Requirements Specification

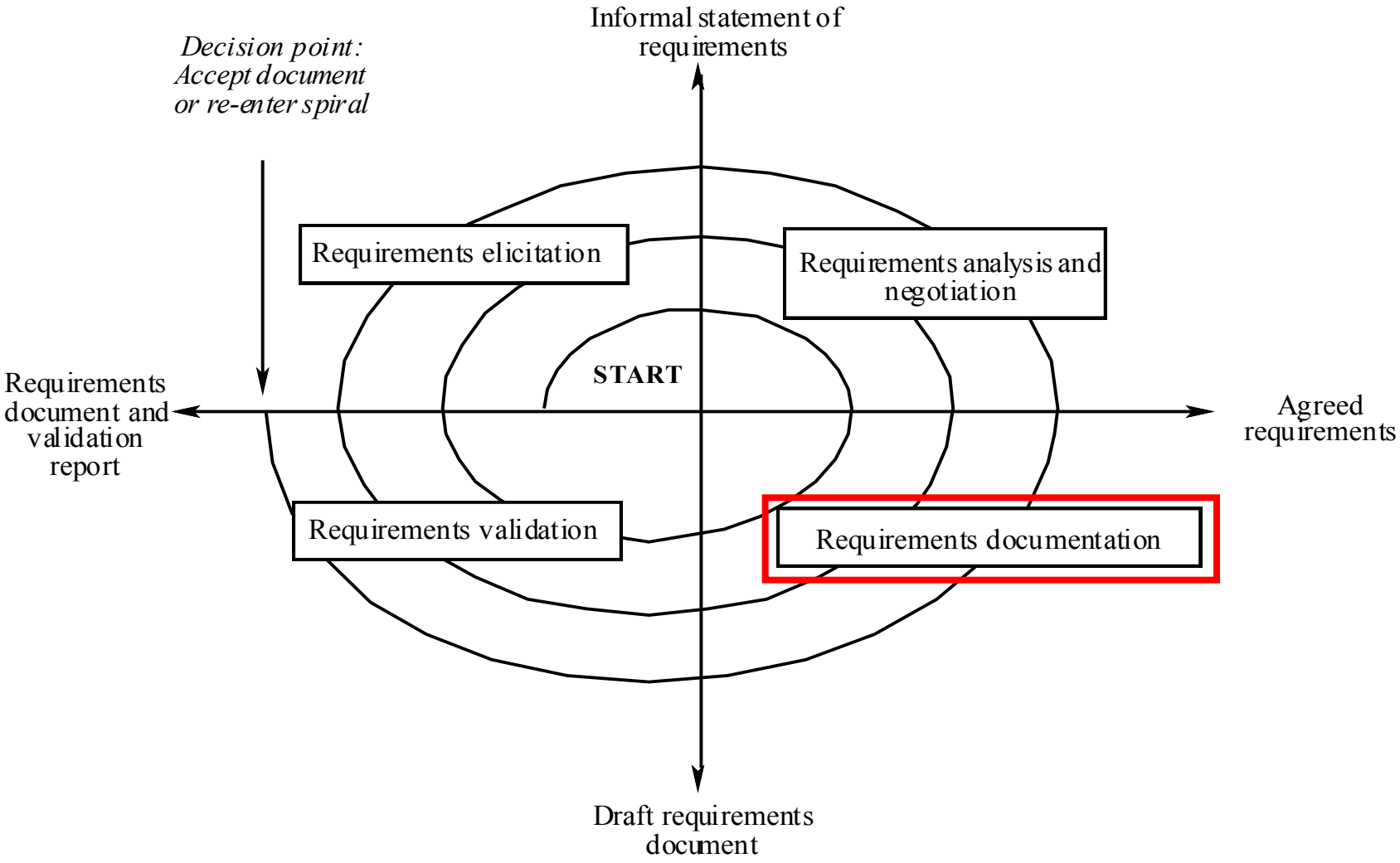
2016-11-17

Outline

"Requirements Specification":

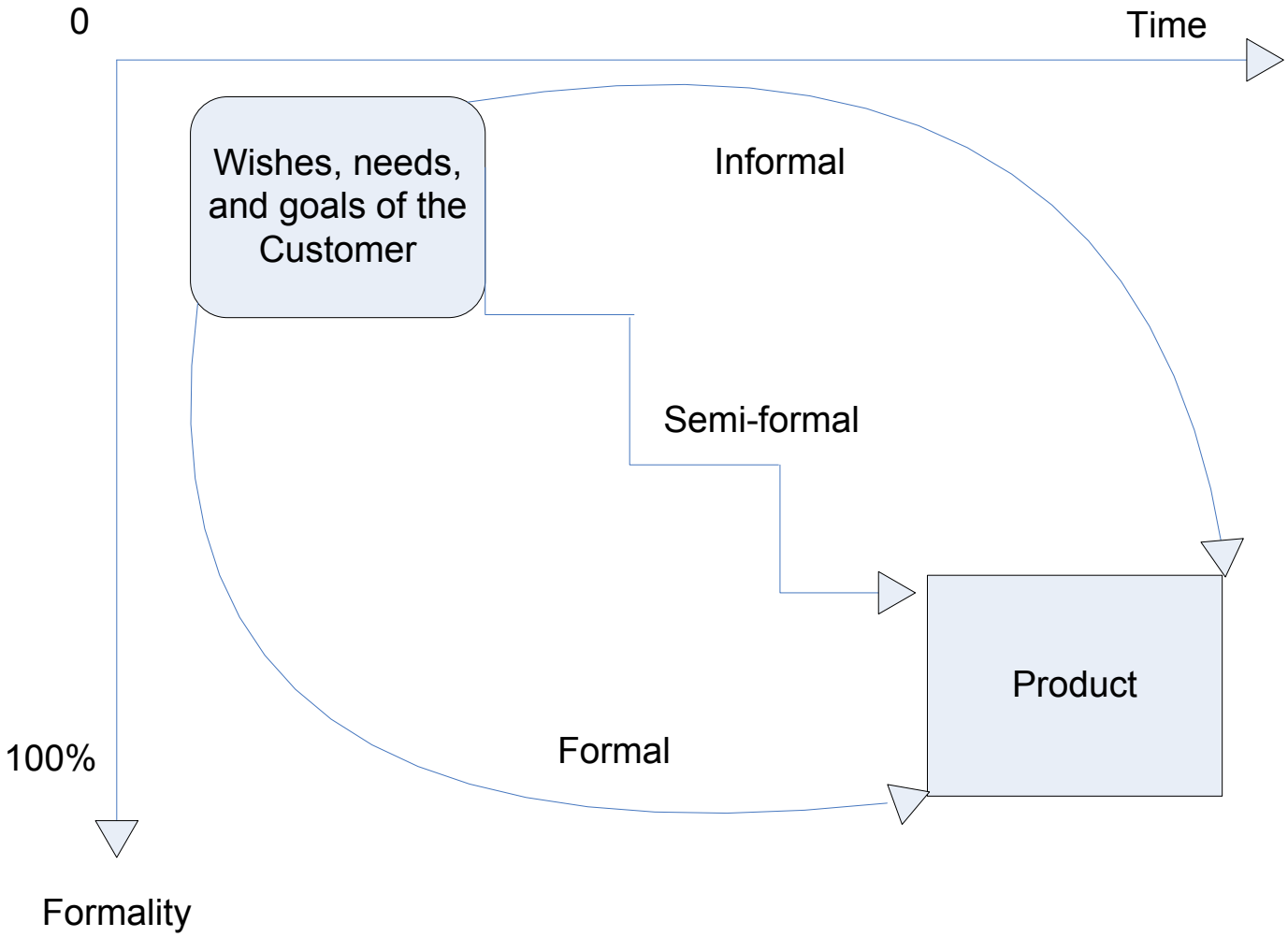
- **Introduction**
- Content of Software Requirements Specifications
  - IEEE Std. 830
  - Volere Template
- Writing Good Requirements
- Requirements Management and Traceability
- Requirements on Requirements Management Tools
- Sample Software Requirements Specification
- Exercise 3: Requirements Inspection and SRS Evaluation

Spiral Requirements Engineering Process



Spiral RE Process (G. Kotonya and I. Sommerville 1998)

Documenting Software Requirements



Different degrees of formality

- “Oral documentation” and unstructured prose
- User Stories
- Prototypes
- **Structured prose**
  (e.g. according IEEE Std. 830, Volere Template)
- Use cases
- UML models (e.g., state charts, activity diagrams)
- Formal approaches (e.g., Z)
- Indirectly via other artifacts (e.g., test cases, user manual)

Outline

"Requirements Specification":

- ✓ Introduction
- **Content of Software Requirements Specifications**
  - **IEEE Std. 830**
  - Volere Template
- Writing Good Requirements
- Requirements Management and Traceability
- Requirements on Requirements Management Tools
- Sample Software Requirements Specification
- Exercise 3: Requirements Inspection and SRS Evaluation

IEEE Std. 830

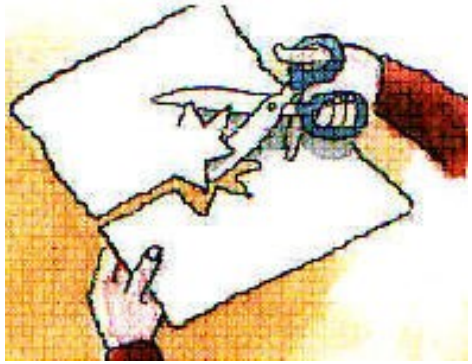
IEEE Std 830-1998

IEEE Recommended Practice for Software Requirements Specifications

A summary

What is IEEE Std. 830?

- **IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications**
  - **Abstract:** The content and qualities of a good **software requirements specification (SRS)** are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 are also provided.
- **Purpose and goal**
  - IEEE Std. 830 describes **recommended approaches for the specification** of software requirements
  - It is based on a model in which the result of the software requirements specification process is an **unambiguous and complete specification document**



Aims

- Helping customers to accurately **describe** what they wish to obtain
- Helping suppliers to **understand** exactly what the customer wants
- Helping individuals to develop a standard software requirements specification (SRS) **outline, format and content** for their own organizations

SRS - Role and Potential Benefits

- Basis for agreement between customers and suppliers on what the software product is to do
- Basis for estimating costs and schedules
- Baseline for validation and verification

- Reduction of development effort
- Easier transfer of software product to new users or machines
- Basis for enhancement of the software product

Nature of the SRS

- ***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, etc.
- ***Attributes.*** What are the portability, correctness, maintainability, security, etc. 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) etc.?

Environment of the SRS

- The SRS writer(s) should be careful not to go beyond the bounds of their role. This means the SRS
  - Should correctly define all of the software requirements
  - Should not describe any design or implementation details. These should be described in the design stage of the project
  - Should not impose additional constraints on the software. These are properly specified in other documents such as a software quality assurance plan
- Therefore, **a properly written SRS limits the range of valid designs, but does not specify any particular design**

Characteristics of a Good SRS

- ***Correct***. Every requirement stated is one that the software shall meet.
- ***Unambiguous***. Every requirement stated has only one interpretation.
- ***Complete***. All significant requirements + Definition of responses of the software to all realizable classes of input data in all realizable classes of situations.
- ***Consistent***. No subset of individual requirements are in conflict.
- ***Ranked*** *for importance and/or stability.*
- ***Verifiable***. There exists some finite cost-effective process with which a person or machine can check that the software product meets a requirement.
- ***Modifiable***. Structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style.
- ***Traceable***. Origin of each of its requirements is clear and the referencing of each requirement in future development or enhancement documentation is facilitated.

Group Discussion

- Discuss in groups the “Characteristics of a Good SRS” (cf. slide 13)
- For each of the characteristics try to answer the following:
  - Is it realistic?
  - How can it be achieved?
  - Under which conditions is it beneficial?
  - Are there drawbacks of trying to achieve it?
- Give a short oral summary of your discussion to the auditorium

Joint Preparation, SRS Evolution, Change Mgmt

- The customer and the supplier should work together to produce a well-written and completely understood SRS
- Requirements should be specified as completely and thoroughly as known at the time, even if evolutionary revisions can be foreseen as inevitable. The fact that they are incomplete should be noted
- A formal change process should be initiated to identify, control, track, and report projected changes

Embedding Design in the SRS ?

