Dataordlista
- Reguljära uttryck
- Virtuella fönster

First read the "gray box" of all styles so that you understand what they are about and their pros and cons. Then read in depth as needed.

Funktionella kravstilar:

- Kontextdiagram
- Händelse- & Funktionslistor
- Produktegenskapskrav
- Skärmbilder & Prototyper
- Uppgiftsbeskrivningar
- Egenskaper från uppgifter
- Uppgifter och stöd
- (Levande) Scenarier
- Högnivåuppgifter
- Användningsfall
- Uppgifter med data
- Dataflödesdiagram
- Standardkrav
- Krav på utvecklingsprocessen

Funktionella detaljer:

- Enkla och sammansatta funktioner
- Tabeller & Beslutstabeller
- Textuella processbeskrivningar
- Tillståndsdiagram
- Övergångsmatriser
- Aktivitetsdiagram
- Klassdiagram
- Samarbetsdiagram
- Sekvensdiagram

Speciella gränssnitt

- Rapporter
- Plattformskrav
- Produktintegration
- Tekniska gränssnitt

All techniques have + and - depending on the context

When is a specific style good?

The answer depends on…
- abstraction level
- project type
- the stakeholders
- tool support
- the amount of requirements
- …

Use a well-balanced combination!
…but how do you know that it all fits together?
-> checking consistency is an important part of validation!

Data requirements

Examples:

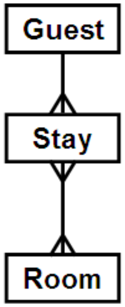
- Mobile Subscriber data (roaming data, phone book items,)
- Image data (date, resolution, name, category),
- Music data (album, artist, genre, name, frequency played, rating), ...

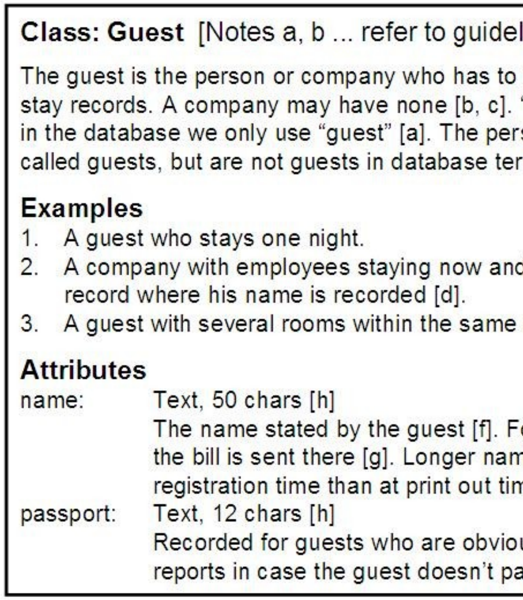
Techniques for modelling data in [Lauesen]:

- Data model (e.g. E/R-diagrams)
- Data dictionary
- Data expressions
- Virtual windows

Data requirements techniques – Summary

- **Data model (E/R-diagr.)**
  - Block diagram describing data inside and outside the product
  - Precise and insensitive to abstraction level
  - Excellent for experts – difficult for users; takes time to learn
  - Easy to verify by experts that the data is handled by the product
  - Difficult to decide how much detail should be included in the model





Class: Guest [Notes a, b ... refer to guidel

The guest is the person or company who has to
stay records. A company may have none [b, c].
in the database we only use "guest" [a]. The pers
called guests, but are not guests in database ter

Examples

1. A guest who stays one night.
2. A company with employees staying now and
   record where his name is recorded [d].
3. A guest with several rooms within the same

Attributes

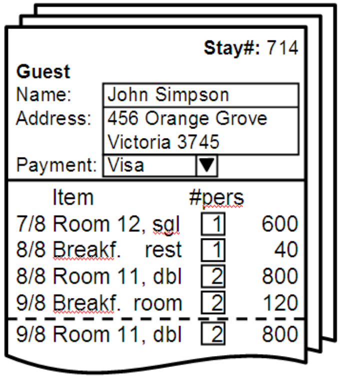
name: Text, 50 chars [h]
The name stated by the guest [f]. F
the bill is sent there [g]. Longer nam
registration time than at print out tim

passport: Text, 12 chars [h]
Recorded for guests who are obvio
reports in case the guest doesn't pa

- **Data dictionary**
  - Textual description of data inside and outside the product
  - Structured and systematic descriptions using verbal text
  - Very expressive, can be used for all levels of detail and special cases
  - Easy to validate by experts and non-experts
  - Takes long time to write; when is it good enough? (Start with difficult parts!!)
- **Data expressions (regular expressions)**
  - Compact formulas for describing data sequences
  - Useful for composite data and message protocolls
  - Excellent for experts, acceptable for many users
  - No visual overview

```
passport number = letter + {digit}*8
room state = { free | booked | occupied | repair }
account data = transfer + {account record}* + done
```

- **Virtual windows**
  - Simplified screens with graphics and realistic data, but no buttons and menues
  - Excellent for both experts and users
  - Easy to validate and verify
  - Risk of overdoing it and start designing the user interface

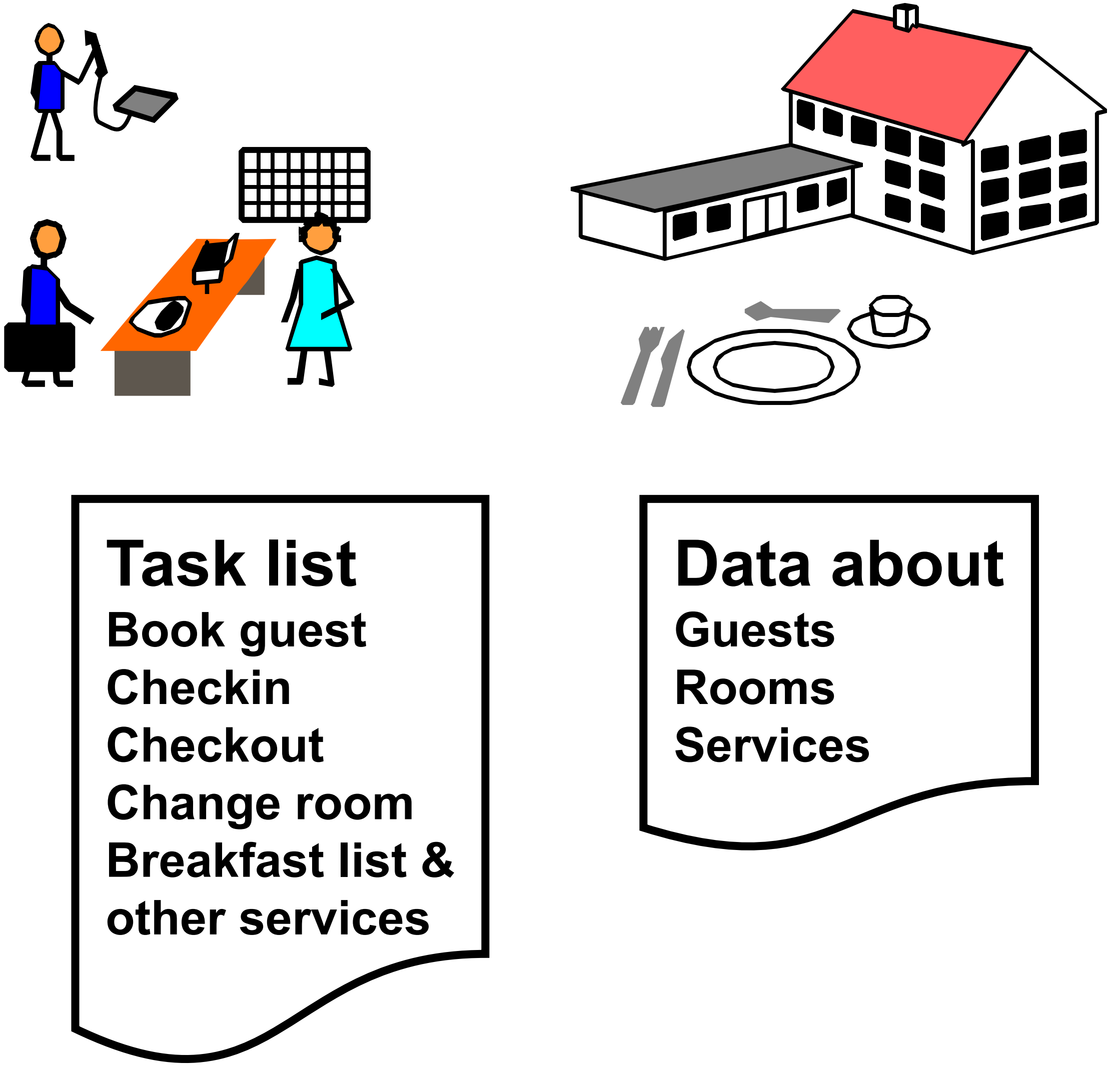


Stay#: 714

Guest
Name: John Simpson
Address: 456 Orange Grove
Victoria 3745
Payment: Visa

Item	#pers	
7/8 Room 12, sgl	1	600
8/8 Breakf. rest	1	40
8/8 Room 11, dbl	2	800
9/8 Breakf. room	2	120
9/8 Room 11, dbl	2	800

Fig 2.1 The hotel system



Task list

Book guest
Checkin
Checkout
Change room
Breakfast list &
other services

Data about

Guests
Rooms
Services

From: Soren Lauesen: Software Requirements

Fig 2.3 Data dictionary

Class: Guest [Notes a, b ... refer to guidelines]

The guest is the person or company who has to pay the bill. A guest has one or more stay records. A company may have none [b, c]. “Customer” is a synonym for guest, but in the database we only use “guest” [a]. The persons staying in the rooms are also called guests, but are not guests in database terms [a].

Examples

1. A guest who stays one night.
2. A company with employees staying now and then, each of them with his own stay record where his name is recorded [d].
3. A guest with several rooms within the same stay.

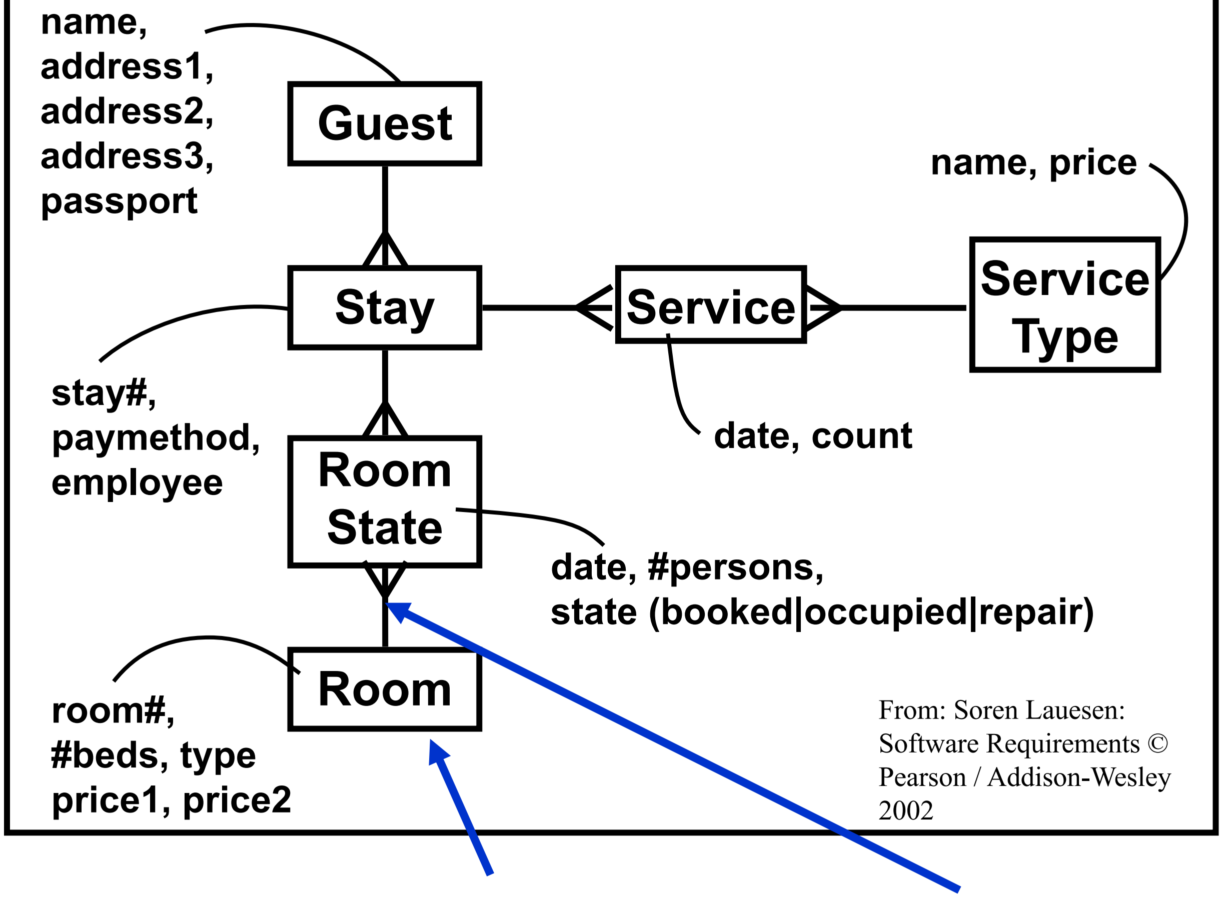
Attributes

name: Text, 50 chars [h]
The name stated by the guest [f]. For companies the official name since the bill is sent there [g]. Longer names exist, but better truncate at registration time than at print out time [g, j].
passport: Text, 12 chars [h]
Recorded for guests who are obviously foreigners [f, i]. Used for police reports in case the guest doesn’t pay [g] . . .

