Scenario-Based Analysis

- Provides a more user-oriented view perspective on the design and development of an interactive system.
- The defining property of a scenario is that it projects a concrete description of an activity that the user engages in when performing a specific task, a description sufficiently detailed so that the design implications can be inferred and reasoned about.

Scenario-Based Analysis (example)

- first shot:
  - check due back date
  - if overdue, collect fine
  - record book as being available again
  - put book back
- as a result of discussion with library employee:
  - what if person returning the book is not registered as a client?
  - what if the book is damaged?
  - how to handle in case the client has other books that are overdue, and/or an outstanding reservation?

Scenario-Based Analysis

<table>
<thead>
<tr>
<th>Scenario view</th>
<th>Standard view</th>
</tr>
</thead>
<tbody>
<tr>
<td>concrete descriptions</td>
<td>abstract descriptions</td>
</tr>
<tr>
<td>focus on particular instances</td>
<td>focus on generic types</td>
</tr>
<tr>
<td>work-driven</td>
<td>technology-driven</td>
</tr>
<tr>
<td>open-ended, fragmentary</td>
<td>complete, exhaustive</td>
</tr>
<tr>
<td>informal, rough, colloquial</td>
<td>formal, rigorous</td>
</tr>
<tr>
<td>envisioned outcomes</td>
<td>specified outcomes</td>
</tr>
</tbody>
</table>

Form analysis

Proceedings request form:

<table>
<thead>
<tr>
<th>Client name</th>
<th>Title</th>
<th>Editor</th>
<th>Place</th>
<th>Publisher</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Certainty vs uncertainty
Prototyping

- The simplest kind: paper prototype.
  - a set of pictures of the system that are shown to users in sequence to explain what would happen
- The most common: a mock-up of the system’s UI
  - Written in a rapid prototyping language
  - Does not normally perform any computations, access any databases or interact with any other systems
  - May prototype a particular aspect of the system

Use case analysis

- Determine the classes of users that will use the facilities of this system (actors)
- Determine the tasks that each actor will need to do with the system

Use-Cases: describing how the user will use the system

- A use case is a typical sequence of actions that a user performs in order to complete a given task
- The objective of use case analysis is to model the system from the point of view of ... how users interact with this system ... when trying to achieve their objectives.
  It is one of the key activities in requirements analysis
- A use case model consists of
  - a set of use cases
  - an optional description or diagram indicating how they are related

Use cases

- A use case should
  - Cover the full sequence of steps from the beginning of a task until the end.
  - Describe the user’s interaction with the system ...
    - Not the computations the system performs.
  - Be written so as to be as independent as possible from any particular user interface design.
  - Only include actions in which the actor interacts with the computer.
    - Not actions a user does manually
Scenarios

- A scenario is an instance of a use case
- A specific occurrence of the use case
  - a specific actor ...
  - at a specific time ...
  - with specific data.

How to describe a single use case

A. Name: Give a short, descriptive name to the use case.
B. Actors: List the actors who can perform this use case.
C. Goals: Explain what the actor or actors are trying to achieve.
D. Preconditions: State of the system before the use case.
E. Summary: Give a short informal description.
F. Related use cases.
G. Steps: Describe each step using a 2-column format.
H. Postconditions: State of the system in following completion.

- A and G are the most important

Use case diagrams

The modeling processes: Choosing use cases on which to focus

- Often one use case (or a very small number) can be identified as central to the system
  - The entire system can be built around this particular use case
- There are other reasons for focusing on particular use cases:
  - Some use cases will represent a high risk because for some reason their implementation is problematic
  - Some use cases will have high political or commercial value
The benefits of basing software development on use cases

- They can
- Help to define the scope of the system
- Be used to plan the development process
- Be used to both develop and validate the requirements
- Form the basis for the definition of test cases
- Be used to structure user manuals

Use cases must not be seen as a panacea

- The use cases themselves must be validated
- Using the requirements validation methods.
- Some aspects of software are not covered by use case analysis.
- Innovative solutions may not be considered.

Requirement documents

- An informal outline of the requirements using a few paragraphs or simple diagrams
- requirements definition
- A long list of specifications that contain thousands of pages of intricate detail
- requirements specification

- Requirements documents for large systems are normally arranged in a hierarchy

- Level of required detail
  - The size of the system
  - The need to interface to other systems
  - The readership
  - The stage in requirements gathering
  - The level of experience with the domain and the technology
  - The cost that would be incurred if the requirements were faulty
Prioritizing requirements (MoSCoW)

- **Must haves**: top priority requirements
- **Should haves**: highly desirable
- **Could haves**: if time allows
- **Won’t haves**: not today

Prioritizing requirements (Kano model)

- **Attractive**: more satisfied if +, not less satisfied if –
- **Must-be**: dissatisfied when -, at most neutral
- **One-dimensional**: satisfaction proportional to number
- **Indifferent**: don’t care
- **Reverse**: opposite of what analyst thought
- **Questionable**: preferences not clear