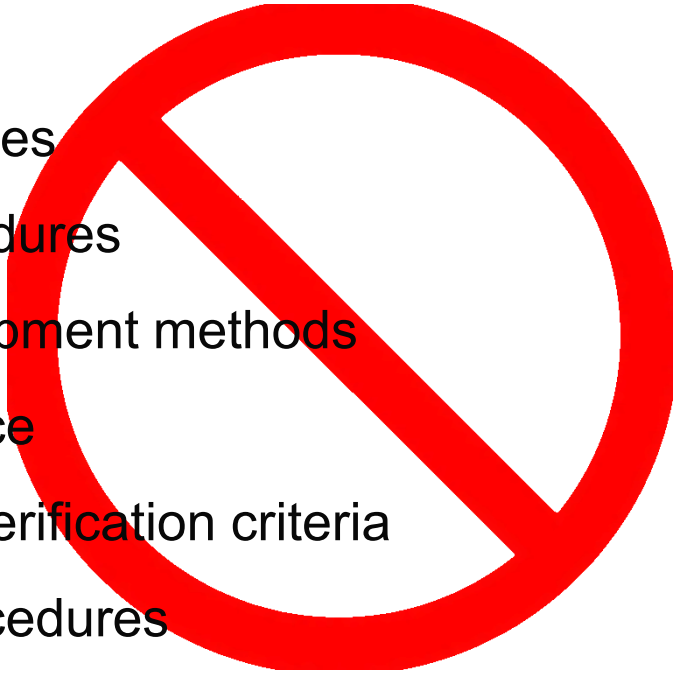
The SRS should normally **not** specify:

- Partitioning the software into modules
- Allocating functions to the modules
- Describing the flow of information or control between modules
- Choosing data structures

Embedding Project Requirements in the SRS ?

The SRS should **address the software product, not the process of producing the software product:**

- Cost
- Delivery schedules
- Reporting procedures
- Software development methods
- Quality assurance
- Validation and verification criteria
- Acceptance procedures



Prototype SRS Outline

Table of Contents

1. Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms, and abbreviations
   1.4 References
   1.5 Overview
2. Overall description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 Constraints
   2.5 Assumptions and dependencies
3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)

Appendixes

Index

Outline

"Requirements Specification":

- ✓ Introduction
- **Content of Software Requirements Specifications**
  - ✓ IEEE Std. 830
  - **Volere Template**
- Writing Good Requirements
- Requirements Management and Traceability
- Requirements on Requirements Management Tools
- Sample Software Requirements Specification
- Exercise 3: Requirements Inspection and SRS Evaluation

Volere Template

SRS Example: The Volere Template

Content of SRS: Volere Template

Requirements Types, Structure of Specifications

PROJECT DRIVERS:

1. The Purpose of the Product
2. Client, Customer, Stakeholders
3. Users of the Product

PROJECT CONSTRAINTS:

4. Mandated Constraints
5. Naming Conventions and Definitions
6. Relevant Facts and Assumptions

FUNCTIONAL REQUIREMENTS:

7. The Scope of the Work
8. The Scope of the Product
9. Functional and Data Requirements

http://www.systemsguild.com

NON-FUNCTIONAL REQUIREMENTS:

10. Look and Feel
11. Usability
12. Performance
13. Operational
14. Maintainability and Portability
15. Security
16. Cultural and Political
17. Legal

PROJECT ISSUES:

18. Open Issues
19. Off-the-shelf Solutions
20. New Problems
21. Tasks
22. Cutover
23. Risks
24. Costs
25. User Documentation
26. Waiting Room
27. Ideas for Solutions

Non-functional Requirements

10. Look and Feel Requirements
11. Usability Requirements
12. Performance Requirements
13. Operational Requirements
14. Maintainability and Portability Requirements
15. Security Requirements
16. Cultural and Political Requirements
17. Legal Requirements

10. Look and Feel Requirements

- The interface
- The style of the product

11. Usability Requirements

- Ease of use
- Personalization and internationalization requirements
- Ease of learning
- Accessibility requirements

12. Performance Requirements

- Speed and latency requirements
- Safety critical requirements
- Precision requirements
- Reliability and availability requirements
- Robustness requirements
- Capacity requirements
- Scalability or extensibility requirements

13. Operational Requirements

- Expected physical environments
- Expected technological environment
- Partner applications
- Productization Requirements

14. Maintainability and Support Rqmts

- How easy must it be to maintain this product?
- Are there special conditions that apply to the maintenance of this product?
- Supportability
- Portability requirements

15. Security Requirements

- Access requirements
- Integrity requirements
- Privacy requirements
- Audit requirements
- Immunity requirements

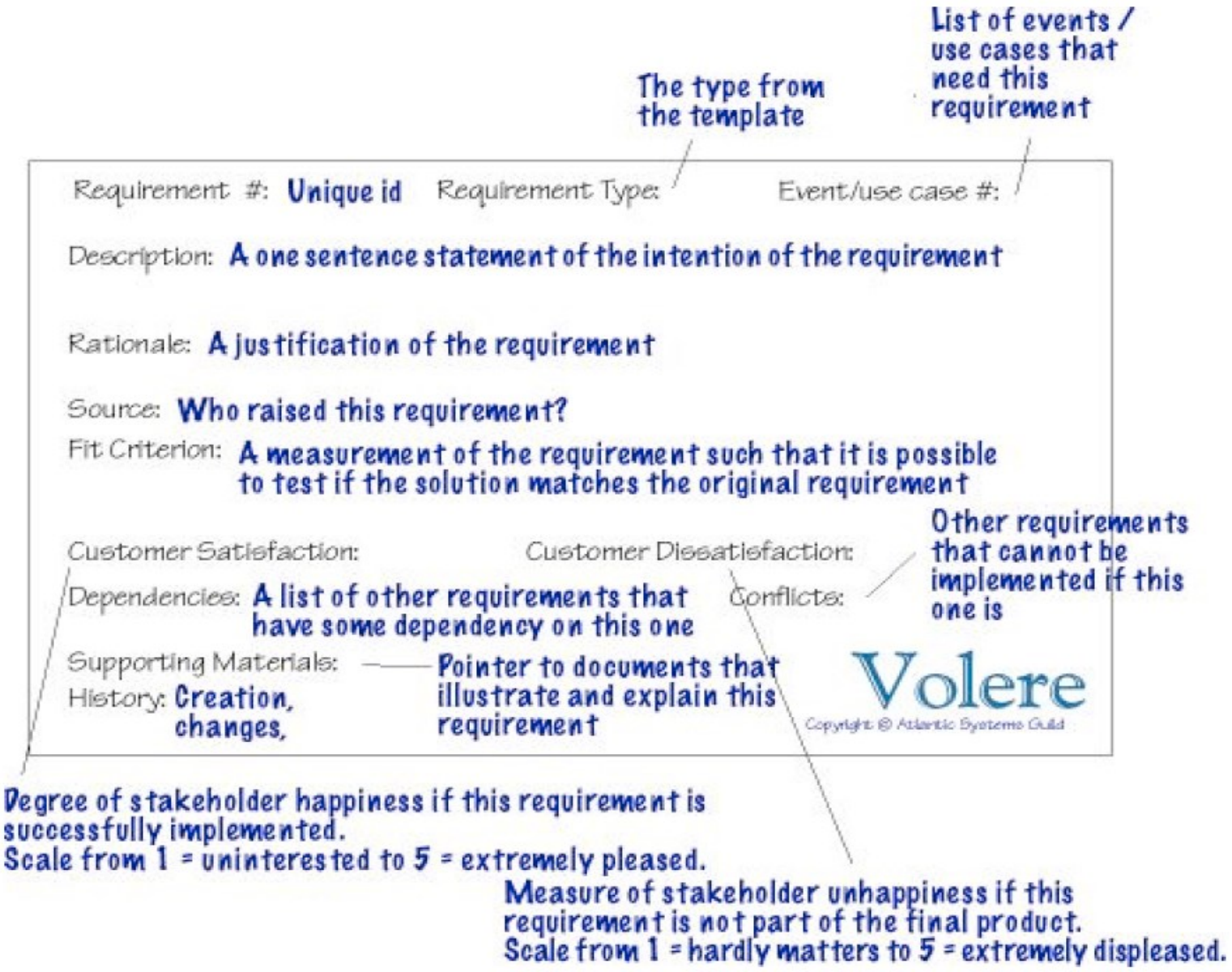
16. Cultural and Political Requirements

- Are there any special factors about the product that would make it unacceptable for some political or cultural reason?

17. Legal Requirements

- Does the system fall under the jurisdiction of any law?
- Are there any standards with which we must comply?

Requirements Attributes in Volere



Fit Criterion

- "… an objective measure of the requirement's meaning; it is the criterion for evaluating whether or not a given solution fits the requirement" [Volere]

- Example
  - Description: "The product shall make the users want to use it."
  - Fit Criterion: "[x %] of the users are regularly using the product after [an agreed time] familiarization period."

Outline

"Requirements Specification":

- ✓ Introduction
- ✓ Content of Software Requirements Specifications
  - ✓ IEEE Std. 830
  - ✓ Volere Template
- ➢ **Writing Good Requirements**
- ▪ Requirements Management and Traceability
- ▪ Requirements on Requirements Management Tools
- ▪ Sample Software Requirements Specification
- ▪ Exercise 3: Requirements Inspection and SRS Evaluation

An Effective Requirement Is … (1/2)

- **Necessary**
  - A condition or capability needed to solve a problem or achieve an objective
- **Verifiable**
  - A requirement must be objectively verifiable (by test, analysis, inspection, or demonstration) to prove compliance
  - The requirement should be written in a fashion such that the means of verification is clearly understood
  - A desired capability that cannot be objectively verified should be written as a design goal (“should”), not a requirement (“shall”)
- **Achievable**
  - If a requirement is not achievable, you cannot satisfy your contractual obligations
  - To be achievable, a requirement must be technically feasible, affordable, and fit within schedule and other constraints

Slides partially based on: “Writing Effective Requirements”, INCOSE, http://www.incose.org

An Effective Requirement Is … (2/2)

- **Simple, concise & easily understood**
  - Vague and ambiguous requirements
    - can be misinterpreted and result in a faulty design solution that fails to satisfy the customers' needs and your contractual obligations
    - cannot form the basis for an objective verification criteria
  - Each requirement must specify one and only one function
- **Unique**
  - Verifying redundant requirements adds unnecessary cost
  - Redundant requirements may result in contradictions
- **Traceable**
  - Relationship with parent requirement(s) is defined and documented
  - Relationship with child requirement(s) is defined and documented
  - Rationale describing allocation is documented

Ineffective Requirements: Example (1/2)

3.2.1.1.3.9 Optical/electronic noise.

Noise patterns in the output video due to optical reflections, electronic processing, microphonics and radiation variations arising in the optical or TV subsystems shall not be evident on a test display at any contrast or brightness level for any illumination input. The test display shall also be free of any significant noise or interference due to the transmission or reception of laser energy, operation of the data link, data processing from the transducers inside the MPS or other sources in the system. The following design areas shall be given special attention to achieve optical electronic noise free video:

a. Laser energy ghosts will be controlled by optics design.
b. Laser pump light will be controlled in laser design.
c. Veiling glare, as defined in MIL-STD-150, shall be less than 1 percent by optical design, which includes metal finishes and coatings of optical finishes and coatings of optical surfaces and lens edges.
d. A ruggedized TV camera tube that has no mechanical resonance below 1 kilohertz will be used to minimize microphonics.

Ineffective Requirements: Example (2/2)

"Advertisement and Specification for a Heavier-Than-Air Flying Machine," U.S. Army Signal Corps Specification No. 486, 1907.

> "10. It should be sufficiently simple in its construction and operation to permit an intelligent man to become proficient in its use within a reasonable length of time."

