

University of Toronto Department of Computer Science

## Lecture 8: Communicating Requirements

**Last Week:**  
 Modeling and Analysis (III)  
 Formal Modeling Techniques  
 Program Spec. vs. Reqts Modeling  
 R'SML, SCR, RML, Telos, Albert II  
 Lightweight formal modeling

**This Week:**  
 Communicating Requirements  
 the Software Requirements Specification (SRS)  
 Documentation Standards  
 Requirements Traceability

**Next Week:**  
 Agreeing Requirements (I)  
 Validation  
 Reviews and Inspections  
 Prototyping

© 2000-2003, Steve Easterbrook 1

University of Toronto Department of Computer Science

## 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

© 2000-2003, Steve Easterbrook 2

University of Toronto Department of Computer Science

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

© 2000-2003, Steve Easterbrook 3

University of Toronto Department of Computer Science

## 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

© 2000-2003, Steve Easterbrook 4

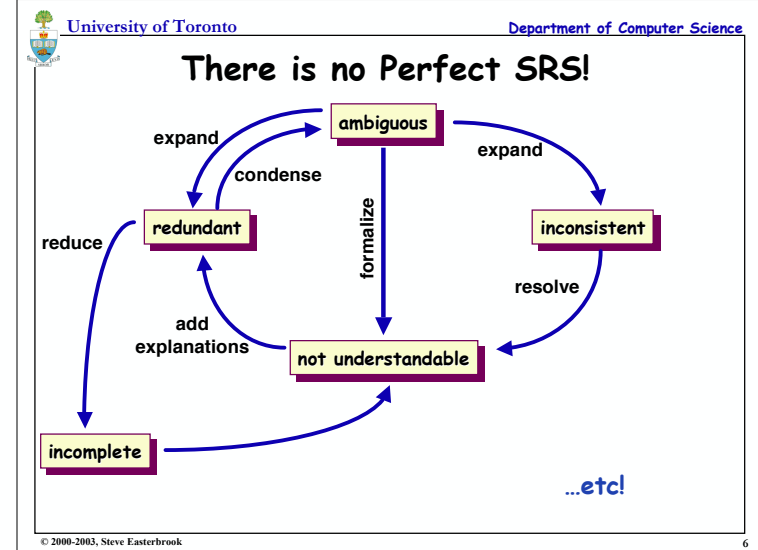
University of Toronto Department of Computer Science

## Desiderata for Specifications

Source: Adapted from Bhui 1992, pp164-5 and the IEEE-STD-830-1993

- **Valid (or "correct")**
  - ↳ Expresses only the real needs of the stakeholders (customers, users,...)
- **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!!!
- **Consistent**
  - ↳ Doesn't contradict itself
    - > I.e. is satisfiable
  - ↳ Uses all terms consistently
  - ↳ **Note: inconsistency can be hard to detect**
    - > especially in timing aspects and condition logic
    - > (Formal specification can help)
- **Necessary**
  - ↳ Doesn't contain anything that isn't "required"
- **Unambiguous**
  - ↳ Every statement can be read in exactly one way
  - ↳ Clearly defines confusing terms
    - > E.g. in a glossary
- **Verifiable**
  - ↳ A process exists to test satisfaction of each requirement
  - ↳ "every requirement is specified behaviorally"
- **Understandable (Clear)**
  - ↳ E.g. by non-computer specialists
- **Modifiable**
  - ↳ Can be changed without difficulty
    - > Good structure and cross-referencing
  - ↳ It must be kept up to date!

© 2000-2003, Steve Easterbrook 5



University of Toronto Department of Computer Science

## 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

© 2000-2003, Steve Easterbrook Source: Adapted from Kovitz, 1999 7

University of Toronto Department of Computer Science

## SRS should not include...

- **Project development plans**
  - > cost, staffing, schedules, methods, tools, etc
  - ↳ Lifetime of SRS is until the software is made obsolete
  - ↳ Lifetime of development plans is much shorter
- **Product assurance plans**
  - > CM, V&V, test, QA, etc
  - ↳ Different audiences
  - ↳ Different lifetimes
- **Designs**
  - ↳ Requirements and designs have different audiences
  - ↳ Analysis and design are different areas of expertise
    - > I.e. requirements analysts shouldn't do design!
  - ↳ **Except where application domain constrains the design**
    - > e.g. limited communication between different subsystems for security reasons.

