CT70A2000
Requirements Engineering

Shola Oyedeji

LUT University

LECTURE 3

Writing Requirements

Requirements Management

Tracing Requirements

RE Tools

Validating Requirements

Characteristics of Excellent Requirements

- **Complete.** Each requirement must fully describe the functionality to be delivered. It must contain all the information necessary for the developer to design and implement that bit offunctionality.
- **Correct.** Each requirement must accurately describe the functionality to be built. A software requirement that conflicts with its parent system requirement is not correct. Only user representatives can determine the corretness of user requirements, which is why users or their close surrogates must review the requirements.
- **Feasible.** It must be possible to implement each requirement within the known capabilities and limitations of the system and its operating evironment. To avoid specifying unattainable requirements, have a developer work with marketing or the requirements analysts throughout the elicitation process.
- **Necessary.** Each requirement should document a capability that the cusotmers really need or one that's requirement for conformance to an external system requirement or a standard. Every requirement should originate from a source that has been authority to specify requirements. Trace each requirement back to specific voice-of-the-customer input.
- **Prioritized.** Assign an implementation priority to each functional requirement feature to indicate how essential it is to a particular product release. If all the requirements are considered equally important, it is hard for the project manager to respond to budget cuts, schedule overruns, personnel losses, or new requirements added during development
- **Unambiguous.** All readers of a requirement statement should arrive at a single, consistent interpretation of it, but natural language is highly prone to ambiuity. Write requirements in simple, concise, straightforward language appropriate to the user domain.
- **Verifiable.** See whether you can devise a few tests or use other verification approaches, such as inspection or demonstration, to determine whether the product properly implements each requirement. If a requirement isn't verifiable, determining whether it was correctly implemented becomes a matter of opinion.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

The system should send user notification of transaction status

The Software Requirements Specification (SRS)

a.k.a. Functional Specification, Product Specification, Requirements document, System Specifiation

- Precisely states the functions and capabilities that a software system must provide and the constraints that it must respect
- Creates basis for all subsequent project planning, design, and coding, as well as the foundation for system testing and user documentation
- It should describe as completely as necessary the system's behaviours under various conditions but NOT contain design, construction, testing, or project management details

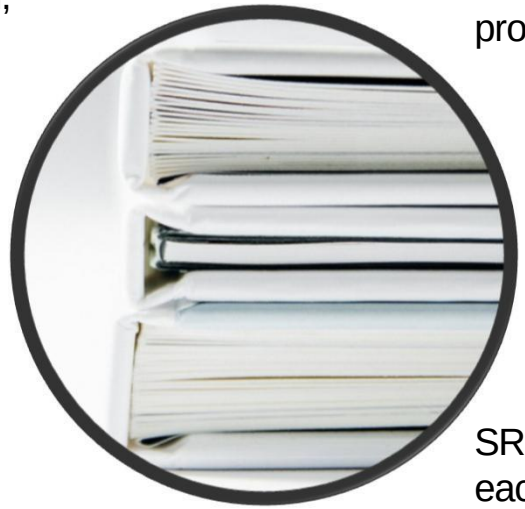
K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Stakeholders relying on SRS

Customers, the marketing deparment, and sales staff need to know what product they can expect to be delivered.

Project managers base their estimates of schedule, effort, and resources on the product description

Subcontractors base their work on, and can be legally held to, the SRS



Software development team need to know what to build

Legal staff ensure that the requirements comply with applicable laws and regulations

Testing group uses the SRS to develop test plans, cases, and procedures

Training personnel use the SRS and user documentation to develop educational materials

SRS tells **maintenance and support staff** what each part of the product is supposed to do

Documentation writers base user manuals and help screens on the SRS and the user interface design

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Requirements Specification Characteristics

- **Complete.** No requirements or necessary information should be absent.
- **Consistent.** Consistent requirements don't conflict with other requirements of the same type or with higher-level business, system, or user requirements. Disagreements between requirements must be resolved before development can proceed.
- **Modifiable.** You must be able to revise the SRS when necessary and to maintain a history of changes made to each requirement. This dictates that each requirement be uniquely labeled and expressed separately from other requirements so that you can refer to it unambiguously. Each requirement should appear only once in the SRS.
- **Traceable.** A traceable requirement can be linked backward to its origin and forward to the design elements and source code that implement it and to the testcases that verify the implementation as correct. Traceable requirements are uniquely labeled with persistent identifiers. They are written in a structured, fine-grained way as opposed to crafting long narrative paragraphs.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

1. **Introduction**
    1. Purpose
    2. Document conventions
    3. Intended audience and Reading Suggestions
    4. Project Scope
    5. References
2. **Overall Description**
    1. Product perspective
    2. Product Features
    3. User classes and characteristics
    4. Operating environment
    5. Design/implementation constraints
    6. User Documentation
    7. Assumptions and dependencies
3. **System Features**
    x. System feature X
        1. Description and priority
        2. Action/result
        3. Functional requirements
4. **External Interface Requirements**
    1. User interfaces
    2. Hardware interfaces
    3. Software interfaces
    4. Communication protocols and interfaces
5. **Other Nonfunctional Requirements**
    1. Performance requirements
    2. Safety requirements
    3. Security requirements
    4. Software quality attributes
6. **Other Requirements**

Appendix A: Terminology/Glossary/Definitions list
Appendix B: Analysis Models
Appendix C: Issues List

SRS Structure

There's not a "standard specifications template" for all projects in all industries

- the individual requirements that populate an SRS are unique not only from company to company, but also from project to project within any one company
- **select an existing template that you can fine - tune for your organizational needs (if you don't have one already)**

Read more from:
https://techwhirl.com/writing-software-requirements-specifications/

This SRS example is an adaptation and extension of the IEEE Standard 830-1998:

1. **Introduction**
    1. Purpose
    2. Document conventions
    3. Intended audience and Reading Suggestions
    4. Project Scope
    5. References
2. **Overall Description**
    1. Product perspective
    2. Product Features
    3. User classes and characteristics
    4. Operating environment
    5. Design/implementation constraints
    6. User Documentation
    7. Assumptions and dependencies
3. **System Features**
    x. System feature X
        1. Description and priority
        2. Action/result
        3. Functional requirements
4. **External Interface Requirements**
    1. User interfaces
    2. Hardware interfaces
    3. Software interfaces
    4. Communication protocols and interfaces
5. **Other Nonfunctional Requirements**
    1. Performance requirements
    2. Safety requirements
    3. Security requirements
    4. Software quality attributes
6. **Other Requirements**

Appendix A: Terminology/Glossary/Definitions list
Appendix B: Analysis Models
Appendix C: Issues List

SRS: 1. Introduction

1.Purpose: Identify the product whose requirements are specified in this document, including the revision or release number

2.Document Conventions: Describe any standards or typographical conventions, including text styles, highlighting, or significant notations.

3.Intended Audience: List the different readers to whom the SRS is directed. Describe what the rest of the SRS contain and how it is organized. Suggest a sequence for reading the document that is most appropriate for each type of reader

4.Project Scope: Provide a short description of the software being specified and its purpose Relate the software to user or corporate goals and to business objectives and strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here. An SRS that specifies an incremental release of an evolving product should contain its own scope statement as a subset of the long-term strategic product vision.

5.References: List any documents or other resources to which this SRS refers, including hyperlinks to them if possible. These might include user interface style guides, contracts, standards, system requirements specifications, use-case documents, interface specifications, concept-of-operations documents, or the SRS for a related product.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

1. **Introduction**
   1. Purpose
   2. Document conventions
   3. Intended audience and Reading Suggestions
   4. Project Scope
   5. References
2. **Overall Description**
   1. Product perspective
   2. Product Features
   3. User classes and characteristics
   4. Operating environment
   5. Design/implementation constraints
   6. User Documentation
   7. Assumptions and dependencies
3. **System Features**
   x. System feature X
      1. Description and priority
      2. Action/result
      3. Functional requirements
4. **External Interface Requirements**
   1. User interfaces
   2. Hardware interfaces
   3. Software interfaces
   4. Communication protocols and interfaces
5. **Other Nonfunctional Requirements**
   1. Performance requirements
   2. Safety requirements
   3. Security requirements
   4. Software quality attributes
6. **Other Requirements**

   Appendix A: Terminology/Glossary/Definitions list

   Appendix B: Analysis Models

   Appendix C: Issues List

SRS:
2. Overall Description

1.Product Perspective: Describe the product's context and origin. Is it the next member of a growing product family, the next version of a mature system, a replacement for an exiting application, or an entirely new product? If this SRS defines a component of a larger system, state how this software relates to the overall system and identify major interfaces between the two

2.Product Features: List the major features the product contains or the significant functions that it performs. You need only a high-level summary here. Details will be provided later in the SRS. A picture of the major groups of requirements and how they are related, such as a top-level data flow diagram, a use-case diagram or a class diagram, might be helpful.

3.User Classes and Characteristics: Identify the various user classes that you anticipate will use this product and describe their pertinent characteristics. Some requirements might pertain only to certain user classes. Identify the favored user classes.

4.Operating Environment: Describe the environment in which the software will operate, including the hardware platform, the operating systems and versions, and the geographical locations of users, servers, and databases. List any other software components or applications with which the system must peacefully coexist.

5.Design and Implementation Contraints: Describe any factors that will restrict the options available to the developers and the rationale for each constraint. Constraints might include: 1) Tools & technologies to be used or avoided, 2) Restrictions caused by the operating environment, 3) Required development conventions and standards, 4) Backward compatibility, 5) Limitations imposed by business rules or hardware, 6) Existing user interface conventions to be followed…

6.User Documentation: List the user documentation components that will be delivered along with the executable software. These could include user manuals, online help, and tutorials.

7.Assumptions and Dependencies: Problems can arise if assumptions are incorrect, are not shared, or change, so certain assumptions will translate into project risks. For example, one SRS reader might assume that the product will conform to a particular user interface convention, whereas another assumes something different. Identify any *dependencies* the project has on external factors outside its control, such as the release date of the next version of an operating system.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

1. **Introduction**
   1. Purpose
   2. Document conventions
   3. Intended audience and Reading Suggestions
   4. Project Scope
   5. References
2. **Overall Description**
   1. Product perspective
   2. Product Features
   3. User classes and characteristics
   4. Operating environment
   5. Design/implementation constraints
   6. User Documentation
   7. Assumptions and dependencies
3. **System Features**
   x. System feature X
      1. Description and priority
      2. Action/result
      3. Functional requirements
4. **External Interface Requirements**
   1. User interfaces
   2. Hardware interfaces
   3. Software interfaces
   4. Communication protocols and interfaces
5. **Other Nonfunctional Requirements**
   1. Performance requirements
   2. Safety requirements
   3. Security requirements
   4. Software quality attributes
6. **Other Requirements**

   Appendix A: Terminology/Glossary/Definitions list
   Appendix B: Analysis Models
   Appendix C: Issues List

SRS:
3. System Features

Organizing the requirements by the system features is just one of the possibilities. Other ways could be to organize by use case, mode of operations, user class, stimulus, response, object class, or functional hierarchy. Combinations of these elements are also possible, such as use cases within user classes. There is no single right choice.

x.System Feature X: State the name of the feature just a few words, such as “3.1 Spell Check”.