From: Soren Lauesen: Software Requirements

Fig 2.2A Data model (E/R-diagram)

R2: The system shall store the following data:



name,
address1,
address2,
address3,
passport

Guest

Stay

Service

Service Type

name, price

stay#,
paymethod,
employee

date, count

Room State

date, #persons,
state (booked|occupied|repair)

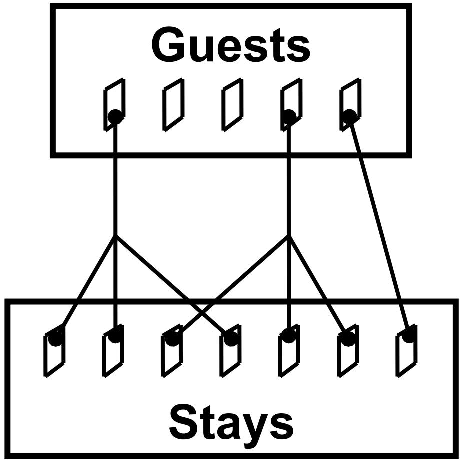
Room

room#,
#beds, type
price1, price2

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

One-to-many (1:m)

Each guest
connected to
zero or more stays



Guests

Stays

Each stay
connected to
one guest record

Entities and Relationships

http://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

Cardinality of relations

Fig 2.4A Data expressions

Notation with plus as concatenator

booking request = guest data + period + room type

guest data = guest name + address + paymethod
+ [passport number]

passport number = letter + {digit}*8

room state = { free | booked | occupied | repair }

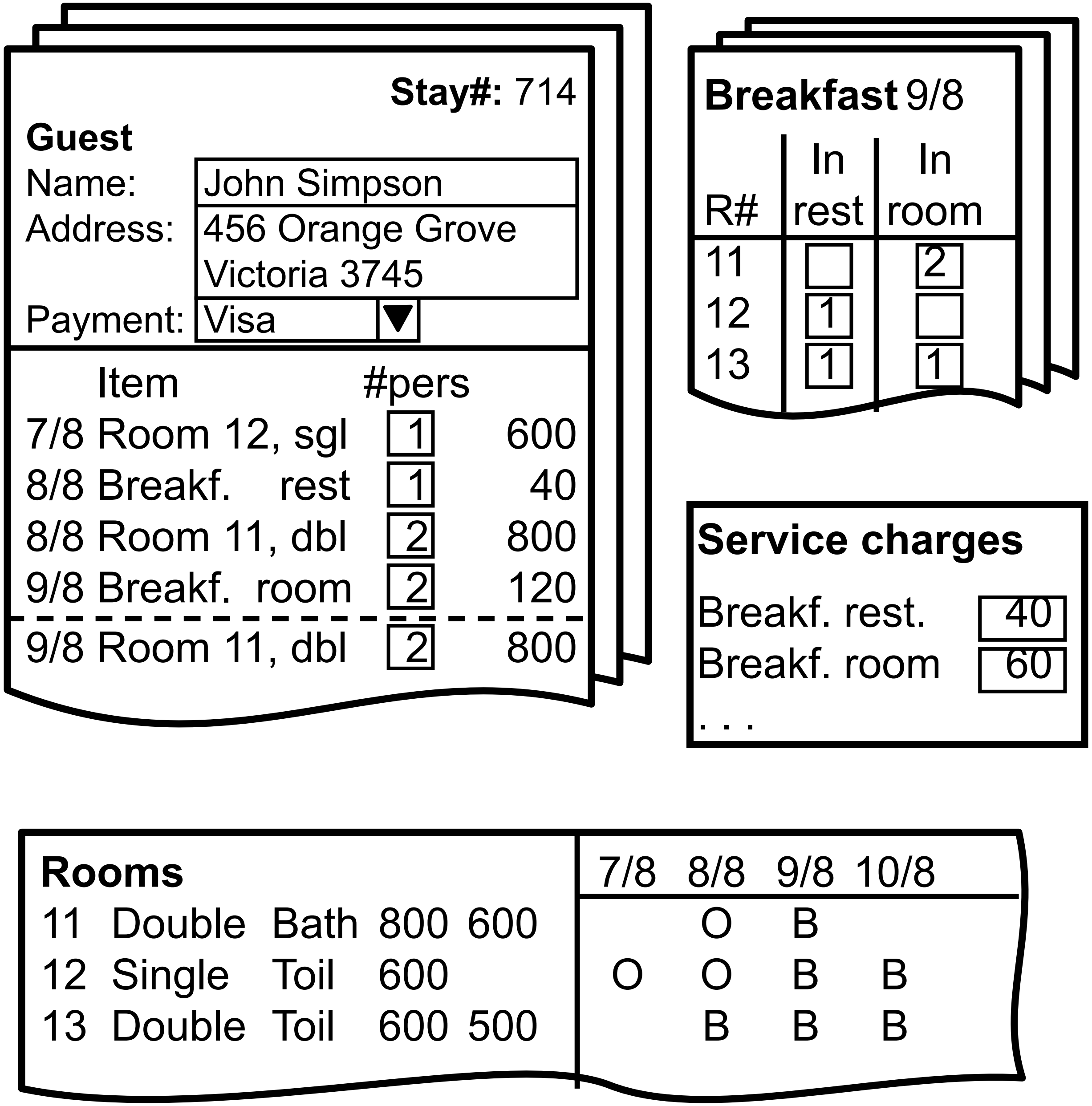
account data = transfer + {account record}* + done

From: Soren Lauesen: Software Requirements

Fig 2.5 Virtual Windows

R1: The product shall store data corresponding to the following virtual windows:

R2: The final screens shall look like the virtual windows ??



Stay#: 714

Guest

Name: John Simpson

Address: 456 Orange Grove
Victoria 3745

Payment: Visa ▼

Item	#pers	
7/8 Room 12, sgl	1	600
8/8 Breakf. rest	1	40
8/8 Room 11, dbl	2	800
9/8 Breakf. room	2	120
9/8 Room 11, dbl	2	800

Breakfast 9/8

R#	In rest	In room
11	□	2
12	1	□
13	1	1

Service charges

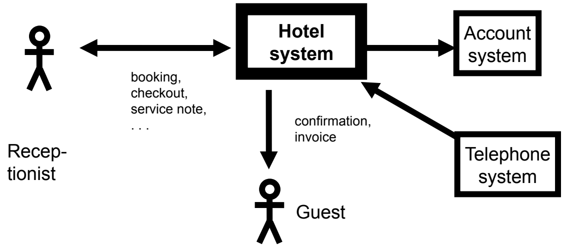
Breakf. rest.	40
Breakf. room	60
. . .	

Rooms					7/8	8/8	9/8	10/8
11	Double	Bath	800	600		O	B	
12	Single	Toil	600		O	O	B	B
13	Double	Toil	600	500		B	B	B

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

Functional Requirements Part 1 Summary

- **Context Diagram**
  - Diagram of product and its surrounding
  - Defining product scope
  - Very useful!



- **Event- and function lists**
  - Lists of events and functions
    - Domain or product level
  - Good as checklists at verification
  - Validation at product level?

R1: The product shall **support** the following business events / user activities / tasks:

R1.1 Guest books
R1.2 Guest checks in
R1.3 …

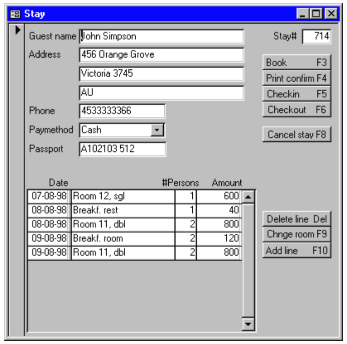
- **Feature requirements**
  - Textual requirement: ”the product shall …”
  - High expressive power
  - Acceptable to most stakheolders
  - Can lead to false sense of security
    - How to ensure that goal-level covered?

R1: The product shall be able to record that a room is occupied for repair in a specified period.

R2: The product shall ….

R3: The product shall ….

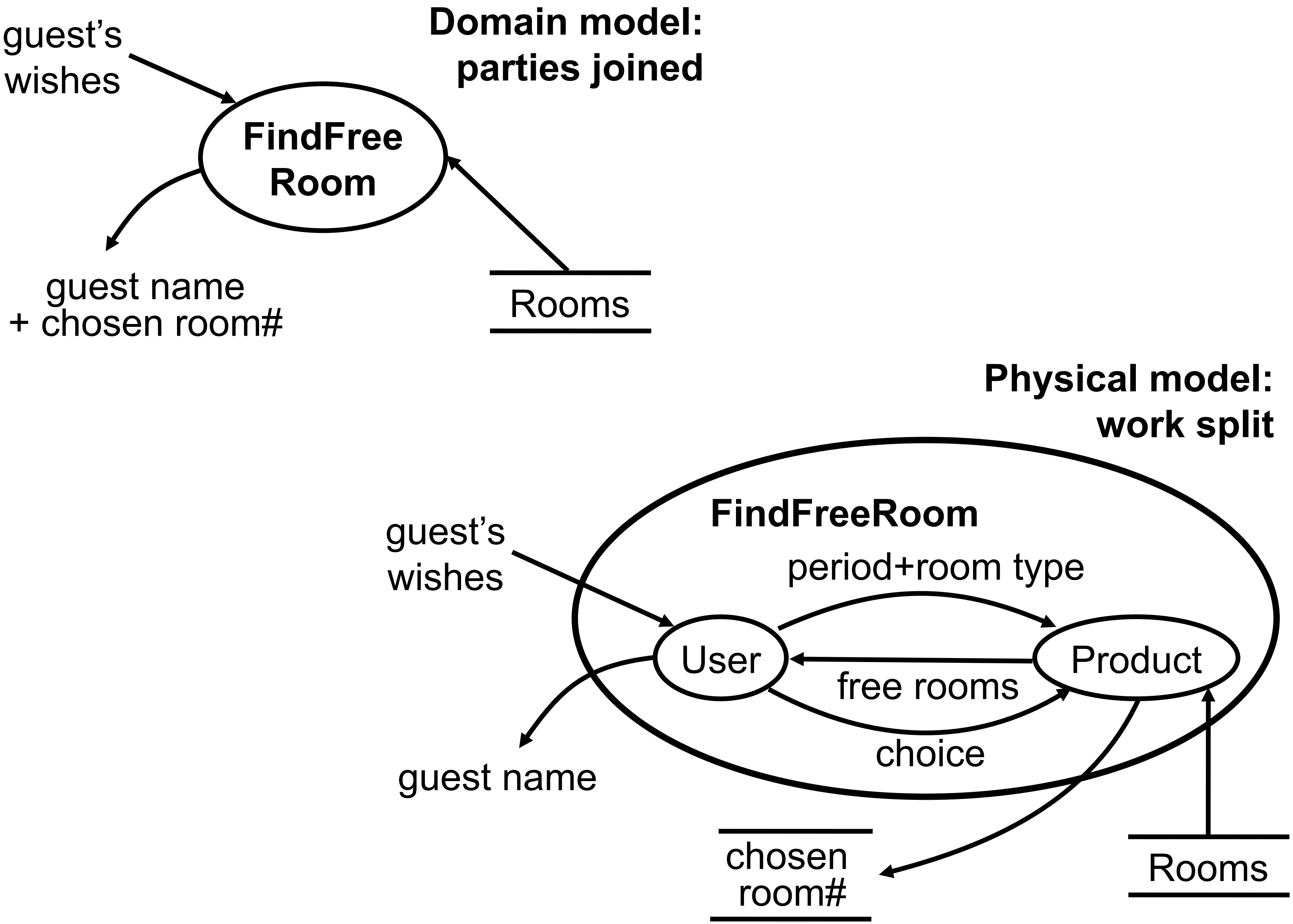
- **Screens and Prototypes**
  - Screen pictures + what buttons do
  - Excellent as design-level requirements if carefully tested
  - Not good when for COTS-based systems



Guest name John Simpson
Stay# 714
Address 456 Orange Grove
Victoria 3745
AU
Phone 4533333366
Paymethod Cash
Passport A102103 512

Date		#Persons	Amount
07-08-98	Room 12, sgl	1	600
08-08-98	Breakf. rest	1	40
08-08-98	Room 11, dbl	2	800
09-08-98	Breakf. room	2	120
09-08-98	Room 11, dbl	2	800

Fig 3.1 Human-computer - who does what?

