Techniques for Requirements Gathering and Definition

Kristian Persson
Principal Product Specialist
Requirements Lifecycle Management

- Elicit and define business/user requirements
- Validate requirements
- Analyze requirements to define the solution
- Drive implementation from the requirements
- Drive testing from the requirements
- Manage traceability and change
Requirements Definition

Requirements Lifecycle Management
- Change Management
- Traceability
- Impact Analysis
- Reporting & Metrics
- Monitoring

Business Definition
- Elicitation
- Specification
- Visualization
- Storyboarding

Analysis
- Definition
- Validation

Systems Analysis & Implementation
- Analysis
- Implementation
- Test
Distinguish between Problem and Solution

Problem definition

Risk of defining the wrong problem

Risk of not meeting the requirements

Risk of unnecessarily constraining the solution

Stakeholder Requirements

System Requirements

Abstract solution

Specific solution

Should be possible to change the design without changing system requirements

Design
Problem and Solution

• What causes apples to fall off trees?

• What is the purpose of apples falling off trees?

• What are the user requirements for a set of traffic lights to control traffic at a road junction?
## Differentiating Problem and Solution

### Problem

**Customer requirements**
- A description of the problem and its context
- Results that stakeholders want from the system
- Do not define the solution, other than for environment
- Quality of results
- Owned by stakeholders or their representatives (e.g. business)

### Solution

**System requirements**
- An *abstract* representation of the solution
- What the system does
- Do not define the design
- How well it does it
- Owned by IT or systems engineers

**“The user shall be able to ....”**

**“The system shall do ....”**
Results of mixing Problem and Solution

• Don’t understand the problem

• Can’t decide on functions

• Developers dominate

• Can’t do acceptance

• User and system constraints muddled

• Unclear ownership

Disaster!
Gathering requirements

• Problem domain
  – Concentrate on what the stakeholders want to be able to do: i.e. Capabilities
  – Do not design the system

• System Level
  – Concentrate on what the system must do (function)
  – Do not define the design (i.e. avoid saying how)

• Subsystem and lower level components
  – Concentrate on what the subsystem or component must do (function)
  – Make design constraints explicit (e.g. interface obligations)
  – Do not over constrain
    • leads to inflexibility,
    • builds in obsolescence,
    • reduces/removes freedom to innovate
  – Do not define the design (i.e. avoid saying how)
Use the four key questions

1. What is the purpose?
   • Focus on problem rather than solution
   • Recognize where a requirement is stated as a solution
   • Express requirement in a solution-free manner

2. What is the underlying goal?
   • Helps to express the requirement properly
   • Helps to quantify the requirement
   • Is it to maximise, minimise or optimise?

3. Is there an implied solution?
   • Understand why a particular solution may be required
   • Collect the constraints that lead to the suggested solution

4. How will you know if the need has been met?
   • Make requirements quantifiable and testable (see later)
Use concise, clear, consistent language

Each requirement statement should be:

1. **Individual**: each statement is a single traceable element
2. **Unique**: each statement is uniquely identified
3. **Clear**: each statement is clearly understandable
4. **Precise**: each statement is precise and concise
5. **Abstract**: does not impose a solution on the next layer
6. **Quantified**: each statement has acceptance criteria
7. **Testable**: each statement can be validated/verified
Six Things to Avoid

1. **Rambling**: conciseness is a virtue

2. **Let-out clauses**: such as “if that should be necessary”; they render the requirements useless

3. **Multiple requirements**: often indicated by “and”, “or”, “but”, “however”

4. **Vague terms**: usually, generally, often, normally, typically, user friendly, versatile, flexible

5. **Wishful thinking**: “100% reliable”, “please all users”, “run on all platforms”, “handle all unexpected failures”, “upgradeable to all future situations”

6. **Speculation**: stick to what you know
Focus on documents as well as statements

Need to balance two aspects:

• Focus on the individual statement of requirement (principle 4):
  – Language
  – Clarity, preciseness
  – Identity, traceability

• Focus on the requirements document:
  – Understanding context
  – Assessing completeness
  – Identifying repetition/conflict
  – Navigating/searching requirements
Structuring Documents - Goals

Organizing requirements into the right structure can help:

- **Understand** large amounts of information
- **Navigate** sets of requirements relating to particular topics
- **Detect** omissions and duplications
- **Minimize** the number of requirements
- **Eliminate** conflicts between requirements
- **Evaluate** requirements sets
- **Reuse** requirements across projects
Seven Criteria for Requirements Documents

Each requirements set should be:

1. **Complete / Sufficient**: all requirements are present
2. **Consistent**: no two requirements are in conflict
3. **Non-redundant**: each requirement is expressed once
4. **Modular**: requirements statements that belong together are close to one another
5. **Structured**: there is a clear structure to the requirements document
6. **Satisfied**: the appropriate degree of design traceability has been achieved
7. **Evaluated**: the appropriate degree of test traceability has been achieved

Define an outline structure at the outset, and improve it as you go.