1.Description and Priority: Provide a short description of the feature and indicate whether it is of high, medium, or low priority.

2.Action/result: List the sequences of input stimuli (user actions, signals from external devices or other triggers) and system responses that define the behaviors for this features. These stimuli correspond to the initial dialog steps of use cases or to external system events.

3.Functional Requirements: Itemize the detailed functional requirements associated with this features. These are the software capabilities that must be present for the user to carry out the feature’s services or to perform a use case. Describe how the product should respond to anticipated error conditions and to invalid inputs and actions. Uniquely label each functional requirement.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

1. **Introduction**
   1. Purpose
   2. Document conventions
   3. Intended audience and Reading Suggestions
   4. Project Scope
   5. References
2. **Overall Description**
   1. Product perspective
   2. Product Features
   3. User classes and characteristics
   4. Operating environment
   5. Design/implementation constraints
   6. User Documentation
   7. Assumptions and dependencies
3. **System Features**
   x. System feature X
      1. Description and priority
      2. Action/result
      3. Functional requirements
4. **External Interface Requirements**
   1. User interfaces
   2. Hardware interfaces
   3. Software interfaces
   4. Communication protocols and interfaces
5. **Other Nonfunctional Requirements**
   1. Performance requirements
   2. Safety requirements
   3. Security requirements
   4. Software quality attributes
6. **Other Requirements**

   Appendix A: Terminology/Glossary/Definitions list
   Appendix B: Analysis Models
   Appendix C: Issues List

SRS: 4. External Interface Requirements

4.1. User Interface: Describe the logical characteristics of each user interface that the system needs. Some possible items to include are:

- References to GUI standards, or product family style guides that are to be followed
- Standards for fonts, icons, button labels, images, color schemes, field tabbing sequences, commonly used controls, and the like
- Screen layout or resolution constraints, standard buttons, functions, or navigation links that will appear on every screen, such as help button
- Shortcut keys, message display conventions, layout standards to facilitate software localization, accommondations for visually impaired users

2.Hardware Interfaces: Describe the characteristics of each interface between the software and hardware components of the system. This description might include the supported device types, the data and control interactions between the software and the hardware, and the communication protocols to be used.

3.Software Interfaces: Describe the connections between this product and other software components (identified by name and version), including databases, operating systems, tools, libraries, and integrated commercial components. State the purpose of the messages, data, and control items exchanges between the software components. Describe the services needed by external software components and the nature of the intercomponent communications. Identify data that will be shared across software components.

4.Communication Interfaces: State the requirements for any communication functions the product will use, including e-mail, Web browser, network communications protocols, and electronic forms. Define any pertinent message formatting. Specify communication security or encryption issues, data transfer rates, and synchronization mechanisms.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

1. **Introduction**
   1. Purpose
   2. Document conventions
   3. Intended audience and Reading Suggestions
   4. Project Scope
   5. References
2. **Overall Description**
   1. Product perspective
   2. Product Features
   3. User classes and characteristics
   4. Operating environment
   5. Design/implementation constraints
   6. User Documentation
   7. Assumptions and dependencies
3. **System Features**
   x. System feature X
      1. Description and priority
      2. Action/result
      3. Functional requirements
4. **External Interface Requirements**
   1. User interfaces
   2. Hardware interfaces
   3. Software interfaces
   4. Communication protocols and interfaces
5. **Other Nonfunctional Requirements**
   1. Performance requirements
   2. Safety requirements
   3. Security requirements
   4. Software quality attributes
6. **Other Requirements**

   Appendix A: Terminology/Glossary/Definitions list
   Appendix B: Analysis Models
   Appendix C: Issues List

SRS: 5. Other Nonfunctional Requirements

1.Performance Requirements: State specific performance requirements for various system operations. Explain their rationale to guide the developers in making appropriate design choices. For instance, stringent database response time demands might lead the designers to mirror the database in multiple geographical locations or to denormalize relational database tables for faster query responses. Specify the number of transactions per second to be supported, response times, computational accuracy, and timing relationships for real-time systems. You could also specify memory and disk space requirements, concurrent user loads, or the maximum number of rows stored in database tables. Quantify the performance requirements as specifically as possible.

2.Safety Requirements: Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as potentially dangerous actions that must be prevented. Identify any safety certifications, policies, or regulations to which the product must conform.

3.Security Requirements: Specify any requirements regarding security, integrity, or privacy issues that affect access to the product, use of the product, and protection of data that the product uses or creates. Security requirements normally originate in business rules, so identify any security or privacy policies or regulations to which the product must conform.

4.Software Quality Attributes: State any additional product quality characteristics that will be important to either customers or developers. These characteristics should be specific, quantitative, and verifiable. Indicate the relative priorities of various attributes such as ease of use over ease of learning, or portability over efficiency.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

1. **Introduction**
   1. Purpose
   2. Document conventions
   3. Intended audience and Reading Suggestions
   4. Project Scope
   5. References
2. **Overall Description**
   1. Product perspective
   2. Product Features
   3. User classes and characteristics
   4. Operating environment
   5. Design/implementation constraints
   6. User Documentation
   7. Assumptions and dependencies
3. **System Features**
   x. System feature X
      1. Description and priority
      2. Action/result
      3. Functional requirements
4. **External Interface Requirements**
   1. User interfaces
   2. Hardware interfaces
   3. Software interfaces
   4. Communication protocols and interfaces
5. **Other Nonfunctional Requirements**
   1. Performance requirements
   2. Safety requirements
   3. Security requirements
   4. Software quality attributes
6. **Other Requirements**

   Appendix A: Terminology/Glossary/Definitions list
   Appendix B: Analysis Models
   Appendix C: Issues List

SRS: 5. Other Nonfunctional Requirements

Building Non-functional Requirements (NFRs)

1.Identify Project Attribute: The first thing a team must consider is whether they are tackling the NFR's that relevant to the project in order to build success indicators. Attributes can fall under:

- Operational attributes:
  Usability, Availability, Confidentiality, Security, Efficiency
- Revisional attributes:
  Flexibility, Maintainability, Modifiability
- Transitional attributes:
  Portability, Reusability, Interoperability

2.Construct your Non-functional Requirement: NFRs are broad and important for project success. Functional requirement defines what a system should do, a non-functional requirement will define how we measure the success of the system. Therefore, it is important to consider the following:

- What are you measuring : application or system or project or process
- What attributes are required for such measurement such as security, maintainability
- What is the goal of such measure and what metrics will determine the success or failure of such goals in the system.

Example:

- We are measuring an e-learning system for LUT
- We are measuring the availability
- We want 100% uptime in the first year. Our measure is 100% uptime, our metric is uptime/first year.

1. **Introduction**
   1. Purpose
   2. Document conventions
   3. Intended audience and Reading Suggestions
   4. Project Scope
   5. References
2. **Overall Description**
   1. Product perspective
   2. Product Features
   3. User classes and characteristics
   4. Operating environment
   5. Design/implementation constraints
   6. User Documentation
   7. Assumptions and dependencies
3. **System Features**
   x. System feature X
      1. Description and priority
      2. Action/result
      3. Functional requirements
4. **External Interface Requirements**
   1. User interfaces
   2. Hardware interfaces
   3. Software interfaces
   4. Communication protocols and interfaces
5. **Other Nonfunctional Requirements**
   1. Performance requirements
   2. Safety requirements
   3. Security requirements
   4. Software quality attributes
6. **Other Requirements**
   Appendix A: Terminology/Glossary/Definitions list
   Appendix B: Analysis Models
   Appendix C: Issues List

SRS: 6. Other Requirements

Define any other requirements that are not covered elsewhere in the SRS. Examples include internationalization requirements and legal requirements. You could also add sections on operations, administration, and maintenance to cover requirements for product installation, configuration, startup and shutdown, recovery and fault tolerance, and logging and monitoring operations.

Appendix A: Glossary: Define any specialized terms that a reader needs to know to properly interpret the SRS, including acronyms and abbreviations. Spell yout each acronym and provide its definition. Consider building an enterprise-level glossary that spans multiple projects. Each SRS would then define only those terms that are specific to an individual project.

Appendix B: Analysis Models: This optional section includes or points to pertinent analysis models such as dataflow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.

Appendix C: Issues List: This is a dynamic list of the open requirements issues that remain to be resolved. Issues could include items flagged as TBD, pending decision, information that is needed, conflicts awaiting resolution, and the like.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Guidelines for Writing Requirements

- Write complete sentences that have proper grammar, spelling, and punctuation. Keep sentences and paragraphs short and direct.
- Use the active voice
- Use terms consistently and as defined in the glossary. Watch out for synonyms and near-synonyms. The SRS is not a place to creatively vary your language in an attempt to keep the reader's interest
- Decompose a vague top-level requirements into sufficient detail to clarify it and remove ambiguity
- State requirements in a consistent fashion, such as "The system shall" or "The user shall", followed by an action verb, followed by the observable result.
- Specify the trigger condition or action that causes the system to perform the specified behavior.
- When stating a requirement in the form "The user shall" identify the specific actor whenever possible (for example, "the Buyer shall…").
- Use lists, figures, graphs, and tables to present information visually. Readers glaze over when confronting a dense mass of turgid text
- Emphasize the most important bits of information. Techniques for emphasis include graphics, sequence (the first item is emphasized), repetition, use of white space, and use of visual contrast such as shading
- Ambiguous language leads to unverifiable requirements, so avoid using vague and subjective terms.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Don’t use weak and unprecise words

- About
- Applicable
- Approximate
- Bad
- Best
- Best Practice
- Between
- Clearly
- Could
- E.G.
- Easily
- Easy
- Effective
- Efficient
- Enable
- Etc.
- Excellent
- Fast
- Finest
- Good
- High
- His / Her
- Ideal
- Include But Shall Not Be Limited To
- Least
- Like
- Low
- Maximize
- May
- Minimum
- Mostly
- Relevant
- Same
- Should
- Similar
- So As
- State Of The Art
- Sufficient
- Support
- Support
- Target
- User Friendly
- Worse
- Would

Example

What weaknesses can you identify from following requirement? Howwould you improve it?

