Visit **gosocrative.com** and enter room name REQENG

Lecture starts at 13:15

- Please fill out Q1 and Q2 in Socrative

You are welcome to share a short break with us and discuss/ask questions

Housekeeping
What is requirements interpretation
Why is it challenging
Functional requirements
Wrapping up

L4 – Interpretation

DAT232/DIT285 Advanced Requirements Engineering

Eric Knauss
eric.knauss@cse.gu.se

September 12, 2025

Outline

1. Housekeeping
2. What is requirements interpretation
3. Why is it challenging
4. Functional requirements
5. Wrapping up

Housekeeping

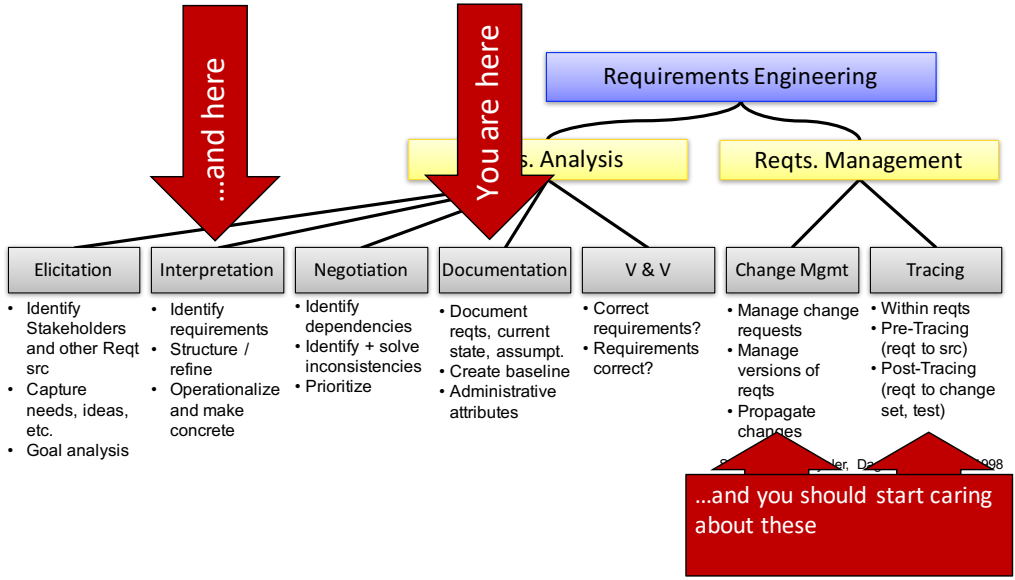
Discussion points

- Student representatives: Meet me after lecture (will take 15min max)

9. Please comment on one thing that you liked and one thing that you wished for in today's lecture.

Hide Answers | Show Names | 13/73 Students Answered

i liked the group break out discussion, i wish there will also be some in the future
no
–
if you can provide more practical example that is better
Regarding the lecture slides, Specifically, when addressing key concepts, it would be maybe beneficial to provide direct explanations of the concept, rather than referencing the perspectives of other individuals or experts
I liked that we had to practise on different kind of requirements.
professor's way of conveying lecture is very good and easy to understand
Wish there are more examples help understanding different types of requirements.
Liked - Socrates. Wished for - More elaborate explanations of differences between performance requirements and specific quality requirements. This was a bit fluffy in my opinion
Everything is great!
I liked the creative explanations of constraint (cooking an omelette), explaining otherwise very text-heavy/category-heavy theory in this way makes learning about requirements fun and memorable.
-
The development team shall use an agile methodology (Scrum) with bi-weekly sprints and sprint reviews.


...and here
You are here
Requirements Engineering
s. Analysis
Reqts. Management
Elicitation
Interpretation
Negotiation
Documentation
V & V
Change Mgmt
Tracing
• Identify Stakeholders and other Reqt src
• Capture needs, ideas, etc.
• Goal analysis
• Identify requirements
• Structure / refine
• Operationalize and make concrete
• Identify dependencies
• Identify + solve inconsistencies
• Prioritize
• Document reqts, current state, assumpt.
• Create baseline
• Administrative attributes
• Correct requirements?
• Requirements correct?
• Manage change requests
• Manage versions of reqts
• Propagate changes
• Within reqts
• Pre-Tracing (reqt to src)
• Post-Tracing (reqt to change set, test)
...and you should start caring about these