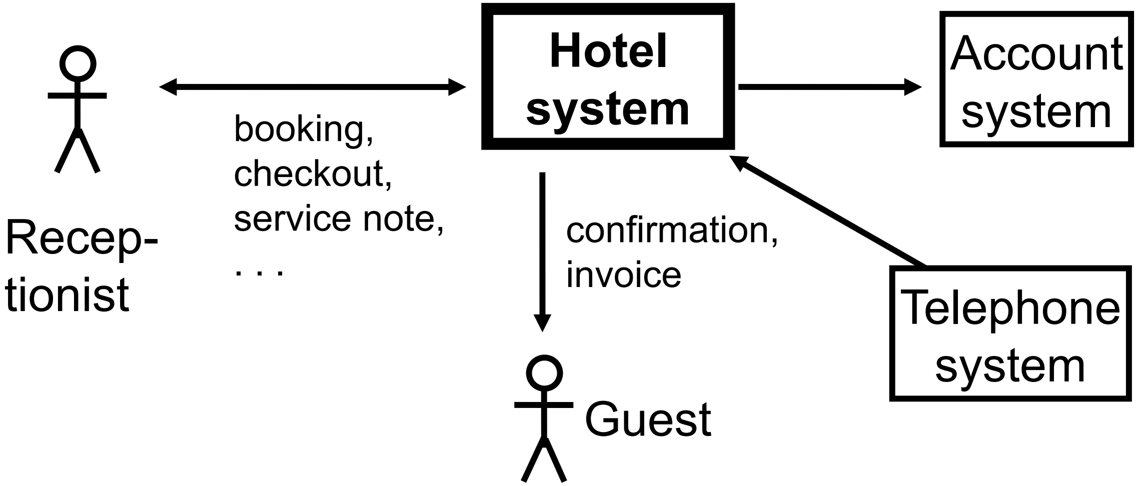

Domain model: parties joined

Physical model: work split

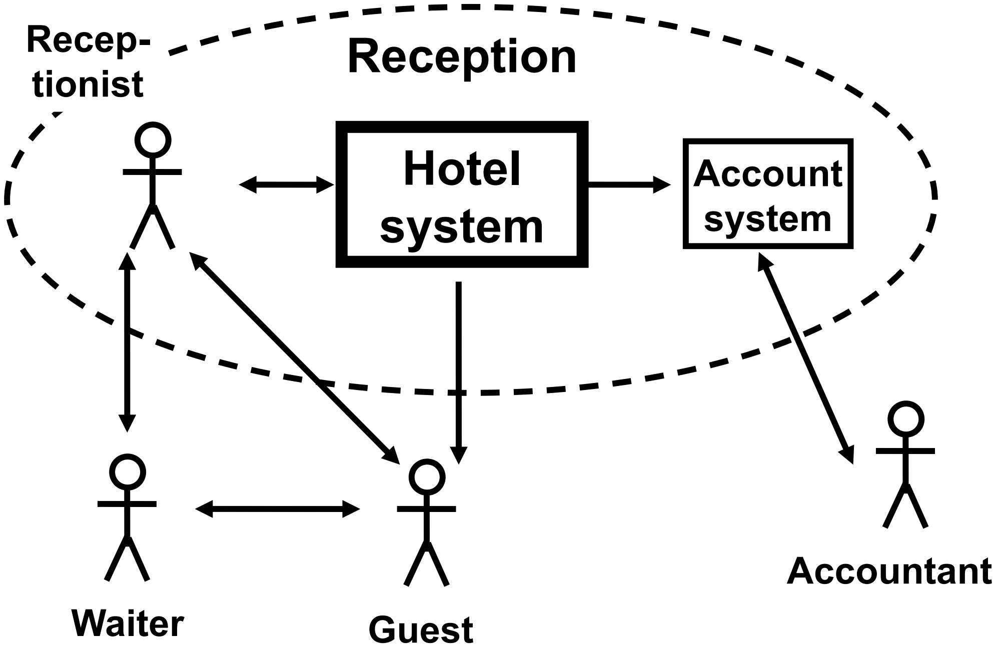
From: Soren Lauesen: Software Requirements

Fig 3.2 Context diagram

R1:
The product shall have the following interfaces:

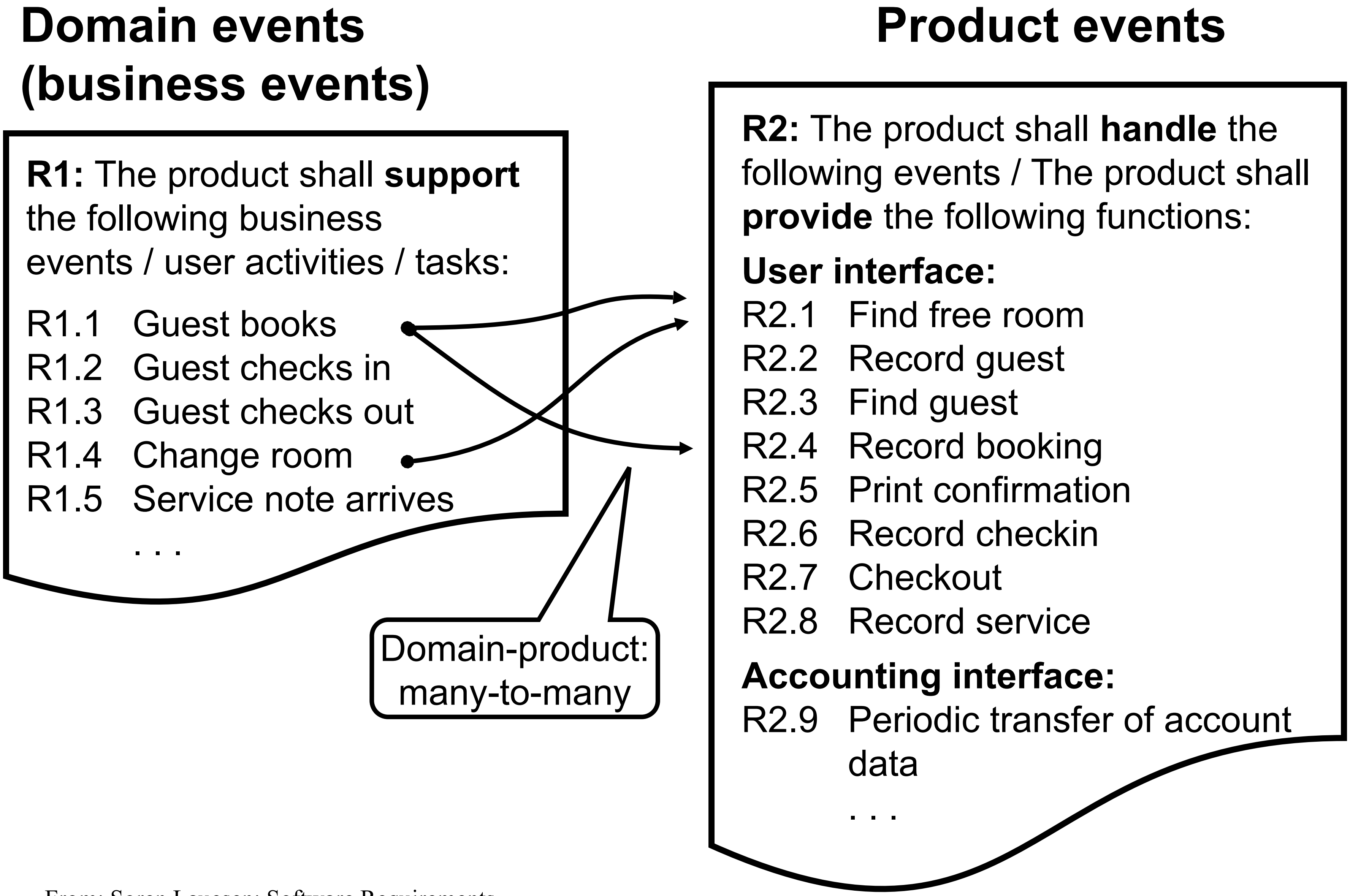


R2 ??:
The reception domain communicates with the surroundings in this way:



From: Soren Lauesen: Software Requirements

Fig 3.3 Event list & function list



Domain events (business events)

R1: The product shall **support** the following business events / user activities / tasks:

R1.1 Guest books
R1.2 Guest checks in
R1.3 Guest checks out
R1.4 Change room
R1.5 Service note arrives
. . .

Product events

R2: The product shall **handle** the following events / The product shall **provide** the following functions:

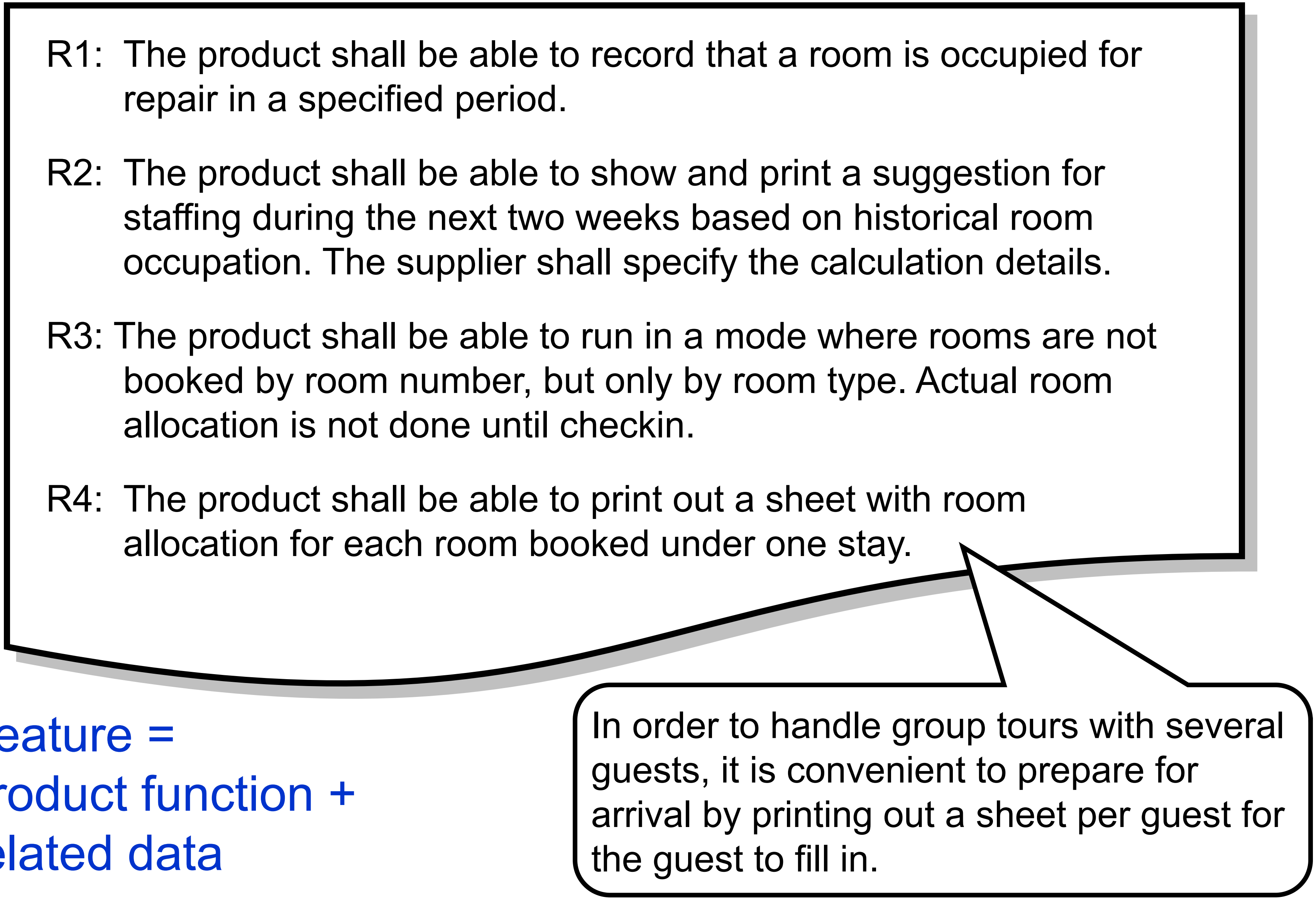
User interface:
R2.1 Find free room
R2.2 Record guest
R2.3 Find guest
R2.4 Record booking
R2.5 Print confirmation
R2.6 Record checkin
R2.7 Checkout
R2.8 Record service

Accounting interface:
R2.9 Periodic transfer of account data
. . .

Domain-product: many-to-many

From: Soren Lauesen: Software Requirements

Fig 3.4 Feature requirements



R1: The product shall be able to record that a room is occupied for repair in a specified period.

R2: The product shall be able to show and print a suggestion for staffing during the next two weeks based on historical room occupation. The supplier shall specify the calculation details.

R3: The product shall be able to run in a mode where rooms are not booked by room number, but only by room type. Actual room allocation is not done until checkin.

R4: The product shall be able to print out a sheet with room allocation for each room booked under one stay.

