Technical Requirements (SRS)

Quality Requirements

Requirements Documentation

- Is a detailed requirements specification necessary?
- How do we know what “correct” means?
  - How do we decide exactly what capabilities code should provide?
  - How do we know which test cases to write and how to interpret the results?
  - How do we know when we are done implementing?
  - How do we know if we’ve built what the customer asked for (may be distinct from “want” or “need”)?
  - Etc...
- Correctness is a relation between a spec and an implementation (M. Young)
- Implication: until you have a spec, you have no standard for “correctness”

Technical Requirements

- Focuses on developing a rigorous specification
  - Should be straight-forward to determine acceptable inputs and outputs
  - Preferably, can systematically check completeness consistency
- Use cases are not sufficient
- Generally accomplished by modeling required behavior
  - Formal model: models based on formal languages
  - Partial and semi-formal models
Formal Models

- Requirements modeling methods based on formal languages, e.g.:
  - SCR: finite state machines
  - Z: formal logic
  - Statecharts: concurrent automata
- Advantages: allows users to:
  - Derive the set of acceptable outputs for given inputs
  - Prove properties like consistency, completeness, safety, liveness
- Disadvantages:
  - Requires rare skills
  - Expensive to produce and change
- Used seldom except where mission/safety critical (e.g., Intel fab after $475M FDIV error)

Semi-formal Modeling

- Many semi-formal methods used:
  - Structured but non-mathematical models
  - Formal but partial models
- E.g. UML models add some rigor to Use Cases:
  - Activity diagrams
  - Sequence diagrams
  - Disadvantage: tends to model design and implementation
- Modeling critical parts of the requirements:
  - Use predicates (i.e., basic Boolean expressions)
  - Use mathematical expressions
  - Use tables
- A little rigor in the right places can help a lot:
  - Adding formality is not an all-or-none decision
  - Use it where it matters most to start
  - Often easier, less time consuming than trying to say the same thing in prose

Example state transition diagram

Does the Address Book have stateful behavior?
What are the states? Transitions?
### Formal Specification Example

#### Type Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Unit</th>
<th>Legal Values</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>integer</td>
<td></td>
<td>0.0 to 100</td>
<td>Speed measured in miles per hour</td>
</tr>
<tr>
<td>Weight</td>
<td>integer</td>
<td></td>
<td>0.0 to 100</td>
<td>Weighting for the body weight</td>
</tr>
<tr>
<td>Time</td>
<td>integer</td>
<td></td>
<td>0 to 60</td>
<td>Timed in seconds</td>
</tr>
</tbody>
</table>

#### Monitored Variable Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Initial Value</th>
<th>Accuracy</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowHighWindSpeed</td>
<td>integer</td>
<td>0</td>
<td>1</td>
<td>Wind speed reported by first high resolution sensor</td>
</tr>
<tr>
<td>HighLowWindSpeed</td>
<td>integer</td>
<td>1</td>
<td>1</td>
<td>Wind speed reported by second high resolution sensor</td>
</tr>
</tbody>
</table>

#### Controlled Variable Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Initial Value</th>
<th>Accuracy</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransmittedWindSpeed</td>
<td>integer</td>
<td></td>
<td></td>
<td>Transmitted value of wind speed</td>
</tr>
</tbody>
</table>

- SCR formal model
  - Define explicit types
  - Variables monitored or controlled

- Inputs and outputs
  - Be explicit about value types and ranges for each input variable (e.g., Name, Zip, phone)
    - How many digits? Other characters?
  - Be explicit about acceptable outputs
    - Export values and formats
    - Values output to printer (i.e., how is the output a function of the stored values?)
    - Easiest to define the inputs and outputs as abstract variables

- Detailed behavioral requirements
  - Specify acceptable results for a sort
  - Specify acceptable search results
  - Specify state changes (if applicable)

### For Your Projects

- Inputs and outputs
  - Be explicit about value types and ranges for each input variable (e.g., Name, Zip, phone)
    - How many digits? Other characters?
  - Be explicit about acceptable outputs
    - Export values and formats
    - Values output to printer (i.e., how is the output a function of the stored values?)
    - Easiest to define the inputs and outputs as abstract variables

