Lecture 8: Specification and Validation

Last Week:
Modeling and Analysis (III)
Non-functional Requirements
Measuring Software Quality

This Week:
Communicating Requirements
the Software Requirements Specification (SRS)
Validation: Reviews, Inspections, etc
Requirements Prioritization

Next Week:
Evolving Requirements
Change management
Inconsistency management
Feature Interaction
Product Families

Software Requirements Specification

→ How do we communicate the Requirements to others?

- It is common practice to capture them in an SRS
  - But an SRS doesn't need to be a single paper document...

→ Purpose

- Communicates an understanding of the requirements
  - explains both the application domain and the system to be developed
- Contractual
  - May be legally binding!
  - Expresses an agreement and a commitment
- Baseline for evaluating subsequent products
  - supports system testing, verification and validation activities
  - should contain enough information to verify whether the delivered system meets requirements
- Baseline for change control
  - requirements change, software evolves

→ Audience

- Users, Purchasers
  - Most interested in system requirements
  - Not generally interested in detailed software requirements
- Systems Analysts, Requirements Analysts
  - Write various specifications that inter-relate
- Developers, Programmers
  - Have to implement the requirements
- Testers
  - Determine that the requirements have been met
- Project Managers
  - Measure and control the analysis and development processes

SRS Contents

Source: *Adapted from IEEE-STD-830*

→ Software Requirements Specification should address:

- Functionality. What is the software supposed to do?
- External interfaces. How does the software interact with people, the system's hardware, other hardware, and other software?
- Performance. What is the speed, availability, response time, recovery time of various software functions, and so on?
- Attributes. What are the portability, correctness, maintainability, security, and other considerations?
- Design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) and so on?

→ Some other topics should be excluded:

- ... should avoid placing either design or project requirements in the SRS
- ... should not describe any design or implementation details. These should be described in the design stage of the project.
- ... should address the software product, not the process of producing the software product.

Appropriate Specification

Source: *Adapted from Blum 1992, p154-5*

→ Consider two different projects:

A) Small project, 1 programmer, 6 months work
programmer talks to customer, then writes up a 5-page memo

B) Large project, 50 programmers, 2 years work
team of analysts model the requirements, then document them in a 500-page SRS

	Project A	*Project B*
Purpose of spec?	Crystalizes programmer's understanding; feedback to customer	Build-to document; must contain enough detail for all the programmers
Management view?	Spec is irrelevant; have already allocated resources	Will use the spec to estimate resource needs and plan the development
Readers?	**Primary**: Spec author; **Secondary**: Customer	**Primary**: all programmers + V&V team, managers; **Secondary**: customers

A complication: Procurement

- An 'SRS' may be written by…
  - …the procurer:
    - so the SRS is really a call for proposals
    - Must be general enough to yield a good selection of bids…
    - …and specific enough to exclude unreasonable bids
  - …the bidders:
    - Represents a proposal to implement a system to meet the CfP
    - must be specific enough to demonstrate feasibility and technical competence
    - …and general enough to avoid over-commitment
  - …the selected developer:
    - reflects the developer's understanding of the customers needs
    - forms the basis for evaluation of contractual performance
  - …or by an independent RE contractor!
- Choice over what point to compete the contract
  - Early (conceptual stage)
    - can only evaluate bids on apparent competence & ability
  - Late (detailed specification stage)
    - more work for procurer; appropriate RE expertise may not be available in-house
  - IEEE Standard recommends SRS jointly developed by procurer & developer

Desiderata for Specifications

Source: *Adapted from IEEE-STD-830-1998*

- Valid (or "correct")
  - Expresses only the real needs of the stakeholders (customers, users,…)
  - Doesn't contain anything that isn't "required"
- Unambiguous
  - Every statement can be read in exactly one way
- Complete
  - Specifies all the things the system must do
  - …and all the things it must not do!
  - Conceptual Completeness
    - E.g. responses to all classes of input
  - Structural Completeness
    - E.g. no TBDs!!!
- Understandable (Clear)
  - E.g. by non-computer specialists
- Consistent
  - Doesn't contradict itself
    - I.e. is satisfiable
  - Uses all terms consistently
- Ranked
  - Must indicate the importance and/or stability of each requirement
- Verifiable
  - A process exists to test satisfaction of each requirement
  - *"every requirement is specified behaviorally"*
- Modifiable
  - Can be changed without difficulty
    - Good structure and cross-referencing
- Traceable
  - Origin of each requirement must be clear
  - Facilitates referencing of requirements in future documentation