In order to handle group tours with several guests, it is convenient to prepare for arrival by printing out a sheet per guest for the guest to fill in.

Feature =
product function +
related data

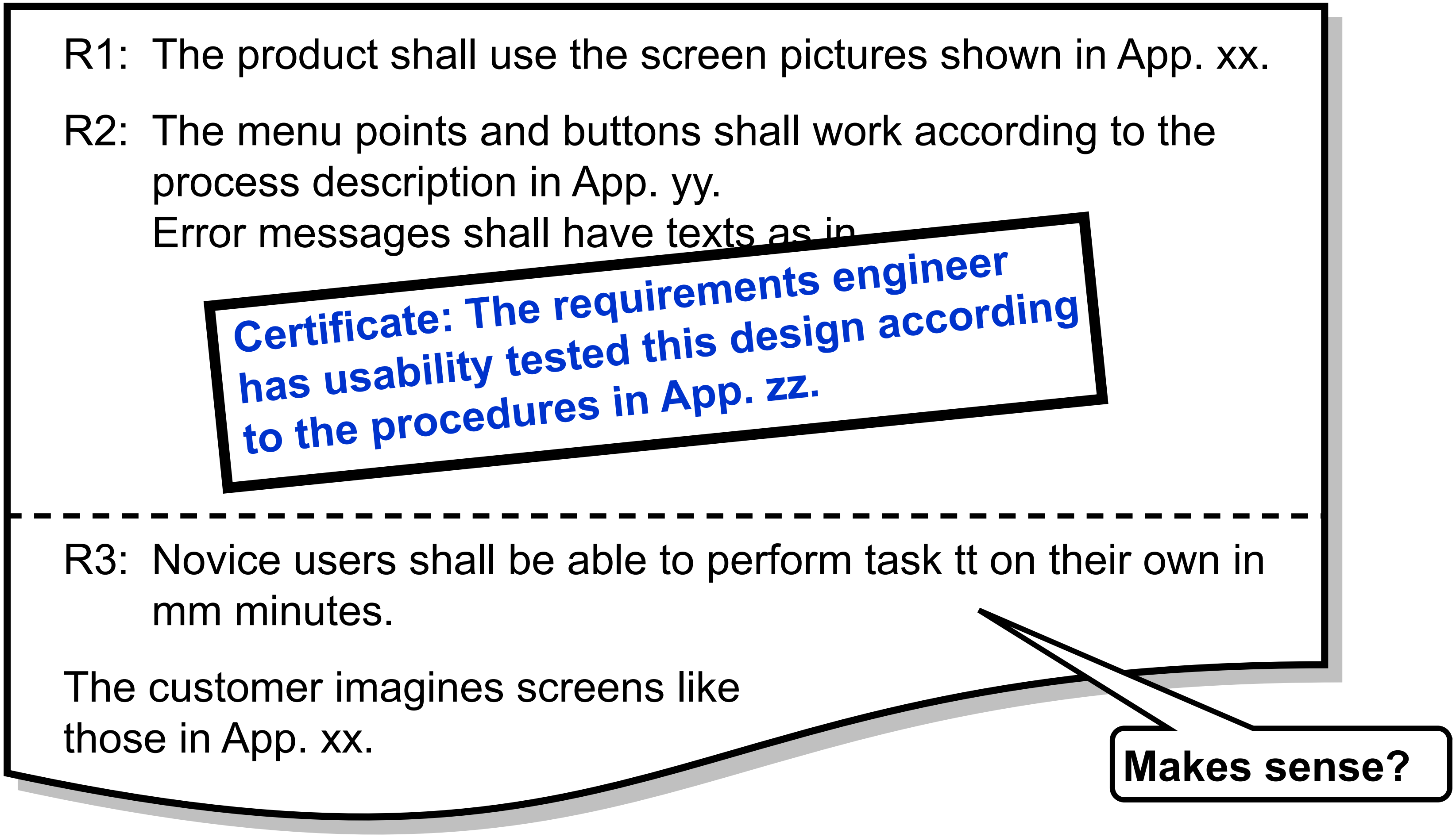
From: Soren Lauesen: Software Requirements

What is a ‘feature’?

Some possible definitions:

1. A textual shall-statement requirement
2. A releasable characteristic of a (software-intensive) product
3. A (high-level, coherent) bundle of requirements
4. A ‘decision unit’ that can be ‘in’ or ‘out’ of a release plan depending on:
   - What it gives (investment return)
   - What it takes (investment costs)
   - Politics, Beliefs, Loyalties, Preferences ...

Fig 3.5A   Screens & prototypes



R1: The product shall use the screen pictures shown in App. xx.

R2: The menu points and buttons shall work according to the process description in App. yy.
Error messages shall have texts as in ...

Certificate: The requirements engineer has usability tested this design according to the procedures in App. zz.

R3: Novice users shall be able to perform task tt on their own in mm minutes.

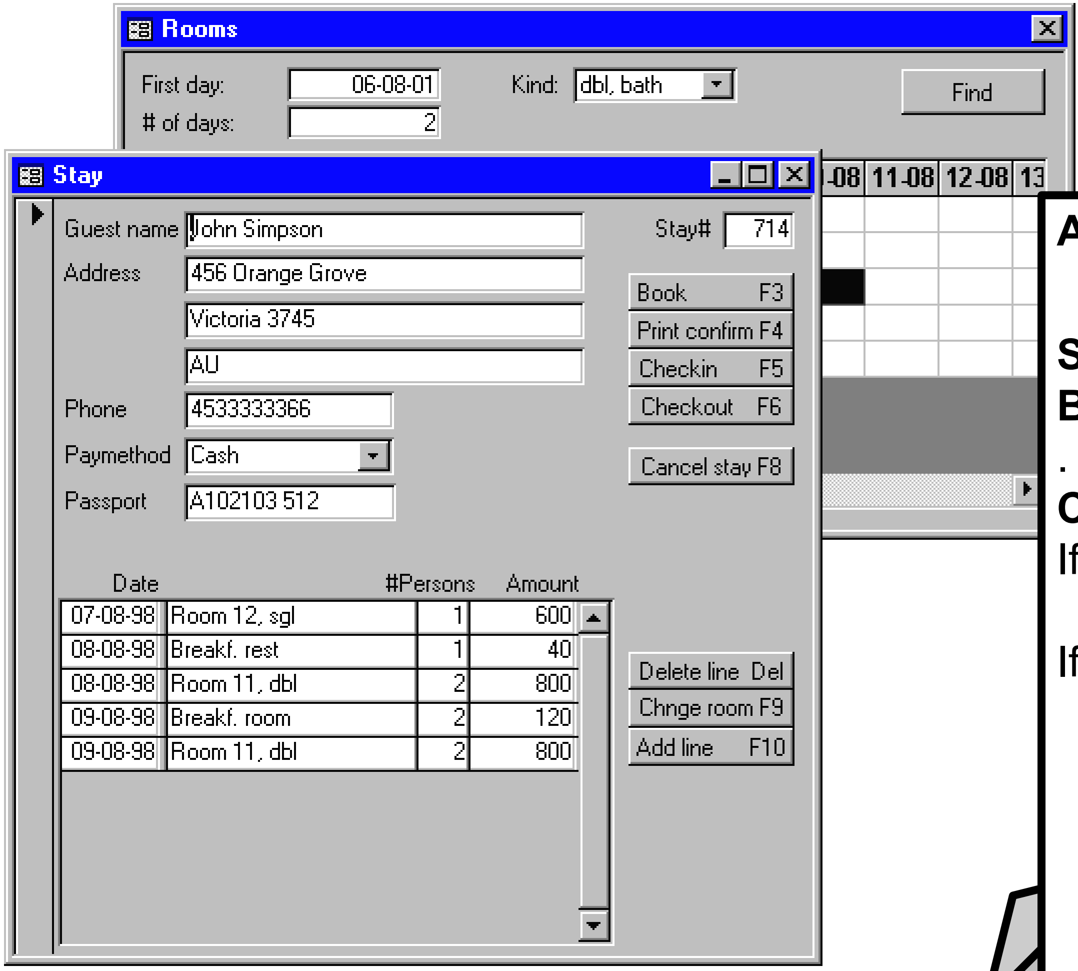
The customer imagines screens like those in App. xx.

Makes sense?

From: Soren Lauesen: Software Requirements

Fig 3.5B Screens & prototypes

Appendix xx. Required screens



Rooms

First day:	06-08-01
# of days:	2
Kind:	dbl, bath

Stay

Guest name	John Simpson
Address	456 Orange Grove
	Victoria 3745
	AU
Phone	4533333366
Paymethod	Cash
Passport	A102103 512
Stay#	714

Date		#Persons	Amount
07-08-98	Room 12, sgl	1	600
08-08-98	Breakf. rest	1	40
08-08-98	Room 11, dbl	2	800
09-08-98	Breakf. room	2	120
09-08-98	Room 11, dbl	2	800

Appendix yy. Required functions

Stay window

Book:

. . .

Checkin:

If stay is booked, record the booked rooms as occupied.

If stay is not recorded,

Check selected rooms free and guest information complete.

Record guest and stay.

Record selected rooms as occupied.

If stay is checked in, . . .

From: Soren Lauesen: Software Requirements

Overview of styles for specifying functional requirements (Swedish terminology)

Datakravstilar:

✓ Datamodell (=E/R-diagr.)

✓ Dataordlista

✓ Reguljära uttryck

✓ Virtuella fönster

Funktionella kravstilar:

✓ Kontextdiagram

✓ Händelse- & Funktionslistor

✓ Produktegenskapskrav

✓ Skärmbilder & Prototyper

- **Uppgiftsbeskrivningar**
- **Egenskaper från uppgifter**
- **Uppgifter och stöd**
- **(Levande) Scenarier**
- **Högnivåuppgifter**
- **Användningsfall**
- Uppgifter med data
- Dataflödesdiagram
- Standardkrav
- Krav på utvecklingsprocessen

Funktionella detaljer:

- Enkla och sammansatta funktioner
- Tabeller & Beslutstabeller
- Textuella processbeskrivningar
- **Tillståndsdiagram**
- Övergångsmatriser
- Aktivitetsdiagram
- **Klassdiagram**
- Samarbetsdiagram
- **Sekvensdiagram**

Speciella gränssnitt

- Rapporter
- Plattformskrav
- Produktintegration
- Tekniska gränssnitt

Different types of requirements abstraction

- Hierarchical decomposition (nested bundling)
- Level of detail
  (degree of completeness)
- Goal-design scale
  - goal: why: intentional level
  - domain: who: context level
  - product: what: functions+data
  - design: how: "inside" product

Complete requirements?

- In practice you cannot specify everything to the last detail!
- What is good enough?
  -> Depends on the context
- Tip: Focus on the reqs that have the largest risk of…
  - misinterpretation by stakeholders
  - misfit of the final system
- Do not spend large efforts on the “easy” requirements that everybody already knows much about
- Do pre-studies: conceptual and feasibility studies, prototypes etc. to ...
  - ... reduce risks
  - “jump” between abstraction levels

Terminology confusion:
Scenario, Task, Use Case, User Story

(sv: scenario, uppgift, användningsfall, användarberättelse)

Scenario =

(1) A general term for all types of example-based dynamic descriptions of system usage (Usability Engieering ’Tasks’, UML ’Use cases’, Scrum ’User Stories’, etc.

(2) A specific realisation (instance) of a use case

(3) A detailed narrative describing an experience of a user, also known as ”vivid scenario”

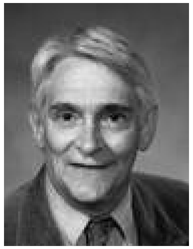
(4) Future scenaries, possible future events /outcomes, in e.g. risk managament

…

In addition there are many variants of Use Cases, Tasks, etc (Jacobson, Cockburn, Lauesen, …)

A brief history of scenarios-based requirements

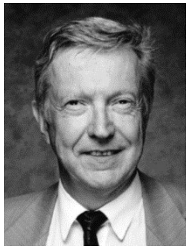
- Scenario-based requirements have been around for a while:
  - Task descriptions from Usability Engineering,
    e.g. J.F. Allen ‘80ies, J.M. Carroll ‘90ies
  - Scenario-based RE. e.g. J.W. Hooper, P. Hsia (1982), Potts (1995), Suttcliffe (1998)
  - Message Sequence Charts within Telecom, SDL’87
- 1992: Ivar Jacobson coined the term ”**use case**” in his book ”OOSE”
- Mid 1990ies: ”*three amigos*” (Booch, Rumbaugh, Jacobson) at Rational (later IBM) -> UML, RUP
- 2001: Beck starts agile movement with ”**user stories**”
  As a **<user>** *I want* **<action>** *so that* **<purpose>**
- 2011: Lauesen publishes study on use cases vs tasks; use cases are questioned...



