



Requirements Engineering III

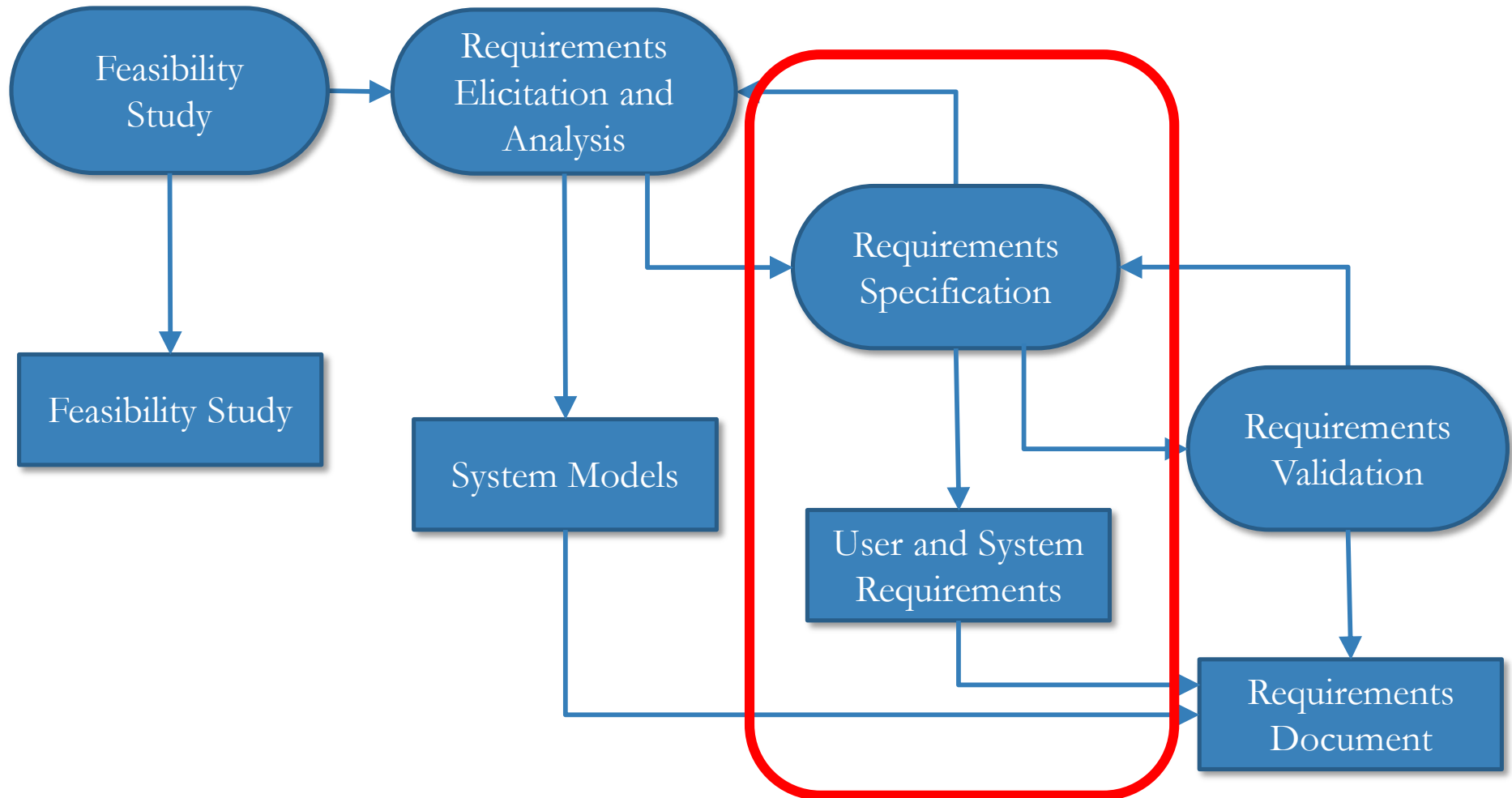
Course Topics

- ~~Introduction~~
- ~~Software Process Models~~
- Requirements Engineering
- Modeling
- Programming Languages
- Software Construction Techniques
- Testing
- Project Management
- Refactoring
- Ethical Issues