- How simple is "sufficiently simple"?
- How intelligent is "intelligent"?
- How proficient is "proficient"?
- What is a "reasonable length of time"?

Common Traps And How To Avoid Them (1/4)

- **Bad or missing information**
  - Leads to over-specifying requirements or failing to specify needed requirements
  - Identify and involve the stakeholders and subject matter experts in every step of the product lifecycle
  - Develop and validate concepts-of-operations, mission scenarios, linkage and flow, constraints (cost, schedule, and technical)
  - Employ requirements analysis tools, checklists, comprehensive specification templates
- **Specifying the "how," not the "what"**
  - Determine the need statement and write the requirement accordingly
- **Verbosity**
  - Extra words mean extra chances to misunderstand!
  - Be concise
  - Use simple, common terms whenever possible – avoid using "buzz words" and "project-speak"

Common Traps And How To Avoid Them (2/4)

- **Using vague or ambiguous words, phrases, and statements**
  - Subject to multiple interpretations
  - Not objectively verifiable
  - Creates an opportunity for the customer to require additional work without additional compensation, or the contractor to demand additional compensation for in-scope work
  - Example words and phrases to avoid
    - "Achievable", "adequate", "approximately", "complete", "damaged", "degraded", "efficient", "effective", "minimize", "maximize", "flexible", "modular", "nominal", "normally", "optimum", "survive", "typically", "usually", "generally", "often", "easy", "to the maximum (or minimum) extent", "as much (or little) as possible", "user-friendly", "scalable", "versatile", "approximately", "and/or", "shall be designed to", "shall be capable of"
  - ➢ Be precise
  - ➢ State the real need

Common Traps And How To Avoid Them (3/4)

- **Multiple requirements ("shalls") per statement**
  - Increases risk that a requirement may be missed in the design
  - May present problems in verification if the requirements have to be verified by different methods, at different times, or at different levels of assembly
  - ➢ Only specify a single requirement per statement
- **Use of negative or passive sense**
  - i.e., "The system shall not perform the following maneuver when …."
  - ➢ Reword to positive statements; use active verbs
- **Misuse of the terms "shall," "will," and "should"**
  - Requirements use "shall", statements of fact use "will", and goals use "should"
  - Terms such as "are", "is", "was", and "must" don't belong in a requirement
  - ➢ Stick with the government and industry standard defined above – to deviate from it will only invite confusion

Common Traps And How To Avoid Them (4/4)

- **Inconsistent use of phrases to reference, specify alternative courses of action, or state limitations**
  - Can create confusion for the reader
  - ➢ Pick a phrase and be consistent throughout a specification or family of specifications
    - For example, use **"as specified in**" when referencing **external documents**, use **"as specified herein"** or **"as specified in x.x.x"** when referencing **within a document**
- **Being over-stringent on parametric requirements**
  - It is not possible to verify absolute values. Test instrumentation has finite measurement accuracy
  - ➢ Place acceptable tolerances on parameters, e.g., dimensions, weight, voltage
  - ➢ Tolerances should be stated as values, not percentages

Cross-Referencing Requirements

- **Cross-referencing within a specification is used only to**
  - Clarify relationship between conditional requirements
  - Avoid inconsistencies and unnecessary repetition
- **The proper language for cross-referencing is:**
  - ... “as specified herein”
    - When referencing to a requirement within the spec that is obvious and easy to find (e.g., requirement is a paragraph title)
  - ... “as specified in n.n.n”
    - When the requirement paragraph is not obvious or may be difficult to find

Requirements Wording Templates

- **Four classes of requirements:**
  - Behavior/Performance
  - Design Production Capability
  - Design Constraint
  - Process Compliance

- **Use of templates as:**
  - guidelines for assessing necessity and sufficiency, or completeness, of requirement statement components
  - guidance for assessing requirements quality

Template for Behavior/Performance Rqmt

The <*System_name*> shall <behavior>
if <conditions>, where <quality factor>.

Upon <conditions>, the <*System_name*> shall <behavior> where <quality factor>.

Examples:

- The ATM shall reject withdrawal requests if the amount requested is not divisible by 20.
- Upon Operator Request, the system shall disable all audible alarms.

Template for Production Capability

The system_name shall produce <output>
for use by <nodes>,
if <conditions>,
using <inputs/outputs>,
where <quality factor>.

Examples:

- The ATM shall produce a receipt for use by bank patrons if a transaction is completed.
- The system shall produce a launch alert message for use by the Missile Defense Agency if a launch is detected within the programmed target area within 2 minutes of launch detection.

Template for Design Constraints

The <system_name> **shall have** <instance>
with this <feature>,
and/or <constraint>.

Examples:

- The ATM shall have an ACME 12.1-inch TFT active-matrix display.
- The Ground Segment Software shall be programmed in ADA.

Template for Process Compliance

The *system_name* shall be <programmatic process> in accordance with <document> where <quality factor>.

Examples:

- The ATM shall be developed in accordance with ISO9001, Quality System Management Guidelines.

Use of Abbreviations & Symbols

- **First use of an abbreviation or symbol in a paragraph**
  - Spell it out in full
  - Place it in parentheses after first use
- **Plural abbreviations**
  - If referring to more than one, you can put a plain “s” at the end as long as the meaning is clear. If the abbreviation is using periods, e.g., M.D., or the plural meaning will not be clear, then use an “’s” or simply spell it out.
- **Don’t:**
  - use an abbreviation or symbol in a paragraph title if at all possible
  - start a sentence with an abbreviation, symbol, or digit number, e.g., “10 minutes shall elapse …”
    - Can get confused with paragraph numbers or numerically ordered lists
    - Instead use “The elapsed time shall be not greater than 10 minutes … .” or “Not greater than 10 minutes shall elapse …”

Summary

- Make sure each requirement is necessary, verifiable, and achievable
- Write clearly, simply, concisely and unambiguously
- Make sure each requirement is unique and traceable
- Use only one “shall” per statement
- Specify “what’s required," not "how to do it". Don't specify a design constraint unless it’s necessary to do so
- Avoid buzz words and project-speak
- Keep the language active and positive vs. passive and negative
- Be consistent with your choice of phrasing throughout

- *Don't assume the reader will know what you meant even if that’s not what you wrote*

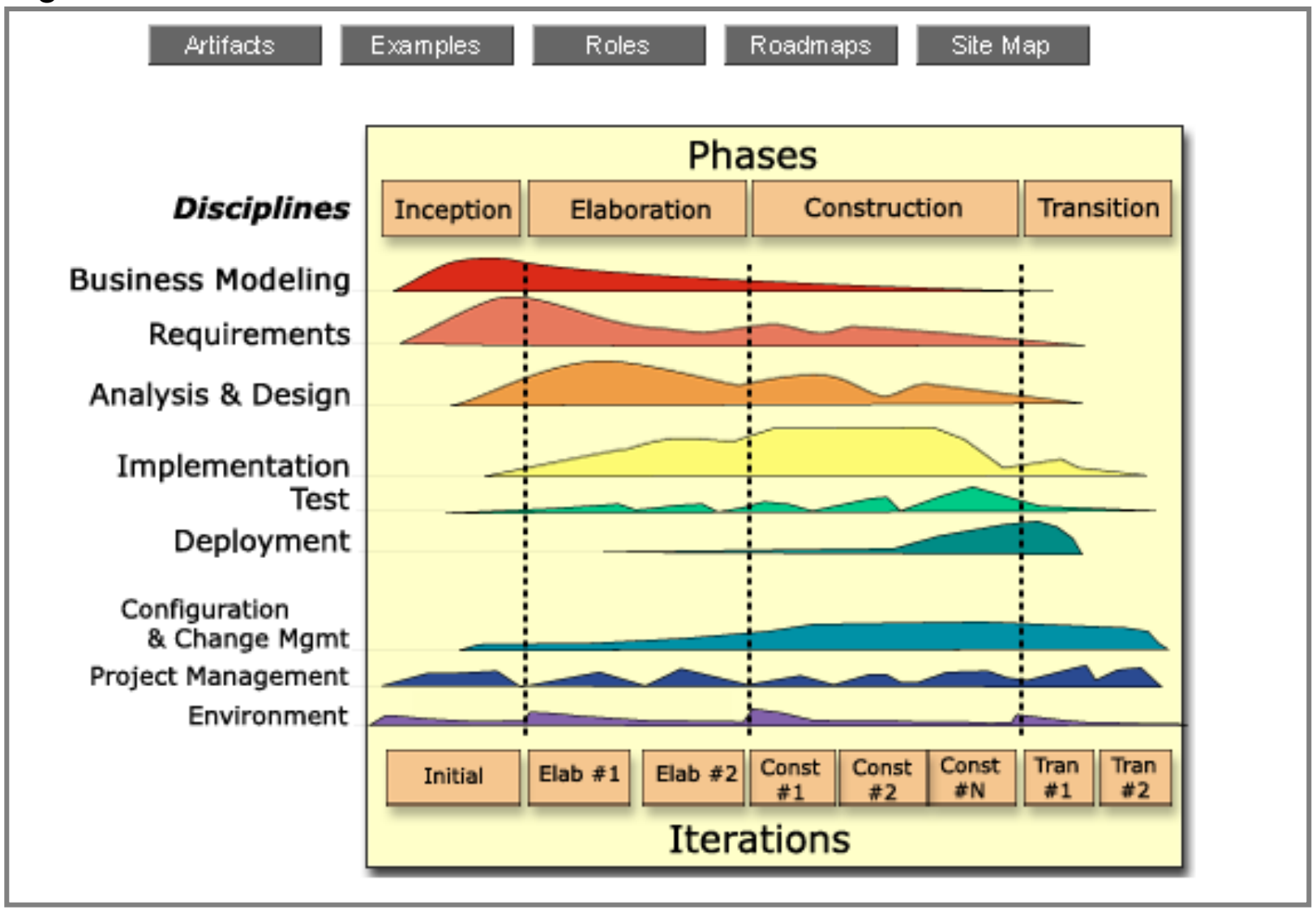
Outline

"Requirements Specification":