Typical mistakes

- Noise
  - the presence of text that carries no relevant information to any feature of the problem.
- Silence
  - a feature that is not covered by any text.
- Over-specification
  - text that describes a feature of the solution, rather than the problem.
- Contradiction
  - text that defines a single feature in a number of incompatible ways.
- Ambiguity
  - text that can be interpreted in at least two different ways.
- Forward reference
  - text that refers to a feature yet to be defined.
- Wishful thinking
  - text that defines a feature that cannot possibly be validated.
- Jigsaw puzzles
  - e.g. distributing requirements across a document and then cross-referencing
- Duckspeak requirements
  - Requirements that are only there to conform to standards
- Unnecessary invention of terminology
  - E.g., 'the user input presentation function', 'airplane reservation data validation function'
- Inconsistent terminology
  - Inventing and then changing terminology
- Putting the onus on the development staff
  - i.e. making the reader work hard to decipher the intent
- Writing for the hostile reader
  - There are fewer of these than friendly readers

Source: *Adapted from Kovitz, 1999*

Ambiguity Test

- Natural Language?
  - "The system shall report to the operator all faults that originate in critical functions or that occur during execution of a critical sequence and for which there is no fault recovery response."
    (adapted from the specifications for the international space station)
- Or a decision table?

Originate in critical functions	F	T	F	T	F	T	F	T
Occur during critical seqeunce	F	F	T	T	F	F	T	T
No fault recovery response	F	F	F	F	T	T	T	T
Report to operator?								

Source: *Adapted from Easterbrook & Callahan, 1997.*

Avoiding ambiguity

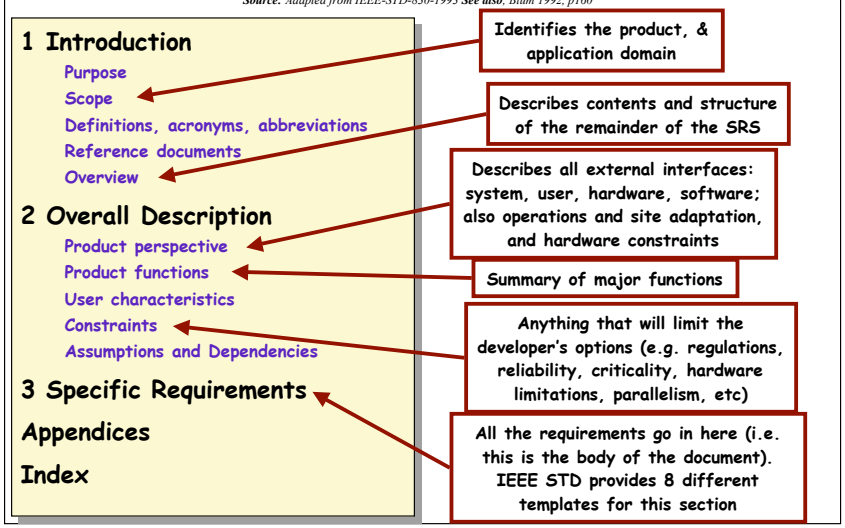
- → Review natural language specs for ambiguity
  - use people with different backgrounds
  - include software people, domain specialists and user communities
  - Must be an independent review (I.e. not by the authors!)
- → Use a specification language
  - E.g. a restricted subset or stylized English
  - E.g. a semi-formal notation (graphical, tabular, etc)
  - E.g. a formal specification language (e.g. Z, VDM, SCR, …)
- → Exploit redundancy
  - Restate a requirement to help the reader confirm her understanding
  - …but clearly indicate the redundancy
  - May want to use a more formal notation for the re-statement

Organizing the Requirements

- → Need a logical organization for the document
  - IEEE standard offers different templates
- → Example Structures - organize by…
  - …External stimulus or external situation
    - e.g., for an aircraft landing system, each different type of landing situation: wind gusts, no fuel, short runway, etc
  - …System feature
    - e.g., for a telephone system: call forwarding, call blocking, conference call, etc
  - …System response
    - e.g., for a payroll system: generate pay-cheques, report costs, print tax info;
  - …External object
    - e.g. for a library information system, organize by book type
  - …User type
    - e.g. for a project support system: manager, technical staff, administrator, etc.
  - …Mode
    - e.g. for word processor: page layout mode, outline mode, text editing mode, etc
  - …Subsystem
    - e.g. for spacecraft: command&control, data handling, comms, instruments, etc.

