Real-Time & Embedded Systems

Agenda

- Requirements Lecture (subset of Ch5)
Requirements engineering process

Adapted from Software Engineering, Sommerville, 2010

© Copyright 2011 Dr. Phillip A. Laplante
Why are Requirements important?

- Verification of meeting customer desires/specification
- Allows design to be traceable (end-to-end)
- Allows transparent testing (i.e. – everyone knows what it should do)
- During engineering, it’s easier to understand the why if requirements are written, and well written
- Allows easier interfacing when outputs/inputs are better documented
Types of requirements

- Functional
- External Interfaces
- Performance
- Logical database
- Design constraints
- Software system attributes
Functional requirements

- All system inputs
- Exact sequence of operations and responses (outputs) to normal and abnormal situations for every input possibility
- May use case-by-case description or other general form of description (e.g. using universal quantification, use cases, user stories)
External interface requirements

- name of item
- description of purpose
- source of input or destination of output
- valid range, accuracy, and/or tolerance
- units of measure
- timing
- relationships to other inputs/outputs
- screen formats/organization
- window formats/organization
- data formats
- command formats
Performance requirements

- Static and dynamic requirements placed on the software or on human interaction with the software as a whole.

- Might include:
  - the number of simultaneous users to be supported
  - the numbers of transactions and tasks the amount of data to be processed within certain time periods for both normal and peak workload conditions
Logical database requirements

- Types of information used by various functions such as:
  - Frequency of use
  - Accessing capabilities
  - Data entities and their relationships
  - Integrity constraints
  - Data retention requirements
Design constraint requirements

- Related to:
  - Standards compliance
  - Hardware limitations (recall, “where do the deadlines come from?”)
Software system attribute requirements

- Reliability
- Availability
- Security
- Maintainability
- Portability
- Just about any other “ility” you can think of
Why real-time specifications are different

- Involve more than one process and often more than one processor and involve specification of timing relationships between processes and processors.
- The time selectivity (the “age” of the data) needs to be explicitly defined, and
- Start and stop times for process activations must be known and or controlled.
Requirements specification for real-time systems

- There is no one way to do it. Various ways include:
  - Top-down process decomposition or structured analysis
  - Object-oriented approaches
  - Program description languages (PDL) or pseudo-code
  - High-level functional specifications that are not further decomposed
  - Ad hoc techniques, including simply natural language and mathematical description, and are always included in virtually every system specification
Organizing the requirements document

- IEEE 830
- Guidelines for writing requirements
- Text structure
- Requirements triage
IEEE 830-1998

- IEEE Standard 830-1998 is the IEEE's Recommended Practice for Software Requirements Specifications (SRS), and provides a template of what an SRS should look like.

- IEEE 830 suggests that requirements documents should be:
  - Correct
  - Unambiguous
  - Complete
  - Consistent
  - Ranked for importance and/or stability
  - Verifiable
  - Modifiable
  - Traceable
IEEE 830

- Requirements can be organized by:
  - Functional mode (e.g. nav, attack, diag)
  - User class (e.g. user, supervisor, diag)
  - Object (define classes/objects, attributes, functions/methods, messages)
  - Feature (what the system provides to the user)
  - Stimulus (e.g. sensor 1, 2,...)
  - Functional hierarchy (e.g. using SA)
  - Mixed (combining two or more of the above)
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>1.1 Purpose</td>
</tr>
<tr>
<td></td>
<td>1.2 Scope</td>
</tr>
<tr>
<td></td>
<td>1.3 Definitions and Acronyms</td>
</tr>
<tr>
<td></td>
<td>1.4 References</td>
</tr>
<tr>
<td></td>
<td>1.5 Overview</td>
</tr>
<tr>
<td>2. Overall Description</td>
<td>2.1 Product Perspective</td>
</tr>
<tr>
<td></td>
<td>2.2 Product Functions</td>
</tr>
<tr>
<td></td>
<td>2.3 User Characteristics</td>
</tr>
<tr>
<td></td>
<td>2.4 Constraints</td>
</tr>
<tr>
<td></td>
<td>2.5 Assumptions and Dependencies</td>
</tr>
<tr>
<td>3. Specific Requirements</td>
<td>Appendix</td>
</tr>
<tr>
<td></td>
<td>Index</td>
</tr>
</tbody>
</table>
Validation involves checking the following:

- Validity, that is does the system provide the functions which best support the customer’s needs?
- Consistency, that is, are there any requirements conflicts?
- Completeness, in other words, are all functions required by the customer included?
- Realism, or, can the requirements be implemented given available budget and technology?
- Verifiability: can the requirements be checked?
Requirements validation and review

- Checking for conformance to the IEEE 830 best practice:
  - Requirements reviews
  - Systematic manual analysis of the requirements
  - Prototyping
  - Using an executable model of the system to check requirements.
  - Test-case generation
  - Developing tests for requirements to check testability
  - Automated consistency analysis
  - Checking the consistency of a structured requirements description
Recommendations and Best Practices

- Use consistent modeling approaches and techniques throughout the specification.
- Use op-down decomposition, Structured Design or Object-Orientation.
- Separate operational specification from descriptive:
  - belongs in a software design document, not a requirements specification.
Recommendations and Best Practices

• Use consistent levels of abstraction within models and conformance between levels of refinement across models.

• Model non-functional requirements as a part of the specification models – in particular timing properties.

• Avoid hardware and software assignment in the specification (this is an aspect of design rather than specification).