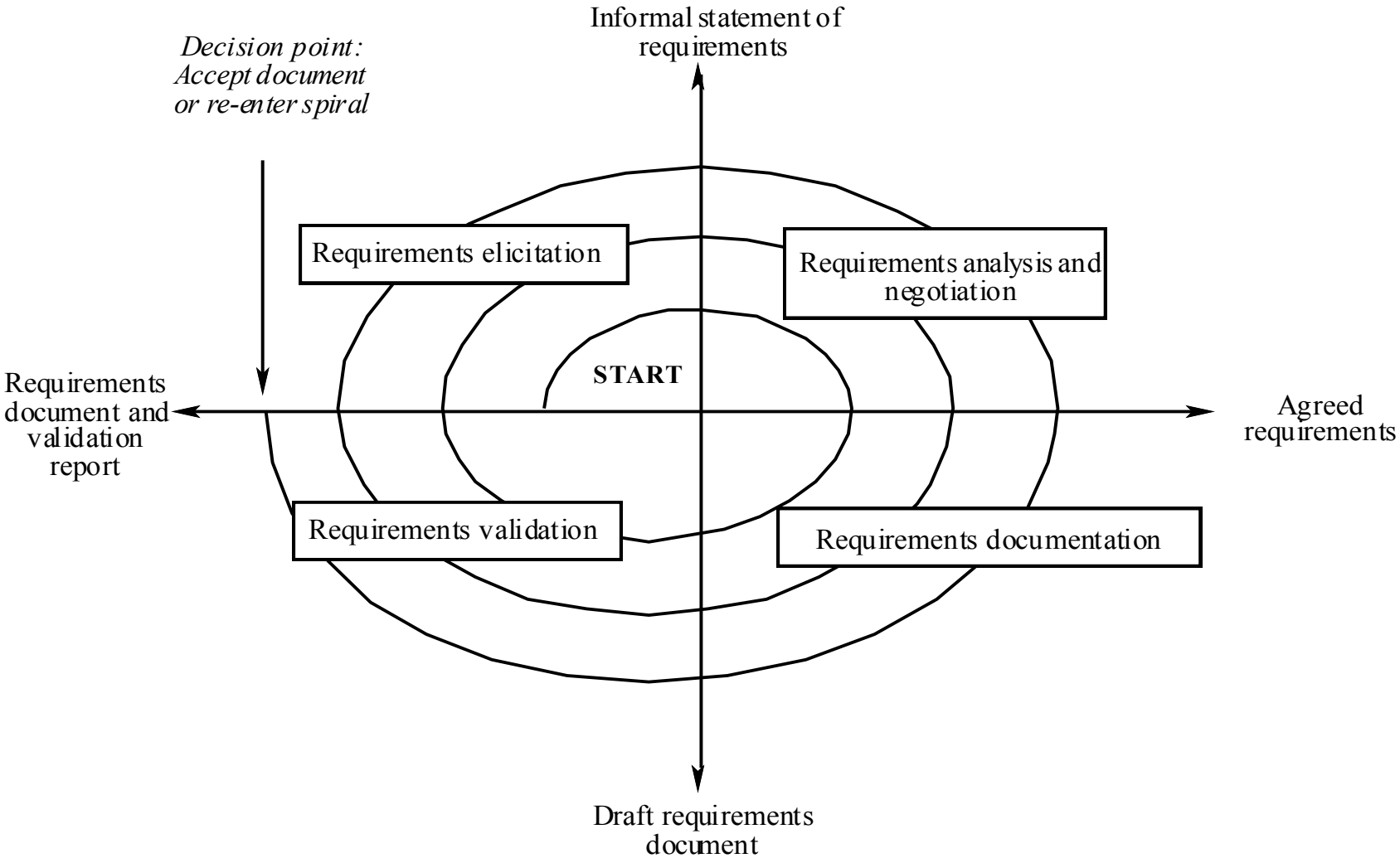
- ✓ Introduction
- ✓ Content of Software Requirements Specifications
  - ✓ IEEE Std. 830
  - ✓ Volere Template
- ✓ Writing Good Requirements
- ➢ **Requirements Management and Traceability**
- ▪ Requirements on Requirements Management Tools
- ▪ Sample Software Requirements Specification
- ▪ Exercise 3: Requirements Inspection and SRS Evaluation

Requirements in Iterative Life Cycle Models

E.g.: **Rational Unified Process**



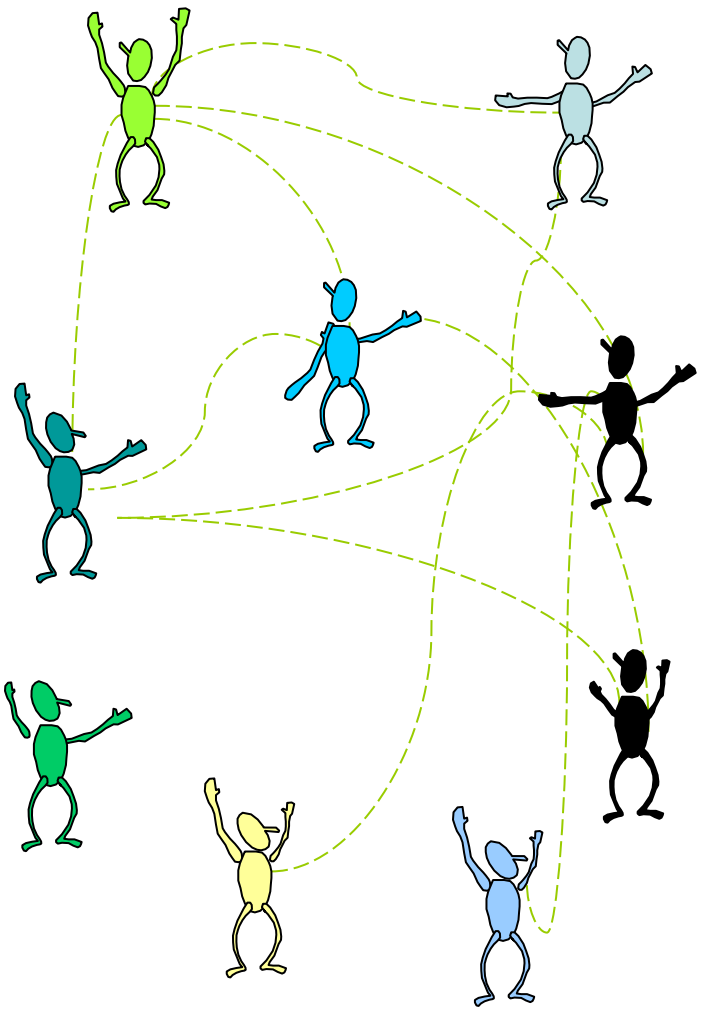
Spiral Requirements Engineering Process



Spiral RE Process (G. Kotonya and I. Sommerville 1998)

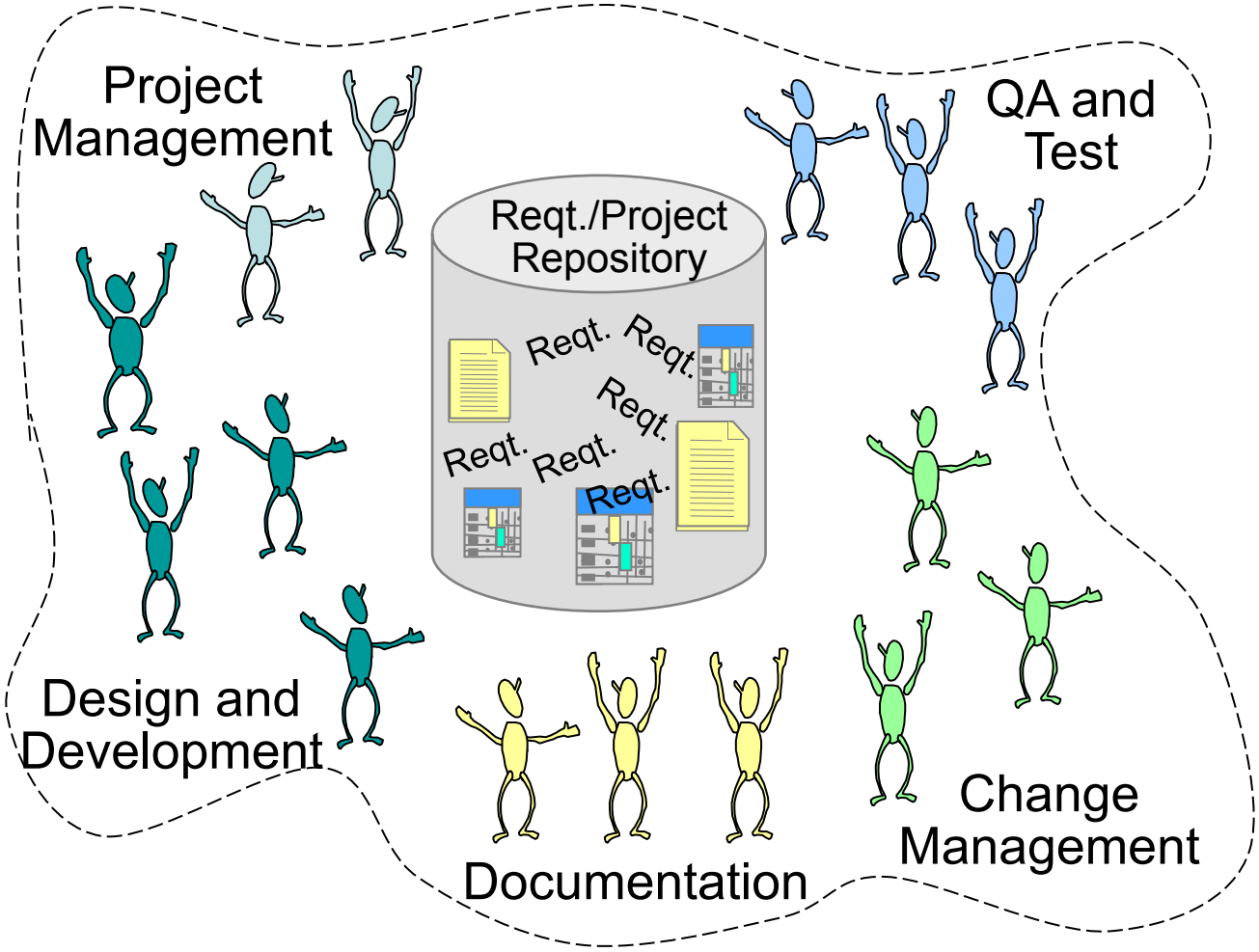
How Does Your Team Communicate?

- Is there a centralized place for reviewing requirements and data?
- How are changes communicated?
- How do you manage change?
- How do you monitor project progress and status?

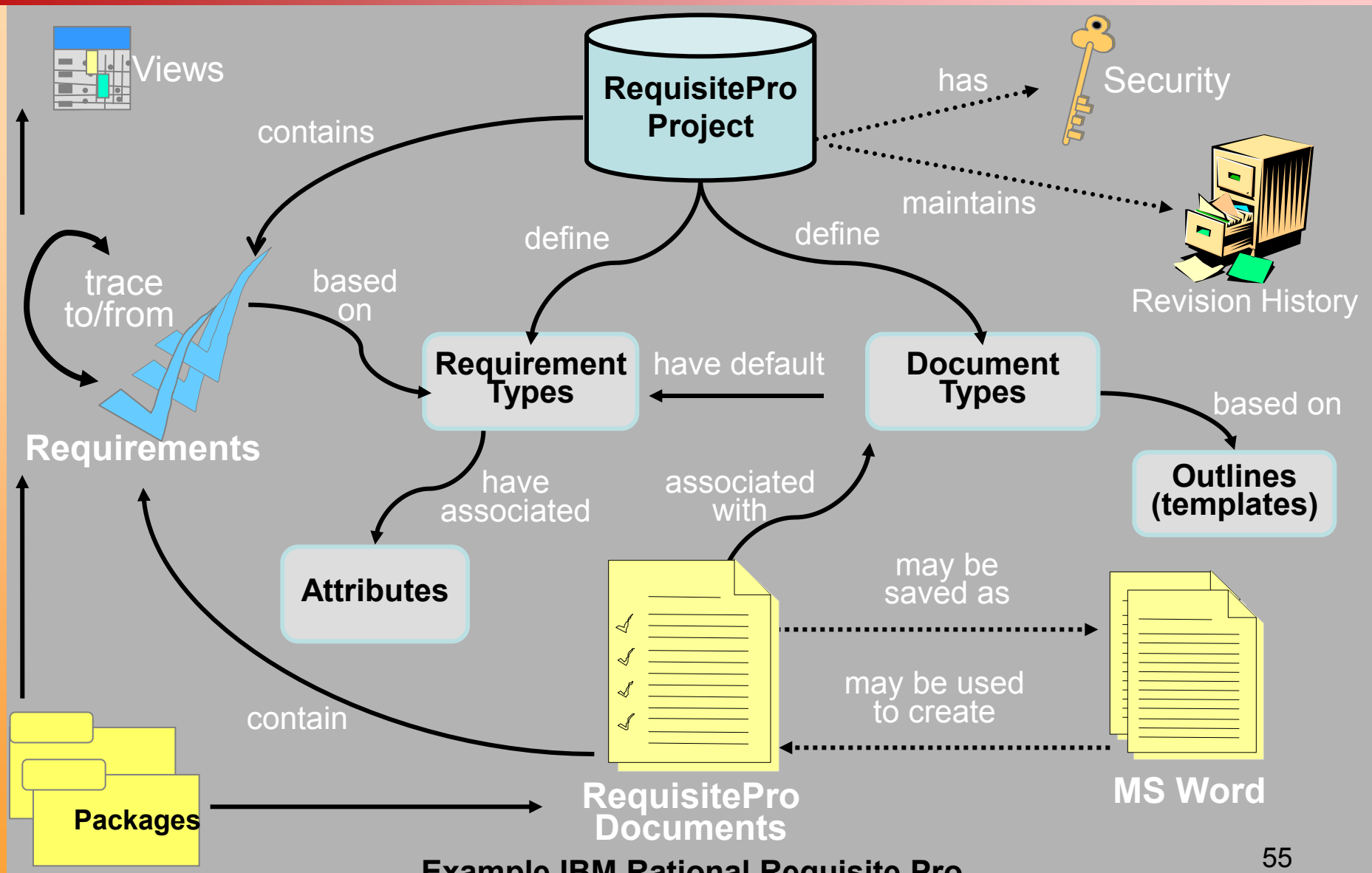


Requirements Management Challenges

- Accessibility of requirements by whole team
- Control of change
- Communication of change
- Understanding of the impact of change
- …