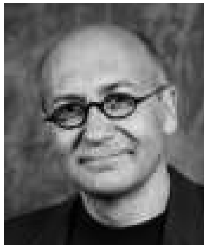
John M. Carroll

Grady Booch

James Rumbaugh



Ivar Jacobson



Colin Potts

Alistair Sutcliffe

Kent Beck

Søren Lauesen

Användningsfall - begrepp
Use case - concepts

Actor
– a category of users, a user role
Use case
– fulfills a goal in a usage context
Scenario (*several different other meanings*)
– a specific realization of a use case

Examples:

- **ATM machine: “Withdraw money” (enter card, enter code...)**
- **Word processor: “Check spelling” (select paragraph, select dictionary...)**

Good for what?

Aktör
– en kategori av användare, roll
Användningsfall
– måluppfyllande användningssituation
Scenario (*används i flera andra betydelser*)
– en specifik realisering

Exempel:

- **Bankomat: “Ta ut pengar” (stoppa in kort, knappa in kod ...)**
- **Ordbehandling: “Kontrollera stavning” (välj stycke, välj ordlista ...)**

Bra till vadå?

Some advantages with (example-based) dynamic models of system usage

- Easy to understand by non-engineers (if not too abstract)
- Gives a dynamic perspective on requirements
- Can relate requirements at different abstraction levels
- Can provide a structure for requirements
- Good for modeling functional requirements
- Can support traceability
- Can be a good basis for test cases

Traps and pitfalls with scenario-based requirements

- Too much details – ”over specification”
- Too few details – ”under specification”
- Fragmentations
- Premature design
- Non-uniform specifications
  - Structure, content, level of abstr., terminology, ...
- Inconsistent specification
  - Mutually contradictory specifications
- Incomplete specifications
- Functional decomposition -> bad OO design

Fig 3.6A 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

Missing sub-task?

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

From: Soren Lauesen: Software Requirements

Fig 3.6B   Triggers, options, preconditions

Task: **Look at your new e-mails**

Purpose: Reply, file, forward, delete, handle later.

Trigger: A mail arrives.

- Someone asks you to look.
- You have been in a meeting and are curious about new mail.

Frequency: . . .

Task: **Change booking**

Purpose: . . .

Precondition: Guest has booked?

Trigger: Guest calls

. . .

Makes sense?

Sub-tasks:

1. Find booking
2. Modify guest data, e.g. address (optional)
3. Modify room data, e.g. two rooms (optional)
4. Cancel booking (optional)

From: Soren Lauesen:
Software Requirements

Fig 3.8A Tasks & Support

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

Sub-tasks:	**Example solution:**
1. Find room. **Problem:** Guest wants neighbor 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.

Past: Problems

Domain level

Future: Computer part

Fig 3.9 Vivid scenario

Scenario: The evening duty

Doug Larsson had studied all afternoon and was a bit exhausted when arriving 6 pm to start his turn in the reception. The first task was to prepare the arrival of the bus of tourists expected 7 pm. He printed out all the checkin sheets and put them on the desk with the appropriate room key on each sheet.

In the middle of that a family arrived asking for rooms. They tried to bargain and Doug always felt uneasy about that. Should he give them a discount? Fortunately Jane came out from the back office and told them with her persuading smile that she could offer 10% discount on the children’s room. They accepted, and Doug was left to assign them their rooms. They wanted an adjoining room for the kids, and as usual he couldn’t remember which rooms were neighbors.

Around 10 pm, everything was quiet, and he tried to do some of his homework, but immediately became sleepy. Too bad - he wasn’t allowed to sleep at work until 1 AM. Fortunately the office computer allowed him to surf the net. That kept him awake and even helped him with some of his homework.

From: Soren Lauesen: Software Requirements

Fig 3.10 Good tasks

Good tasks:

- Closed: goal reached, pleasant feeling
- Session: Small, related tasks in one description
- Don’t 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

Fig 3.11 High-level tasks

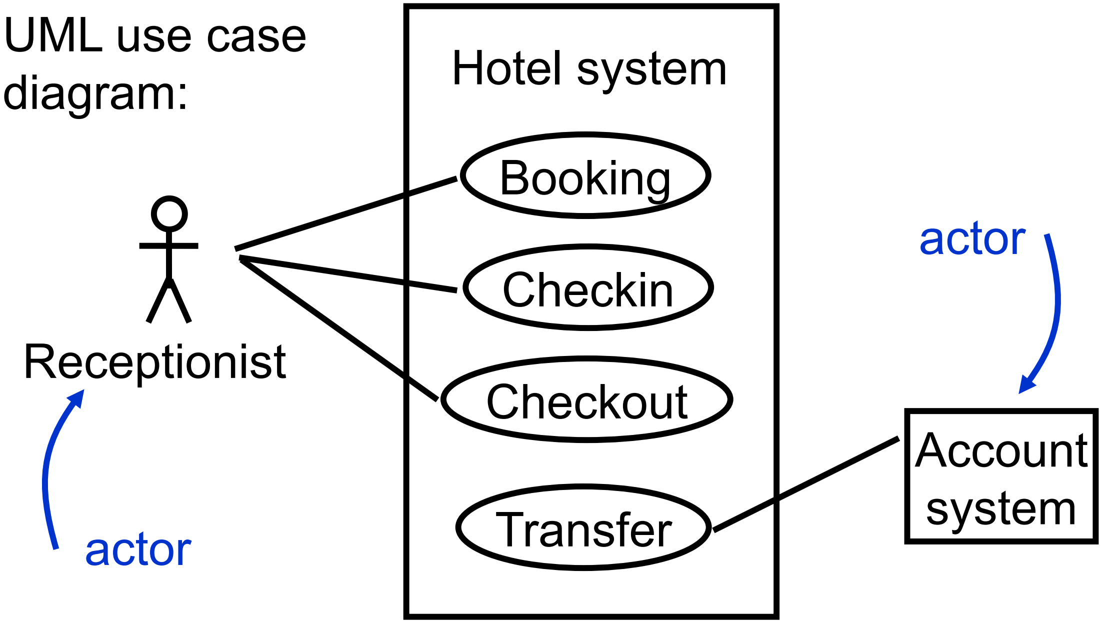
Task: 1. A stay at the hotel
Actor: The guest
Purpose: . . .

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.

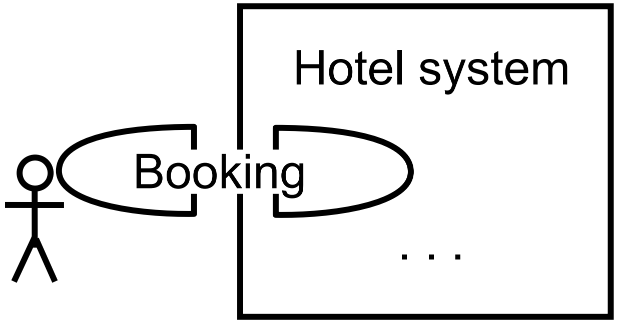
From: Soren Lauesen: Software Requirements

Fig 3.12A Use cases vs. tasks

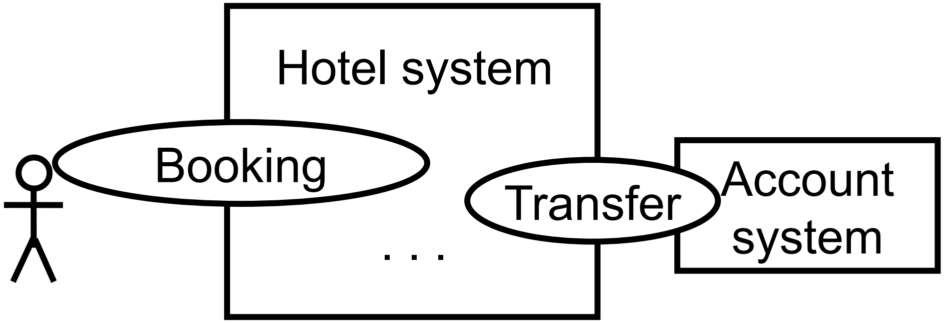
UML use case diagram:



Human and computer separated:



Task descriptions. Split postponed:



From: Soren Lauesen: Software Requirements

Fig 3.12B   Human and/or computer

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

Computer-centric use case

Use case: Check in a booked guest

Trigger: Receptionist selects check in

Read booking number
Display guest and booking details
Read and store modifications
Wait for checkin command
Select free room(s)
Mark them as occupied
Add them to guest details
Display room number(s)

End use case

From: Soren Lauesen: Software Requirements

Fig 3.15 Standards as requirements

R1: Data transfer to the account package shall be done through a file with the format described in WonderAccount Interface Guide xx.yy. The account numbers shall be . . .

R2: The user interface shall follow MS Windows Style Guide, xx.yy. The MS Word user interface should be used as a model where appropriate.

R3: Shall run under MS-Windows release xx.yy. Supplier shall port product to new releases within ______ months.

R4: Shall follow good accounting practice. The supplier shall obtain the necessary certification.

R5: The supplier shall update the payroll computations in accordance with new union agreements within one month after release of the agreement.

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

Fig 3.16 Development process as requirement

R1: System development shall use iterative development based on prototypes as described in App. xx.

Generates new requirements?

R2: Supplier shall deliver additional screens with a complexity like screen S3 at a price of $____ per screen.

R3: All developers shall spend at least two days working with the users on their daily tasks.

R4: A special review shall be conducted at the end of each development activity to verify that all requirements and system goals are duly considered. The customer's representative shall participate in the review.

R5: Customer and supplier shall meet at least two hours bi-weekly to review requests for change and decide what to do, based on cost/benefit estimates of the changes.

From: Soren Lauesen: Software Requirements

Functional Requirements – Summary

- **Context Diagram**
  - Diagram of product and its surrounding
  - Defining product scope
  - Very useful!
- **Event- and function lists**
  - Lists of events and functions
    - Domain or product level
  - Good as checklists at verification
  - Validation at product level?
- **Feature requirements**
  - Textual requirement: ”the product shall …”
  - High expressive power
  - Acceptable to most stakheolders
  - Can lead to false sense of security
    - How to ensure that goal-level covered?
- **Task descriptions**
  - Structured text describing user tasks
  - Easy to understand and verify
  - Good at domain level
- **(Vivid) Scenarios**
  - Rich descriptions of specific cases
  - Improves developer intuition and imagination
  - Products of elicitation but not “real” requirements
- **High-level tasks**
  - Client view of goal-related tasks
  - Independent of existing domain-level tasks
  - Good for business process re-engineering
- **Use Cases**
  - Widely used in many styles and variants
  - Some styles are good for design level (UI)
  - Can be used at different levels
  - Risk of pre-mature desin
- **Standards as requirements**
  - Textual requirement:
    “the product shall follow standard xxx”
  - Transfer the problem to the supplier
  - Sometimes lead to false sense of security
- **Development process requirements**
  - A requirement to follow a certain procedure
    - Use prototypes
    - Use specific reviews at certain points
    - Test in a specific way
    - Max number of simultaneous change reports
    - …etc
  - Validation? Difficult to say how process quality relates to product quality

Functional details Lau:4

- Skim read so that you know what is in there and see if anything is relevant for your project
- If you have studied UML you already know some of it, BUT it is very important to consider at which level to use the diagrams (domain, product, or design)…

Functional details & Special interfaces

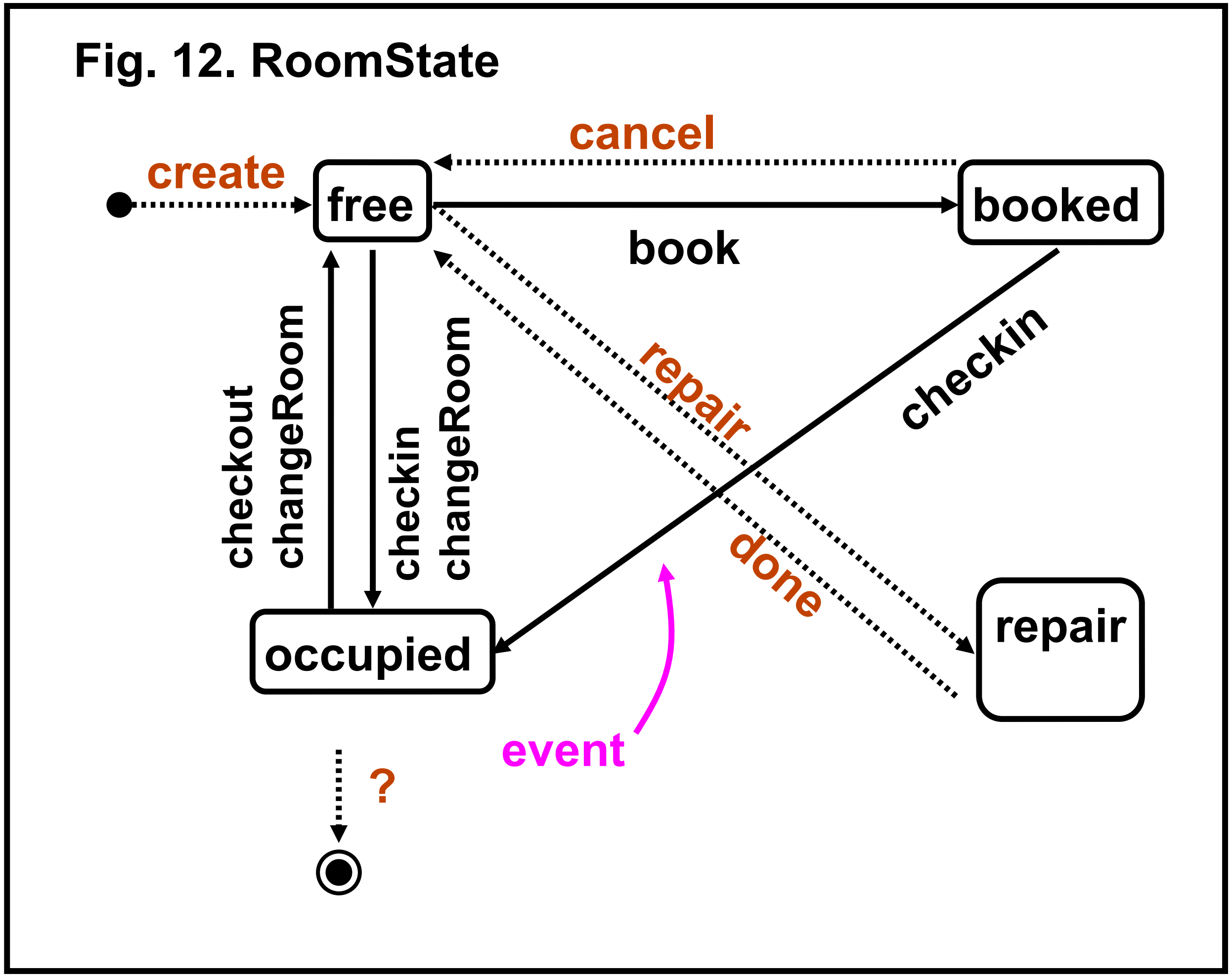
- Complex & simple functions
- Tables & decision tables
- Textual process descr.
- **State diagrams**
- State-transition matrices
- Activity diagrams
- **Class diagrams**
- Collaboration diagrams
- **Sequence diagrams**

