Introduction to Software Requirements Engineering

Part II: Functionality

Björn Regnell

Updated: 2024-04-02

Part II: Functionality
Outline

Requirements Modeling

A requirements model...

- is an informative **representation** of the intended system.
- is an **abstraction** of the intended system
  - only *some* aspects are represented,
  - while other aspects are *excluded*.
- captures important knowledge gained from **elicitation**.
- is often textual or a diagram, or both.
- is often part of a **specification**.
- should be **validated** by (some) stakeholders.

Different aspects of requirements to model

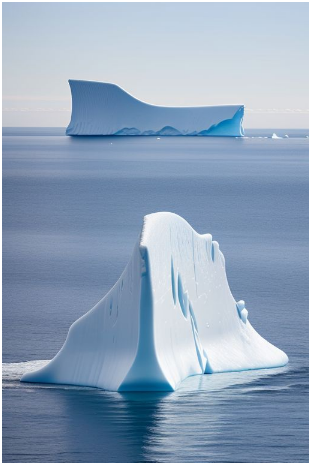
- Functional aspects:
  - Data aspects:
    - What is stored and processed by the system?
    - What is the format of input and output data?
  - Business Logic aspects:
    - How should the system behave in different usage contexts?
    - What output should be produced, given input and state?

- Quality aspects:
  - What is a 'good' function from stakeholders' viewpoints?
  - More on quality modeling in Part III.

Which modeling technique is best?

- A modeling technique may be more or less suitable for representing some aspects of requirements from some stakeholders' viewpoint.
- A **well-balanced combination** of modeling techniques can capture a more aspects in a better way than a single technique.
- What modeling technique is best depends on...
  - abstraction level
  - project type
  - stakeholders
  - tool support
  - the number and kind of requirements
  - ...
- Model interpretation often requires specific knowledge:
  - Choose modeling technique based on stakeholders' ability to understand and validate!
- How do you know that your chosen mix of models fit together and does not contradict each other?
  - Make sure to check model inter-consistency in validation!

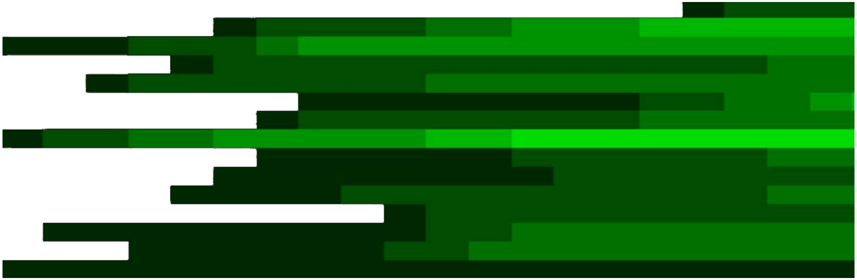
Abstraction and level of detail



- The iceberg metaphor:
  - Initial requirements are only the tip of a requirements "iceberg".
  - Most requirements are still hidden under the surface.
- Abstraction means simplification:
  - focusing on what's important,
  - reduction of less relevant details.
- Different parts of the req space need different levels of detail.
- Which parts of the under-surface iceberg should be unravel first?

Mixed levels of detail over time

- The picture below illustrates elaboration of reqs over time:
  - Time on x-axis. Different reqs on y-axis.
  - Lighter color means deeper knowledge, improved spec.



- It is not cost-effective to spend the same amount of effort on all requirements.
- Some requirements need more effort than others in elicitation, specification, validation and selection.

Ontology

- Ontology is the philosophical study of being:
  - What types of entities exist?
  - How are entities grouped into categories?
  - How are entities related to one another?
- Example of ontological concepts
  - Particulars and universals
  - Abstract and concrete
  - Identity and persistence over time
  - Modality: what is possible, actual, necessary?
  - Properties and relations
- Further reading:
  - `https://en.wikipedia.org/wiki/Ontology`
  - `https://en.wikipedia.org/wiki/Ontology_engineering`

Examples of functional modeling techniques

Context	***Data***	***Business Logic***
Business model	Data dictionary	User stories
Stakeholder model	Data views	Use cases and tasks
Goal model	E/R-diagrams	Narratives
Context diagram	Class diagrams	State diagrams
	Regular expressions	Interaction diagrams
	Protocol data specs	Data-flow diagrams