”The Background Task Manager shall provide status messages at regular intervals not less than every 60 seconds”

Possible improvement:

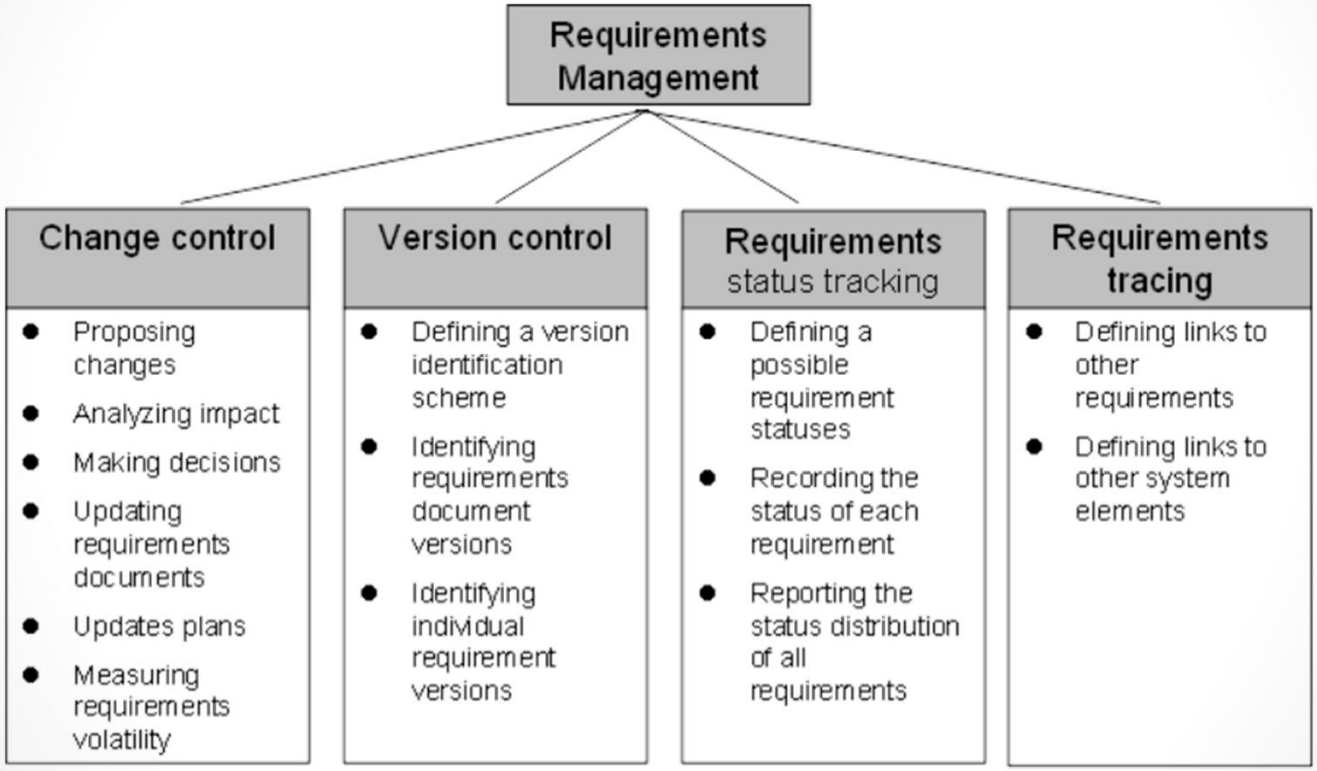
1. The Background Task Manager (BTM) shall display status messages in a designated area of the user interface
    1. *The messages shall be updated every 60 plus or minus 10 seconds after background task processing begins*
    2. *The message shall remain visible continuously*
    3. *Whenever communication with the background task process is possible, the BTM shall display the percent completed on the background task*
    4. *The BTM shall display a ”Done” message when the background task is completed*
    5. *The BTM shall display a message if the bakground task has stalled*

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

LECTURE 3

Writing Requirements
Requirements Management
Tracing Requirements
RE Tools
Validating Requirements

Requirements Management Activities



K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Agreeing on Requirements

Sign-off vs. Establishing Requirements Baseline

Risks of sign-off:

Customer: *I sign because otherwise the the developer's won't start coding.*

Development: *This is what we will build. Requirements are now frozen.*

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Establishing Requirements Baseline

"I agree that this document represents our best understanding of the requirements for this project today and that the system described will satisfy our needs. I agree to make future changes in this baseline through the project's defined change process. I realize that approved changes might require us to renegotiate the costs, resource, and schedule commitments for this project"

Benefits:

- Customer management is confident that the project scope won't explode out of control, because customers manage the scope change decisions
- User representatives have confidence that development will work with them to deliver the right system, even if the representatives didn't think of every requirement before construction began.
- Development management has confidence because the development team has a business partner who will keep the project focused on achieving its objectives and will work with development to balance schedule, cost, functionality, and quality
- Requirements analysts are confident because they know that they can manage changes to the project in a way that will keep chaos to a minimum.

Requirements Baseline

- Set of functional and nonfunctional requirements that the development team has committed to implement in a specific release
- Gives the project stakeholders a shared understanding of the capabilities and properties they can expect to see in the release
- Changes to the baseline can be made only through the project's defined change-control process.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Change Control

- The project incorporates requirements changes in a consistent fashion
- Proposed requirements changes are carefully evaluated before committed to
- The appropriate individuals make informed business decisions about requested changes
- Approved changes are communicated to all affected participants
- Requirements Analyst should incorporate approved changes in the project's requirements documentation
  - requirements documentation should always be accurate and up to date

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

The Change Control Board (CCB)

- Decides which of the proposed requirement changes and newly suggested features are accepted for inclusion in the product
- Also decides which reported defects to correct and when to correct them.
- Reviews and approves changes to any baselined work product
- Typically consists members from the following areas:
  - Project or program management
  - Product management or requirements analyst
  - Development
  - Testing or quality assurance
  - Marketing or customer representatives
  - User documentation
  - Technical support or help desk
  - Configuration management

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Proposed Template for Describing Change Control Process

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions
2. Roles and Responsibilities
3. Change-Request Status
4. Entry Criteria
5. Tasks
   - 5.1 Evaluate Request
   - 5.2 Make Decision
   - 5.3 Make Change
   - 5.4 Notify All Affected Parties
6. Verification
   - 6.1 Verify Change
   - 6.2 Install Product
7. Exit Criteria
8. Change-Control Status Reporting

Appendix: Data Items Stored for Each Request

1. Introduction

- Describes the purpose of this process and identifies the organizational scope to which it applies.
- If this process covers changes only in certain work products, identify them here.
- Also indicate whether any specific kinds of changes are exempted, such as changes in interim or temporary work products created during the course of a project.
- Define any terms that are necessary for understanding the rest of the document in section 1.3.

https://www.jamasoftware.com/blog/a-change-control-process-description/

Proposed Template for Describing Change Control Process

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions
2. Roles and Responsibilities
3. Change-Request Status
4. Entry Criteria
5. Tasks
   - 5.1 Evaluate Request
   - 5.2 Make Decision
   - 5.3 Make Change
   - 5.4 Notify All Affected Parties
6. Verification
   - 6.1 Verify Change
   - 6.2 Install Product
7. Exit Criteria
8. Change-Control Status Reporting

Appendix: Data Items Stored for Each Request

2. Roles and Responsibilities

List the project team members—by role, not by name—who participate in the change-control activities and describe their responsibilities. Different individuals need not fill each role. For example, the CCB Chair might also receive submitted change requests. Several—perhaps all—roles can be filled by the same person on a small project.

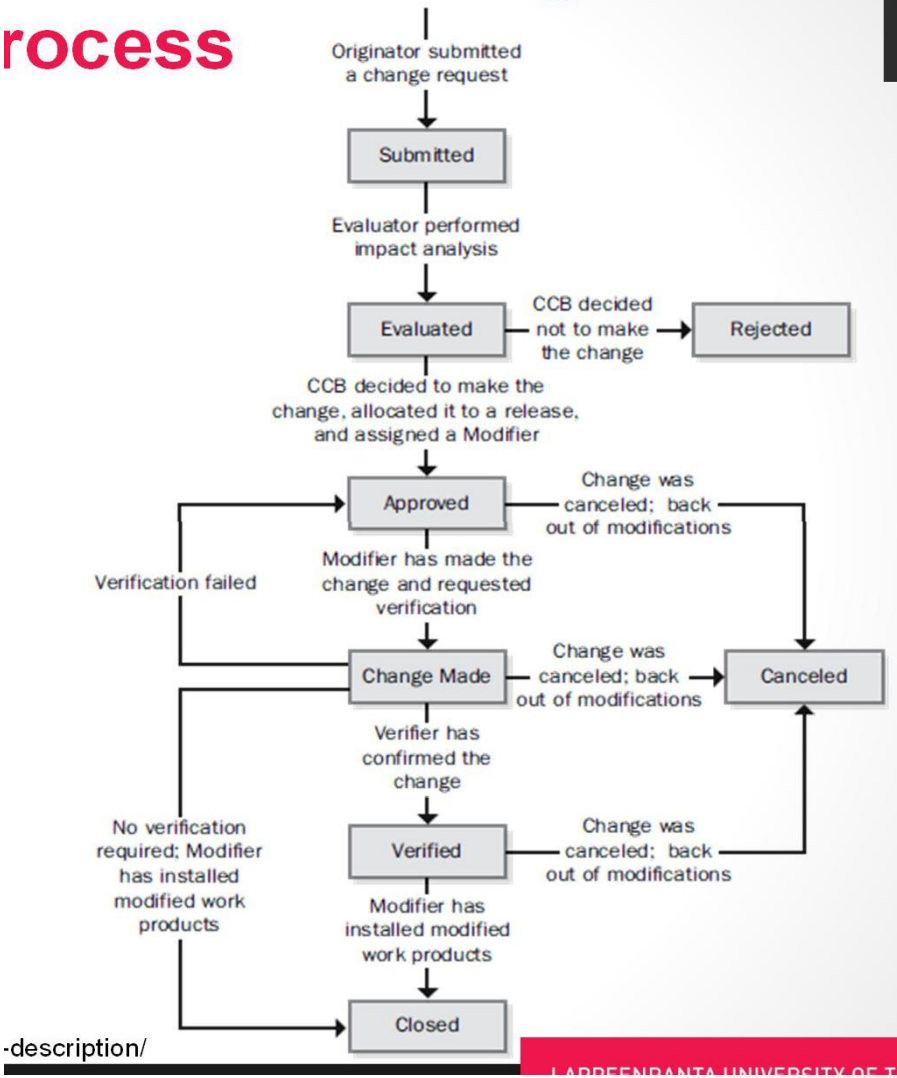
Role	Description and Responsibilities
CCB Chair	Chairperson of the change control board; generally has final decision-making authority if the CCB does not reach agreement; selects the Evaluator and the Modifier for each change request
CCB	The group that decides to approve or reject proposed changes for a specific project
Evaluator	The person whom the CCB Chair asks to analyze the impact of a proposed change; could be a technical person, a customer, a marketing person, or a combination
Modifier	The person responsible for making changes in a work product in response to an approved change request
Originator	Someone who submits a new change request
Request Receiver	The person to whom new change requests are submitted
Verifier	The person who determines whether the change was made correctly

https://www.jamasoftware.com/blog/a-change-control-process-description/

Proposed Template for Describing Change Control Process

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions
2. Roles and Responsibilities
3. Change-Request Status
4. Entry Criteria
5. Tasks
   - 5.1 Evaluate Request
   - 5.2 Make Decision
   - 5.3 Make Change
   - 5.4 Notify All Affected Parties
6. Verification
   - 6.1 Verify Change
   - 6.2 Install Product
7. Exit Criteria
8. Change-Control Status Reporting