Kano model

Requirements specification

- readable
- understandable
- non-ambiguous
- complete
- verifiable
- consistent
- modifiable
- traceable
- usable
- …
IEEE Standard 830

1. Introduction
   1.1. Purpose
   1.2. Scope
   1.3. Definitions, acronyms and abbreviations
   1.4. References
   1.5. Overview

2. General description
   2.1. Product perspective
   2.2. Product functions
   2.3. User characteristics
   2.4. Constraints
   2.5. Assumptions and dependencies

3. Specific requirements
   3.1. External interface requirements
   3.1.1. User interfaces
   3.1.2. Hardware interfaces
   3.1.3. Software interfaces
   3.1.4. Comm. interfaces
   3.2. Functional requirements
   3.2.1. User class 1
   3.2.1.1. Functional req. 1.1
   3.2.1.2. Functional req. 1.2
   3.2.2. User class 2
   3.3. Performance requirements
   3.4. Design constraints
   3.5. Software system attributes
   3.6. Other requirements

Requirements management

- Requirements identification (number, goal-hierarchy numbering, version information, attributes)
- Requirements change management (CM)
- Requirements traceability:
  - Where is requirement implemented?
  - Do we need this requirement?
  - Are all requirements linked to solution elements?
  - What is the impact of this requirement?
  - Which requirement does this test case cover?
- Related to Design Space Analysis
The 7 sins of the analyst

- noise
- silence
- overspecification
- contradictions
- ambiguity
- forward references
- wishful thinking

Functional vs. Non-Functional Requirements

- functional requirements: the system services which are expected by the users of the system.
- non-functional (quality) requirements: the set of constraints the system must satisfy and the standards which must be met by the delivered system.
  - speed
  - size
  - ease-of-use
  - reliability
  - robustness
  - portability

Reviewing requirements

- Each individual requirement should
  - Have benefits that outweigh the costs of development
  - Be important for the solution of the current problem
  - Be expressed using a clear and consistent notation
  - Be unambiguous
  - Be logically consistent
  - Lead to a system of sufficient quality
  - Be realistic with available resources
  - Be verifiable
  - Be uniquely identifiable
  - Does not over-constrain the design of the system

Validation of requirements

- Inspection of the requirement specification w.r.t. correctness, completeness, consistency, accuracy, readability, and testability.
- Some aids:
  - structured walkthroughs
  - prototypes
  - simulation
  - use cases and scenarios analysis
  - develop a test plan
  - tool support for formal specifications
Requirements Review Checklist

1. Does the (software) product have a succinct name, and a clearly described purpose?
2. Are the characteristics of users and of typical usage mentioned? (No user categories missing.)
3. Are all external interfaces of the software explicitly mentioned? (No interfaces missing.)
4. Does each specific requirement have a unique identifier?
5. Is each requirement atomic and simply formulated? (Typically a single sentence. Composite requirements must be split.)
6. Are requirements organized into coherent groups? (If necessary, hierarchical; not more than about ten per group.)
7. Is each requirement prioritized? (Is the meaning of the priority levels clear?)

8. Are all unstable requirements marked as such? (TBC=‘To Be Confirmed’, TBD=‘To Be Defined’)
9. Is each requirement verifiable (in a provisional acceptance test)? (Measurable: where possible, quantify; capacity, performance, accuracy)
10. Are the requirements consistent? (Non-conflicting.)
11. Are the requirements sufficiently precise and unambiguous? (Which interfaces are involved, who has the initiative, who supplies what data, no passive voice.)
12. Are the requirements complete? Can everything not explicitly constrained indeed be viewed as developer freedom? Is a product that satisfies every requirement indeed acceptable? (No requirements missing.)
13. Are the requirements understandable to those who will need to work with them later?
14. Are the requirements realizable within budget?
15. Do the requirements express actual customer needs (in the language of the problem domain), rather than solutions (in developer jargon)?

Requirements documents...

- The document should be:
  - sufficiently complete
  - well organized
  - clear
  - agreed to by all the stakeholders

- Traceability:

  Requirements document

  1.1 XXXX

  1.2 YYYY

  ...due to requirement 1.2

  Design document

  rationale

  ...due to requirement 1.2
Managing Changing Requirements

- Requirements change because:
  - Business process changes
  - Technology changes
  - The problem becomes better understood

- Requirements analysis never stops
  - Continue to interact with the clients and users
  - The benefits of changes must outweigh the costs.
    - Certain small changes (e.g. look and feel of the UI) are usually quick and easy to make at relatively little cost.
    - Larger-scale changes have to be carefully assessed
      - Forcing unexpected changes into a partially built system will probably result in a poor design and late delivery
  - Some changes are enhancements in disguise
    - Avoid making the system bigger, only make it better