© 2000-2003, Steve Easterbrook Source: Adapted from Davis, 1990, p183 8

University of Toronto Department of Computer Science

## Text Analysis to measure Quality

- Can do textual analysis of an SRS
  - ↳ measure current practice
  - ↳ establish norms for an organisation
- E.g. NASA SEL used nine quality indicators:
  - ↳ Imperatives
    - > identified by words such as "shall", "must", "is required", etc.
    - > Imperatives measure how explicit a SRS is.
  - ↳ Continuances follow an imperative and introduce requirements
    - > indicated by "below:", "as follows:" etc.
    - > measure the structure of an SRS.
  - ↳ Option
    - > indicated by words such as "can", "may", "optionally" etc.
    - > measure how much latitude does an SRS leave
  - ↳ Weak phrases
    - > cause uncertainty
    - > e.g. "adequate", "as applicable" etc.
  - ↳ Directives
    - > indicated by tables, figures etc
    - > these strengthen the quality of the document
  - ↳ Size
    - > ...in terms of lines of text, indicators and subjects
    - > roughly, the number of subjects for all the imperatives
  - ↳ Text structure
    - > measures the number of statement identifiers
  - ↳ Specification depth
    - > measures how deep are the subsections of the SRS (e.g., 3.2.5.1)
    - > gives an indication of SRS structure.
  - ↳ Readability statistics
    - > e.g average number of syllables per word, number of words per sentence etc.

© 2000-2003, Steve Easterbrook 9

University of Toronto Department of Computer Science

## 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 sequence	F	F	T	T	F	F	T	T
No fault recovery response	F	F	F	F	T	T	T	T
Report to operator?								

© 2000-2003, Steve Easterbrook Source: Adapted from Easterbrook & Callahan, 1997. 10

University of Toronto Department of Computer Science

## 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

© 2000-2003, Steve Easterbrook 11

University of Toronto Department of Computer Science

## SRS format and style

- Modifiability
  - ↳ well-structured, indexed, cross-referenced, etc.
  - ↳ redundancy should be avoided or must be clearly marked as such
  - ↳ An SRS is not modifiable if it is not traceable...
- Traceability
  - ↳ Backwards - the specification must be "traced"
    - > each requirement traces back to a source or authority
    - > e.g. a requirement in the system spec; a stakeholder; etc
  - ↳ Forwards - the specification must be "traceable"
    - > each requirement will eventually trace forwards to parts of the design that satisfy it
    - > Hence we will need a way of referring to each requirement
  - ↳ Note: traceability links are two-way
    - > other documents will be traced into the SRS
    - > Every requirement must have a unique label.
- Useful Annotations
  - ↳ E.g. relative necessity and relative stability

© 2000-2003, Steve Easterbrook Source: Adapted from Davis, 1990, p192-5 12

University of Toronto Department of Computer Science

## 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 Agate case study: campaign management, staff management, etc.

© 2000-2003, Steve Easterbrook 13

University of Toronto Department of Computer Science

## IEEE Standard for SRS

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

The diagram shows the IEEE Standard for SRS structure with callouts explaining each section:

- 1 Introduction**
  - Purpose: Identifies the product, & application domain
  - Scope: Describes contents and structure of the remainder of the SRS
  - Definitions, acronyms, abbreviations: Describes all external interfaces: system, user, hardware, software; also operations and site adaptation, and hardware constraints
  - Reference documents: Summary of major functions
  - Overview: Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)
- 2 Overall Description**
  - Product perspective: All the requirements go in here (i.e. this is the body of the document). IEEE STD provides 8 different templates for this section
  - Product functions: Summary of major functions
  - User characteristics: Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)
  - Constraints: Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)
  - Assumptions and Dependencies: Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)
- 3 Specific Requirements**
- Appendices**
- Index**

© 2000-2003, Steve Easterbrook 14

University of Toronto Department of Computer Science

## IEEE STD Section 3 (example)

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

<p><b>3.1 External Interface Requirements</b></p> <ul style="list-style-type: none"> <li>3.1.1 User Interfaces</li> <li>3.1.2 Hardware Interfaces</li> <li>3.1.3 Software Interfaces</li> <li>3.1.4 Communication Interfaces</li> </ul> <p><b>3.2 Functional Requirements</b> this section organized by mode, user class, feature, etc. For example:</p> <ul style="list-style-type: none"> <li>3.2.1 Mode 1           <ul style="list-style-type: none"> <li>3.2.1.1 Functional Requirement 1.1</li> <li>...</li> </ul> </li> <li>3.2.2 Mode 2           <ul style="list-style-type: none"> <li>3.2.1.1 Functional Requirement 1.1</li> <li>...</li> </ul> </li> <li>...</li> <li>3.2.2 Mode n           <ul style="list-style-type: none"> <li>...</li> </ul> </li> </ul>	<p><b>3.3 Performance Requirements</b> <i>Remember to state this in measurable terms!</i></p> <p><b>3.4 Design Constraints</b></p> <ul style="list-style-type: none"> <li>3.4.1 Standards compliance</li> <li>3.4.2 Hardware limitations etc.</li> </ul> <p><b>3.5 Software System Attributes</b></p> <ul style="list-style-type: none"> <li>3.5.1 Reliability</li> <li>3.5.2 Availability</li> <li>3.5.3 Security</li> <li>3.5.4 Maintainability</li> <li>3.5.5 Portability</li> </ul> <p><b>3.6 Other Requirements</b></p>
---	--