Appendix: Data Items Stored for Each Request



-description/

https://www.jamasoftware.com/blog/a-change-control-process-description/

LUT University

Proposed Template for Describing Change Control Process

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions
2. Roles and Responsibilities
3. Change-Request Status
4. Entry Criteria
5. Tasks
   - 5.1 Evaluate Request
   - 5.2 Make Decision
   - 5.3 Make Change
   - 5.4 Notify All Affected Parties
6. Verification
   - 6.1 Verify Change
   - 6.2 Install Product
7. Exit Criteria
8. Change-Control Status Reporting

Appendix: Data Items Stored for Each Request

4. Entry Criteria

The basic entry criterion for your change control process is that a valid change request has been received through an approved channel. All potential originators should know how to submit a change request, whether it's by completing a paper or Web-based form, sending an email message, or entering the information into a change-control tool. Assign a unique identification tag to every change request, and route them all to a single point of contact, the Request Receiver.

5. Tasks

The next step is to evaluate the request for technical feasibility, cost, and alignment with the project's business requirements and resource constraints. The change control board (CCB) Chair might assign an evaluator to perform an impact analysis, risk analysis, hazard analysis, or other assessments. This analysis ensures that the potential consequences of accepting the change are well understood. The evaluator and the CCB should also consider the business and technical implications of rejecting the change.

The appropriate decision makers, chartered as the CCB, then elect whether to approve or reject the requested change. The CCB gives each approved change a priority level or target implementation date, or it allocates the change to a specific build or release number. The CCB communicates the decision by updating the request's status and notifying all team members who might have to modify work products. Affected work products could include the requirements documentation, design descriptions and models, user interface components, code, test documentation, help screens, and user manuals. The modifier(s) update the affected work products as necessary.

https://www.jamasoftware.com/blog/a-change-control-process-description/

Proposed Template for Describing Change Control Process

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions
2. Roles and Responsibilities
3. Change-Request Status
4. Entry Criteria
5. Tasks
   - 5.1 Evaluate Request
   - 5.2 Make Decision
   - 5.3 Make Change
   - 5.4 Notify All Affected Parties
6. Verification
   - 6.1 Verify Change
   - 6.2 Install Product
7. Exit Criteria
8. Change-Control Status Reporting

Appendix: Data Items Stored for Each Request

6. Verification

Requirements changes are typically verified through a peer review to ensure that modified specifications, use cases, and models correctly reflect all aspects of the change. Use traceability information to find all the parts of the system that the change touched and that therefore must be verified. Multiple team members might verify the changes made in downstream work products through testing or review. Following verification, the modifier installs the updated work products to make them available to the rest of the team and redefines the baseline to reflect the changes.

7. Exit Criteria

All of the following exit criteria must be satisfied to properly complete an execution of your change control process:

- The status of the request is Rejected, Closed, or Canceled.
- All modified work products are installed into the correct locations.
- The originator, CCB Chair, project manager, and other relevant project participants have been notified of the change details and the current status of the change request.
- If you have requirements traceability matrix for your project, it has been updated.

https://www.jamasoftware.com/blog/a-change-control-process-description/

Proposed Template for Describing Change Control Process

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions
2. Roles and Responsibilities
3. Change-Request Status
4. Entry Criteria
5. Tasks
   - 5.1 Evaluate Request
   - 5.2 Make Decision
   - 5.3 Make Change
   - 5.4 Notify All Affected Parties
6. Verification
   - 6.1 Verify Change
   - 6.2 Install Product
7. Exit Criteria
8. Change-Control Status Reporting

Appendix: Data Items Stored for Each Request

Item	Description
Change Origin	Functional area that requested the change; possible groups include marketing, management, customer, software engineering, hardware engineering, and testing
Change-Request ID	Identification tag or sequence number assigned to the request
Change Type	Type of change request, such as a requirement change, a proposed enhancement, or a defect report
Date Submitted	Date the originator submitted the change request
Date Updated	Date the change request was most recently modified
Description	Free-form text description of the change being requested
Implementation Priority	The relative importance of making the change as determined by the CCB: low, medium, or high
Modifier	Name of the person who is primarily responsible for implementing the change
Originator	Name of the person who submitted this change request; consider including the originator's contact information
Originator Priority	The relative importance of making the change from the originator's point of view: low, medium, or high
Planned Release	Product release or build number for which an approved change is scheduled
Project	Name of the project in which a change is being requested
Response	Free-form text of responses made to the change request; multiple responses can be made over time; do not change existing responses when entering a new one
Status	The current status of the change request, selected from the options in Figure 19-2 (on page 335)
Title	One-line summary of the proposed change
Verifier	Name of the person who is responsible for determining whether the change was made correctly

https://www.jamasoftware.com/blog/a-change-control-process-description/

Impact Analysis

- Provides accurate understanding of the implications of a proposed change, which help the teams make informed business decisions about which proposals to approved
- Impact analysis has three aspects:
  1. Understand the possible implications of making the change. Change often produces a large ripple effect. Stuffing too much functionality into a product can reduce its performance to unacceptable levels.
  2. Identify all the files, models, and documents that might have to be modified if the team incorporates the requested change.
  3. Identify the tasks required to implement the change, and estimate the effort needed to complete those tasks.
- Traceability data that links the affected requirement to other downstream deliverables helps greatly with impact analysis.
- Understanding the impact enables teams to quickly and accurately respond to change requests. The team can be responsive while maintaining control over scope and the customer expectations.
- Lastly, impact analysis is essential on projects where quality and safety is an issue such as in healthcare, automotive and aerospace projects.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Impact Analysis checklists

Checklist of possible implications of a proposed change.

- ❑ Do any existing requirements in the baseline conflict with the proposed change?
- ❑ Do any other pending requirements changes conflict with the proposed change?
- ❑ What are the business or technical consequences of not making the change?
- ❑ What are possible adverse side effects or other risks of making the proposed change?
- ❑ Will the proposed change adversely affect performance requirements or other quality attributes?
- ❑ Is the proposed change feasible within known technical constraints and current staff skills?
- ❑ Will the proposed change place unacceptable demands on any computer resources required for the development, test, or operating environments?
- ❑ Must any tools be acquired to implement and test the change?
- ❑ How will the proposed change affect the sequence, dependencies, effort, or duration of any tasks currently in the project plan?
- ❑ Will prototyping or other user input be required to verify the proposed change?
- ❑ How much effort that has already been invested in the project will be lost if this change is accepted?
- ❑ Will the proposed change cause an increase in product unit cost, such as by increasing third-party product licensing fees?
- ❑ Will the change affect any marketing, manufacturing, training, or customer support plans?

Checklist of possible software elements affected by a proposed change.

- ❑ Identify any user interface changes, additions, or deletions required.
- ❑ Identify any changes, additions, or deletions required in reports, databases, or files.
- ❑ Identify the design components that must be created, modified, or deleted.
- ❑ Identify the source code files that must be created, modified, or deleted.
- ❑ Identify any changes required in build files or procedures.
- ❑ Identify existing unit, integration, system, and acceptance test cases that must be modified or deleted.
- ❑ Estimate the number of new unit, integration, system, and acceptance test cases that will be required.
- ❑ Identify any help screens, training materials, or other user documentation that must be created or modified.
- ❑ Identify any other applications, libraries, or hardware components affected by the change.
- ❑ Identify any third-party software that must be purchased or licensed.
- ❑ Identify any impact the proposed change will have on the project's software project management plan, quality assurance plan, configuration management plan, or other plans.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Analyzing impact

Estimating effort for requirement change

Effort (Labor Hours)	Task
_____	Update the SRS or requirements database.
_____	Develop and evaluate a prototype.
_____	Create new design components.
_____	Modify existing design components.
_____	Develop new user interface components.
_____	Modify existing user interface components.
_____	Develop new user documentation and help screens.
_____	Modify existing user documentation and help screens.
_____	Develop new source code.
_____	Modify existing source code.
_____	Purchase and integrate third-party software.
_____	Modify build files.
_____	Develop new unit and integration tests.
_____	Modify existing unit and integration tests.
_____	Perform unit and integration testing after implementation.
_____	Write new system and acceptance test cases.
_____	Modify existing system and acceptance test cases.
_____	Modify automated test drivers.
_____	Perform regression testing.
_____	Develop new reports.
_____	Modify existing reports.
_____	Develop new database elements.
_____	Modify existing database elements.
_____	Develop new data files.
_____	Modify existing data files.
_____	Modify various project plans.
_____	Update other documentation.
_____	Update the requirements traceability matrix.
_____	Review modified work products.
_____	Perform rework following reviews and testing.
_____	Other additional tasks.
_____	**Total Estimated Effort**

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

1. Work through the checklists
2. Use the worksheet to estimate the effort required for the anticipated tasks. Most change requests will require only a portion of the tasks on the worksheet, but some could involve additional tasks.
3. Total the effort estimates.
4. Identify the sequence in which the tasks must be performed and how they can be interleaved with currently planned tasks.
5. Determine whether the change is on the project's critical path. If a task on the critical path slips, the project's completion date will slip. Every change consumes resources, but if you can plan a change to avoid affecting tasks that are currently on the critical path, the change won't cause the entire project to slip.
6. Estimate the impact of the proposed change on the project's schedule and cost.
7. Evaluate the change's priority by estimating the relative benefit, penalty, cost, and technical risk compared to other discretionary requirements.
8. Report the impact analysis results to the CCB so that they can use the information to help them decide whether to approve or reject the change request.