Learning Objectives

Knowledge	Skills	Judgement
K1 Identify a common RE challenge in a given software development context.	S1 Plan suitable RE practices in a team with respect to a given software development context.	J1 Assess new requirements engineering knowledge (challenge, principle, practice) and relate them to the framework in this course.
K2 Choose an appropriate RE practice in a given software development context.	S2 Effectively apply a suitable RE practice in a team in a given software development context.	J2 Suggest suitable actions to overcome a lack of requirements knowledge in a software development context.
K3 Compare suitability as well as advantages and disadvantages of given RE practices in a given software development context.	S3 Analyze the effect and quality of the outcome of a set of or individual RE practices in a given software development context.	J3 Consider inter-team, program level and social/ethical implications of a set of RE practices in a given software development context.
K4 Explain the current state of practice and research in requirements engineering.		J4 Critically assess the effectivity of a set of RE practices from the perspective of the student's master program (e.g. Software Engineering & Technology/Management, Interaction Design, Game Design, Data Science, ...)

Learning Objectives

Interpretation for ...Interpretation:

K1,K2,K3	Explain common challenges of interpretation and documentation, describe practices of interpretation and documentation of reqts
K4	Be aware of current research challenges in interpretation
S1-S3	Apply this knowledge to your project
J1-J3	Look out for opportunities to get beyond course literature
J4	Reflect on how your background changes your view on documentation (different documentation for games, data-intense projects, with technology/management focus?)

Outline

1 Housekeeping

2 What is requirements interpretation

3 Why is it challenging

4 Functional requirements

5 Wrapping up

What is requirements interpretation

Activity	*Explanation*
Identify requirements	Elicitation provides us with candidate requirements. Those things mentioned by our stakeholders: Are those really needed to solve the problem?
Structure and refine	Go through the candidate requirements and analyse them.
Operationalize and make concrete	It is easy to agree with vague and abstract requirements. The system should be usable. Sure. But how usable and at what cost?

Quality criteria

What is requirements interpretation

A good requirements specification is [Lauesen, 2002, Fig.9.1]:

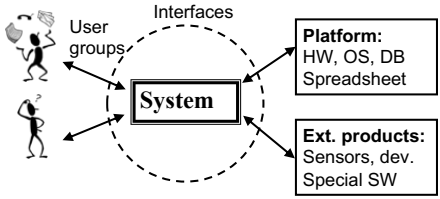
Correct	Each requirement reflects a need.
Complete	All necessary requirements included.
Unambiguous	All parties agree on meaning.
Consistent	All parts match, e.g. E/R and event list.
Ranked for importance and stability	Priority and expected changes per requirement.
Modifiable	Easy to change, maintaining consistency.
Verifiable	Possible to see whether requirement is met.
Traceable	To goals/purposes, to design/code.
Additional:	
Traceable	from goals to requirements.
Understandable	by customer and developer.

How to fulfil those quality criteria?

Common approach: **Structured, top-down**

For everything mentioned by stakeholders:

1. Is it a requirement? No: Add as information, if helpful. Yes: Continue
2. What type of requirement? Functional? Data? Quality? Other? Write at correct place.
3. Check, whether each part has been filled out and whether information is consistent.
4. Add tracelinks to indicate dependencies.



Data requirements:

System state: Database, comm. states
Input/output formats

Functional requirements, each interface:

Record, compute, transform, transmit
Theory: F(input, state) -> (output, state)
Function list, pseudocode, activity diagram
Screen prototype, support tasks xx to yy

From: Soren Lauesen: Software Requirements

Quality reqs:

Performance
Usability
Maintainability
. . .

Other deliverables:

Documentation
Install, convert, train . . .

Managerial reqs:

Delivery time
Legal
Development process . . .

Helping the reader:

Business goals
Definitions
Diagrams . . .

How to fulfil those quality criteria?

Common approach: **Feature based**

Each requirement is one statement about a feature of the system.

Example

$F_{x.y}$ The system shall support check-in of a guest into a pre-booked room.