IEEE Standard for SRS

*Source: Adapted from IEEE-STD-830-1993 **See also**, Blum 1992, p160*



IEEE STD Section 3 (example)

*Source: Adapted from IEEE-STD-830-1993. **See also**, Blum 1992, p160*

3.1 External Interface Requirements
- 3.1.1 User Interfaces
- 3.1.2 Hardware Interfaces
- 3.1.3 Software Interfaces
- 3.1.4 Communication Interfaces

3.2 Functional Requirements
- *this section organized by mode, user class, feature, etc. For example:*
- 3.2.1 Mode 1
  - *3.2.1.1 Functional Requirement 1.1*
  - …
- 3.2.2 Mode 2
  - *3.2.1.1 Functional Requirement 1.1*
  - …
- …
- 3.2.2 Mode n
  - …

3.3 Performance Requirements
- *Remember to state this in measurable terms!*

3.4 Design Constraints
- *3.4.1 Standards compliance*
- *3.4.2 Hardware limitations*
- *etc.*

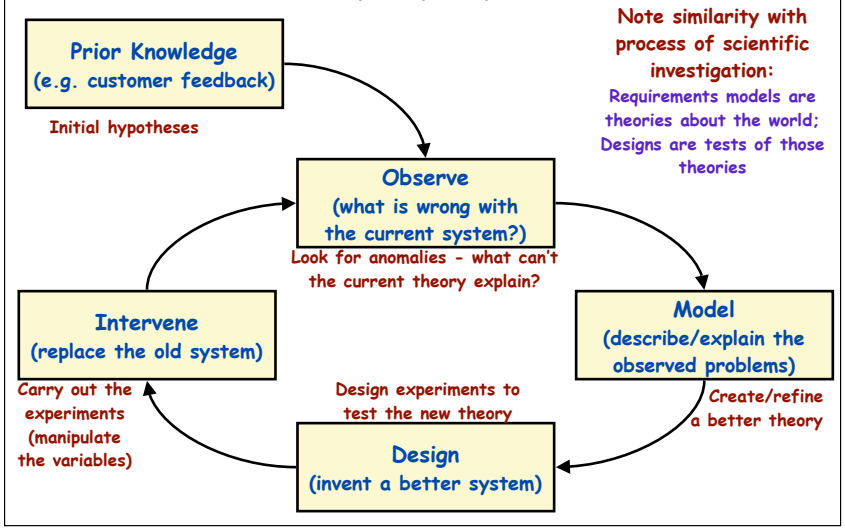
3.5 Software System Attributes
- 3.5.1 Reliability
- 3.5.2 Availability
- 3.5.3 Security
- 3.5.4 Maintainability
- 3.5.5 Portability

3.6 Other Requirements

Agreeing on a specification

- Two key problems for getting agreement:
  - 1) the problem of validation
    - Like validating scientific theories
    - If we build to this spec, will the customer's expectations be met?
  - 2) the problem of negotiation
    - How do you reconcile conflicting goals in a complex socio-cognitive setting?
- Validating Requirements
  - Inspections and Reviews
  - Prototyping
- Negotiating Requirements
  - Requirements Prioritization
  - Conflict and Conflict Resolution
  - Requirements Negotiation Techniques

Inquiry Cycle



The problem of validation

- logical positivist view:
  - "there is an objective world that can be modeled by building a consistent body of knowledge grounded in empirical observation"
  - In RE, assumes there is an objective problem that exists in the world
    - Build a consistent model; make sufficient empirical observations to check validity
    - Use tools that test consistency and completeness of the model
    - Use reviews, prototyping, etc to demonstrate the model is "valid"
- Popper's modification to logical positivism:
  - "theories can't be proven correct, they can only be refuted by finding exceptions"
  - In RE, design your requirements models to be refutable
    - Look for evidence that the model is wrong
    - E.g. collect scenarios and check the model supports them
- post-modernist view:
  - "there is no privileged viewpoint; all observation is value-laden; scientific investigation is culturally embedded"
  - E.g. Kuhn: science moves through paradigms
  - E.g. Toulmin: scientific theories are judged with respect to a *weltanschauung*
  - In RE, validation is always subjective and contextualised
    - Use stakeholder involvement so that they 'own' the requirements models
    - Use ethnographic techniques to understand the weltanschauungen