Impact Analysis report

Change Request ID: _____
Title: _____
Description: _____
_____
Analyst: _____
Date Prepared: _____
Prioritization Estimates:
- Relative Benefit: _____ (1–9)
- Relative Penalty: _____ (1–9)
- Relative Cost: _____ (1–9)
- Relative Risk: _____ (1–9)
- Calculated Priority: _____ (relative to other pending requirements)

Estimated total effort: _____ labor hours
Estimated lost effort: _____ labor hours (from discarded work)
Estimated schedule impact: _____ days
Additional cost impact: _____ dollars
Quality impact: _____
_____
Other requirements affected: _____
Other tasks affected: _____
Integration issues: _____
Life-cycle cost issues: _____
Other components to examine for possible changes: _____
_____

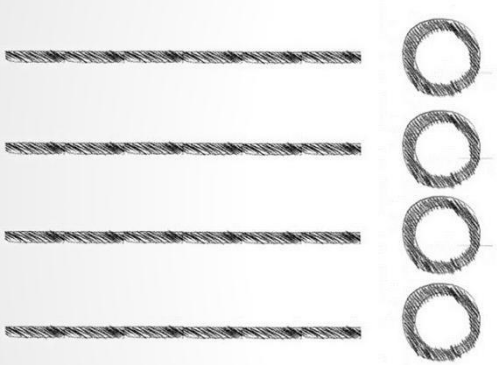
K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Version Control

- Every version of the requirements documents must be uniquely identifiable
- Every team member must be able to access the current version of the requirements, and changes must be clearly documented and communicated
- Permit only designated individuals to update the requirements and make sure that the version identifier changes whenever a requirement changes
- Requirements document should include a revision history that identifies:
  - the changes made
  - the date of each change
  - the individual who made the change
  - the reason for each change

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Requirements Attributes



Consider specifying attributes such as the following for each requirement:

- Date the requirement was created
- Its current version number
- Author who wrote the requirement
- Person who is responsible for ensuring that the requirement is satisfied
- Owner of the requirement or a list of stakeholders (to make decisions about proposed changes)
- Requirement status
- Origin or source of the requirement
- The rationale behind the requirement
- Subsystem (or subsystems) to which the requirement is allocated
- Product release number to which the requirement is allocated
- Verification method to be used or acceptance test criteria
- Implementation priority
- Stability (an indicator of how likely it is that the requirement will change in the future; unstable requirements might reflect ill-defined or volatile business processes or business rules)

Select the smallest set of attributes that will do the job for you!

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Requirements Status

- **Proposed** (someone suggested it)
- **Approved** (it was allocated to a baseline)
- **Implemented** (the code was designed, written, and unit tested)
- **Verified** (the requirement passed its tests after integration into the product)
- **Deferred** (the requirement will be implemented in a future release)
- **Deleted** (you decided not to implement it at all)
- **Rejected** (the idea was never approved)

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

LECTURE 3

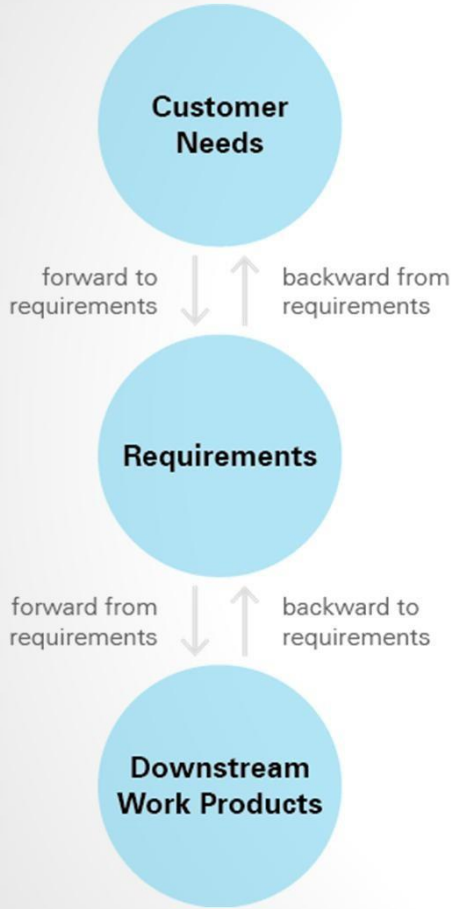
Writing Requirements
Requirements Management
Tracing Requirements
RE Tools
Validating Requirements

Tracing Requirements

- Simple requirements changes often have far-reaching impacts, necessitating that many parts of the product be modified.
- It's hard to find all the system components that might be affected by a requirement modification. Assessing the impact of a proposed change is easier if you have a road map that shows where each requirement or business rule was implemented in the software.
- Traceability links allow you to follow the life of a requirement both forward and backward, from origin through implementation.
- To permit traceability, each requirement must have a unique and persistent label so you can refer to it unambiguously throughout the project.
- Write the requirements in a fine-grained fashion, rather than having large paragraphs containing many individual functional requirements that lead to an explosion of design and code elements.

https://www.jamasoftware.com/blog/requirements-traceability-links-in-the-requirements-chain-part-1/

Traceability links



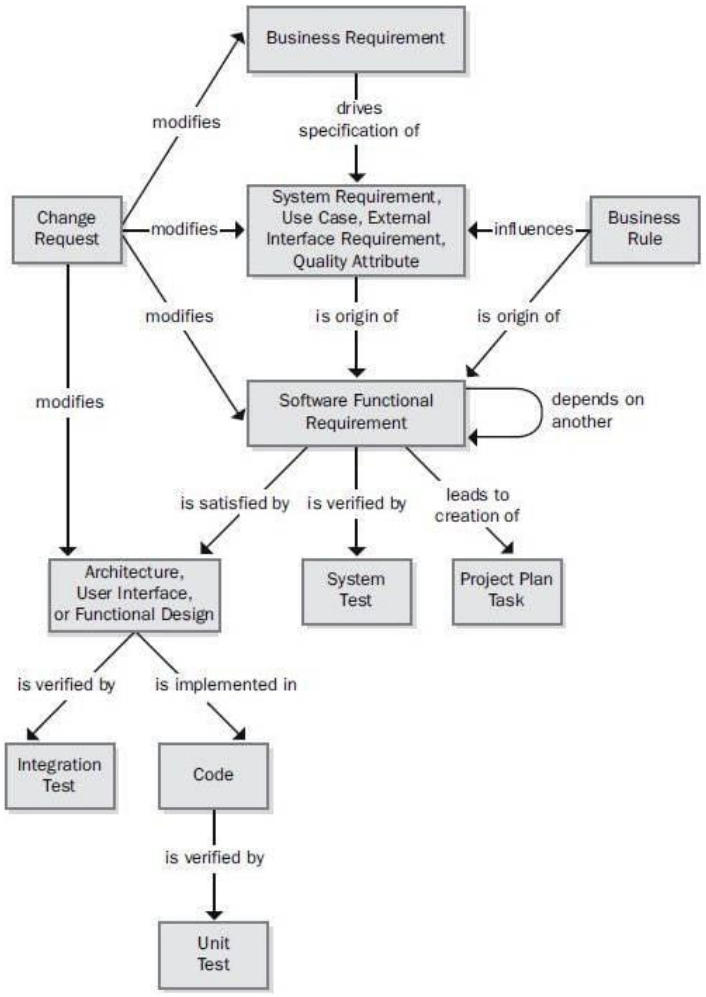
Four types of requirements traceability links:

1. Customer needs are traced *forward to requirements*, so you can tell which requirements will be affected if those needs change during or after development. This also gives you confidence that the requirements specification has addressed all stated customer needs.
2. Conversely, you can trace *backward from requirements* to customer needs to identify the origin of each software requirement. If you represented customer needs in the form of use cases, the top half of Figure illustrates tracing between use cases and functional requirements.
3. The bottom half of Figure indicates that, as requirements flow into downstream deliverables during development, you can trace *forward from requirements* by defining links between individual requirements and specific product elements. This type of link assures that you've satisfied every requirement because you know which components address each one.
4. The fourth type of link traces specific work products *backward to requirements* so that you know why each item was created. Most applications include code that doesn't relate directly to user-specified requirements, but you should know why someone wrote every line of code.

https://www.jamasoftware.com/blog/requirements-traceability-links-in-the-requirements-chain-part-1/

Some Possible Requirements Traceability Links

- Decide which links are pertinent to your project and can contribute the most to successful development and efficient maintenance.
- Don’t ask team members to spend time recording information unless you have a clear idea of how you expect to use it



https://www.jamasoftware.com/blog/requirements-traceability-links-in-the-re quirements-chain-part-1/

Motivations for Tracing Requirements

- ***Certification.*** Traceability information can be used when certifying a safety-critical product to demonstrate that all requirements were implemented (although it doesn't confirm that they were implemented correctly or completely!). Of course, if the requirements are incorrect or key requirements are missing, even the best traceability data won't help you.
- ***Change impact analysis.*** Without traceability information, there's a high probability of overlooking a system element that would be affected if you add, delete, or modify a particular requirement.
- ***Maintenance.*** Reliable traceability information facilitates making changes correctly and completely during maintenance, which improves your productivity. When corporate policies or government regulations change, software applications often require updating. A table that shows where each applicable business rule was implemented in the functional requirements, designs, and code makes it easier to make the necessary changes properly.
- ***Project tracking.*** If you diligently record the traceability data during development, you'll have an accurate record of the implementation status of planned functionality. Missing links indicate work products that have not yet been created.
- ***Reengineering.*** You can list the functions in a legacy system you're replacing and record where they were addressed in the new system's requirements and software components. Defining traceability links is a way to capture some of what you learn through reverse engineering of an existing system.
- ***Reuse.*** Traceability information facilitates reusing product components by identifying packages of related requirements, designs, code, and tests.
- ***Risk reduction.*** Documenting the component interconnections reduces the risk if a key team member with essential knowledge about the system leaves the project.
- ***Testing.*** The links between tests, requirements, and code point you toward likely parts of the code to examine for a bug when a test yields an unexpected result. Knowing which tests verify which requirements can save time by letting you eliminate redundant tests.