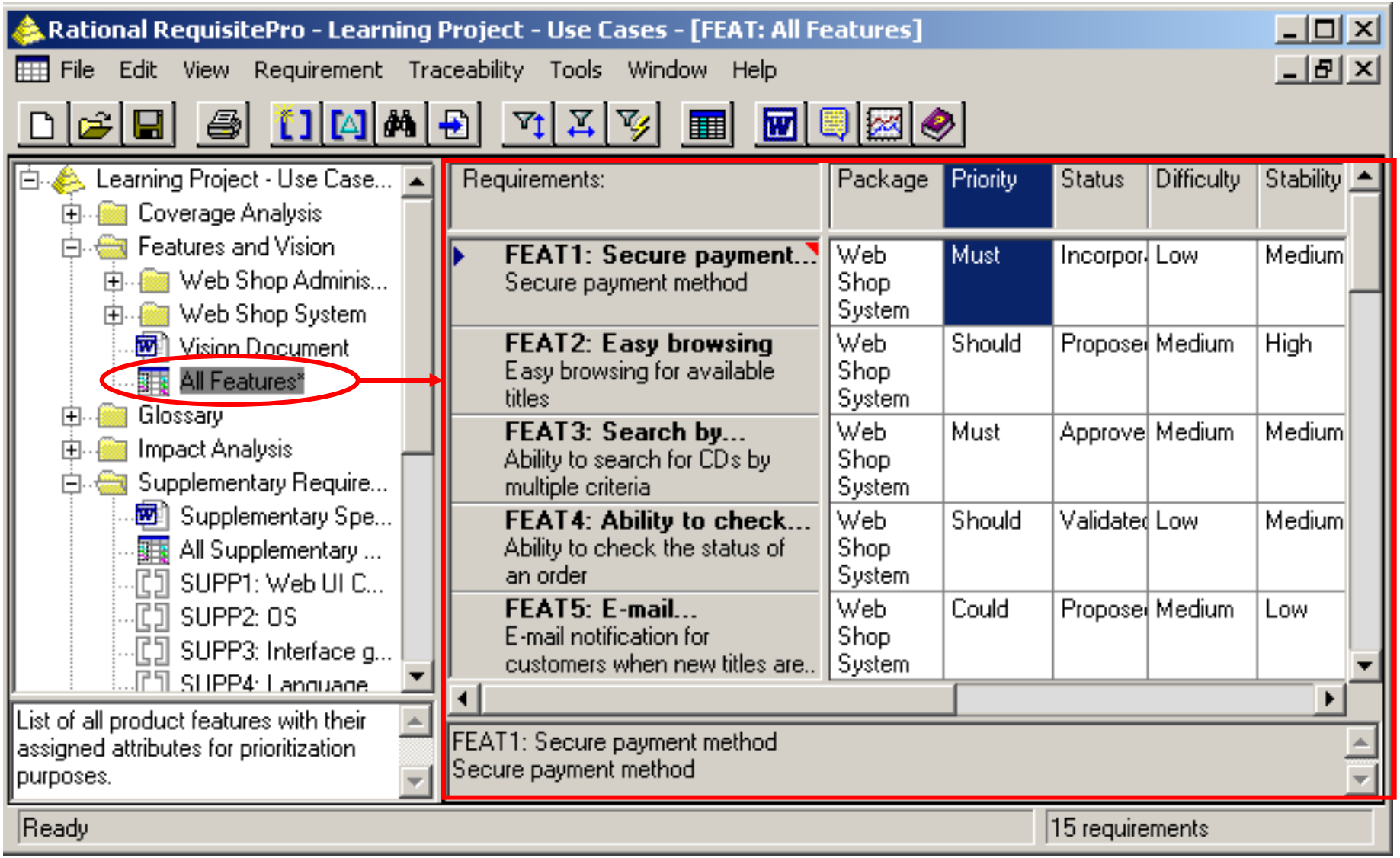


Tool- / Repository-based Approaches



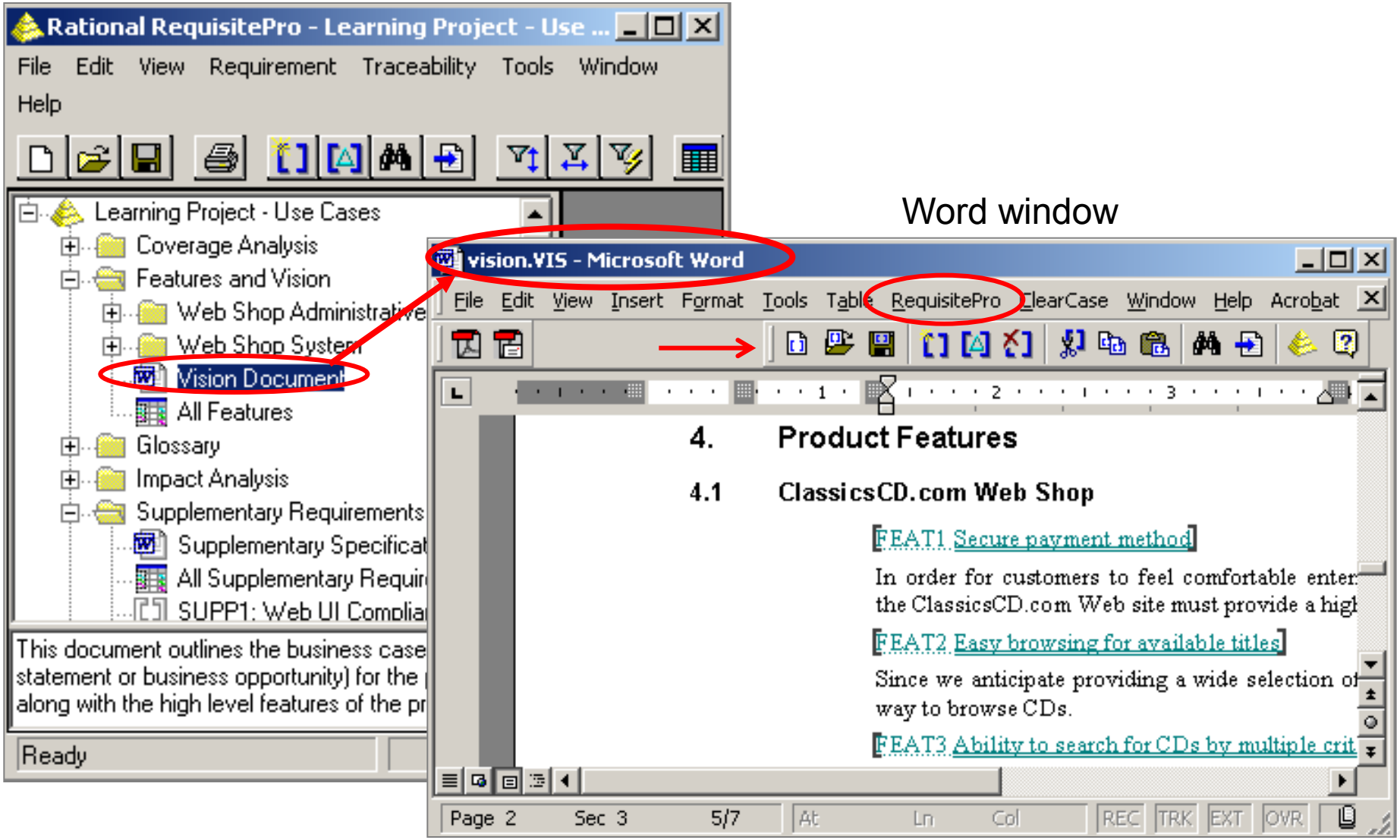
Example IBM Rational Requisite Pro

Working in Views



Example IBM Rational Requisite Pro

Working in a Document



Example IBM Rational Requisite Pro

Requirements Management Plan and Project Structure

- Define Requirements Management Strategy
  - Identify project artifacts that help plan and define the project
- Describe Requirements Engineering and Management relevant components for Project Structure
  - Requirement types
  - Requirement attributes and their values
  - Document types
  - Traceability criteria

Define Requirement Types

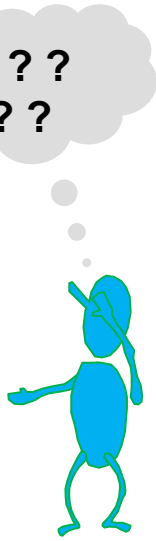
- Example: RequisitePro Requirements
  - Any tracked item: Inputs and outputs to the system; Functions of the system; Attributes of the system and its environment; Features; Use Cases; Supplementary requirements; Stakeholder requests; …

? **Environmental** **?** **Use cases** **Stakeholder Requests**

Features **?** **Usability** **?**

Functional **?** **Stakeholder Needs**

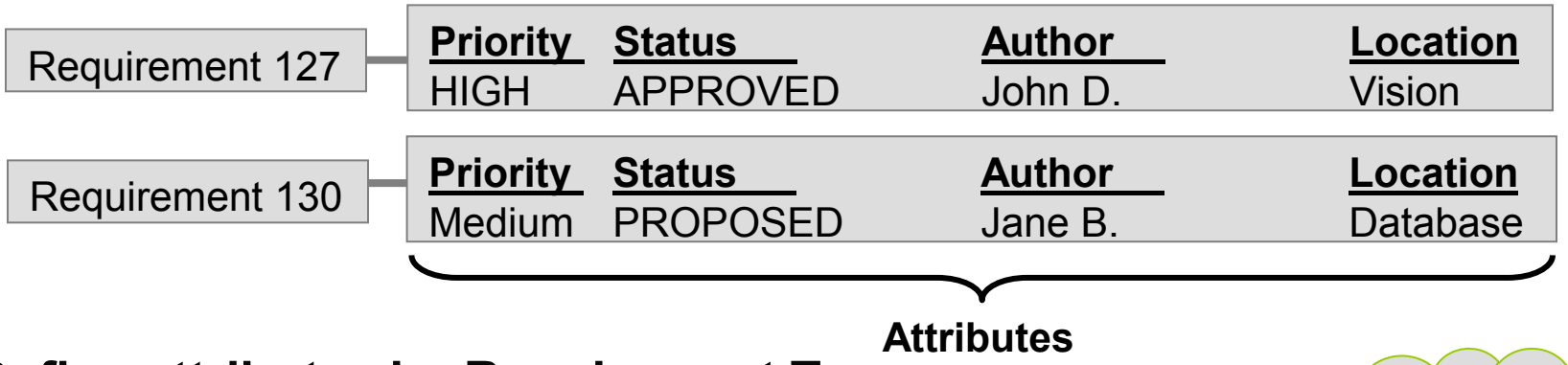
Reliability **Supplementary** **?**



- What types of project requirements do you want to document, track, manage?

Define Requirement Attributes

- **What Is a Requirement Attribute?**
  - Information attached to a requirement
  - Important details about the requirement



- **Define attributes by Requirement Type**
  - What information do you want to track?