- Reports
- **Platform requirements**
- Product integration
- **Technical interfaces**

Fig 4.4 State diagrams

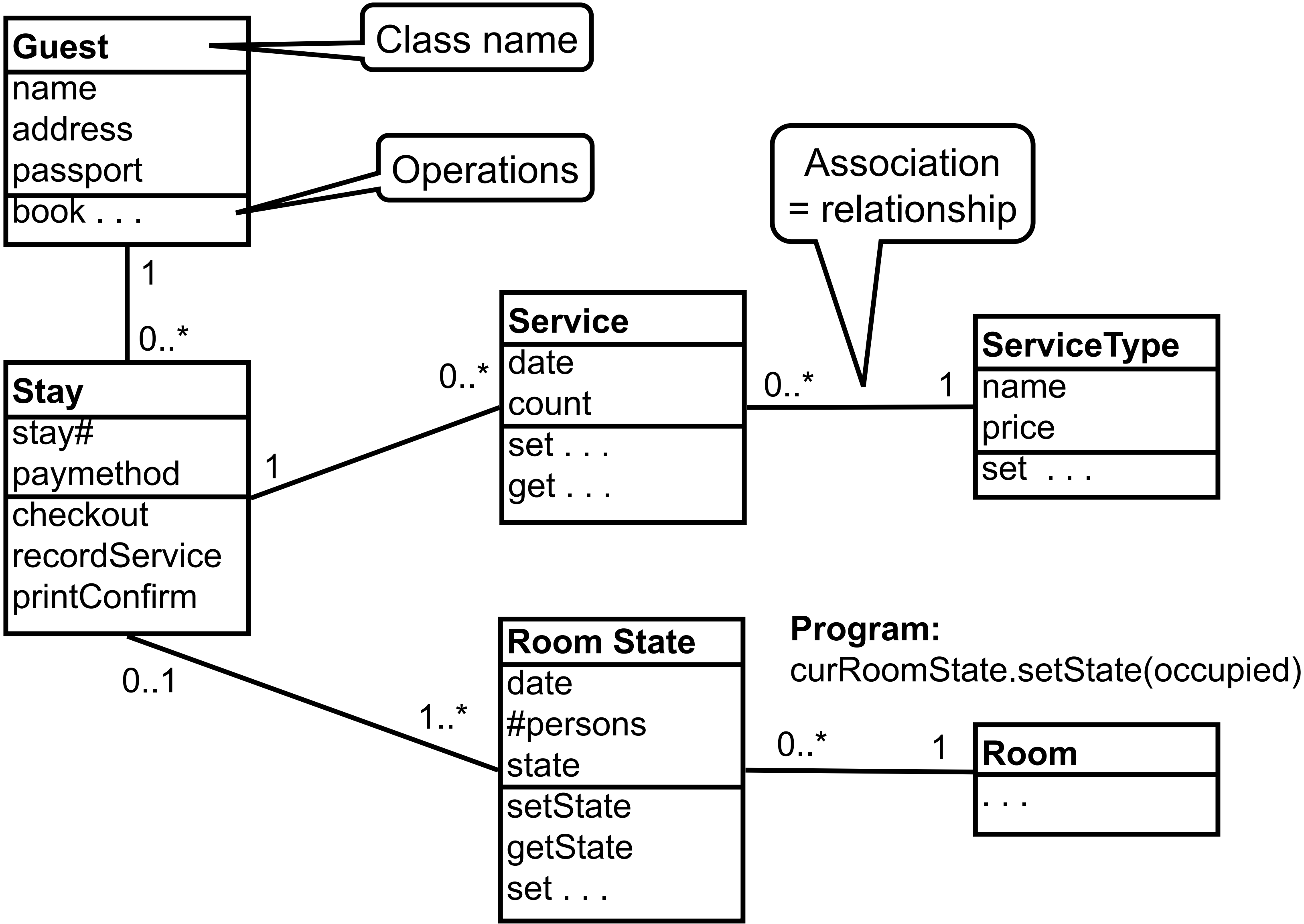
Rooms have a RoomState for each day in the planning period. The status shows whether the room is free, occupied, etc. that day.

R12: RoomState shall change as shown in Fig. 12.



From: Soren Lauesen:
Software Requirements

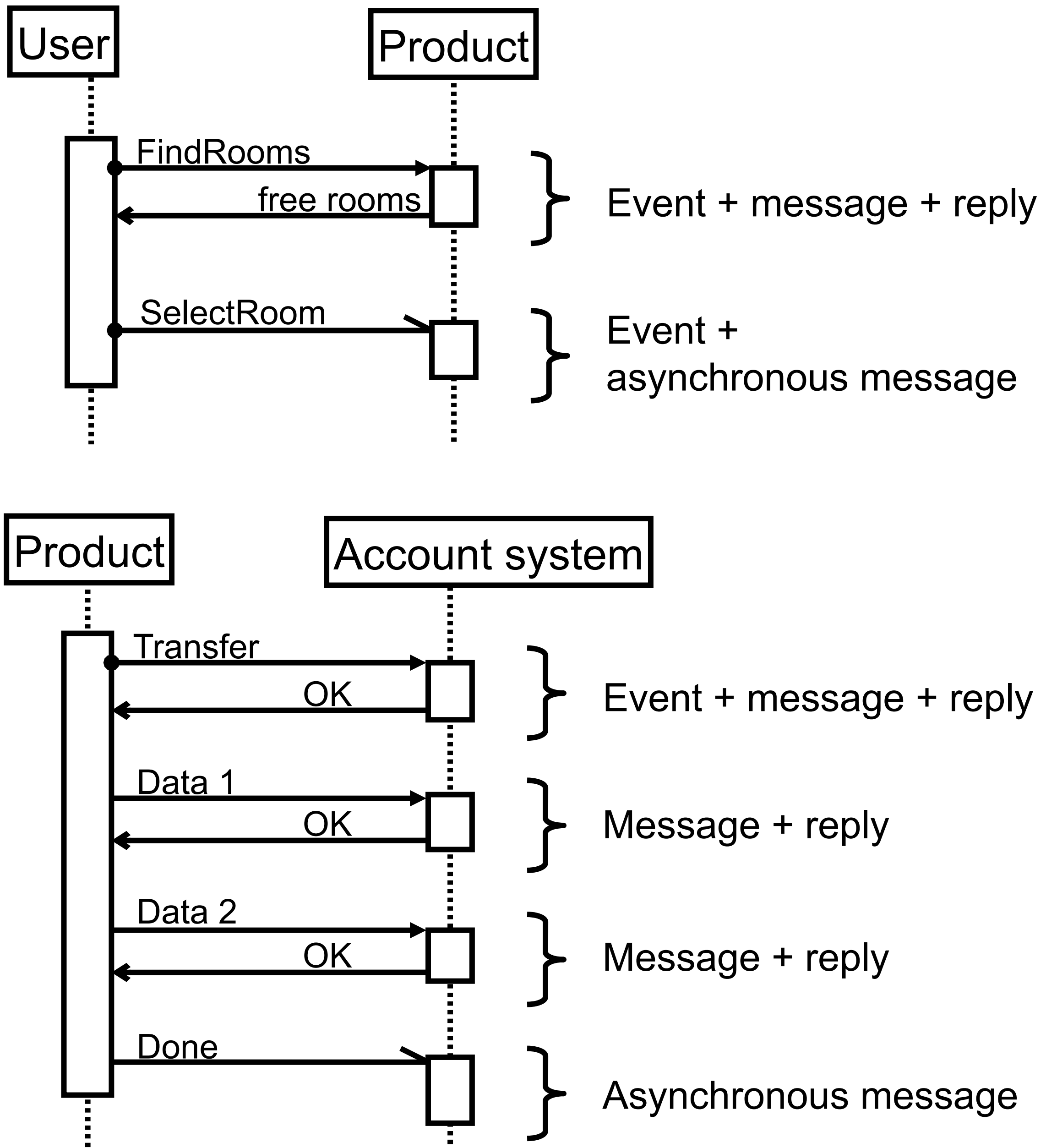
Fig 4.7A UML Class Diagram



Program:
curRoomState.setState(occupied)

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

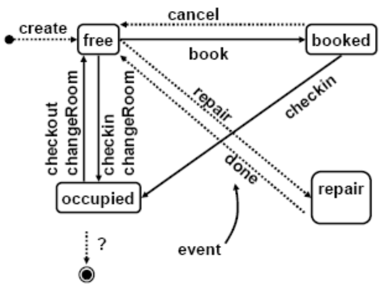
Fig 4.9 Sequence diagram



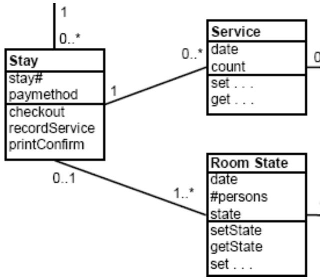
From: Soren Lauesen:
Software Requirements
© Pearson / Addison-Wesley 2002

Functional details Summary

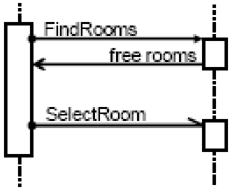
- **State diagrams**
  - Diagram showing how something changes from one state to another
  - Good for finding missing functions
  - Both on domain and product level
  - Can sometimes be very complex and difficult to read



- **Class diagrams**
  - A data model with operations on data
  - Harder to understand than E/R-diagrams
  - Widely used even when not good
  - Not good for higher levels



- **Sequence diagram**
  - Time diagram for how objects communicate
  - Good for describing (simple) communication protocols
  - Useful at design-level



- Activity Diagram…
- Collaboration diagrams …

reqT

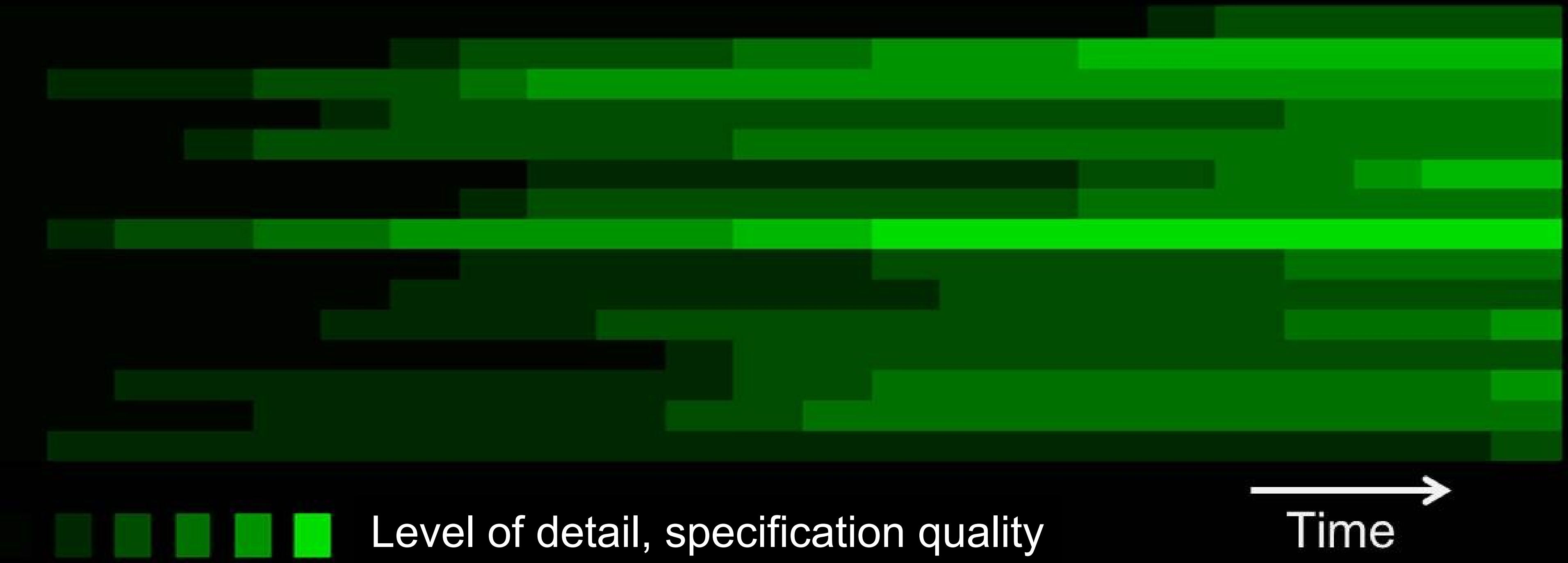
reqT tutorial and lab prep

Getting started with reqT:

- Check out https://reqT.github.io
- Download the Desktop app in reqT.jar
- Run with `java -jar reqT.jar`
- Investigate the GUI, check out the Templates and Tools
- Start the repl with `java -jar reqT.jar repl`
  - Try some simple Models to see test how it works

```
m"* Feature: hi has Spec: hello"

Model(Feature("hi").has(Spec("hello")))
```

Evolving mix of levels of detail & quality in continuous requirements engineering



The reqT tool supports evolving requirements based on a modeling language with essential RE concepts suitable for teaching & learning.

The idea behind reqT

- Be methodology agnostic: 'bag of concepts'
- Graphical UI and Terminal UI for power users
- Turn requirements into code by a scalable data structure, from 1 to 10E4 reqts
- Scriptable to the power of Scala and the JDK
- Integrate with git and similar code/text tools
- Solve requirements constraints problems
- Open source: https://github.com/reqT

Research papers on reqT:
https://reqt.org/documentation.html#pub

https://reqT.github.io

- **New in reqT v4**:
  Simple syntax based on **markdown bullet lists**

- Especially developed for **this course**
- **Essential** Requirements Engineering concepts from literature
- Generates **visualizations** using Graphviz
- Generates **documents** via export to html, latex, pdf

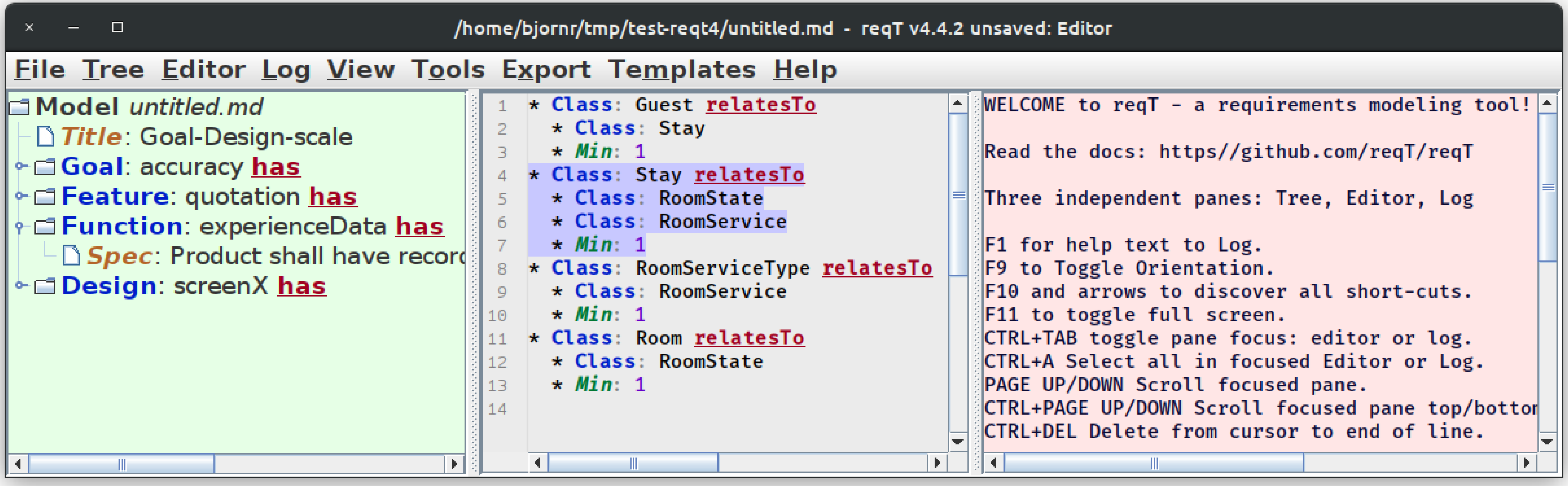
- Discuss in your project if/how you want to use reqT

Screenshot of reqT

Tree pane

Editor pane

Log pane


× – □
/home/bjornr/tmp/test-reqt4/untitled.md - reqT v4.4.2 unsaved: Editor
File Tree Editor Log View Tools Export Templates Help

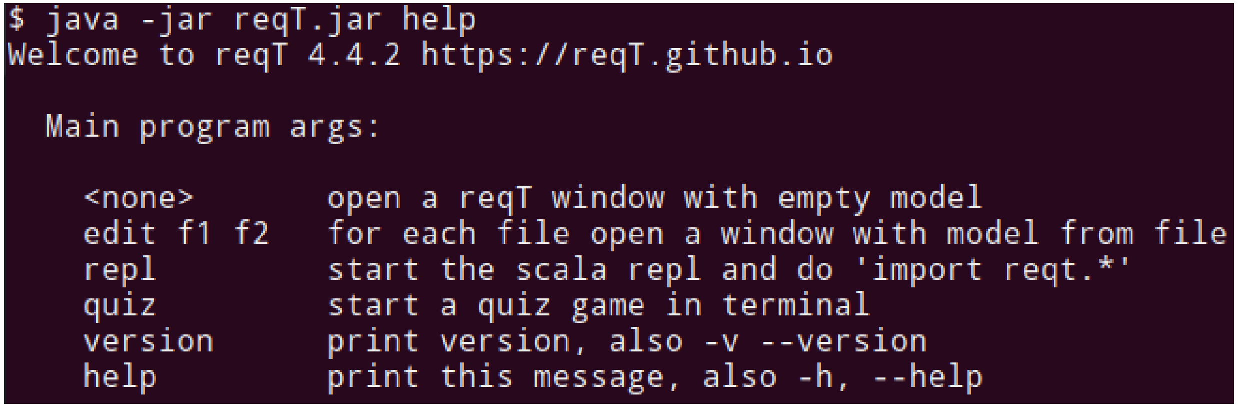
Model *untitled.md*
- *Title*: Goal-Design-scale
- **Goal**: accuracy **has**
- **Feature**: quotation **has**
- **Function**: experienceData **has**
  - *Spec*: Product shall have recor
- **Design**: screenX **has**

```
* Class: Guest relatesTo
  * Class: Stay
  * Min: 1
* Class: Stay relatesTo
  * Class: RoomState
  * Class: RoomService
  * Min: 1
* Class: RoomServiceType relatesTo
  * Class: RoomService
  * Min: 1
* Class: Room relatesTo
  * Class: RoomState
  * Min: 1
```

```
WELCOME to reqT - a requirements modeling tool!

Read the docs: https//github.com/reqT/reqT

Three independent panes: Tree, Editor, Log

F1 for help text to Log.
F9 to Toggle Orientation.
F10 and arrows to discover all short-cuts.
F11 to toggle full screen.
CTRL+TAB toggle pane focus: editor or log.
CTRL+A Select all in focused Editor or Log.
PAGE UP/DOWN Scroll focused pane.
CTRL+PAGE UP/DOWN Scroll focused pane top/bottor
CTRL+DEL Delete from cursor to end of line.
```

See terminal help:

`java -jar reqT.jar help`



```
$ java -jar reqT.jar help
Welcome to reqT 4.4.2 https://reqT.github.io

 Main program args:

   <none>       open a reqT window with empty model
   edit f1 f2   for each file open a window with model from file
   repl         start the scala repl and do 'import reqt.*'
   quiz         start a quiz game in terminal
   version      print version, also -v --version
   help         print this message, also -h, --help
```

Requirements Entities

Examples from the reqT metamodel

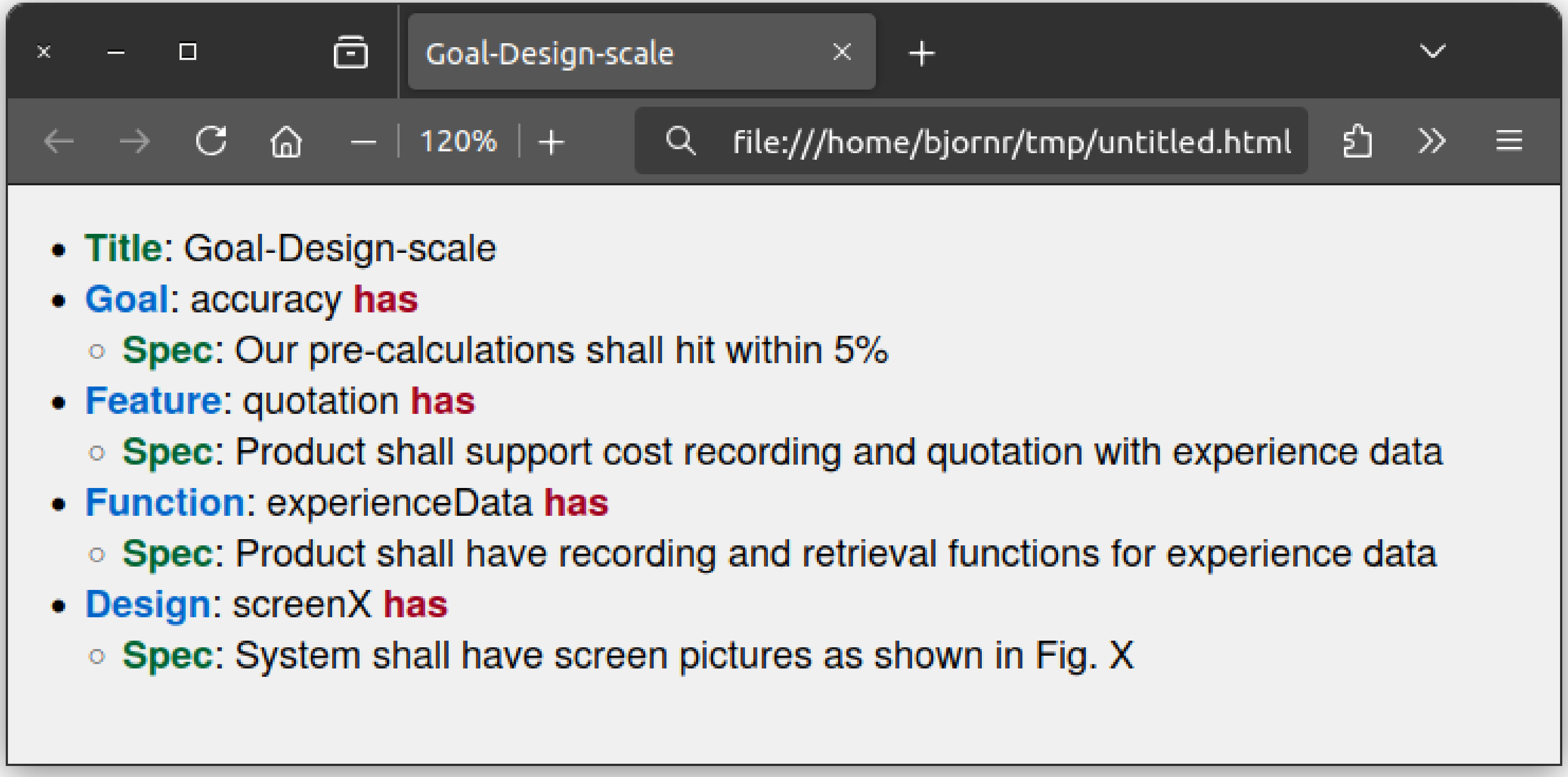
```
Product, Interface,
Stakeholder, Idea, Goal,
Feature, Data, Function,
State, Event,  Quality,
Design, Scenario, Story,
UseCase, Risk, Release,
Issue, Test, Variant, Req
```

The goal-design scale in reqT

```
* Goal: accuracy has
  * Spec: Our pre-calculations shall hit within 5%

* Feature: quotation has
  * Spec: Product shall support cost recording
    and quotation with experience data

* Function: experienceData has
  * Spec: Product shall have recording and
    retrieval functions for experience data

* Design: screenX has
  * Spec: System shall have screen pictures
    as shown in Fig. X
```

Shipyard example [Lau]

```
* Product: reqT has
  * Feature: toHtml
```



- **Title**: Goal-Design-scale
- **Goal**: accuracy **has**
  - **Spec**: Our pre-calculations shall hit within 5%
- **Feature**: quotation **has**
  - **Spec**: Product shall support cost recording and quotation with experience data
- **Function**: experienceData **has**
  - **Spec**: Product shall have recording and retrieval functions for experience data
- **Design**: screenX **has**
  - **Spec**: System shall have screen pictures as shown in Fig. X

The reqT metamodel

A **Model** is a sequence of elements.