https://www.jamasoftware.com/blog/requirements-traceability-links-in-the-requirements-chain-part-1/

The Requirements Traceability Matrix

- The most common way to represent the links between requirements and other system element
- You should fill in the information as the work gets done, not as it gets planned. This way a reader knows that populated cells in the requirements traceability matrix indicate completed work, not just good intentions

User Requirement	Functional Requirement	Design Element	Code Module	Test Case
UC-28	catalog.query.sort	Class catalog	catalog.sort()	search.7 search.8
UC-29	catalog.query.import	Class catalog	catalog.import() catalog.validate()	search.12 search.13 search.14

https://www.jamasoftware.com/blog/links-in-the-requirements-chain-part-2/

Requirements Traceability Matrix Showing Links Between Use Cases and Functional Requirements

Functional Requirement	Use Case			
	UC-1	UC-2	UC-3	UC-4
FR-1	↵			
FR-2	↵			
FR-3			↵	
FR-4			↵	
FR-5		↵		↵
FR-6			↵	

https://www.jamasoftware.com/blog/links-in-the-requirements-chain-part-2/

Likely Sources of Traceability Link Information

Link Source Object Type	Link Target Object Type	Information Source
System requirement	Software requirement	System engineer
Use case	Functional requirement	Requirements analyst
Functional requirement	Functional requirement	Requirements analyst
Functional requirement	Test case	Test engineer
Functional requirement	Software architecture element	Software architect
Functional requirement	Other design elements	Designer or Developer
Design element	Code	Developer
Business rule	Functional requirement	Requirements analyst

https://www.jamasoftware.com/blog/links-in-the-requirements-chain-part-2/

Setting up Requirements Traceability Procedure

1. **Select the link relationships you want to define**
2. **Choose the type of traceability matrix you want to use.** Select a mechanism for storing the data—a table in a text document, a spreadsheet, or a commercial requirements management tool.
3. **Identify the parts of the product for which you want to maintain traceability information.** Start with the critical core functions, the high-risk portions, or the portions that you expect to undergo the most maintenance and evolution over the product's life.
4. **Modify your development procedures and checklists to remind developers to update the links after implementing a requirement or an approved change.** The traceability data should be updated as soon as someone completes a task that creates or changes a link in the requirements chain.
5. **Define the tagging conventions you will use to uniquely identify all system elements so they can be linked together.** If necessary, write scripts that will parse the system files to construct and update the traceability matrices. If you don't have unique and persistent labels on requirements, design elements, and other system elements, there's no way you can document the connections between them.
6. **Educate the team about the concepts and importance of requirements tracing**, your objectives for this activity, where you will store the traceability data, and the techniques for defining the links—for example, using the tracing features of a requirements management tool.
7. **Identify the individuals who will supply each type of link information and the person who will coordinate the traceability activities and manage the data**. Obtain commitments from all of them to do their part.
8. As development proceeds, **have each participant provide the requested traceability information as they complete small bodies of work**. Stress the need to assemble the traceability data as they work, rather than attempting to reconstruct it at a major milestone or at the end of the project.
9. **Audit the traceability information periodically to make sure it is being kept current.** If a requirement is reported as implemented and verified, yet its traceability data is incomplete or inaccurate, your traceability process isn't working as you intend.

https://www.jamasoftware.com/blog/links-in-the-requirements-chain-part-3/

LECTURE 3

Writing Requirements
Requirements Management
Tracing Requirements
RE Tools
Validating Requirements

Tools for Requirements Management

A document-based approach to storing requirements has numerous limitations, including the following:

- It's difficult to keep the documents current and synchronized.
- Communicating changes to all affected team members is a manual process.
- It's not easy to store supplementary information (attributes) about each requirement.
- It's hard to define links between functional requirements and other system elements.
- Tracking requirements status is cumbersome.
- Concurrently managing sets of requirements that are planned for different releases or for related products is difficult.
- Reusing a requirement means that the business analyst (BA) must copy the text from the original software requirements specification (SRS) into the SRS for each other system or product where the requirement is to be used.
- It's difficult for multiple project participants to modify the requirements safely, particularly if the participants are geographically separated.
- There's no convenient place to store requirements that were proposed and then rejected or requirements that were deleted from a baseline.

A commercial requirements management (RM) tool that stores information in a multiuser database provides a robust solution to these restrictions. Such products:

- let users import requirements from different sources
- export requirements in various formats
- define attribute values
- filter and display the database contents
- define traceability links and connect requirements to items stored in other software development tools.
- Some examples: IBM Engineering Requirements Management DOORS (DOORS Next); Azure DevOps; ReqSuite® RM; Jira and Confluence

https://www.jamasoftware.com/resource/getting-the-most-from-a-requirements-management-tool/

Seven Benefits of Using a Requirements Management Tool

1. Manage Versions and Changes.
- Flexible baselining functions.
- Tools to maintain a history of the changes made to every requirement.
- Possibility record the rationale behind each change decision and revert to a previous version of a requirement if necessary.

2. Store Requirements Attributes.
- Possibility to record and view several descriptive attributes for each requirement.
- Possibility to define rights to update/view requirements
- Automated generation of several system-defined attributes, such as the date a requirement was created and its current version number, and they let you define additional attributes of various data types.
- Ability to search requirements based on attributes. For instance, you might ask to see a list of all the requirements originating from a specific business rule so that you can judge the consequences of a change in that rule.

3. Facilitate Impact Analysis.
- Enabling requirements tracing by letting you define links between different types of requirements, between requirements in different subsystems, and between individual requirements and related system components (for example, designs, code modules, tests and user documentation).
- Some of the tools let you establish traceability links between requirements in the database and objects stored in other, third-party tools, such as problem reports, change requests, design model objects, source code files and project task lists.

4. Track Requirements Status.
- Collecting requirements in a database lets you know how many discrete requirements you've specified for the product.
- Tracking the status of each requirement during development supports the overall status tracking of the project. A project manager has good insight into project status if he or she knows that 55 percent of the requirements committed to the next release have been verified, 28 percent have been implemented but not verified and 17 percent are not yet fully implemented.

5. Control Access.
- Requirements management tools let you define access permissions for individuals or groups of users and share information with a geographically dispersed team through a web interface to the database.

6. Communicate With Stakeholders.
- Some tools permit team members to discuss requirements issues electronically through threaded conversations. Automatically triggered email messages notify affected individuals when a new discussion entry is made or when a specific requirement is modified.

7. Reuse Requirements.
- Storing requirements in a database facilitates reusing them in multiple projects or subprojects. Requirements that logically fit into multiple parts of the product description can be stored once and referenced whenever necessary to avoid duplicates.

https://www.jamasoftware.com/resource/getting-the-most-from-a-requirements-management-tool/

Some Requirements Management Tool Capabilities

- **Let you define different requirement types** (sometimes called classes), such as business requirements, use cases, functional requirements, hardware requirements and constraints. This lets you differentiate individual objects that you want to treat as requirements from other useful information contained in the SRS.
- **Provide strong capabilities for defining attributes for each requirement type**, which is a great advantage over the typical document-based SRS approach.
- **Requirements traceability features** let you define links between objects of two requirement types, or even within the same requirement type.
- **Integrate with Microsoft Word** to some degree. The higher-end tools support a rich variety of import and export file formats. Several of the tools let you mark text in a Word document to be treated as a discrete requirement. Some tools can parse documents in various fashions to extract individual requirements and load them into the database.
- **Support hierarchical numeric requirement labels**, in addition to maintaining a unique internal identifier for each requirement. These identifiers typically consist of a short text prefix that indicates the requirement type—such as UR for a user requirement—followed by a unique integer. Some tools provide efficient displays to let you manipulate the hierarchical requirements tree.
- **Output capabilities** from the tools include the ability to generate a requirements document, either in a user-specified format or as a tabular report. Several tools let you define an SRS template in Word and then populate this template with information it selects from the database according to user-defined query criteria to produce a customized specification document. This SRS is, therefore, a report generated from selected database contents.
- **The ability to set up user groups and define permissions for selected users** or groups to create, read, update and delete projects, requirements, attributes and attribute values.
- **Let you incorporate nontextual objects** such as graphics and spreadsheets into the requirements repository.

Tools will move your requirements management practices to a higher plane of sophistication and capability. However, the diligence of the tools' users remains a critical success factor. Dedicated, disciplined and knowledgeable people will make progress even with mediocre tools, whereas the best tools won't pay for themselves in the hands of unmotivated or ill-trained users.

https://www.jamasoftware.com/resource/getting-the-most-from-a-requirements-management-tool/

Get the Most Out of Your Requirements Management Tool

- **Write Good Requirements First.** The tools are not a substitute for effective requirements development processes and techniques. They will help you manage and track whatever information you store in them.
- **Expect a Culture Change**. Organizations that are accustomed to storing requirements in documents already have mechanisms in place for creating, reviewing, approving, storing, distributing and modifying those documents. A requirements management tool brings a significant new paradigm to these organizations.
- **Don't Create Too Many Requirement Types or Attributes.** It takes effort to create and maintain them.
- **Train the Tool Users.** For the smoothest transition, assign a tool advocate, a local enthusiast who learns the tool's ins and outs, mentors other users, and sees that it gets employed as intended.
- **Assign Responsibilities.** If responsibilities are not made clear and accepted by the team members, important work won't get done. This degrades the quantity, quality and value of the data stored in the tool.
- **Time Activities Sensibly.** Don't try to capture requirements directly in the tool during the early elicitation workshops. As the requirements begin to stabilize, though, storing them in the tool makes them visible to the workshop participants for review and refinement. Also, don't define traceability links until the requirements stabilize, such as when you define a baseline for a particular subset targeted at a particular iteration or release. Otherwise, you can count on doing a lot of work to revise the links as requirements continue to change.
- **Take Advantage of Tool Features.** One of the strongest arguments for requirements management tools is having the ability to define traceability links. The more robust requirements management products even allow analysts to establish such links to objects stored in other tools, such as to design elements stored in a modeling tool, code segments in a version control tool and tests in a test management tool. If you don't use such features, it diminishes the value of keeping the requirements in a database. The tools also let you define groups and individuals with different permission levels, to identify who can read, create, and modify the contents of the database. Access controls are an important consideration for companies that have employees in multiple countries. Take advantage of these tool capabilities to ensure that all the right people—and only the right people—can access the requirements.
- **Invest the Effort.** Recognize that it will take effort to load a project's requirements into the database, define attributes and traceability links, keep the database's contents current, define access groups and their privileges, and train users. Management must allocate the resources needed for these operations.