In complement or alternative to above techniques:

- **Feature requirements** in natural language:
  - explicit textual requirements "The system shall..."
  - a product property that can be selected for a release or postponed
  - can combine functional aspects (data, logic) and quality aspects
  - link to above models

Data Modeling

- **Data dictionary**:
  - a list of data entities (classes) with textual descriptions of data attributes (fields) and relations among entities.
- **Data views**: (a.k.a ”virtual windows”)
  - examples of data values (instances) shown in a specific usage context sketched as a mockup screen
  - *not* intended as interface design – instead the focus is on modeling of stored and processed data
- **Data diagrams**:
  - boxes and arrows with data entities (classes), attributes (fields) and relations
  - **E/R-diagram**: focus on entities and their relations
  - **Class diagram**: also inheritance, aggregation, methods
- **Data format specs**:
  - Reqular expressions (regexp)
  - Protocol buffers (protobuf)

Data dictionary example

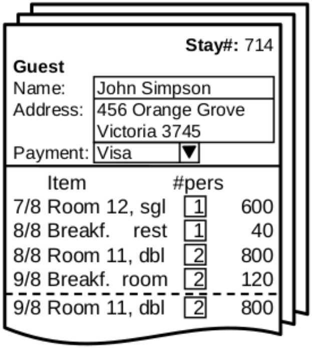
Each data entity (class) together with attributes (fields) are specified in structured natural language.

```
* Product: MusicStreamingApp
* Data: Dictionary
 * Class: listener has
  * Spec: A user that listens to music.
  * Field: id has
   * Spec: A unique identifier of listener. Must only contain letters and digits.
   * Example: abc123
  * Field: password has
   * Spec: ...
 * Class: artist has Spec: ...
 * Class: track has Spec: ...
 * Class: album has Spec: ...
```

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

Data Views

- Example data shown in a mockup screen.
- Good for data model validation with stakeholders.
  - Are examples relevant?
  - Any missing data?
- Also called ”virtual window”.
- Not a user interface design.



Stay#: 714

Guest

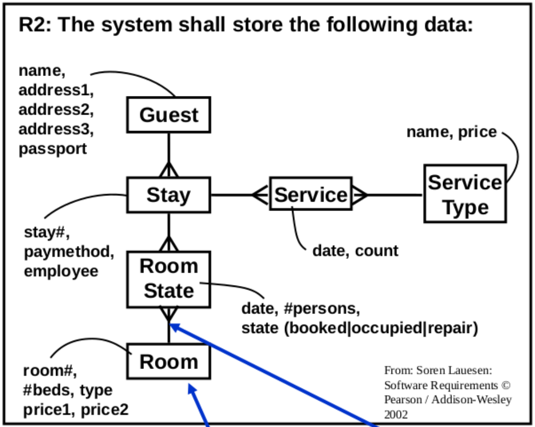
Name:	John Simpson
Address:	456 Orange Grove Victoria 3745
Payment:	Visa ▼

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

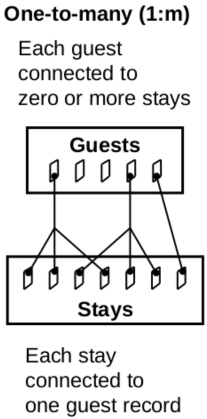
From: S. Lauesen ”Software Requirements” © Addison-Wesley 2002

Entity-Relationship-diagram

https://en.wikipedia.org/wiki/Entity-relationship_model


R2: The system shall store the following data:
name, address1, address2, address3, passport
Guest
name, price
Stay
Service
Service Type
stay#, paymethod, employee
date, count
Room State
date, #persons, state (booked|occupied|repair)
Room
room#, #beds, type price1, price2
From: Soren Lauesen: Software Requirements © Pearson / Addison-Wesley 2002