Prototyping

- Definitions
  - "A software prototype is a partial implementation constructed primarily to enable customers, users, or developers to learn more about a problem or its solution." [Davis 1990]
  - "Prototyping is the process of building a working model of the system" [Agresti 1986]
- Approaches to prototyping
  - Presentation Prototypes
    - explain, demonstrate and inform - then throw away
    - e.g. used for proof of concept; explaining design features; etc.
  - Exploratory Prototypes
    - used to determine problems, elicit needs, clarify goals, compare design options
    - informal, unstructured and thrown away.
  - Breadboards or Experimental Prototypes
    - explore technical feasibility; test suitability of a technology
    - Typically no user/customer involvement
  - Evolutionary (e.g. "operational prototypes", "pilot systems"):
    - development seen as continuous process of adapting the system
    - "prototype" is an early deliverable, to be continually improved.

Throwaway or Evolve?

→ Throwaway Prototyping
- Purpose:
  - to learn more about the problem or its solution…
  - hence discard after the desired knowledge is gained.
- Use:
  - early or late
- Approach:
  - horizontal - build only one layer (e.g. UI)
  - "quick and dirty"
- Advantages:
  - Learning medium for better convergence
  - Early delivery → early testing → less cost
  - Successful even if it fails!
- Disadvantages:
  - Wasted effort if requirements change rapidly
  - Often replaces proper documentation of the requirements
  - May set customers' expectations too high
  - Can get developed into final product

→ Evolutionary Prototyping
- Purpose
  - to learn more about the problem or its solution…
  - …and to reduce risk by building parts of the system early
- Use:
  - incremental; evolutionary
- Approach:
  - vertical - partial implementation of all layers;
  - designed to be extended/adapted
- Advantages:
  - Requirements not frozen
  - Return to last increment if error is found
  - Flexible(?)
- Disadvantages:
  - Can end up with complex, unstructured system which is hard to maintain
  - early architectural choice may be poor
  - Optimal solutions not guaranteed
  - Lacks control and direction

Brooks: "Plan to throw one away - you will anyway!"

Reviews, Inspections, Walkthroughs…

Source: Adapted from Blum, 1992, pp369-373

→ Note: these terms are not widely agreed
- formality
  - informal: from meetings over coffee, to team get-togethers
  - formal: scheduled meetings, prepared participants, defined agenda, specific format, documented output
- "Management reviews"
  - E.g. preliminary design review (PDR), critical design review (CDR), …
  - Used to provide confidence that the design is sound
  - Attended by management and sponsors (customers)
  - Usually a "dog-and-pony show"
- "Walkthroughs"
  - developer technique (usually informal)
  - used by development teams to improve quality of product
  - focus is on finding defects
- "(Fagan) Inspections"
  - a process management tool (always formal)
  - used to improve quality of the development process
  - collect defect data to analyze the quality of the process
  - written output is important
  - major role in training junior staff and transferring expertise

Benefits of formal inspection

Source: Adapted from Blum, 1992, pp369-373 & Freedman and Weinberg, 1990.

→ Formal inspection works well for programming:
- For applications programming:
  - more effective than testing
  - most reviewed programs run correctly first time
  - compare: 10-50 attempts for test/debug approach
- Data from large projects
  - error reduction by a factor of 5; (10 in some reported cases)
  - improvement in productivity: 14% to 25%
  - percentage of errors found by inspection: 58% to 82%
  - cost reduction of 50%-80% for V&V (even including cost of inspection)
- Effects on staff competence:
  - increased morale, reduced turnover
  - better estimation and scheduling (more knowledge about defect profiles)
  - better management recognition of staff ability

→ These benefits also apply to requirements inspections
- E.g. See studies by Porter et. al.; Regnell et. al.;…

Inspection Constraints

Source: Adapted from Blum, 1992, pp369-373 & Freedman and Weinberg, 1990.