© 2000-2003, Steve Easterbrook 15

University of Toronto Department of Computer Science

## MIL-STD-498

→ MIL-STD-498 is the main US DoD standard for software development and documentation

- ↳ replaces DOD-STD-2167A and DOD-STD7935A

→ Consists of:

- ↳ a guidebook,
- ↳ a list of process requirements
- ↳ 22 Data Items Descriptions (DIDs)

→ DIDs are the documents produced during software development. e.g.

- ↳ OCD - Operational Concept Description
- ↳ SSS - System/Subsystem Specification
- ↳ SRS - Software Requirements Specification
- ↳ IRS - Interface Requirements Specification
- ↳ etc

© 2000-2003, Steve Easterbrook Source: Adapted from MIL-STD-498 16

University of Toronto Department of Computer Science

## System Structure

→ MIL-STD-498 uses the following system structure:

HWCI = Hardware Configuration Item  
 CSCI = Computer Software Configuration Item

```

graph TD
    A[System or Segment (SSS)] --> B[System or Segment (SSS)]
    A --> C[System or Segment (SSS)]
    B --> D[CSCI (SRS)]
    B --> E[CSCI (SRS)]
    B --> F[HWCI (PIDS)]
    C --> G[CSCI (SRS)]
    C --> H[CSCI (SRS)]
    C --> I[HWCI (PIDS)]
    C --> J[Interfaces (IRS)]
  
```

© 2000-2003, Steve Easterbrook Source: Adapted from MIL-STD-498 17

University of Toronto Department of Computer Science

## SRS DID from MIL-STD-498

- 1 Scope
  - 1.1 Identification
  - 1.2 System Overview
  - 1.3 Document Overview
- 2 Referenced Documents
- 3 Requirements
  - 3.1 Required States and Modes
  - 3.2 CSCI Capability Requirements
    - 3.2.x Capability X...
  - 3.3 CSCI External Interface Requirements
    - 3.3.1 Interface Identification and diagrams
    - 3.3.x Project Unique Identifier
  - 3.4 CSCI Internal Interface Requirements
  - 3.5 CSCI Internal Data Requirements
  - 3.6 Adaptation Requirements
  - 3.7 Safety Requirements
  - 3.8 Security and Privacy Requirements
  - 3.9 CSCI Environment Requirements
  - 3.10 Computer Resource Requirements
  - 3.11 Software Quality Factors
  - 3.12 Design and Implementation Constraints
  - 3.13 Personnel-related Requirements
  - 3.14 Training-related Requirements
  - 3.15 Logistics-related Requirements
  - 3.16 Other Requirements
  - 3.17 Packaging Requirements
  - 3.18 Precedence and criticality of Requirements
- 4 Qualification Provisions
- 5 Requirements Traceability
- 6 Notes
- Appendices

© 2000-2003, Steve Easterbrook Source: Adapted from MIL-STD-498 18

University of Toronto Department of Computer Science

## Requirements Traceability

→ Definition (DOD-STD-2167A):

- (1) The document in question contains or implements all applicable stipulations in the predecessor document
- (2) a given term, acronym, or abbreviation means the same thing in all documents
- (3) a given item or concept is referred to by the same name or description in the documents
- (4) all material in the successor document has its basis in the predecessor document, that is, no untraceable material has been introduced
- (5) the two documents do not contradict one another"

→ In short:

- ☞ A demonstration of completeness, necessity and consistency
- ☞ a clear allocation/flowdown path (down through the document hierarchy)
- ☞ a clear derivation path (up through the document hierarchy)

© 2000-2003, Steve Easterbrook Source: Adapted from Palmer, 1996, p.367 19

University of Toronto Department of Computer Science

## Importance of Traceability

- Verification and Validation
  - ☞ assessing adequacy of test suite
  - ☞ assessing conformance to requirements
  - ☞ assessing completeness, consistency, impact analysis
  - ☞ assessing over- and under-design
  - ☞ investigating high level behavior impact on detailed specifications
  - ☞ detecting requirements conflicts
  - ☞ checking consistency of decision making across the lifecycle
- Document access
  - ☞ ability to find information quickly in large documents
- Process visibility
  - ☞ ability to see how the software was developed
  - ☞ provides an audit trail
- Management
  - ☞ change management
  - ☞ risk management
  - ☞ control of the development process
- Maintenance
  - ☞ Assessing change requests
  - ☞ Tracing design rationale