Advantages	*Potential problems / risks*
Easy to understand	Tendency to mix abstraction levels
Easy to manage (change, lifecycle)	Hard to assess completeness
Support traceability	Can be ambiguous or missing important information
Good way of listing system requirements	For user requirements, user stories or task descriptions may be more appropriate

$\rightarrow$ In a mixed approach, use as principle list, refer to other artifacts

How to fulfil those quality criteria?

Uncommon approach: **Model based**

Textual or graphical (e.g. UML) models with defined semantics.

Example

- Use case diagram to list user goals,
- Activity diagram or Sequence diagram for scenarios,
- Class or ER diagram for data,
- ...

Advantages	*Potential problems / risks*
Avoid ambiguities	Hard to understand for those who did not model
Easy to verify, if formal enough	Hard to assess completeness
Support testing	Hard to manage changes
	Hard to scale

$\rightarrow$ In a mixed approach, use models to provide details for specific reqts

Common approach: **Scenario based**

Describing a potential way of using (a) feature(s) of the system under construction.

Example

- Use cases (table with ID, title, Stakeholders, pre-/post-condition, main- and alternative scenarios, ...),
- Task description
- Specification by example

How to fulfil those quality criteria?

Advantages	*Potential problems / risks*
Easy to understand	Tendency to overspecify (too much detail)
Relate requirements at different abstraction levels	Tendency to underspecify (not enough detail)
Provide structure	Inconsistent form (level of abstraction, terminology, structure)
Provide sense of completeness (locally, within scenario)	Inconsistent content (contradictions, global incompleteness)
Support traceability Good foundation for testing	Hard to manage changes

→ In a mixed approach, use to break down user goals into system features

Principle approaches to requirements interpretation and documentation

Combine for best results. How? Depends on context.

- Structured approach
- Feature based approach
- Model based approach
- Scenario based approach

→ `socrative.com`, Room REQENG, Q3 and Q4

By the way:

On a good day, user stories combine most advantages of feature and scenario based approaches. On a bad day, they only combine their disadvantages.

Housekeeping

What is requirements interpretation

Why is it challenging

Functional requirements

Wrapping up

Outline

1 Housekeeping

2 What is requirements interpretation

3 Why is it challenging

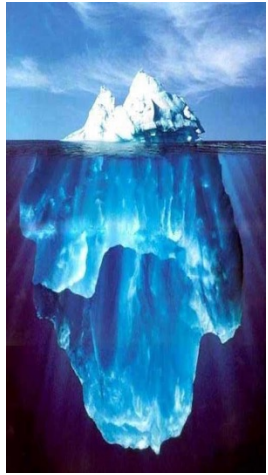
4 Functional requirements

5 Wrapping up

Complete requirements?

Why is it challenging?

- 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
  - Breadth first, but also...
  - "jump" between abstraction levels



How does context matter?

Why is it challenging?

Small, agile project:

- User stories, face-to-face communication to make up for lack of detail

Continuous / market-driven SE:

- Architecture provides structure
- For each new feature:
  - Understand business value and change impact (benefit/cost)
  - Attach user stories
  - Update system requirements

Large-scale:

- More levels and decomposition? Let's discuss in agile RE lecture.

10 min Break

Outline

1 Housekeeping

2 What is requirements interpretation

3 Why is it challenging

4 Functional requirements

5 Wrapping up

Functional requirements

Data requirements

- Data model(=E/R-diagr.)
- Data dictionary
- Data expressions
- Virtual windows

Special interfaces:

- Reports
- Platform requirements
- Product integration
- Technical interfaces

Functional requirements:

- Context diagram
- Event- and function lists
- Feature requirements
- Screens and prototypes
- Task descriptions
- Task & support
- (Vivid) Scenarios
- High-level tasks
- Use Cases
- Data flow diagrams
- Standards as requirements
- Development process requirements

Functional details:

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

When to use which technique?

Functional requirements

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 verification and validation!

Data requirements

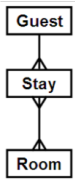
While not strictly functional, both [Lauesen, 2002] and [Glinz, 2007] classify them as such.

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

Example

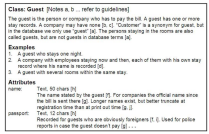
Example data from the mobile domain:
Subscriber data, roaming data, phone book data, image data (when, resolution, name, category), music data (album, artist, genre, name, frequency played, rating), etc.