As the use of a requirements management tool becomes ingrained in your culture, project stakeholders will begin to regard requirements as life cycle assets, just as they do code. The team will discover ways to use the tool to speed up the process of documenting requirements, communicating them and managing changes to them.

https://www.jamasoftware.com/resource/getting-the-most-from-a-requirements-management-tool/

Example

A university plans to develop a **Sustainability Awareness Application** aimed at educating software engineering students about the environmental impacts of software systems and IT services. The app will feature interactive modules, quizzes, and personalized recommendations for eco-friendly software development practices.

Your team is tasked with gathering detailed requirements to ensure the app meets the following goals:

- Engages students with interactive learning.
- Provides accurate sustainability content.
- Offers tailored recommendations based on user behavior.
- Integrates seamlessly into students' existing digital platforms (e.g., LMS or mobile).

LECTURE 3

Writing Requirements
Requirements Management
Tracing Requirements
RE Tools
Validating Requirements

Requirements Validation

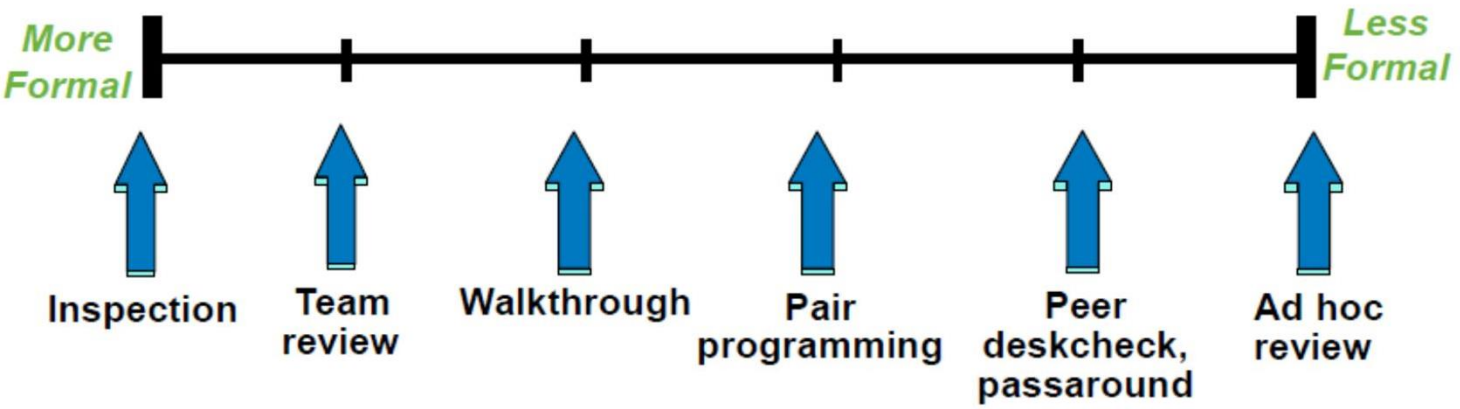
Attempts to ensure that:

- The SRS correctly describes the intended system capabilities and characteristics that will satisfy the various stakeholders' needs
- The software requirements were correctly derived from the system requirements, business rules, or other sources
- The requirements are complete and of high quality
- All requirements representations are consistent with each other
- The requirements provide an adequate basis to proceed with design and construction

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Peer Review Formality Spectrum

Reviewing requirements document is a powerful technique for identifying ambiguous or unverifiable requirements.



SMARTBEAR

https://pdfs.semanticscholar.org/presentation/c178/f52433c0dd0e80618c7be6b2c5888912e5d7.pdf

Reviewing Requirements: Informal reviews

- Informal reviews are not systematic, thorough, or performed in a consisten way
- Informal reviews are useful for educating other people about the product and collecting unstructured feedback.
- Informal review approaches include:
  - A *peer deskcheck*, in which you ask one colleague to look over your work product
  - A *passaround*, in which you invite several colleagues to examine a deliverable concurrently
  - A *walkthrough*, during which the author describes a deliverable and solicits comments on it.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Formal Reviews

- Follow a well-defined process
- Produces a report that identifies the material, the reviewers, and the review team's judgment as to whether the product is acceptable
- Pincipal delivery: a summary of the defects found and the issues raised
- One effective form of formal peer review is an *inspection*

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Inspection Participants

- The author of the work product and perhaps peers of the author
- The author of any predecessor work product or specification for the item being inspected
- People who will do work based on the item being inspected
- People who are responsible for work products that interface with the item being inspected

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Inspection Roles

- **Author** takes a passive role during an inspection. The author may not assume any of the other assigned roles. The author can listen to the comments from other inspectors, respond to-but not debate-their questions, and think. The author will often spot errors that other inpectors don'tsee
- **Moderator** or *inspection leader*
  - plans the inspection with the author,
  - coordinates the activities and facilitates the inspection meeting.
  - reports the inspection results to management or to someone who collects data from multiple inspections
  - follows up on proposed changes with the author to ensure that the defects and issues that came out of inspection were addressed properly.
- **Reader**, who paraphrases the SRS one requirement at a time. The other participants then point out potential defects and issues. By stating a requirement in her own words, the reader provides an interpretation that might differ from that held by other inspectors. This is a good way to reveal an ambiguity, a possible defect, or an assumption. It also underscores the value of having someone other than the author serve as the reader.
- **Recorder** or *scribe* uses standard forms to document the issues raised and the defects found during the inspection meeting. The recorder should review aloud what he wrote to confirm the record's accuracy. The other inspectors should help the recorder capture the essence of each issue in a way that clearly communicates to the author the location and nature of the issue.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

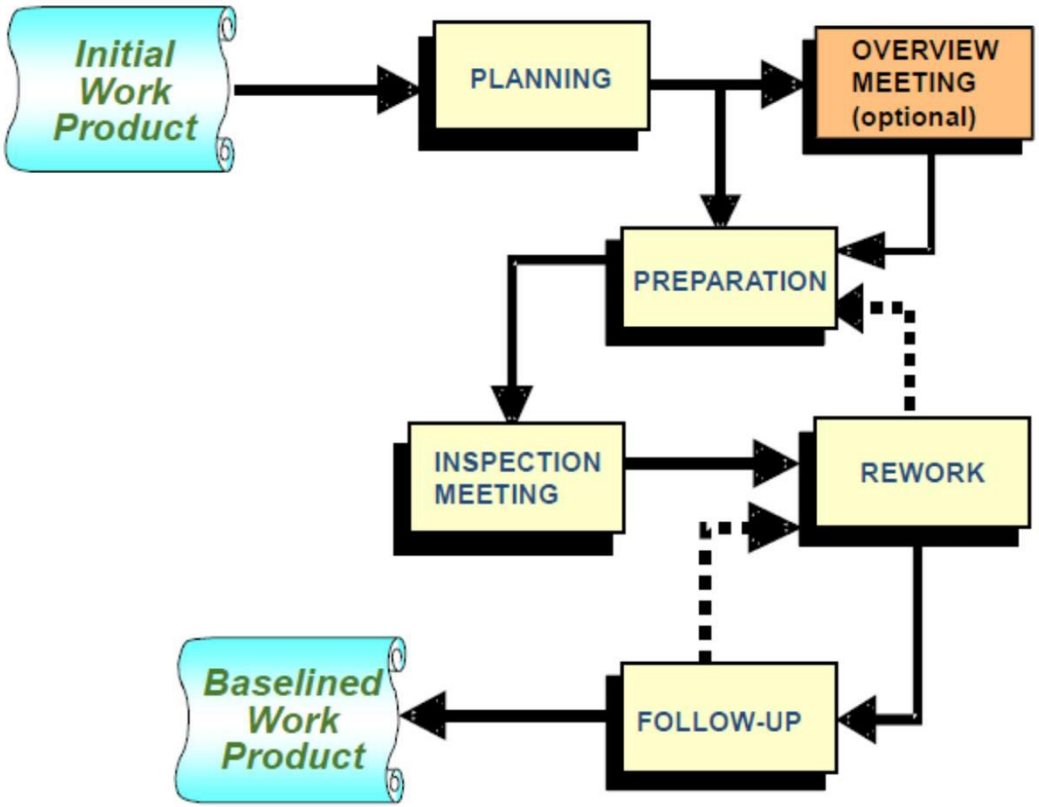
When Inspection can be initiated?

Suggested entry criteria for requirements document:

- The document conforms to the standard template
- The document has been spell-checked
- The author has visually examined the document for any layout errors
- Any predecessor or reference documents that the inspectors will require to examine the document are available
- Line numbers are printed on the document to facilitate referring to specific locations during the inspection meeting
- All open issues are marked as TBD (to be determined)
- The moderator didn't find more than three major defects in a ten-minute examination of a representative sample of the document

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

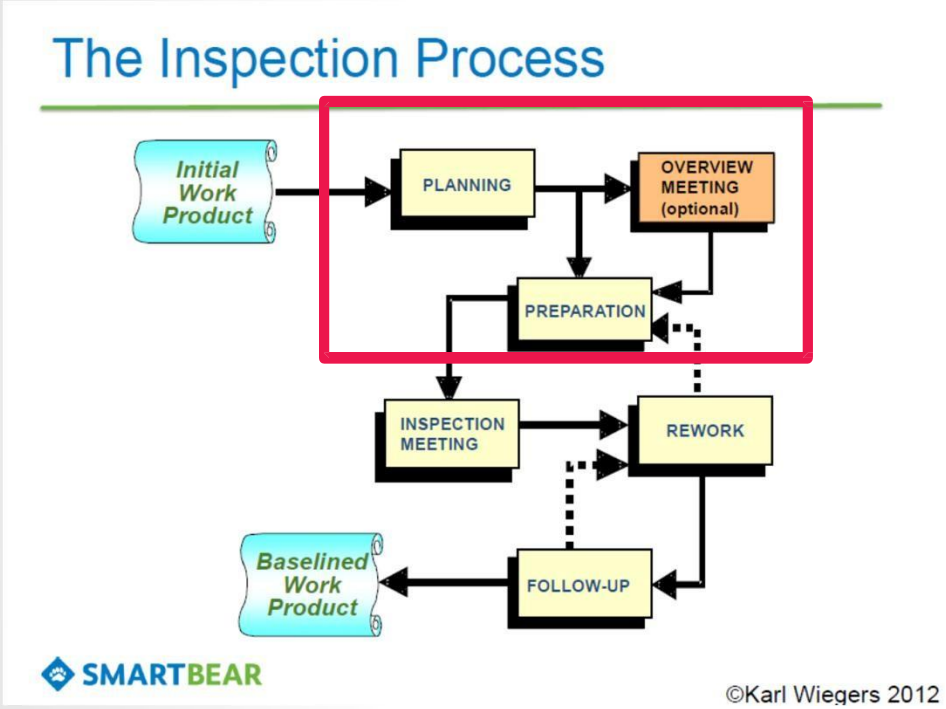
The Inspection Process



SMARTBEAR

https://pdfs.semanticscholar.org/presentation/c178/f52433c0dd0e80618c7be6b2c5888912e5d7.pdf