© 2000-2003, Steve Easterbrook Source: Adapted from Palmer, 1996, p.365 20

University of Toronto Department of Computer Science

## Traceability Difficulties

- **Cost**
  - ↳ very little automated support
  - ↳ full traceability is very expensive and time-consuming
- **Delayed gratification**
  - ↳ the people defining traceability links are not the people who benefit from it
    - > development vs. V&V
  - ↳ much of the benefit comes late in the lifecycle
    - > testing, integration, maintenance
- **Size and diversity**
  - ↳ Huge range of different document types, tools, decisions, responsibilities,...
  - ↳ No common schema exists for classifying and cataloging these
  - ↳ In practice, traceability concentrates only on baselined requirements

© 2000-2003, Steve Easterbrook Source: Adapted from Palmer, 1996, p365-6 21

University of Toronto Department of Computer Science

## Current Practice

- **Coverage:**
  - ↳ links from requirements forward to designs, code, test cases,
  - ↳ links back from designs, code, test cases to requirements
  - ↳ links between requirements at different levels
- **Traceability process**
  - ↳ Assign each sentence or paragraph a unique id number
  - ↳ Manually identify linkages
  - ↳ Use manual tables to record linkages in a document
  - ↳ Use a traceability tool (database) for project wide traceability
  - ↳ Tool then offers ability to
    - > follow links
    - > find missing links
    - > measure overall traceability

© 2000-2003, Steve Easterbrook Source: Adapted from Palmer, 1996, p367-8 22

University of Toronto Department of Computer Science

## Traceability Tools

- **Approaches:**
  - ↳ **hypertext linking**
    - > hotwords are identified manually, tool records them
  - ↳ **unique identifiers**
    - > each requirement gets a unique id; database contains cross references
  - ↳ **syntactic similarity coefficients**
    - > searches for occurrence of patterns of words
- **Limitations**
  - ↳ All require a great deal of manual effort to define the links
  - ↳ All rely on purely syntactic information, with no semantics or context
- **Examples**
  - ↳ **single phase tools:**
    - > TeamWork (Cadre) for structured analysis
  - ↳ **database tools, with queries and report generation**
    - > RTM (Marconi)
    - > SLATE (TD Technologies)
    - > DOORS (Zycad Corp)
  - ↳ **hypertext-based tools**
    - > Document Director
    - > Any web browser
  - ↳ **general development tools that provide traceability**
    - > RDD-100 (Ascent Logic) - documents system conceptual models
    - > Foresight - maintains data dictionary and document management

© 2000-2003, Steve Easterbrook Source: Adapted from Palmer, 1996, p372 23

University of Toronto Department of Computer Science

## Limitations of Current Tools

- **Informational Problems**
  - ↳ Tools fail to track *useful* traceability information
    - > e.g cannot answer queries such as "who is responsible for this piece of information?"
  - ↳ inadequate pre-requirements traceability
    - > "where did this requirement come from?"
- **Lack of agreement...**
  - ↳ ...over the quantity and type of information to trace
- **Informal Communication**
  - ↳ People attach great importance to personal contact and informal communication
    - > These always supplement what is recorded in a traceability database
  - ↳ But then the traceability database only tells part of the story!
    - > Even so, finding the appropriate people is a significant problem

© 2000-2003, Steve Easterbrook Source: Adapted from Gostel & Finkelstein, 1993, p100 24



## Problematic Questions

### → Involvement

- ↳ Who has been involved in the production of this requirement and how?

### → Responsibility & Remit

- ↳ Who is responsible for this requirement?
  - > who is currently responsible for it?
  - > at what points in its life has this responsibility changed hands?
- ↳ Within which group's remit are decisions about this requirement?

### → Change

- ↳ At what points in the life of this requirements has working arrangements of all involved been changed?

### → Notification

- ↳ Who needs to be involved in, or informed of, any changes proposed to this requirement?

### → Loss of knowledge

- ↳ What are the ramifications regarding the loss of project knowledge if a specific individual or group leaves?



## Contribution Structures

### → 'author' attribute too weak

- ↳ does not adequately capture ownership of information
- ↳ refers to person that wrote the document rather than the person who originated the content
- ↳ fail to capture situations where many people participate
- ↳ fail to capture changing patterns of participation

### → Contribution structures

- ↳ link requirements artifacts (contributions) to agents (contributors) via contribution relations

### → Roles

- ↳ Principal
  - > who motivated the artefact (responsible for consequences)
- ↳ Author
  - > who chose the structure and content (responsible for semantics)
- ↳ Documentor
  - > who recorded/transcribed the content (responsible for appearance)