→ Size
- "enough people so that all the relevant expertise is available"
- min: 3 (4 if author is present)
- max: 7 (less if leader is inexperienced)

→ Duration
- never more than 2 hours
  - concentration will flag if longer

→ Outputs
- all reviewers must agree on the result
  - accept; re-work; re-inspect;
- all findings should be documented
  - summary report (for management)
  - detailed list of issues

→ Scope
- focus on small part of a design, not the whole thing

→ Timing
- Examines a product once its author has finished it
- not too soon
  - product not ready - find problems the author is already aware of
- not too late
  - product in use - errors are now very costly to fix

→ Purpose
- Remember the biggest gains come from fixing the process
  - collect data to help you not to make the same errors next time

Inspection Guidelines

Source: Adapted from Freedman and Weinberg, 1990.

→ Prior to the review

- schedule Formal Reviews into the project planning
- train all reviewers
- ensure all attendees prepare in advance

→ During the review

- review the product, not its author
  - keep comments constructive, professional and task-focussed
- stick to the agenda
  - leader must prevent drift
- limit debate and rebuttal
  - record issues for later discussion/resolution
- identify problems but don't try to solve them
- take written notes

→ After the review

- review the review process

Choosing Reviewers

Source: Adapted from Freedman and Weinberg, 1990.

→ Possibilities

- specialists in reviewing (e.g. QA people)
- people from the same team as the author
- people invited for specialist expertise
- people with an interest in the product
- visitors who have something to contribute
- people from other parts of the organization

→ Exclude

- anyone responsible for reviewing the author
  - i.e. line manager, appraiser, etc.
- anyone with known personality clashes with other reviewers
- anyone who is not qualified to contribute
- all management
- anyone whose presence creates a conflict of interest

Structuring the inspection

Source: Adapted from Porter, Votta and Basili, 1995

→ Can structure the review in different ways

- Ad-hoc
  - Rely on expertise of the reviewers
- Checklist
  - uses a checklist of questions/issues
  - checklists tailored to the kind of document (Porter et. al. have examples)
- active reviews (perspective based reading)
  - each reviewer reads for a specific purpose, using specialized questionnaires
  - effectively different reviewers take different perspectives

→ The differences may matter

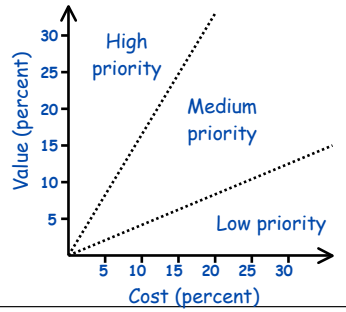
- E.g. Porter et. al. study indicates that:
  - active reviews find more faults than ad hoc or checklist methods
  - no effective different between ad hoc and checklist methods
  - the inspection meeting might be superfluous!

Requirements Prioritization

Source: Adapted from Karlsson & Ryan 1997

→ Usually there are too many requirements

- Decide which to include in the first release
  - Balancing quality, cost and time-to-market
- Assess each requirement's importance to the project as a whole
- Assess the relative cost of each requirement
- Compute the cost-value trade-off:



Analytic Hierarchy Process (AHP)

Source: Adapted from Karlsson & Ryan 1997

- Create n x n matrix (for n requirements)
- Compare each pair of requirements
  - For element (x,y) in the matrix enter:
    - 1 - if x and y are of equal value
    - 3 - if x is slightly more preferred than y
    - 5 - if x is strongly more preferred than y
    - 7 - if x is very strongly more preferred than y
    - 9 - if x is extremely more preferred than y
  - ...and for (y,x) enter the reciprocal.
- Estimate the eigenvalues:
  - E.g. "averaging over normalized columns"
    - Calculate the sum of each column
    - Divide each element in the matrix by the sum of it's column
    - Calculate the sum of each row
    - Divide each row sum by the number of rows
- This gives a value for each reqt:
  - ...based on estimated percentage of total value of the project

AHP example

Source: Adapted from Karlsson & Ryan 1997

	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

Normalise columns

	Req1	Req2	Req3	Req4
Req1	0.21	0.18	0.18	0.48
Req2	0.63	0.54	0.45	0.36
Req3	0.11	0.11	0.09	0.04
Req4	0.05	0.18	0.27	0.12

Sum the rows

sum	sum/4
1.05	0.26
1.98	0.50
0.34	0.09
0.62	0.16

...Also: should compute the consistency index (because the pairwise comparisons may not be consistent)