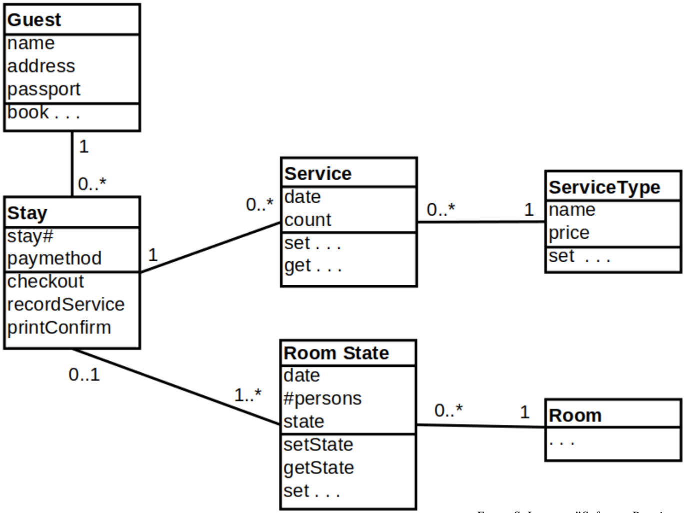
Entities and Relationships


One-to-many (1:m)
Each guest connected to zero or more stays
Guests
Stays
Each stay connected to one guest record

From: S. Lauesen "Software Requirements" © Addison-Wesley 2002

Class diagram

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



From: S. Lauesen "Software Requirements" © Addison-Wesley 2002

Business Logic Modeling

- Contextual usage modeling:
  - User story: short description "user ... wants ... to ..."
  - Task: domain-focused model of user's work
  - Use case: product-focused model of user's interaction
  - Narrative: rich, detailed story of personas' experiences
  - ...
- System behavior modelling:
  - State machines: events trigger transitions between states
  - Interaction diagrams: user–system interaction sequences
  - Data-flow diagrams: flow of input/output to/from functions
  - ...

Work Split: When to decide who does what?

- Domain-focused logic:
  - users' work and system behaviour is joint
  - the work split is decided in later design
- Product-focused logic:
  - users' work and system behaviour is separated
  - the work split is explicitly specified in req spec

Contextual usage modeling

Also called **scenario-based requirements**

https://en.wikipedia.org/wiki/Scenario_(computing)

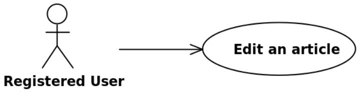
- User story:
  - Short description of what a user role wants to do with the system in order to achieve a goal.
  - https://en.wikipedia.org/wiki/User_story
- Use cases and tasks:
  - *Domain-focus:* often called (user) **task**
    A piece of work by users, potentially supported by a system.
  - *Product-focus:* often called **use case**
    A goal-fulfilling interaction between users and the product.
  - https://en.wikipedia.org/wiki/Use_case
- Narrative:
  - A detailed story from a user's perspective including thoughts and emotions when using the system for a specific purpose.

User Stories

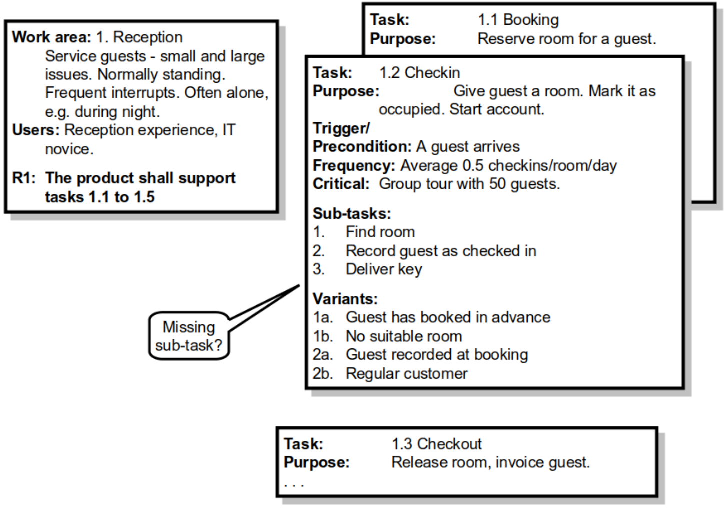
- Short description of what a user role wants to do with the system in order to achieve a goal.
- Some proposed templates:
  - `As a <role> I can <capability>, so that <receive benefit>.`
  - `In order to <receive benefit> as a <role>, I can <desire>.`
  - `As <who> <when> <where>, I want <what> because <why>.`