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 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 [b]
The name stated by the guest [f]. For companies the official name since the bill is sent there [g]. Certain names exist, but better truncate at registration time than at print out time [g, j].

passport: Text, 12 chars [k]
Recorded for guests who are obviously foreigners [f, i]. Used for police reports in case the guest doesn't pay [g] . . .

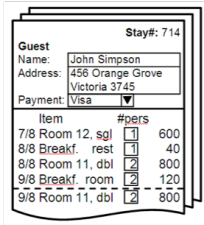
Data dictionary

- 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!!)

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

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



Stay#: 714

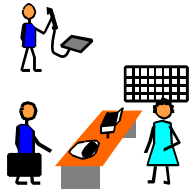
Guest
Name: John Simpson
Address: 456 Orange Grove
Victoria 3745
Payment: Visa

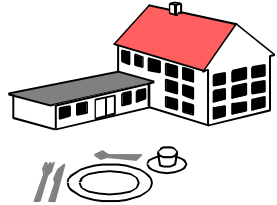
Item	#pers	
7/8 Room 12, sgl	1	600
8/8 Breakf. rest	1	40
8/8 Room 11, dbl	2	800
9/8 Breakf. room	2	120
9/8 Room 11, dbl	2	800

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

Fig 2.1 The hotel system



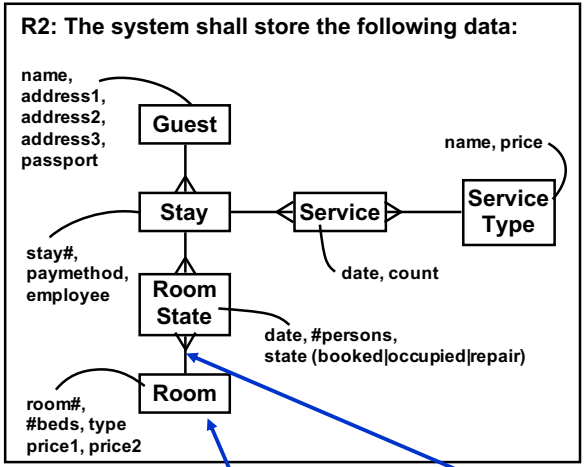


Task list
Book guest
Checkin
Checkout
Change room
Breakfast list & other services

Data about
Guests
Rooms
Services

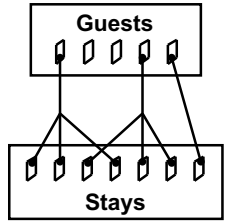
From: Soren Lauesen: Software Requirements

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



One-to-many (1:m)

Each guest connected to zero or more stays



Each stay connected to one guest record

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

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 2.4A 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 }

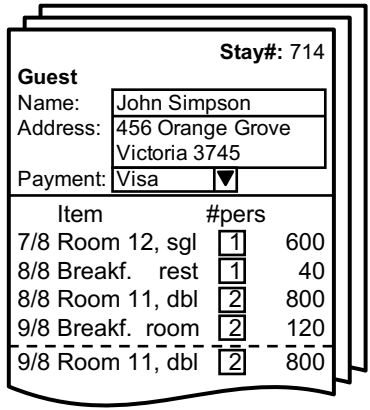
account data = transfer + {account record}* + done

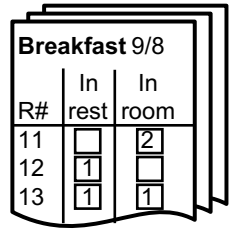
From: Soren Lauesen: Software Requirements

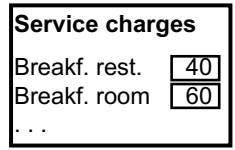
Fig 2.5 Virtual Windows

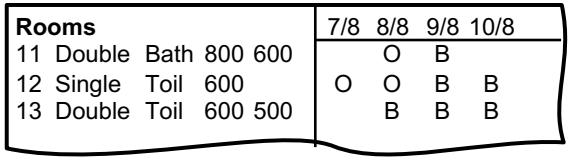
R1: The product shall store data corresponding to the following virtual windows:

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









From: Soren Lauesen:
Software Requirements

Data requirements

→ `socrative.com`, REQENG, Q5

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

Functional requirements

Data requirements

- Data model(=E/R-diagr.)
- Data dictionary
- Data expressions
- Virtual windows

Special interfaces:

- Reports
- Platform requirements
- Product integration
- Technical interfaces

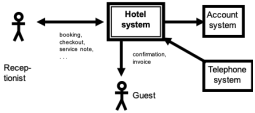
Functional requirements:

- Context diagram
- Event- and function lists
- Feature requirements
- Screens and prototypes
- Task descriptions
- Task & support
- (Vivid) Scenarios
- High-level tasks
- Use Cases
- Data flow diagrams
- Standards as requirements
- Development process requirements

Functional details:

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

Funct. requirements techniques – Summary



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

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

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 ....

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

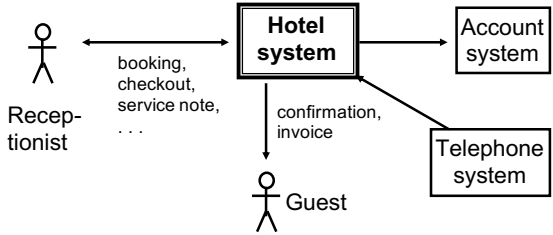
- Textual requirement: “the product shall ...”
- High expressive power
- Acceptable to most stakeholders
- Can lead to false sense of security
- How to ensure that goal-level covered?

Screens and Prototypes

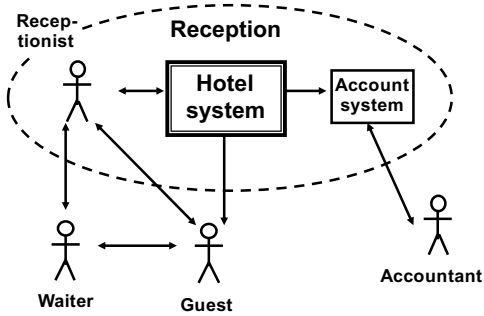
- Screen pictures & what buttons do
- Excellent as design-level requirements
- if carefully tested
- Not good when for COTS-based systems

Fig 3.2 Context diagram

R1:
The product shall have the following interfaces:

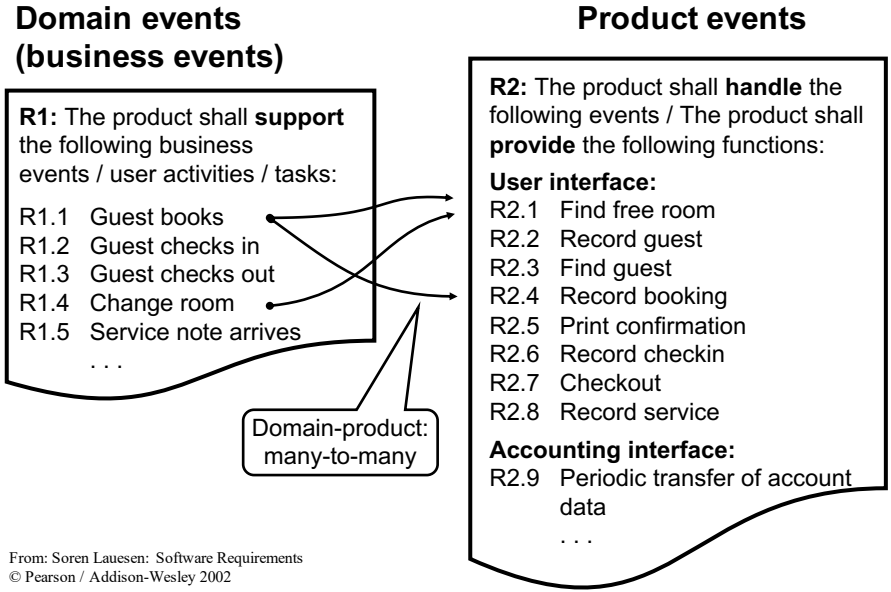


R2 ??:
The reception domain communicates with the surroundings in this way:



From: Soren Lauesen: Software Requirements

Fig 3.3 Event list & function list



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.

