Technical Requirements (SRS)

Quality Requirements

Review: Use Cases

- Natural language narrative describing how a user interacts with the system to accomplish a specific task
- Defines a *subset of functional requirements*
  - Captures requirements visible to the user
  - Focuses on most important user tasks
- Missing specifications of quality requirements (performance, security, etc.)
Requirements Documentation

• Is a more formal, 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

<table>
<thead>
<tr>
<th>Type Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Speed</td>
</tr>
<tr>
<td>Weight</td>
</tr>
<tr>
<td>time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monitored Variable Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>LowResWS1</td>
</tr>
<tr>
<td>LowResWS2</td>
</tr>
<tr>
<td>HighResWS1</td>
</tr>
<tr>
<td>HighResWS2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controlled Variable Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>TransmitWindSpeed</td>
</tr>
</tbody>
</table>

- SCR formal model
  - Define explicit types
  - Variables monitored or controlled
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 displayed or printed
  - 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 run time.

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 up to 100 concurrent users while maintaining a response time under 15 ms.”
  – 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

For all the demand functions, the rate of demand is so low that it will not constitute a significant CPU-load.

For the stated entries, the desired maximum delay is not known; the entry is the maximum delay in the current OFF, which we will use as an approximation. In one case, both the current and desired values are given. The current value would be good enough to satisfy requirements, but the desired rate would be preferred.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Maximum Delay to Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMs</td>
<td>*200 ms</td>
</tr>
<tr>
<td>Switch AUTOCAL light on/off</td>
<td>*200 ms</td>
</tr>
<tr>
<td>Switch computer control on/off</td>
<td>*200 ms</td>
</tr>
<tr>
<td>Issue computer failure</td>
<td>not significant</td>
</tr>
<tr>
<td>Change scale factor</td>
<td>*200 ms</td>
</tr>
<tr>
<td>Switch X slew on/off</td>
<td>*200 ms</td>
</tr>
<tr>
<td>Switch Y slew on/off</td>
<td>*200 ms</td>
</tr>
<tr>
<td>Switch Z slew on/off</td>
<td>*200 ms</td>
</tr>
<tr>
<td>Change latitude-greater-than-70-degrees</td>
<td>*200 ms</td>
</tr>
<tr>
<td>Switch IMA light on/off</td>
<td>*200 ms</td>
</tr>
<tr>
<td>FLR:</td>
<td></td>
</tr>
<tr>
<td>Enable radar cursor</td>
<td>200 ms</td>
</tr>
<tr>
<td>Slave or release slave</td>
<td>40 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?

Midway Review: Expected Progress

- Look for progress against plan (limited evaluation of quality)
- Home page: name, logo, directory
- Plan and schedule:
  - Up-to-date plan: risks, process, etc.
  - Schedule tracking progress
  - Meeting notes and Developer Logs
- Requirements
  - Use cases to current iteration or beyond
  - Some detailed requirements
- Design: initial design (up to iteration)
- QA Plan: planned reviews and test cases
Grading Rubric

• For final deliverables: evaluate extent team demonstrates control?
• Managerial
  – How well does the team establish then follow a plan?
  – Does the team effectively use resources to 1) meet deadlines or 2) re-plan when needed?
• Intellectual
  – Does the team effectively establish what they intend to build?
    • To what extent is it consistent with customer?
  – Does the team build to the specifications?
  – How well does the team demonstrate correctness?
• Individual: to what extent did you contribute?