Use cases

- Definition according to `https://en.wikipedia.org/wiki/Use_case`
  - A usage scenario for a piece of software.
  - A potential scenario in which a system receives an external request and responds to it.
- Contents of textual use case:
  - Title
  - Primary Actor
  - Brief (corresponds to user story)
  - Stakeholders
  - Pre- and postconditions
  - Triggers
  - Basic flows
  - Extensions



Tasks



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

R1: The product shall support tasks 1.1 to 1.5

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

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

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

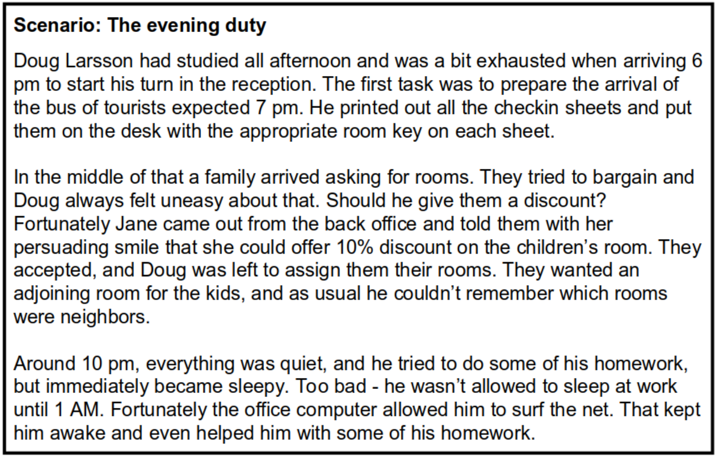
Variants:
1a. Guest has booked in advance
1b. No suitable room
2a. Guest recorded at booking
2b. Regular customer

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

From: S. Lauesen "Software Requirements" © Addison-Wesley 2002

Narratives and personas

- https://en.wikipedia.org/wiki/Scenario_(computing)
- https://en.wikipedia.org/wiki/Persona_(user_experience)



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.

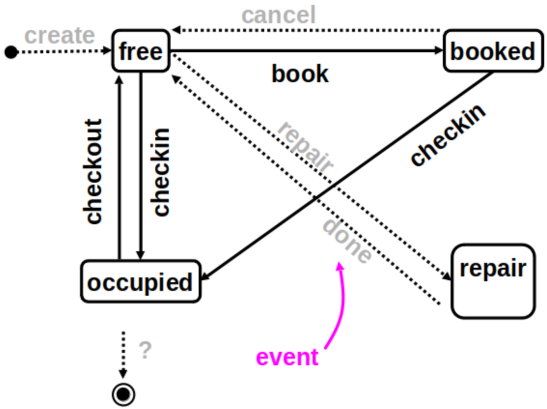
Around 10 pm, everything was quiet, and he tried to do some of his homework, but immediately became sleepy. Too bad - he wasn't allowed to sleep at work until 1 AM. Fortunately the office computer allowed him to surf the net. That kept him awake and even helped him with some of his homework.

From: S. Lauesen "Software Requirements" © Addison-Wesley 2002

System behavior modeling

- State diagrams
  - Events trigger transitions between states
  - A model of many possible event sequences
  - `https://en.wikipedia.org/wiki/State_diagram`
- Interaction diagrams
  - Messages between the system and its surroundings
  - A model of one specific interaction scenario
  - `https://en.wikipedia.org/wiki/Message_sequence_chart`
  - `https://en.wikipedia.org/wiki/Sequence_diagram`
- Data-flow diagrams
  - Flow of input/output to/from functions
  - A model of how functions are connected with data stores
  - `https://en.wikipedia.org/wiki/Data-flow_diagram`