Good Practice: Focus on documents as well as statements
Understand the role of modelling

- The requirements are the “bread and butter” of development.
- What is a sandwich without the bread?
- Requirements alone are a little dry.
- Modeling is what makes the whole rather more interesting.
- The filling holds the bread together.
- It is the bread and the filling together that make a sandwich.
Complementary techniques

• Requirements management:
  – capture of and traceability between individual textual requirements

• Modelling:
  – multiple views of structured information
  – consistency can be checked across the system using the model data dictionary
  – allows animation to be used as a validation technique

• The model is not the requirement
  – non-functional requirements are typically not captured in a model
  – a graphical model is generally insufficient as a contractual basis
The Role of Models

• Modeling is used to understand and analyze a problem

• Models provide a vital means of communication with
  – stakeholders
  – team members

• Modeling provides a basis for reasoning about:
  – a problem,
  – potential solutions, etc

• Modeling complements requirements engineering
Models Bridge Layers of Requirements

- Requirements layer
- Modeling layer
- Requirements layer
- Modeling layer
- Requirements layer
- Modeling layer

Statement of need

- e.g. Goal / Usage modeling
- e.g. Functional modeling
- e.g. Architecture modeling

Stakeholder Requirements

- System Requirements

Design Specification
Rational Requirements Composer*

Requirements Definition for DOORS and RequisitePro

- **Rich Authoring Environment**
  - Rich Text Requirements
  - Glossaries
  - UI Sketching and Storyboarding
  - Process Sketching

- **Web Review and Approval**
  - Wiki style interface
  - Categorize / Tag
  - Comment
  - Review / Approve
  - Share work instantly
  - Users / teams / authorizations
  - Linking between all artifacts
  - Versioning

*Not yet available; details subject to change
Requirements Validation

Requirements Lifecycle Management
- Change Management
- Traceability
- Impact Analysis
- Reporting & Metrics
- Monitoring

Validation
- Prototyping
- Simulation
- Prioritization
- Review
- Define Test & Acceptance Criteria
Model simulation for requirements validation

• Use requirement scenarios to validate the design
• Simulate to verify that model is correct
  – Avoid errors
  – Reduce costly rework
  – Increase quality
• Virtual prototype / Panel graphics support
  – Ideal communications aid for design reviews and to share information.
Prioritization
Quantifying requirements for testing

• Of every requirement statement, ask:
  – “How will you know if the need has been met?”

• Improves the way the requirement is expressed
  – Is it quantified?
  – What are the success criteria?
  – Add requirements to make system testable

• Plan the tests now, not later:
  – What kind of tests will be used?
  – When will the tests be performed?

• Preparing the tests may take months or years:
  – Collect requirements for test facilities

• Trace tests to requirements
  – Include tests in impact analysis
Quantifying Requirements

• Quantities relate to availability, coverage, timeliness, readiness...

• May be related to capabilities, functions or constraints
  – sometimes maximum or minimum level set as constraint

• Defines the trade-off space
  – by indicating the scope for negotiation between conflicting goals

• Gives requirements test criteria
Quantification Example

**Requirement**
- The system shall handle simultaneous users

<table>
<thead>
<tr>
<th>Performance</th>
<th>Value to user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best 200</td>
<td>Fail 100</td>
</tr>
<tr>
<td>Plan 100</td>
<td>Pass 200</td>
</tr>
<tr>
<td>Minimum acceptable 50</td>
<td>Value curve</td>
</tr>
</tbody>
</table>

Good Practice: Quantify requirements for testing
Characterising Quantification

- **Maximise**: Performance vs. value for 50, 100, and 200 users.
- **Exceed**: Performance vs. value for 50, 100, and 200 users.
- **Minimise**: Performance vs. value for 5, 10, and 20 kg.
- **Optimise**: Performance vs. value for 3, 7, and 10 Krpm.
Requirements Lifecycle Management

- Elicit and define business/user requirements
- Validate requirements
- Analyze requirements to define the solution
- Drive implementation from the requirements
- Drive testing from the requirements
- Manage traceability and change