→ `socrative.com`, REQENG, Q6

Feature = product function + related data

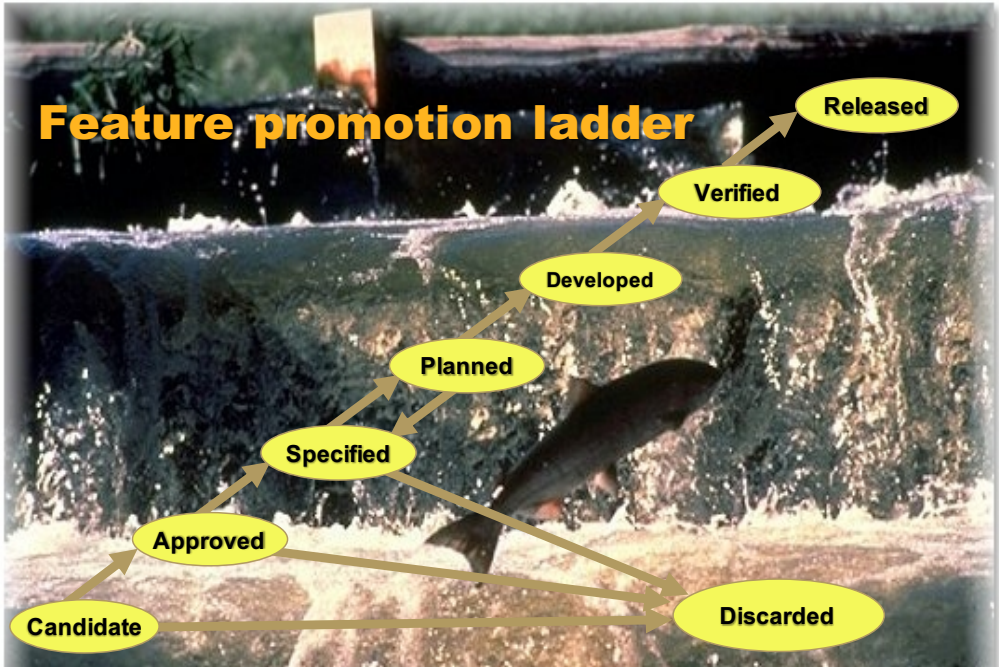
In order to handle group tours with several guests, it is convenient to prepare for arrival by printing out a sheet per guest for the guest to fill in.

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 ...

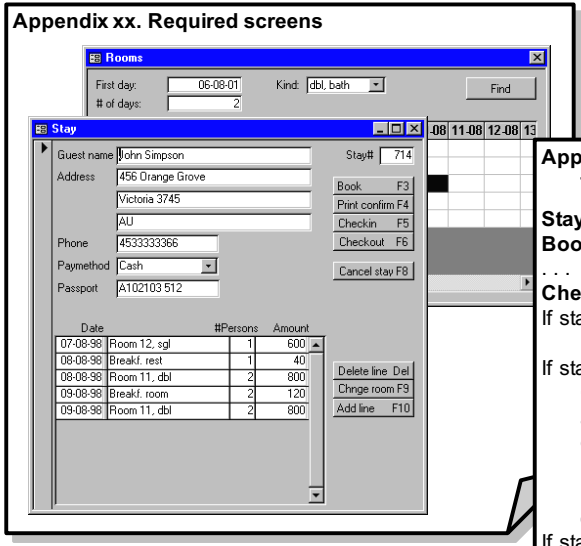

Feature promotion ladder
Released
Verified
Developed
Planned
Specified
Approved
Candidate
Discarded

Database-attributes of features

Example

Attribute	*Value*	*Assigned in State*
State	C / A / S / Di / P / De / V / R	-
ID	Unique identity	Candidate
Submitter	Who issued it?	Candidate
Company	Submitter's company	Candidate
Domain	Functional domain	Candidate
Label	Good descriptive name	Candidate
Description	Short textual description	Candidate
Contract	Link to sales contract enforcing requirement	Candidate
Priority	Importance category (1,2,3)	Approved
Motivation	Rationale: Why is it important?	Approved
Line of Business	Market segment for which requirement is important	Approved
Specification	Links to Use Case, Textual Specification	Specified
Decomposition	Parent-of / Child-of – links to other req's	Specified
Estimation	Effort estimation in hours	Specified
Schedule	Release for which it is planned for	Planned
Design	Links to design documents	Developed
Test	Links to test documents	Verified
Release version	Official release name	Released

Fig 3.5B Screens & prototypes



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

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?

