Requirements Engineering
Requirements engineering

• Process of figuring out
  • Services the customer needs
  • Constraints of operation

• It is about WHAT will be built!
Why Develop Requirements Specs?

I believe that on any non-trivial project (more than about 1 week of coding or more than 1 programmer), if you don't have a spec, you will always spend more time and create lower quality code.

Joel Spolsky
http://www.joelonsoftware.com
Requirement

- Descriptions of
  - system services
  - constraints

- Gathered during the requirements engineering process.
What is a requirement?

• Depends...
  • high-level abstract statement
  • detailed mathematical functional specification
Formal Specification -- VDM

\[ p_1 : \text{Path} = \text{mk\_token} \hspace{1em} ("A1\text{North}"); \]
\[ p_2 : \text{Path} = \text{mk\_token} \hspace{1em} ("A1\text{South}"); \]
\[ p_3 : \text{Path} = \text{mk\_token} \hspace{1em} ("A66\text{East}"); \]
\[ p_4 : \text{Path} = \text{mk\_token} \hspace{1em} ("A66\text{West}"); \]

\[ \text{lights} : \text{map Path to Light} = \{ p_1 |\to <\text{Red}>, \]
\[ \hspace{1em} p_2 |\to <\text{Red}>, \]
\[ \hspace{1em} p_3 |\to <\text{Green}>, \]
\[ \hspace{1em} p_4 |\to <\text{Green}>\}; \]

\[ = \{ \text{mk\_Conflict}(p_1,p_3), \]
\[ \hspace{1em} \text{mk\_Conflict}(p_1,p_4), \]
\[ \hspace{1em} \text{mk\_Conflict}(p_2,p_3), \]
\[ \hspace{1em} \text{mk\_Conflict}(p_2,p_4), \]
\[ \hspace{1em} \text{mk\_Conflict}(p_3,p_1), \]
\[ \hspace{1em} \text{mk\_Conflict}(p_4,p_1), \]
\[ \hspace{1em} \text{mk\_Conflict}(p_3,p_2), \]
\[ \hspace{1em} \text{mk\_Conflict}(p_4,p_2)\}; \]

\text{Types}
\[
\text{Light} = <\text{Red}> \mid <\text{Amber}> \mid <\text{Green}>;
\]
Requirement Spec Use

- Design
- Communicate
- Test
Types of Requirements

- Functional
  - Behavior of system
  - From users point of view
- Non-functional
  - Non behavior related constraints
Types of requirement

- **User** requirements
  - Written for customers
  - Natural Language
  - Diagrams

- **System** requirements
  - Detailed descriptions system functions, services, and operational constraints.
Types of requirements

• Functional
  • Services the system must provide

• Non-functional
  • Constraints on the services
    • i.e timing, development process, standards, etc.
  • Apply to whole system rather than functions
Good SRS

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable
Clear description

• Must be precise

• Ambiguous requirements
  • Different interpretation

• How can we avoid ambiguity?
Completeness & Consistency

• Should be complete & consistent
• Complete
  • All required functionality is stated
• Consistent
  • There are no conflicts between requirements

• In practice: Impossible
Non-functional requirements

- System properties and constraints
  - Up time
  - Response time
  - Storage requirements
  - Usability
- Process requirements
  - IDE
  - Programming language
  - Development method.
- May be more critical than functional requirements
Verifiable Non-functional Requirement Description

- Verifiable non-functional requirement
  - Measurable
  - Can be tested

- Difficult to state *precisely* → difficult to verify.
## Metrics for nonfunctional requirements

<table>
<thead>
<tr>
<th>Property</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Processed transactions/second</td>
</tr>
<tr>
<td></td>
<td>User/event response time</td>
</tr>
<tr>
<td></td>
<td>Screen refresh time</td>
</tr>
<tr>
<td>Size</td>
<td>Mbytes</td>
</tr>
<tr>
<td></td>
<td>Number of ROM chips</td>
</tr>
<tr>
<td>Ease of use</td>
<td>Training time</td>
</tr>
<tr>
<td></td>
<td>Number of help frames</td>
</tr>
</tbody>
</table>
## Metrics for NF Req. (cont.)

<table>
<thead>
<tr>
<th>Category</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Mean time to failure, Probability of unavailability, Rate of failure occurrence, Availability</td>
</tr>
<tr>
<td>Robustness</td>
<td>Time to restart after failure, Percentage of events causing failure, Probability of data corruption on failure</td>
</tr>
<tr>
<td>Portability</td>
<td>Number of target systems</td>
</tr>
</tbody>
</table>
Guidelines for writing requirements

- Use a standard format
- Be consistent
- Use **shall** for mandatory requirements
- Highlight key parts
- Use **structure** to group related requirements
- Enumerate
Requirement Language

- Requirements are often written in natural language (e.g., English).
- inherently ambiguous
- should be **reviewed** by an independent party to identify ambiguous language so that it can be corrected
Consistency

• With external objects
  • Incorrect descriptions of real objects
  • Ex: Blue background vs Green background
• Logical (A x B vs A / B)
• Temporal (A after B vs A and B simultaneously)
• Note: Use consistent and precise terminology
• Agreement with terminology in a project team is crucial
Requirements engineering processes

• Requirements elicitation
• Requirements analysis
• Requirements validation
• Requirements management

• In practice
  • iterative activity
  • processes are interleaved.
Requirements elicitation and analysis

- Requirements discovery
- Requirements classification and organization
- Requirements prioritization and negotiation
- Requirements specification
Scenarios

• Scenarios are real-life examples

• Consists of
  • Starting situation
  • Normal flow of events
  • What can go wrong
  • Information about other concurrent activities
  • Finishing situation
Use cases

• Scenario based technique in the UML

• Identifies the actors and the interaction

• A set of use cases should describe all possible interactions with the system.
Use cases for Hospital System

Medical receptionist
- Register patient
- View personal info.
- View record
- Edit record
- Setup consultation

Nurse

Manager
- Export statistics
- Generate report

Doctor

Requirements engineering
Requirements validation

• Do the requirements define the system that the customer really wants?

• Requirements error is very costly
Requirements checking

• Validity. Does the system provide the functions which support customer needs?
• Consistency. Are there requirements conflicts?
• Completeness. Are all functions required by the customer included?
• Realism. Can the requirements be implemented given available budget/technology
• Verifiability. Can the requirements be checked?
Requirements validation

• Requirements reviews
  • Systematic manual analysis of requirements.
• Prototyping
  • Using an executable model of the system to check requirements.
• Test-case generation
  • Developing tests for requirements to check testability.
Review checks

• Verifiability
  • Is the requirement realistically testable?

• Comprehensibility
  • Is the requirement properly understood?

• Traceability
  • Is the origin of the requirement clearly stated?

• Adaptability
  • Can the requirement be changed without a large impact on other requirements?
Summary

- What software requirements are
- How to write requirements
- Good practices
- Elicitation
- Validation