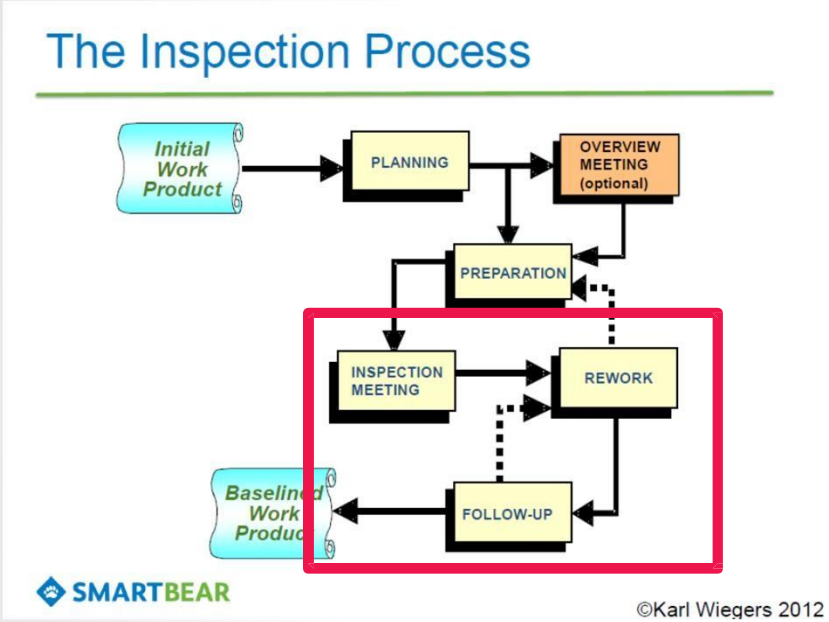
Inspection Preparation Activities



- **Planning:** The author and moderator plan the inspection together. They determine:
  - who should participate
  - what materials the inspectors should receive prior to the inspection meeting
  - how many inspection meetings will be needed to cover the material (2-4 pages per hour is a practical guideline)
- **Overview meeting (optional):** The author describes the background of the material the team will be inspecting, any assumptions he made, and his specific inspection objectives.
- **Preparation:** Prior to the inspection meeting, each inspector examines the product to identify possible defects and issues to be raised, using the checklist of typical defects.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Inspection meeting and afterwork



- **Inspection meeting:** The reader leads the other inspectors through the SRS, describing one requirement at a time in his or her own words. As inspectors bring up possible defects and other issues, the recorder captures them on a form that becomes the action item list for the requirements author. The purpose of the meeting is to identify as many major defects in the requirements document as possible. Meeting shouldn't last more than two hours because tired people aren't effective inspectors.
  At the meeting's conclusion, the team decides whether to accept the requirements document as is, accept it with minor revisions, or indicate that major revision is needed.
- **Rework:** Improve requirements according to the raised issues
- **Follow-up:** Moderator or a designated individual works with the author to ensure that all open issues were resolved and that errors were corrected properly.

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Defect checklist for SRS

Organization and Completeness

- Are all internal cross-references to other requirements correct?
- Are all requirements written at a consistent and appropriate level of detail?
- Do the requirements provide an adequate basis for design?
- Is the implementation priority of each requirement included?
- Are all external hardware, software, and communication interfaces defined?
- Have algorithms intrinsic to the functional requirements been defined?
- Does the SRS include all of the known customer or system needs?
- Is any necessary information missing from a requirement? If so, is it identified as TBD?
- Is the expected behavior documented for all anticipated error conditions?

Correctness

- Do any requirements conflict with or duplicate other requirements?
- Is each requirement written in clear, concise, unambiguous language?
- Is each requirement verifiable by testing, demonstration, review, or analysis?
- Is each requirement in scope for the project?
- Is each requirement free from content and grammatical errors?
- Can all of the requirements be implemented within known constraints?
- Are any specified error messages unique and meaningful?

Quality Attributes

- Are all performance objectives properly specified?
- Are all security and safety considerations properly specified?
- Are other pertinent quality attribute goals explicitly documented and quantified, with the acceptable tradeoffs specified?

Traceability

- Is each requirement uniquely and correctly identified?
- Is each software functional requirement traceable to a higher-level requirement (e.g., system requirement, use case)?

Special Issues

- Are all requirements actually requirements, not design or implementation solutions?
- Are the time-critical functions identified, and timing criteria specified for them?
- Have internationalization issues been adequately addressed?

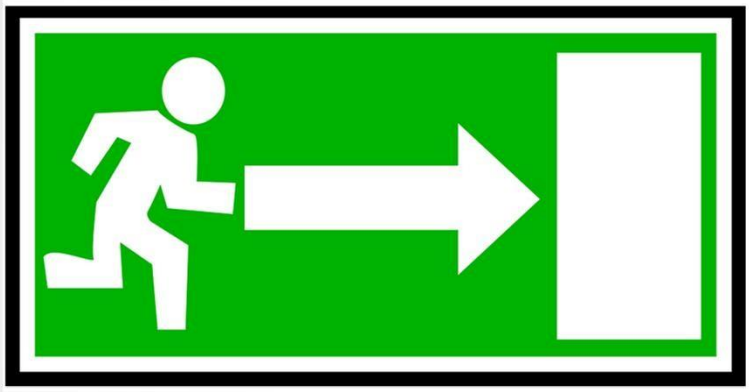
K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Defect checklist for Use Case Documents

- Is the use case a stand-alone, discrete task?
- Is the goal, or measurable value, of the use case clear?
- Is it clear wich actor or actors benefit from the use case?
- Is the use case written at the essential level, free of unnecessary design and implementation details?
- Are all anticipated alternative courses documented?
- Are all known exception conditions documented?
- Are there any common action sequences that could be split into separate use cases?
- Are the steps for each course clearly written, unambiguous, and complete?
- Is each actor and step in the use case pertinent to performing that task?
- Is each alternative course defined in the use case feasible and verifiable?
- Do the preconditions and postconditions properly frame the use case?

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Inspection Exit Criteria



- All issues raised during the inspection have been addressed
- Any changes made in the document and related work products were made correctly
- The revised document has been spell-checked
- All TBDs have been resolved, or each TBD's resolution process, target date, and owner has been documented
- The document has been checked into the project's configuration management system

K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

Example

A university plans to develop a **Sustainability Awareness Application** aimed at educating software engineering students about the environmental impacts of software systems and IT services. The app will feature interactive modules, quizzes, and personalized recommendations for eco-friendly software development practices.

Your team is tasked with gathering detailed requirements to ensure the app meets the following goals:

- Engages students with interactive learning.
- Provides ***accurate*** sustainability content.
- Offers tailored recommendations based on user behavior.
- Integrates seamlessly into students' existing digital platforms (e.g., LMS or mobile).

Functional Requirements

Interactive Learning Modules

- The system shall provide modules with interactive content, including videos, simulations, and gamified experiences that engage students in sustainability topics.
- The system shall enable quizzes at the end of each module to test student knowledge and provide instant feedback.
- **Priority:** High
- **Validation:** Run user testing sessions with students to ensure interactivity meets learning needs.

Sustainability Content

- The system shall provide up-to-date and accurate information on the environmental impact of software systems, based on widely recognized standards (e.g., Green IT).
- The system shall offer content related to sustainable software development practices.
- **Priority:** High
- **Validation:** Have subject-matter experts review the content for accuracy before launch.

Functional Requirements

Tailored Recommendations

- The system shall track user behavior (e.g., quiz responses, interaction time) to provide personalized recommendations on eco-friendly software development practices.
- The system shall allow users to set preferences for specific sustainability areas, and recommendations shall adjust accordingly.
- **Priority:** Medium
- **Validation:** Develop test cases that simulate user behavior to ensure recommendations adapt accurately.

Seamless Integration

- The system shall integrate with existing Learning Management Systems (LMS) such as Moodle or Blackboard.
- The system shall be accessible via mobile devices (iOS and Android) and provide a responsive design for tablet and desktop views.
- **Priority:** High
- **Validation:** Perform system integration testing with LMS environments and user accessibility testing on mobile devices.

Non-Functional Requirements

Usability

- The system interface shall be easy to navigate, with a maximum of three clicks to access any learning content.
- **Priority:** Medium
- **Validation:** Conduct usability tests with students to ensure smooth navigation.

Performance

- The system shall load all content modules in under 2 seconds on a standard mobile network.
- **Priority:** High
- **Validation:** Performance testing in different network conditions.

Scalability

- The system shall handle at least 1,000 concurrent users without performance degradation.
- **Priority:** Medium
- **Validation:** Load testing under varying user loads.

Prioritizing Requirements

You can prioritize requirements based on their importance for achieving the MVP:

Must-Have (High Priority):

- Interactive Learning Modules
- ***Accurate*** Sustainability Content
- Seamless Integration into LMS and mobile platforms

Should-Have (Medium Priority):

- Tailored Recommendations
- Usability and performance benchmarks

Could-Have (Low Priority):

- Advanced features like community forums or progress tracking dashboards (may be out of scope for MVP).

Validation of Requirements and Traceability

1. **Functional Testing:** Validate each functional requirement through unit tests, integration tests, and end-to-end tests.
2. **User Acceptance Testing (UAT):** Engage a sample group of students and instructors to ensure that the application delivers the required functionality and content accuracy.
3. **Feedback Loops:** Set up continuous feedback loops for future enhancements based on user interactions with the MVP.

Traceability: Requirement to MVP

ID	Requirement Description	MVP Feature	Validation Method
FR-01	Provide interactive learning modules	Interactive Learning Pages	User Testing
FR-02	Provide accurate sustainability content	Learning Content Pages	Expert Review
FR-03	Offer tailored eco-friendly recommendations	Recommendation System	Simulation of User Scenarios
NFR-01	Integrate with LMS	LMS Integration API	Integration Testing
NFR-02	Ensure system usability	User Interface	Usability Testing

IMPORTANT DATES!

- **October 29th, 2024, at 23:00.** Group assignment must be submitted.
- **November 15th, 2024, at 23:00.** Peer reviews must be submitted.