Requirements Phase Goals

• What does “getting the requirements right” mean in the systems development context?
• Only three goals
  1. Understand precisely what is required of the software
  2. Communicate that understanding to all of the parties involved in the development (stakeholders)
  3. Control production to ensure the final system satisfies the requirements
• Sounds easy but hard to do in practice
• Understanding what makes these goals difficult to accomplish helps us understand how to mitigate the risks

A Requirements Process Framework

• Requirements Understanding
  – Requirements Elicitation - establish “what people want”
  – Requirements Negotiation - resolve stakeholder conflicts
• Requirements Specification
  – Concept of Operations - communicate with non-programming audiences
  – Software Requirements Specification - specify precisely what the software must do
• Requirements Validation and Verification
  – Establish that we have the right requirements (feedback)
  – Ensure our specification is good quality
Technical Specification

The SRS
The role of rigorous specification

Requirements Documentation

• Is a detailed requirements specification necessary?
• How do we know what “correct” means?
  – How do we decide exactly what capabilities the modules 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

• Focus on developing a technical specification
  – Should be straightforward to determine acceptable inputs and outputs
  – Preferably, can systematically check completeness consistency
• 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 (critical parts, potentially ambiguous parts)
  – Often easier, less time consuming than trying to say the same thing in prose
• E.g. in describing conditions or cases
  – Use predicates (i.e., basic Boolean expressions)
  – Use mathematical expressions
  – Use tables where possible
Example state transition diagram

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>Mentioned Variable Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>LowResWS Speed</td>
</tr>
<tr>
<td>LowResWS Speed</td>
</tr>
<tr>
<td>HighResWS Speed</td>
</tr>
<tr>
<td>HighResWS Speed</td>
</tr>
</tbody>
</table>

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

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

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 Attributes - any constraints on the system's static construction
  - Maintainability, reusability, ease of change (mutability)
  - Measures of these qualities are necessarily relativistic (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 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 input event."
    - "System output Y must be updated at a frequency of no less than 20 ms."

Example Timing Requirements

### 5.1. TIMING REQUIREMENTS FOR DEMAND FUNCTIONS

- For all the demand functions, the due date is a minimum duration to avoid a significant GIC (long)

  For the current system, the desired minimum delay is an interval, the delay to the minimum delay at the current 15, which we will assume as an approximation. If no one, load the system and observe delays, the observed delay would be the delay interval. Therefore, we cannot accurately determine delay in steady state. But, the duration is very small.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Maximum delay in completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTRC:</td>
<td></td>
</tr>
<tr>
<td>switchAIT, turn off</td>
<td>420 ms</td>
</tr>
<tr>
<td>switchAIT switch on</td>
<td>2100 ms</td>
</tr>
<tr>
<td>switchAIT, turn on</td>
<td>2100 ms</td>
</tr>
<tr>
<td>change side-face</td>
<td>2000 ms</td>
</tr>
<tr>
<td>switch X: turn off</td>
<td>2100 ms</td>
</tr>
<tr>
<td>switch X, turn on</td>
<td>2100 ms</td>
</tr>
<tr>
<td>switch Y: turn on</td>
<td>2100 ms</td>
</tr>
<tr>
<td>switch • clearing on</td>
<td>2100 ms</td>
</tr>
<tr>
<td>switch • clearing off</td>
<td>2100 ms</td>
</tr>
<tr>
<td>switch • change on, off</td>
<td>2100 ms</td>
</tr>
<tr>
<td>switch • change off, on</td>
<td>2100 ms</td>
</tr>
<tr>
<td>switch • change off, on, off</td>
<td>2100 ms</td>
</tr>
</tbody>
</table>

### FLR:
- Rail edge curve
- More or release check

- 40 ms

Requirements Validation and Verification

- **Feedback-control for requirements**
- **Should answer two distinct questions:**
  - **Validation:** "Are we building to the right requirements?"
  - **Verification:** "Are we building what we specified?"
- **Validation requires going back to the stakeholders:**
  - Review of specifications
  - Prototyping
  - Story-boarding
  - Use case walkthroughs
  - Review software iterations
- **Verification requires checking work products against specifications**
  - Review
  - Testing
  - Formal modeling and analysis
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
• Requirements activities must be incorporated in the project plan
  – Requirements baseline
  – Requirements change management

Questions?