- **Use Requirement Attributes to:**
  - Assign resources
  - Assess status
  - Calculate software metrics
  - Manage project risk
  - Estimate costs and time
  - Manage project scope
  - Prioritize requirements

Organize Document Types

- Define the types of documents you want to create:

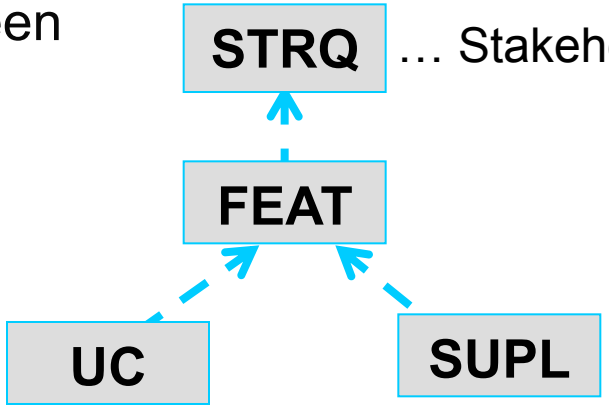
Glossary	RM Plan
Vision	Supplementary Specification
Use Cases	…

- Identify which Requirement Type will be captured in each Document Type:

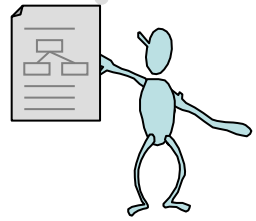
TERM	➔ Glossary	RMP	➔ RM Plan
FEAT	➔ Vision	SUPL	➔ Supplementary Specification
UC	➔ Use Cases	…	

Define Traceability Relationships

- What is Requirements Traceability?
  - A relationship between two requirements



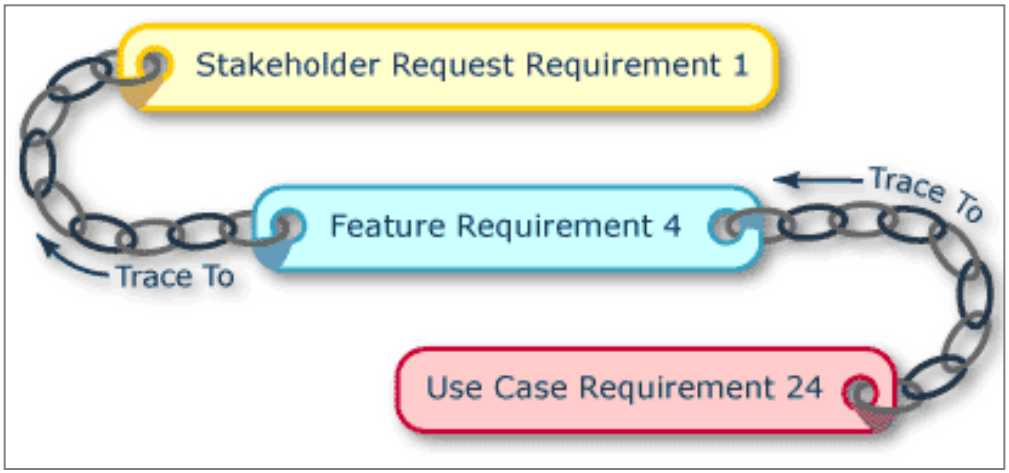
Reminder: This is defined and documented in the RM Plan.



- Use:
  - Query requirements data (based on traceability links and requirements attributes) :
    - Project status (schedule, budget, progress)
    - Impact analysis
    - Coverage analysis
    - Feature creep
  - Create requirement statistics (metrics)

Traceability Relationships

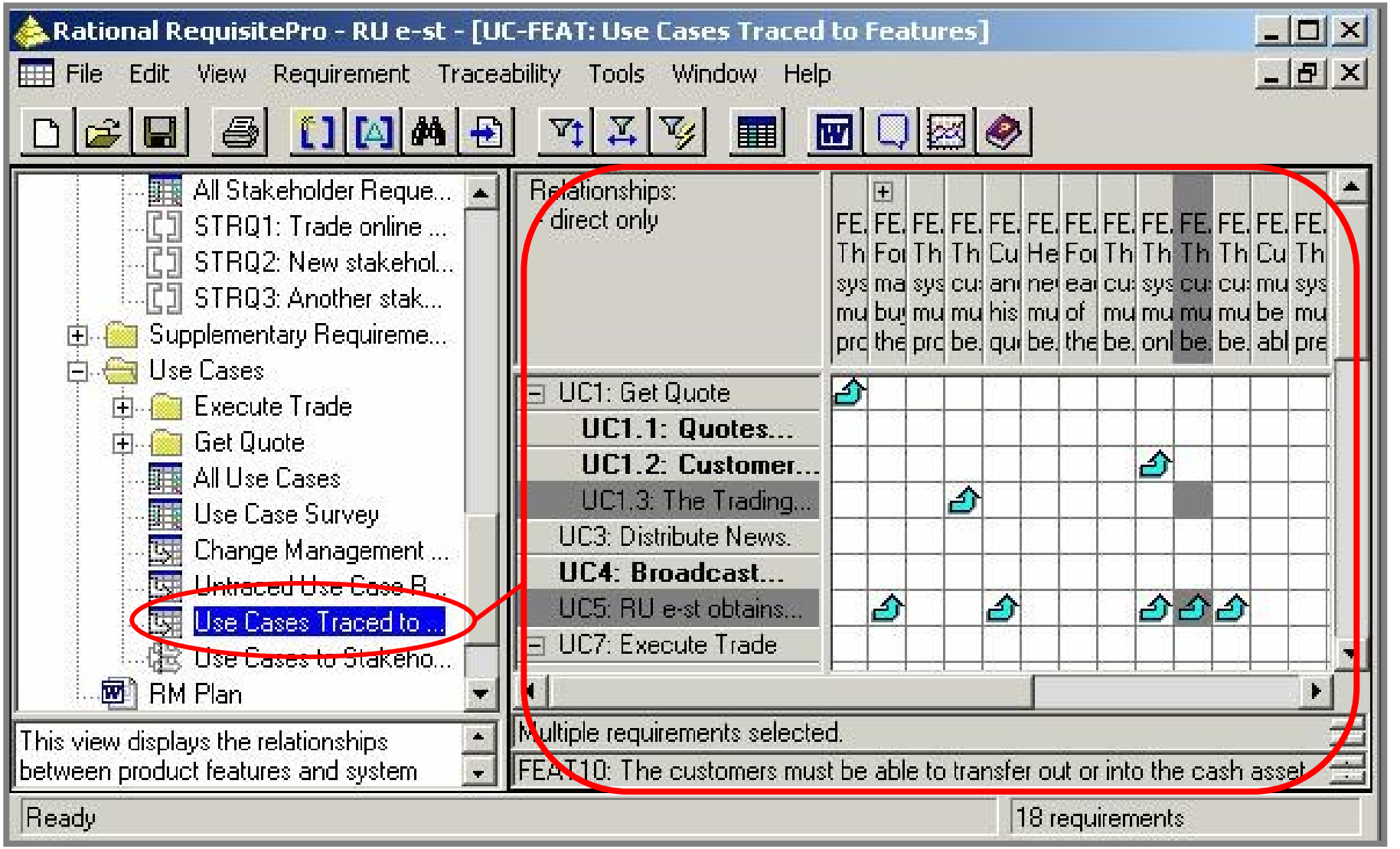
- Link two requirements to each other



- Help manage change
- Display in views:
  - Traceability Matrix
  - Traceability Tree