- Detailed behavioral requirements
  - Specify acceptable results for a sort
  - Specify acceptable search results
  - Specify state changes (if applicable)

### Quality Requirements
Terminology

- Avoid "functional" and non-functional" classification
- Behavioral Requirements – any information necessary to determine if the run-time behavior of a given implementation constitutes an acceptable system
  - All quantitative constraints on the system's run-time behavior
  - Other objective measures (safety, performance, fault-tolerance)
  - In theory all can be validated by observing the running system and measuring the results
- Developmental Quality Requirements - any constraints on the system's static construction
  - Maintainability, reusability, ease of change (mutability)
  - Measures of these qualities are necessarily relative (i.e., in comparison to something else)

Behavioral vs. Developmental

<table>
<thead>
<tr>
<th>Behavioral (observable)</th>
<th>Developmental Qualities</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Performance</td>
<td>• Modifiability (ease of change)</td>
</tr>
<tr>
<td>• Security</td>
<td>• Portability</td>
</tr>
<tr>
<td>• Availability</td>
<td>• Reusability</td>
</tr>
<tr>
<td>• Reliability</td>
<td>• Ease of integration</td>
</tr>
<tr>
<td>• Usability</td>
<td>• Understandability</td>
</tr>
<tr>
<td></td>
<td>• Support concurrent development</td>
</tr>
</tbody>
</table>

Properties resulting from the behavior of components, connectors and interfaces that exist at runtime.

<table>
<thead>
<tr>
<th>Developmental Qualities</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Modifiability (ease of change)</td>
</tr>
<tr>
<td>• Portability</td>
</tr>
<tr>
<td>• Reusability</td>
</tr>
<tr>
<td>• Ease of integration</td>
</tr>
<tr>
<td>• Understandability</td>
</tr>
<tr>
<td>• Support concurrent development</td>
</tr>
</tbody>
</table>

Properties resulting from the structure of components, connectors and interfaces that exist at design time whether or not they have any distinct run-time manifestation.

Specifying Quality Requirements

- Is it important to specify the quality requirements explicitly? Unambiguously?
  - Hint: what role would quality requirements play in customer acceptance?
- Are these kinds of specifications adequate?
  - "The system interface shall be easy to use."
  - "The system shall support the maximum possible number of simultaneous users"
Specifying Quality Requirements

- When using natural language, write objectively verifiable requirements when possible
  - Load handling: “The system will support 15 or more concurrent users while staying within required performance bounds.”
  - Maintainability: “The following kinds of requirements changes will require changes in no more than one module of the system...”
  - Performance:
    - “System output X has a deadline of 5 ms from the triggering input event.”
    - “System output Y must be updated at a frequency of no less than 20 ms.”
- Provides unambiguous requirement even if it is not practical to test for compliance

Example Timing Requirements

5.2. TIMING REQUIREMENTS FOR DEMAND FUNCTIONS

<table>
<thead>
<tr>
<th>Function name</th>
<th>Minimum delay to completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Headlight on/off</td>
<td>500 ms</td>
</tr>
<tr>
<td>Brake light on/off</td>
<td>1000 ms</td>
</tr>
<tr>
<td>Windshield wiper on/off</td>
<td>2000 ms</td>
</tr>
<tr>
<td>Change mode select</td>
<td>2000 ms</td>
</tr>
<tr>
<td>Speed limit exceed</td>
<td>1500 ms</td>
</tr>
<tr>
<td>System output X</td>
<td>2000 ms</td>
</tr>
<tr>
<td>System output Y</td>
<td>5000 ms</td>
</tr>
<tr>
<td>System output Z</td>
<td>3000 ms</td>
</tr>
</tbody>
</table>

Summary

- Requirements characterize “correct” system behavior
- Being in control of development requires:
  - Getting the right requirements
  - Communicating them to the stakeholders
  - Using them to guide development
  - Using them to check the quality of the implemented system
Questions?