An **element** can be a node or a relation.

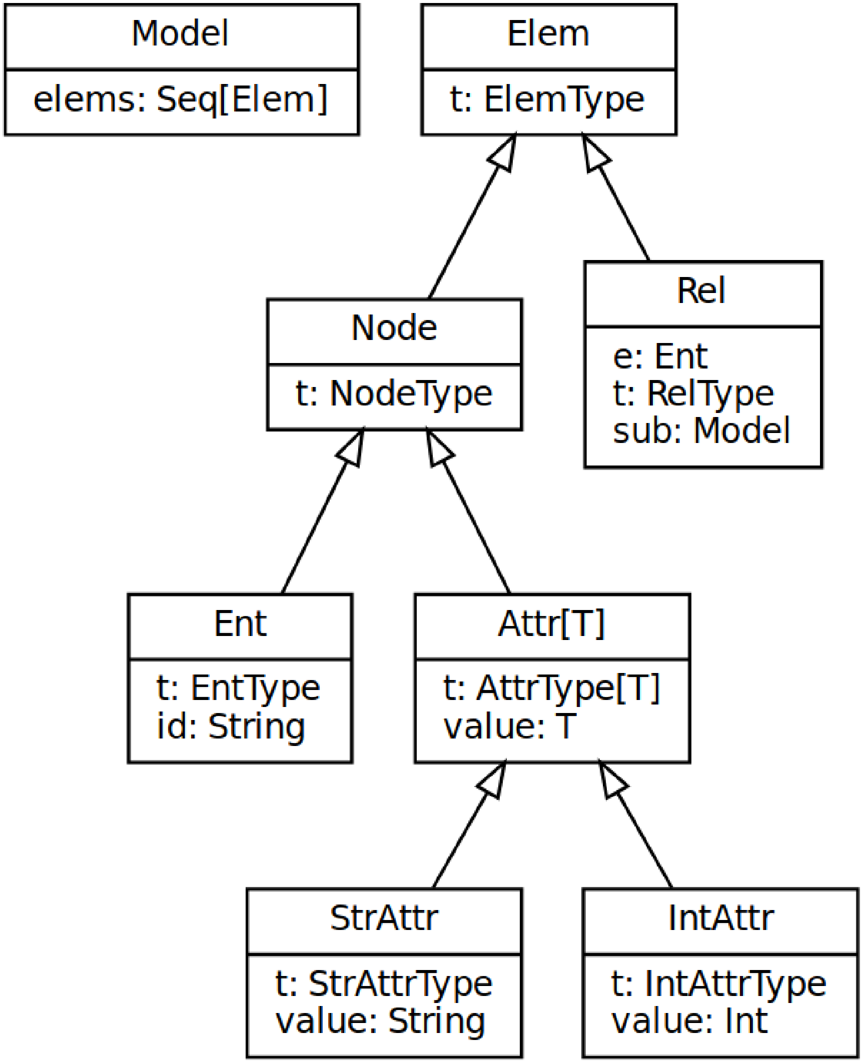
A **node** can be an entity or an attribute.

An **entity** has a type and an id.

An **attribute** has a type and a value.

An attribute can be
a **string** attribute or
an **integer** attribute.

A **relation** connects an entity
to a sub-model via a relation type.



Three views of a reqT Model

- **Markdown bullets**

```
* Feature hi: has
  * Spec: hello
```

- **Scala constructors**

```
Model(
  Feature("hi").has(
    Spec("hello")))
```

- **Underlying Scala classes of the reqT metamodel**

```
Model(
  Rel(Ent(Feature,"hi"),Has,
    Model(StrAttr(Spec,"hello"))))
```

The reqT **Tools** menu

- The tools operate on the Editor pane. Example workflow scenario:
  1. load the things you want to apply a tool to in the Editor pane
  2. select a tool from the menu
  3. the Editor pane is updated
  4. transfer what you want to keep to the Tree pane and save the Tree
- What the tools do:
  - **Format model**: standardized reqT markdown pretty-print
  - **Distinct model**: remove duplicates on all levels
  - **Keep distinct entities**: filter Ent instances, remove duplicates
  - **Entity ordering**: append Order relations, in order of apperance
  - **$100 normalized votes**: weighted total priorities and benefits, assumes that your model has same shape as *Template -> Prioritization: 100$ test*
  - **Id pairs as comparison constraints**: append all pairs of ids in Constraints
  - **Solve Comparison Constraint Problem**: try to satisfy Constraints by searching for a solution that fulfills all comparisons; relaxed by allowing deviations if needed

Automate model merging, analysis, doc building, ...

Download **hello-reqt.scala** from
https://github.com/reqT/reqT.github.io/blob/master/src/hello-reqt.scala
and run with: **scala run . -M hello**

```
//> using scala 3.6.3
//> using dep "reqt:reqt:4.4.2,url=https://github.com/reqT/reqT/releases/download/v4.4.2/reqT-4.4
```

```
import reqt.*

val m = Model(
    Feature("helloWorld").has(
      Spec("Print a nice greeting."),
      Why("First step to get started."),
      )
    )

@main def hello = println(m.toMarkdown)
```

Some things you can do with reqt.Model in Scala

try below in Scala repl: `java -jar reqT.jar repl`

```
var m1 = Model()                                       // an empty Model
val id = "hello"                                       // an immutable reference to a String

val m2 = m"* Feature: $id has Spec: print greeting"    // reqT special String interpolator m
                                                       // parses String and returns a Model
m1 = m"""                                              // multi-line String after """
         * Feature $id has
           * Spec: print greeting"""
m1.show                                                // pretty-print model
m1.toMarkdown                                          // generate Markdown from a Model
m1.toHtml                                              // generate Html from a Model, also
toHtmlBody

Sys.loadLines("mymodel.md")                            // parse markdown model from file
Sys.saveTo(m1.toMarkdown)("mymodel-2.md")                // save markdown to file

m1 = m1.append(Model(Feature("y").has(Spec("more stuff"))))   // append two models also :++
m1 = m1.distinctElemsDeep                                     // remove duplicates

m1.elems   // a sequence of elems, type is immutable Vector[Elem]

for e <- m1.elems yield                                // iterate over all elems e in m1
  e match                                              // match on e of type Elem
    case Ent(id, t) => id                              // pattern-match on Ent, yield the id
    case StrAttr(t, value) => value
    case Rel(e, t, sub) => e.id
    case _ => "or else this string"
```

Some questions for you

- How will you **partition** your reqts space?
- How will you **synchronize** your work?
- What entity **id** policy will you have?
- How will you manage **versions**?
- How will you **build** your document from requirements fragments?

Which tools are you going to use?

- Office apps e.g. Google Docs, LibreOffice, MS Office (spreadsheet, wordprocessor, database)
- Latex
- Web publishing
- Configuration management (git, GitHub, GitLab, Bitbucket...)
- Prototyping tools, gui-builders
- Issue trackers / ticket managers / Trello etc.
- `reqT`

- **Who will be tool responsible?**

Kommersiella verktyg för kravhantering

https://en.wikipedia.org/wiki/Requirements_engineering_tools

Några verktygsexempel:

- Siemens Polarion
- Atlassian Jira
- Github (using issues+labels)
- IBM Doors/Doors Next/Jazz
- IBM Focal Point

  https://public.dhe.ibm.com/software/pdf/fi/IBM-Rational-Focal-Point.pdf

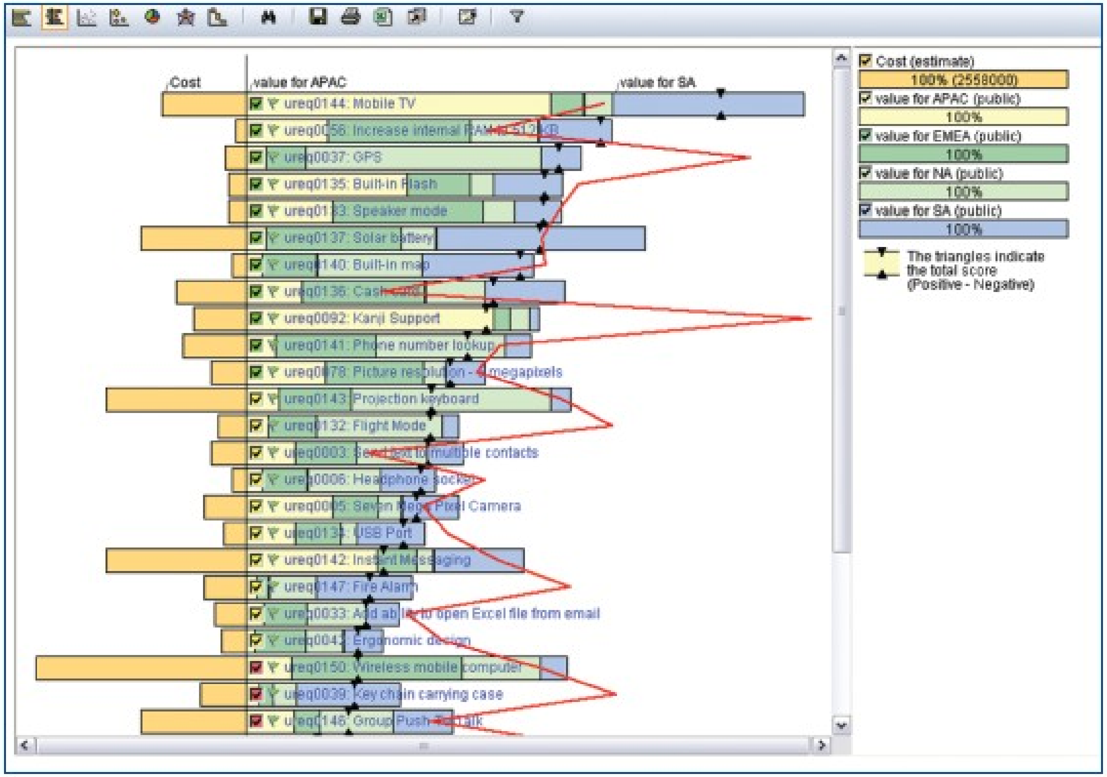

Cost
value for APAC
value for SA
ureq0144: Mobile TV
ureq0056: Increase internal RAM to 512KB
ureq0037: GPS
ureq0135: Built-in Flash
ureq0133: Speaker mode
ureq0137: Solar battery
ureq0140: Built-in map
ureq0136: Cash card
ureq0092: Kanji Support
ureq0141: Phone number lookup
ureq0178: Picture resolution - 4 megapixels
ureq0143: Projection keyboard
ureq0132: Flight Mode
ureq0003: Send text to multiple contacts
ureq0006: Headphone socket
ureq0005: Seven Mega Pixel Camera
ureq0134: USB Port
ureq0142: Instant Messaging
ureq0147: Fire Alarm
ureq0033: Add ability to open Excel file from email
ureq0042: Ergonomic design
ureq0150: Wireless mobile computer
ureq0039: Key chain carrying case
ureq0148: Group Push to talk
☑ Cost (estimate)
100% (2558000)
☑ value for APAC (public)
100%
☑ value for EMEA (public)
100%
☑ value for NA (public)
100%
☑ value for SA (public)
100%
The triangles indicate the total score (Positive - Negative)

Figure 2: Rational Focal Point software enables you to visualize value and trade-offs from various stakeholders—viewing where stakeholders agree and disagree.

To do...

- **Read** these chapters in the textbook by Lauesen: **2, 3,** 4, 5
  First read all gray boxes, then prioritize things from lectures
- **Exercise E2**: Elicitation, really important for your project work, based on Lauesen Chaper 8 (see previous lecture L2)
- **Lab 1**: Context, Features and Priorities reqT.github.io
  - Work in pairs (or individually), book your slot in Canvas
  - Get `java -jar reqT.jar` running in terminal
  - Complete all preparations *before* the lab: https://cs.lth.se/krav/labs/
  - Read paper on prioritization [PRIO] (see previous lecture L2)
- Work in the **project**:
  - Book meeting with supervisor to discuss Project Mission v2
  - Project Mission v2 handed in via Canvas, deadline: see https://cs.lth.se/krav/proj/
- **Lectures next week:**
  - L5: Tuesday 15-17: Prototyping, Agile RE
  - L6: Wednesday 15-17: Open Source RE, Release planning + help to prepare lab 2