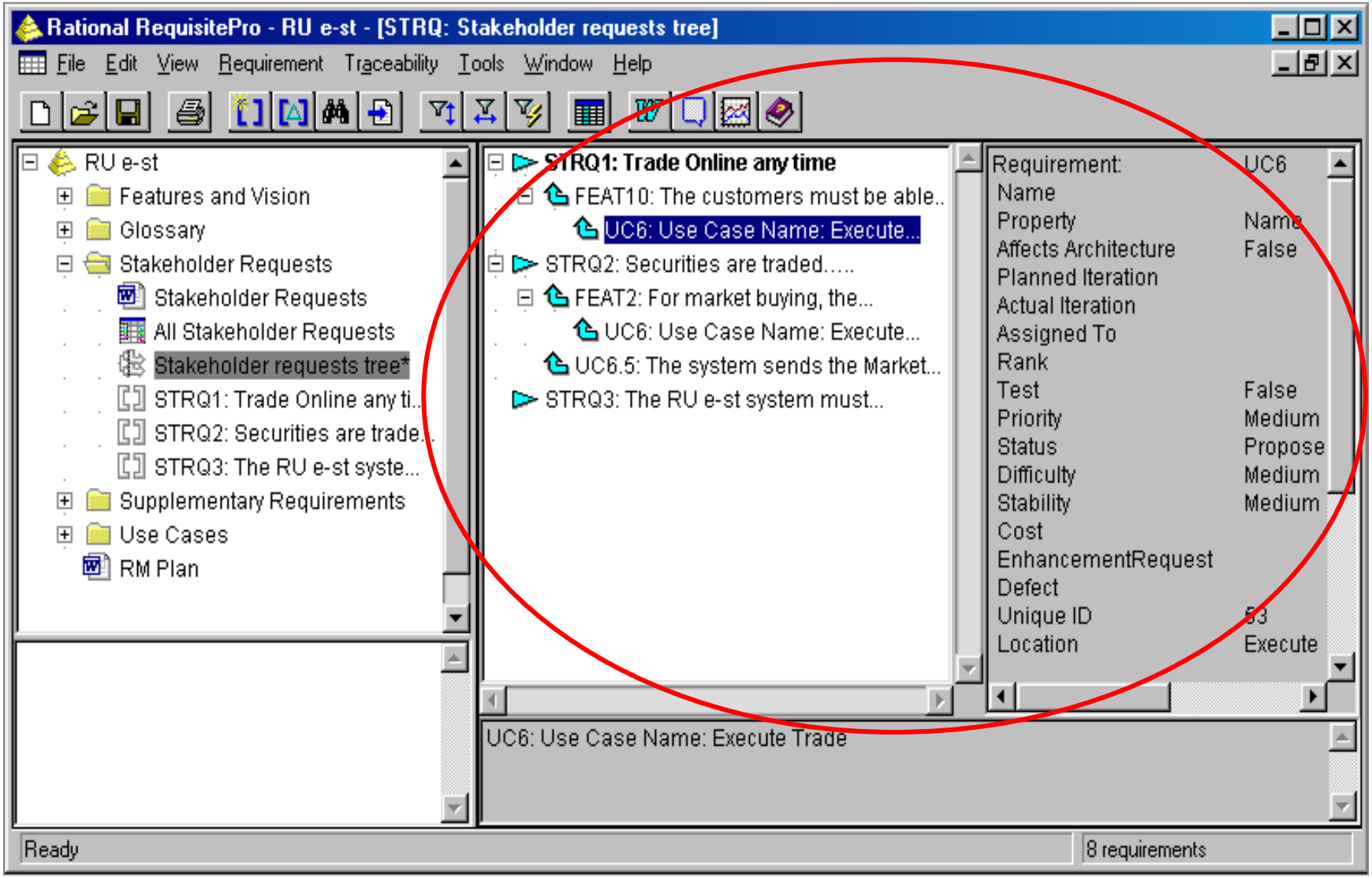
Views are created by Requirement Type

Example: Traceability Matrix View



Example IBM Rational Requisite Pro

Example: Traceability Tree View



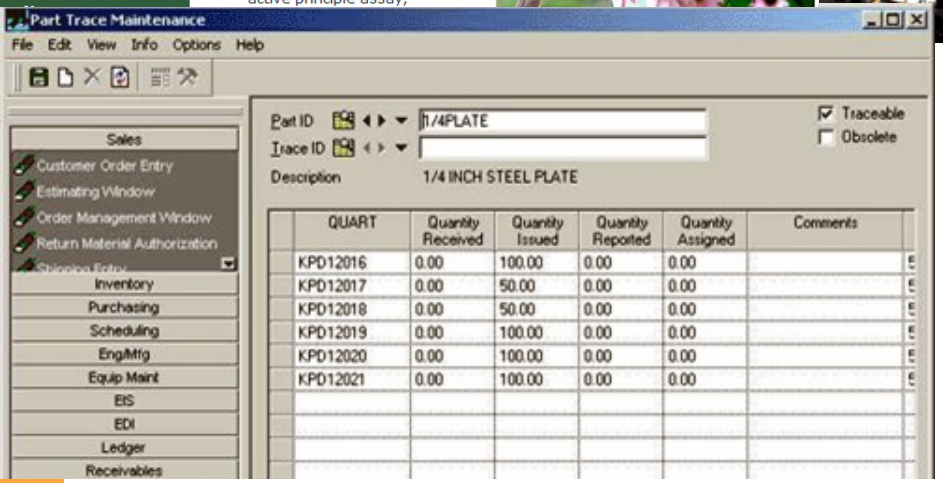
Example IBM Rational Requisite Pro

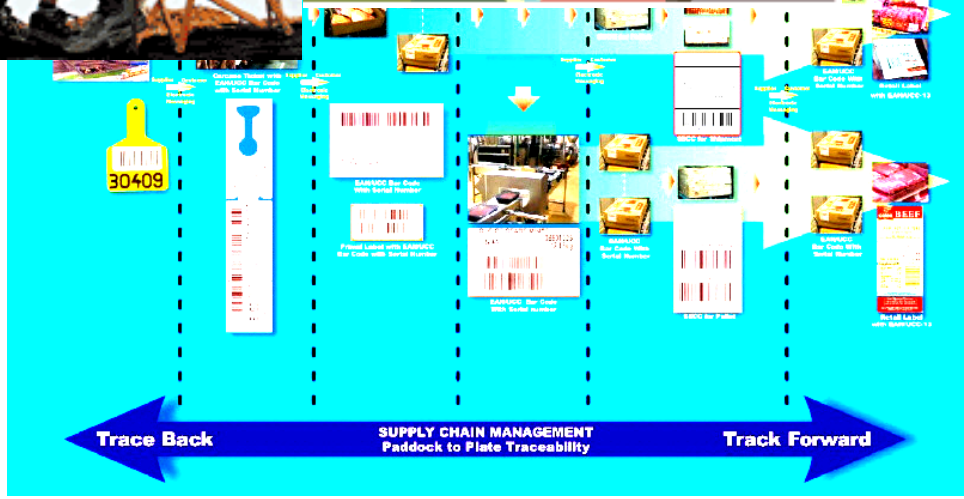
Traceability Definitions

- „… ability to describe and follow the life of a requirement in both a forward and backward direction“
  (Gotel & Finkelstein, ICRE 1994)
- „The degree to which a relationship can be established between two or more products of the development process“ (IEEE standard glossary)

- Numerous standards require traceability
  - ISO 15504, CMMI, IEEE Std 830
  - Many companies are mandated to implement traceability

Traceability is relevant in many domains

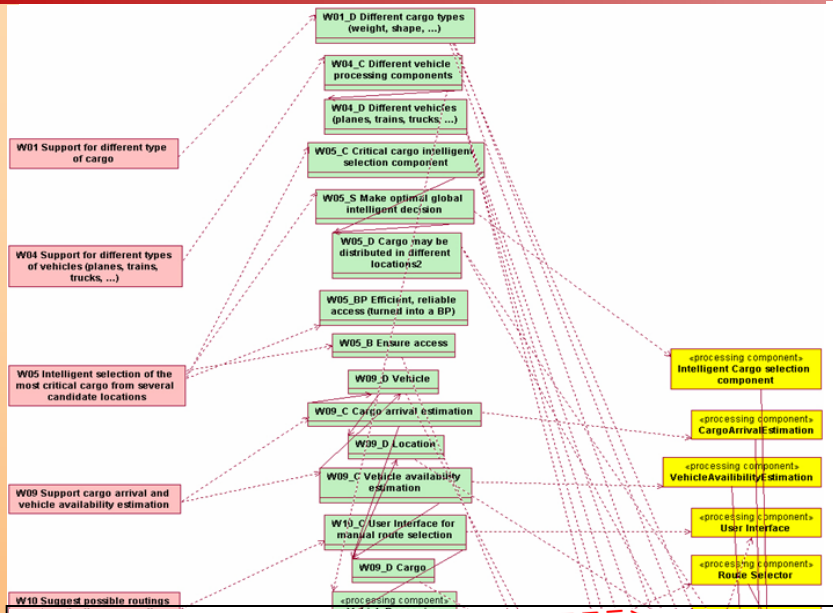




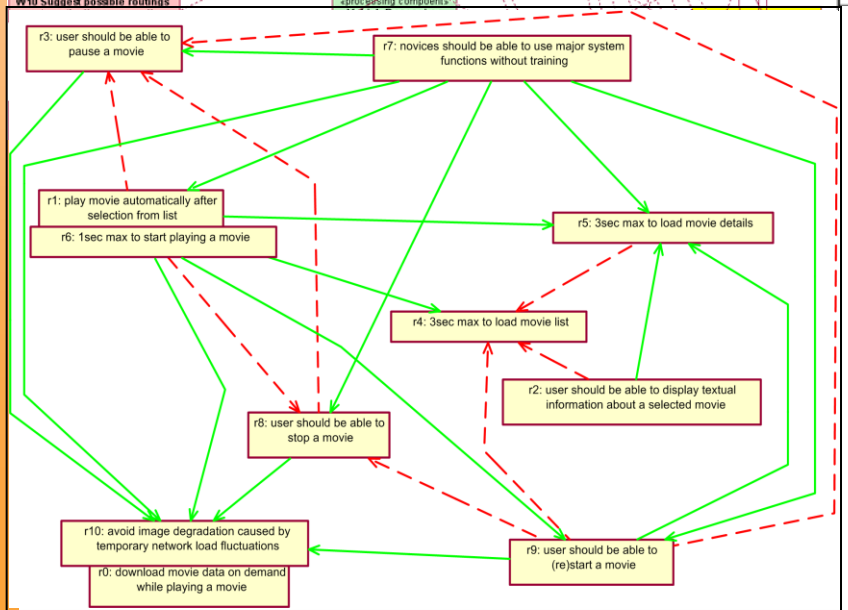
Traceabilty helps to answer questions such as

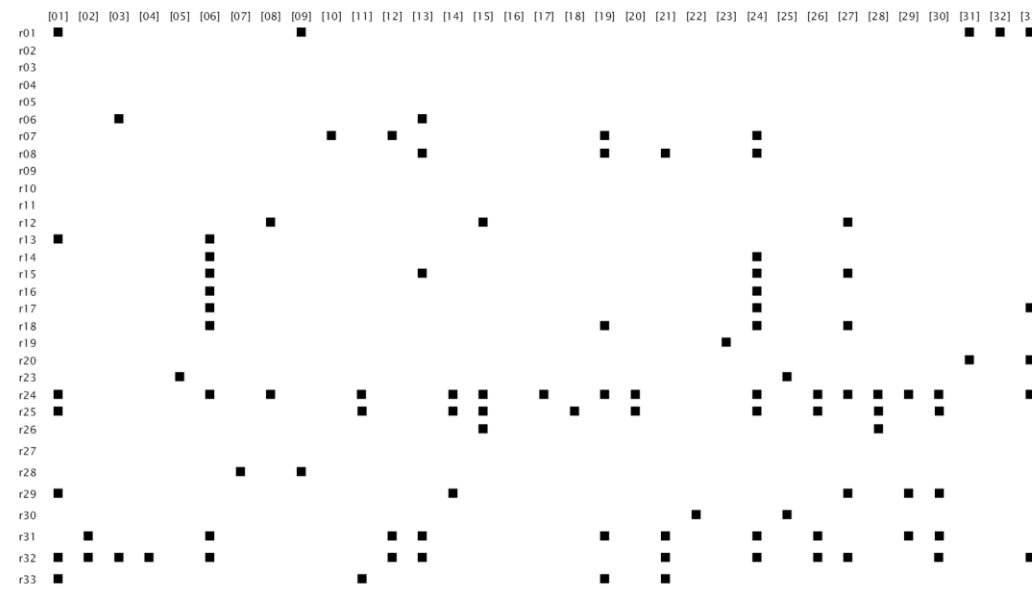
- What is the impact of changing this requirement?
- Why is this component part of the system?
- May I delete this table from the database?
- Which requirements are not covered by test cases?
- Are we finished already?

Traceability Techniques



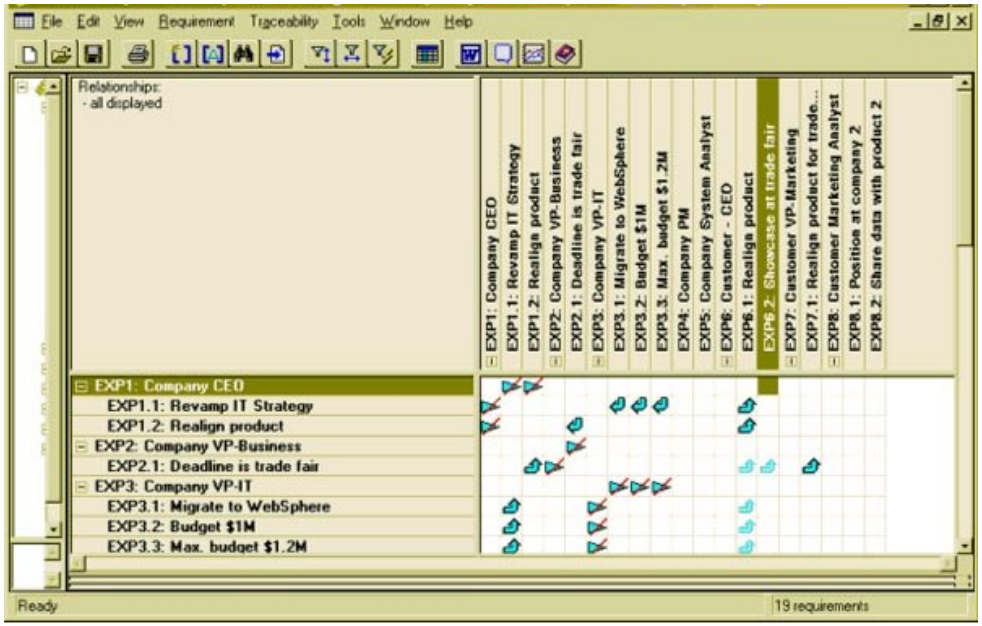
Requirement Numbers	reqs tested	REQ1 UC 1.1	REQ1 UC 1.2	REQ1 UC 1.3	REQ1 UC 2.1	REQ1 UC 2.2	REQ1 UC 2.3.1	REQ1 UC 2.3.2	REQ1 UC 2.3.3	REQ1 UC 2.4	REQ1 UC 3.1	REQ1 UC 3.2	REQ1 TECH 1.1
Test Cases	21	3	2	3	1	1	1	1	1	1	2	3	1
tested implicitly	0												
1.1.1	1	x											
1.1.2	2		x	x									
1.1.3	2	x											x
1.1.4	1			x									
1.1.5	2	x											
1.1.6	1		x										
1.1.7	1			x									
1.2.1	2				x		x						
1.2.2	2					x		x					





Traceability Observations

- Enormous complexity, high costs
- No widespread use in (small) software companies
- Used typically if mandated by standards
- Keeping trace links up-to-date even harder than creating them
- Tools manage manually acquired links
- Weak tool support for
  - Trace acquisition
  - Trace utilization



Outline

"Requirements Specification":

- ✓ Introduction
- ✓ Content of Software Requirements Specifications
  - ✓ IEEE Std. 830
  - ✓ Volere Template
- ✓ Writing Good Requirements
- ✓ Requirements Management and Traceability
- ➢ **Requirements on Requirements Management Tools**
- ▪ Sample Software Requirements Specification
- ▪ Exercise 3: Requirements Inspection and SRS Evaluation

INCOSE Requirements Management Tools Survey

- Performed by the **Tools Database Working Group** (TDWG) of the **International Council on Systems Engineering** (INCOSE)

- Gathering of information on Requirements Management tools since the 1990's
  - Survey questions for Requirements Management tools developed and enhanced by INCOSE/TDGW
  - Survey responses, including rating of compliance with each question or feature, provided directly by tool vendors
  - TDWG reserves the right to review and correct any "informational injustices" (i.e. exaggerated answers)
- Results provided through TDWG's **Tools Database --** unfortunately: *"… currently not supported, and INCOSE does not have a timeline for its return"*
  (http://oldsite.incose.org/ProductsPubs/products/toolsdatabase.aspx)

… however:
a rich source for **requirements** on **Requirements Management Tools** !!!