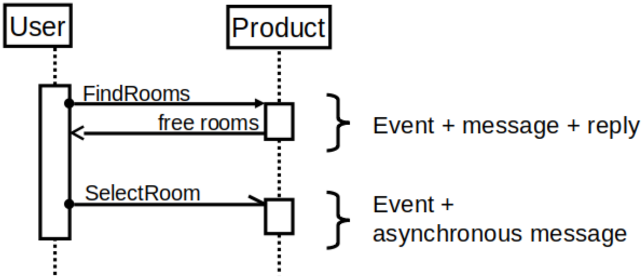
State diagram example



From: S. Lauesen "Software Requirements" © Addison-Wesley 2002

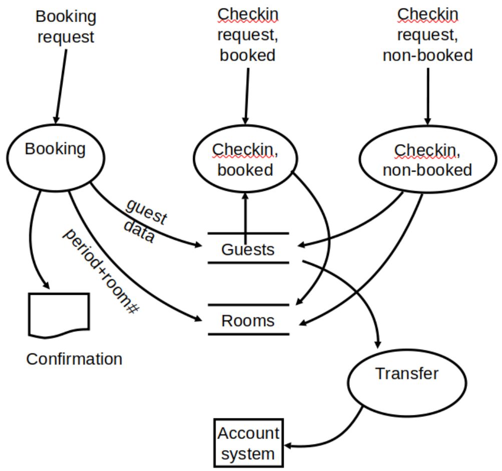
Interaction Diagrams

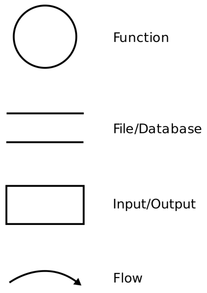
- Time progresses downwards
- One timeline for each user, product, other systems
- Events/messages are drawn as arrows between timelines
- Example sequence diagram:



From: S. Lauesen "Software Requirements" © Addison-Wesley 2002

Data-flow diagram example





From: S. Lauesen "Software Requirements" © Addison-Wesley 2002

What is prototyping?

- Prototyping is a creative practice within product design.
- A prototype can range from
  - a simple paper **sketch**, through
  - a computer-generated **mock-up**, to an
  - incomplete version of the production software
- Aspects of a prototype:
  - Purpose
  - Scope
  - Media
  - Usage

Why prototype?

- Exploration and learning
- Communication: sales, alignment
- Incremental development
- Quality improvement
- Validation and testing
    - investigate problem-solution / product-market fit
    - investigate technical feasibility
    - usability testing

Prototype Scoping

- Breadth of functionality
- Functional refinement:
  - Visual appearance
  - Interactive and haptic behavior
  - Data realism

Prototyp Media

- Sketches and mockups
  - paper drawing
  - computer drawing
  - presentation tool
  - GUI-builder
- Video
  - enacted usage
  - interview
- Executable software
  - demonstrate single feature
  - MVP (minimum viable product)

Prototype Usage

- Reviewers
  - internal
  - with families, friends and foes
  - external
- Prototype interaction
  - yes
  - no (demo)
- Review approach
  - scenario-based
  - free
- Usage environment
  - *in vitro*: lab setting, different from final product usage
  - *in vivo*: real world setting, similar to final product usage

Exploration Strategy

- How to traverse the solution space?
  - Number of iterations:
    - single iteration
    - multiple iterations
    - parallell explorations
  - Iteration focus:
    - business
    - product
    - feature
    - optimisation
  - Iteration size:
    - effort
    - lead-time

Delegated Requirements

- Refer to an **existing artifact** that acts as a **proxy** for requirements.
- Examples:
  - Standards as requirements
    - require a standard query language, SQL
    - require a standard communication protocol, e.g. HTTPS
  - Regulatory requirements
    - General Data Protection Regulation
    - Artificial Intelligence Act
    - Cyber Resilience Act
  - Test cases as requirements
  - Development process requirements
    - require usability testing to instead of explicit usability requirements
    - require regression testing to instead of explicit reliability requirements

Delegated Requirements: Pros and cons

- Advantages:
  - less work: the knowledge is already elicited, specified, validated
  - more concise specification
  - aligned with accepted domain practice
  - mature proxies have high quality, high credibility
- Disadvantages:
  - gives a false sense of security: proxy may be ambiguous, incomplete
  - proxy may impose premature design
  - proxy may be difficult to understand
  - proxy may be "over-kill", include unnecessary requirements
  - All or nothing if you want to claim compliance
    $\rightarrow$ unwanted requirements must be implemented