→ `socrative.com`, REQENG, Q7

From: Soren Lauesen: Software Requirements

Functional requirements

Data requirements

- Data model(=E/R-diagr.)
- Data dictionary
- Data expressions
- Virtual windows

Special interfaces:

- Reports
- Platform requirements
- Product integration
- Technical interfaces

Functional requirements:

- Context diagram
- Event- and function lists
- Feature requirements
- Screens and prototypes
- Task descriptions
- Task & support
- (Vivid) Scenarios
- High-level tasks
- Use Cases
- Data flow diagrams
- Standards as requirements
- Development process requirements

Functional details:

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

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.
. . .

→ socrative.com, REQENG, Q8

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)

→ `socrative.com`, REQENG, Q9

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

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

→ socrative.com, REQENG, Q10

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

→ `socrative.com`, REQENG, Q11

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

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

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 details and special interfaces

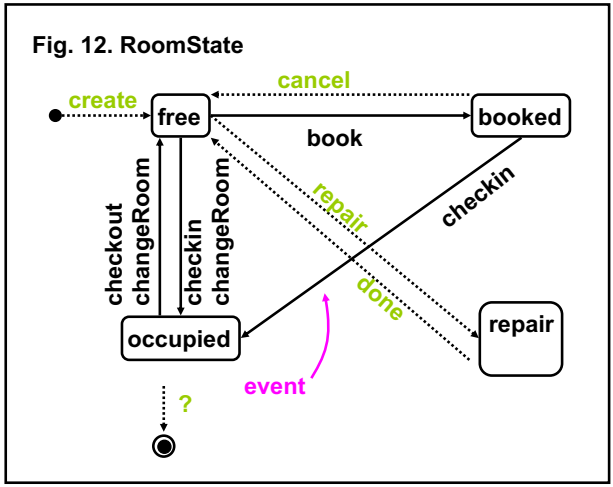
- Complex & simple functions
- Tables & decision tables
- Textual process descr.
- State diagrams
- State-transition matrices
- Activity diagrams
- Class diagrams
- Collaboration diagrams
- Sequence diagrams

- 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.

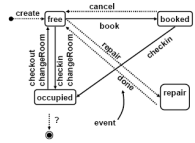
R12: RoomState shall change as shown in Fig. 12.



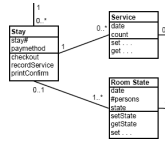
From: Soren Lauesen:
Software Requirements
© Pearson / Addison-Wesley 2002

Functional details Summary

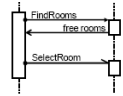
- **State diagrams**
  - Diagram showing how something changes from one state to another
  - Good for finding missing functions
  - Both on domain and product level
  - Can sometimes be very complex and difficult to read



- **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 …

Housekeeping

What is requirements interpretation

Why is it challenging

Functional requirements

Wrapping up

Outline

1. Housekeeping
2. What is requirements interpretation
3. Why is it challenging
4. Functional requirements
5. Wrapping up

Todo

- Read [CREA] for next week's lecture and workshop
- Lecture L6: Continue from today, add Negotiation (inter-dependencies, inconsistencies, prioritization)
- Work in your groups: Continue to do elicitation, start writing requirements, complement with creativity workshop next week.

→ `socrative.com`, REQENG, Q12

L5 Creativity

Slot 2 (Sep-18, 13:15-15:00, J122)

- Group 7
- Group 8
- Group 9
- Group 10
- Group 11
- Group 12
- Group 13

Slot 4 (Sep-19, 13:15-15:00, J121)

- Group 21
- Group 22
- Group 23
- Group 24
- Group 25
- Group 26
- Group 27

Slot 1 (Sep-16, 15:15-17:00, Omega)

- Group 0
- Group 1
- Group 2
- Group 3
- Group 4
- Group 5
- Group 6

Slot 3 (Sep-18, 15:15-17:00, J122)

- Group 14
- Group 15
- Group 16
- Group 17
- Group 18
- Group 19
- Group 20

References I

Glinz, M. (2007).
On non-functional requirements.
In *Proc. of 15th IEEE Int. RE Conf. (RE)*, pages 21–26, New Delhi, India.

Lauesen, S. (2002).
Software Requirements.
Pearson / Addison-Wesley.
the course book (Lau).