INCOSE RM Tools Survey: Categories, Sample Questions

1. **Capturing Requirements/ identification**
2. **Capturing system element structure**
3. **Requirements flowdown**
4. **Traceability analysis**
5. **Configuration Management**
6. **Documents and other output media**
7. **Groupware**
8. **Interfaces to other tools**
9. **System Environment**
10. **User Interfaces**
11. **Standards**
12. **Support and Maintenance**
13. **Training**
14. **Other Comments**

1. Capturing Requirements/identification
1.1. Input document enrichment/analysis: Using existing document information (such as glossary, index, etc.), aid the user in requirements analysis, identification of requirements, etc.
1.1.1. Input document change/comparison analysis: The ability to compare/contrast two different versions of a source document.
1.2. Automatic parsing of requirements: A mechanism for automatic identification of requirements by key words, structure, unique identifiers, etc. to create requirements from the text.
1.3. Interactive/semi-automatic requirement identification: The ability to identify requirements from a text file via interactive means such as mouse highlighting of the requirement text or prompting by the system "is this a requirement?"
1.4. Manual requirement identification: A manual means of identifying or creating requirements.
1.5. Batch mode operation: A mechanism for inputting/identifying requirements from outside of the tool.
1.5.1. Batch-mode document/source-link update: Does the tool have the ability to update existing linked documents from new/changed versions of the source documents without having to re-establish traceability links.
1.6. Requirement classification: Does the tool have the ability to classify/categorize requirements during identification

INCOSE RM Tools Survey: Responses IBM Rational DOORS v9.2

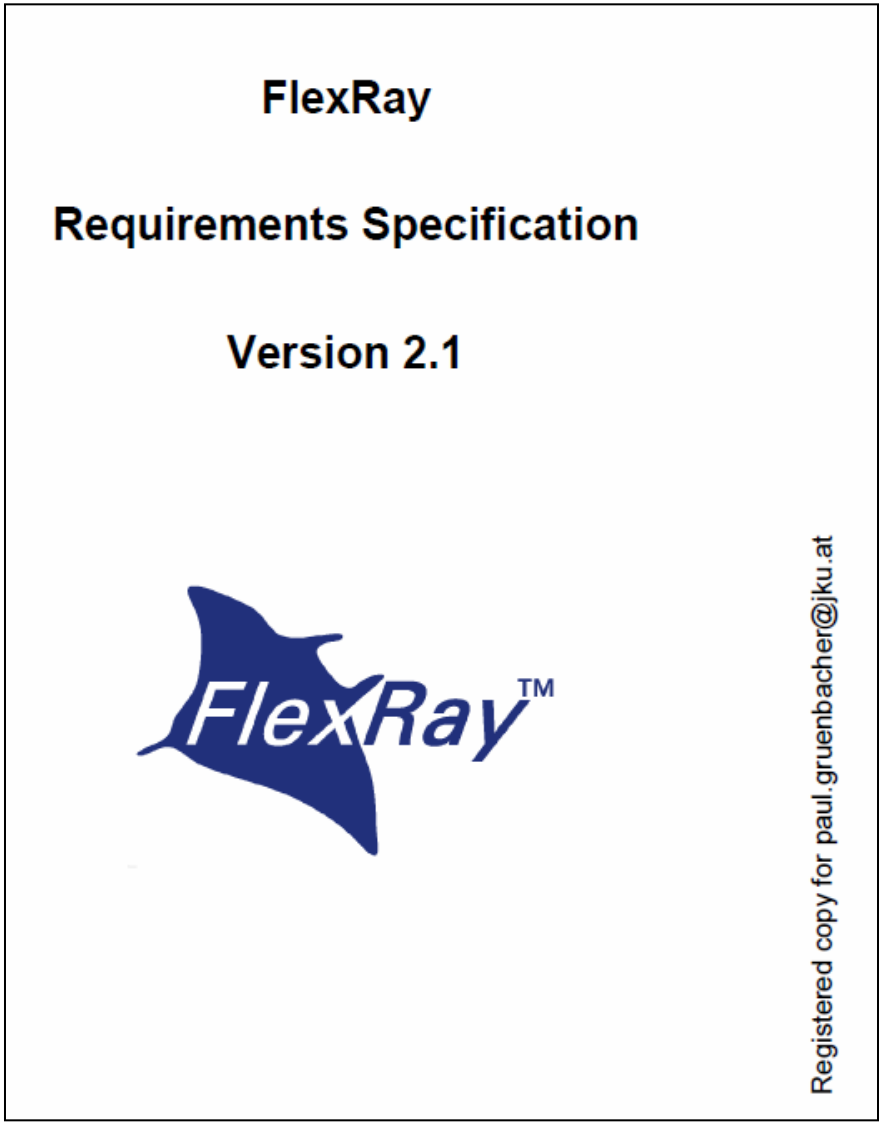
Requirements Management Survey Questions	Compliance (Full, Partial, None)	Comments
1. Capturing Requirements/identification		
1.1. Input document enrichment/analysis: Using existing document information (such as glossary, index, etc.), aid the user in requirements analysis, identification of requirements, etc.	Full	Automatic input parsers analyze text for keywords and create attributes for recognized data such as references and security classification. Following parsing requirements can be automatically labeled based on any search criteria.
1.1.1. Input document change/comparison analysis: The ability to compare/contrast two different versions of a source document.	Full	The spreadsheet import does automatic updates of new versions. Other parsers may be user modified to compare existing with new data. Also, a document compare function can be used to compare two documents that have been separately imported
1.2. Automatic parsing of requirements: A mechanism for automatic identification of requirements by key words, structure, unique identifiers, etc. to create requirements from the text.	Full	Multiple parsers are available to read all kinds of data. All parsers may be configured to fit the users' particular needs.
1.3. Interactive/semi-automatic requirement identification: The ability to identify requirements from a text file via interactive means such as mouse highlighting of the requirement text or prompting by the system "is this a requirement?"	Full	Automatic parsers will recognize requirements without intervention unless input data is ambiguous in which case the user will be prompted
1.4. Manual requirement identification: A manual means of identifying or creating requirements.	Full	Requirements can be entered manually into DOORS.
1.5. Batch mode operation: A mechanism for inputting/identifying requirements from outside of the tool.	Full	Full batch loading of requirements from multiple source formats is provided
1.5.1. Batch-mode document/source-link update: Does the tool have the ability to update existing linked documents from new/changed versions of the source documents without having to re-establish traceability links.	Full	Requirements that are updated, either directly or in batch operations, retain their links. New versions of documents may be used to update the requirements, however, the use of constant requirements identifiers in the source documents significantly aids the process. It should be noted however, that DOORS provides a fully featured Microsoft Word-like editing environment to negate the need for external modification and in many instances remove the need for repeated update from external sources. Where needed links can also be loaded in batch mode from external files
1.6. Requirement classification: Does the tool have the ability to classify/categorize requirements during identification	Full	DOORS provides supports for user-defined attributes. An attribute to classify /categorize requirements can be created.

Outline

"Requirements Specification":

- ✓ Introduction
- ✓ Content of Software Requirements Specifications
  - ✓ IEEE Std. 830
  - ✓ Volere Template
- ✓ Writing Good Requirements
- ✓ Requirements Management and Traceability
- ✓ Requirements on Requirements Management Tools
- ➢ **Sample Software Requirements Specification**
- ▪ Exercise 3: Requirements Inspection and SRS Evaluation

FlexRay Requirements Specification V2.1



FlexRay SRS V2.1 - Sample Evaluation (1/5)

- **General Observations:**
  - Real-world Example
  - SRS according Volere
  - SRS Purpose:
    - Publication of Specification
    - → Requirements plus additional explanations

FlexRay SRS V2.1 - Sample Evaluation (2/5)

- Evaluation against **IEEE Std. 830** "Characteristics of a Good SRS" (cf. slide 13)
- Evaluation Results - Overview

Correct	Unambiguous	Complete	Consistent	Ranked	Verifiable	Modifiable	Traceable
0	2	0	1	1	2	1	1

- Used Scale:
  - 1 ... Criterion **fully** fulfilled
  - 2 ... Criterion **largely** fulfilled
  - 3 ... Criterion **partly** fulfilled
  - 4 ... Criterion **not** fulfilled
  - 0 ... Criterion **not applicable**

FlexRay SRS V2.1 - Sample Evaluation (3/5)

- **Unambiguous: 2**
  - **In favor:**
    - Definition of terms
    - Definition of notation
    - Explanations (subordinate requirements)
  - **Problems:**
    - Use of prose
    - p.12, ID301: “state of the art automotive environments”
    - p.94, ID1708: “independent (as far as possible)”

FlexRay SRS V2.1 - Sample Evaluation (4/5)

- **Consistent: 1**
  - Not applicable sections marked as such
- **Ranked for importance and/or stability: 1**
  - **Importance:** Expressed via “Bindingness” (“shall/should/may”)

Bindingness	Keyword
The terms *shall* is used to define mandatory behavior. • The defined behavior is binding. • The compliance of the product with the requirement is a must. • The acceptance of the product can be refused if a shall-requirement is not met.	shall
The term *should* is used to describe desirable behavior. Wishes • are not binding. • do not need to be satisfied. • serve improved cooperation of stakeholders and developers. • increase stakeholder satisfaction.	should
The term *may* is used to define behavior that is acceptable, but not necessarily desirable. • Acceptable behaviour is not binding.	may
The term *(optionally)* is used for requirements for optional modules. If the modules are present then the bindingness of the requirement is according to the keyword as described above.	(optionally)

  - **Stability:** Sections on “Open Issues” and “Waiting Room”

FlexRay SRS V2.1 - Sample Evaluation (5/5)

- **Verifiable: 2**
  - Requirements are partly “abstract” and require interpretation within a concrete solution
    - p.25, ID426: 'FlexRay' shall support presence of up to two active stars in a system.
    - p.24, ID425: Presence of nodes on the link between 'repeater modules' is not required to be supported.
  - Linguistic ambiguities (cf. Criterion “Unambiguous”)
  - Referenceable requirements support requirements-based testing
- **Modifiable: 1**
- **Traceable: 1**

Outline

"Requirements Specification":

- ✓ Introduction
- ✓ Content of Software Requirements Specifications
  - ✓ IEEE Std. 830
  - ✓ Volere Template
- ✓ Writing Good Requirements
- ✓ Requirements Management and Traceability
- ✓ Requirements on Requirements Management Tools
- ✓ Sample Software Requirements Specification
- ➢ **Exercise 3: Requirements Inspection and SRS Evaluation**

Exercise 3: Requirements Inspection and SRS Evaluation

Analyze and evaluate the provided sample software requirements specification PACEMAKER and produce an evaluation report.

Perform the following steps:

1) Identify the purpose and context of the PACEMAKER requirements specification.
2) Identify how the structure and content of the PACEMAKER requirements specification match with the sample outlines for requirements specifications provided by IEEE Std 830-1998 and through the Volere Template.
3) Select a few examples of – if possible good and bad – requirements specified in the PACEMAKER requirements specification. Use therefor the criteria and material provided in the lecture on “Writing Good Requirements”.
4) Evaluate the overall PACEMAKER requirements specification against the “Characteristics of a Good SRS” as provided by IEEE Std. 830-1998. Use therefor the rating scale presented in the lecture.

Use a group-based approach for performing above steps, discuss and agree on your ratings in the group, and provide clear rationales and evidence for your judgments.

Submit a report of your evaluation via TUWEL. Each group is expected to submit a pdf document named Exercise3Team<YourTeamNumber>. Please list all team members on the front page of your report.

Outline

"Requirements Specification":

- ✓ Introduction
- ✓ Content of Software Requirements Specifications
  - ✓ IEEE Std. 830
  - ✓ Volere Template
- ✓ Writing Good Requirements
- ✓ Requirements Management and Traceability
- ✓ Requirements on Requirements Management Tools
- ✓ Sample Software Requirements Specification
- ✓ Exercise 3: Requirements Inspection and SRS Evaluation