Lecture Objectives

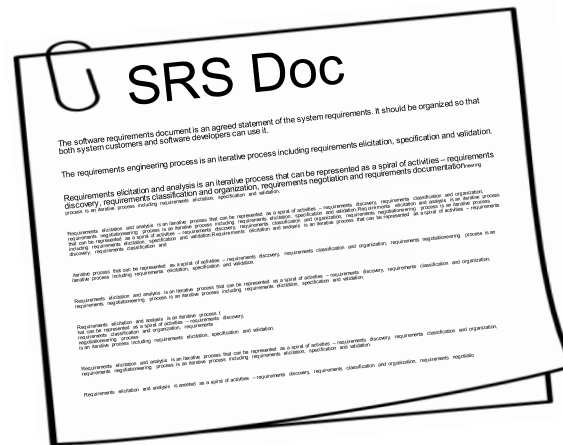
- ✓ Software Requirement Document
- ✓ Requirements Specification
 - Natural Language Specification
 - Structured Specification





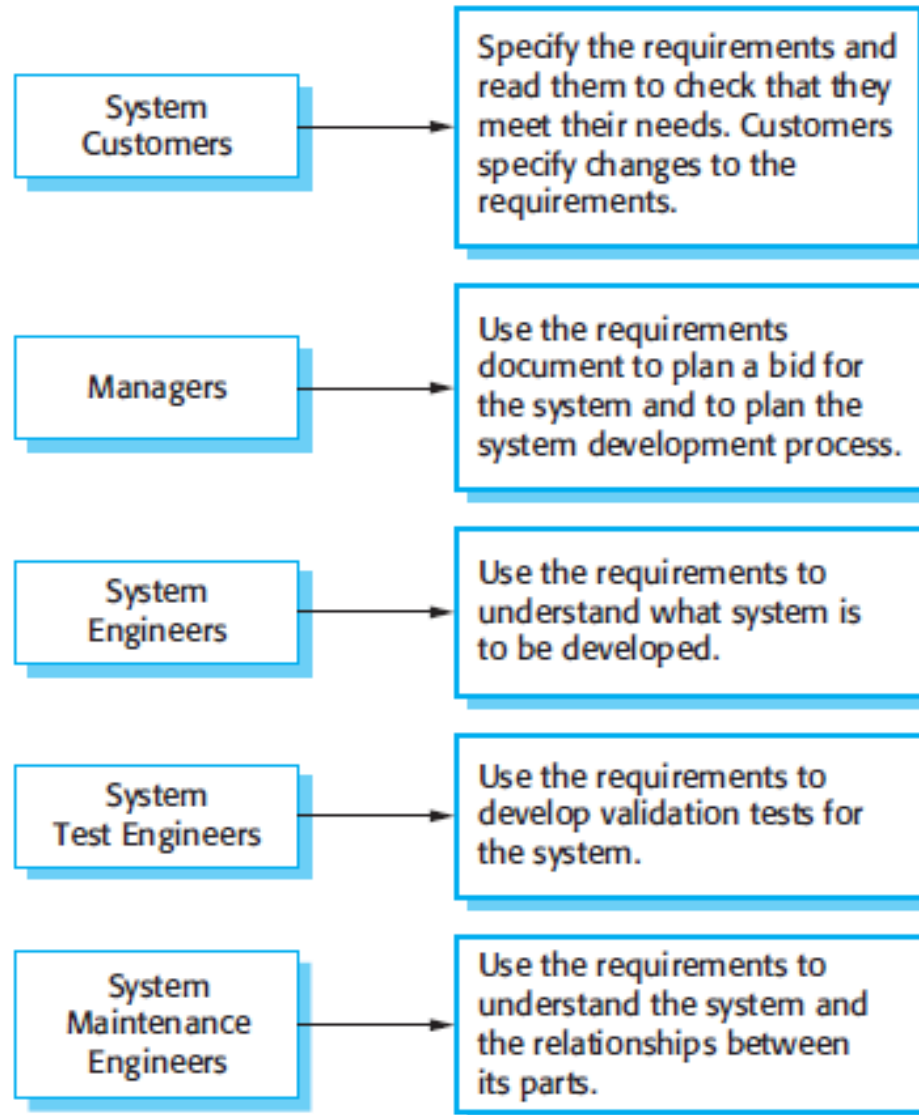
Software requirements document

- The software requirements document is the **official statement** of what is required of the system developers.
- Should include both a definition of **user requirements** and a specification of the **system requirements**.




- It is **NOT** a **design document**. As far as possible, it should set **WHAT** the system should do rather than **HOW** it should do it.

Users of a requirements document




Requirements document variability



- Information in requirements document depends on the type of system and the approach to development used.
 - Systems developed **incrementally** will, typically, have **less detail** in the requirements document.
 - Requirements documents standards have been designed e.g. **IEEE standard**. These are mostly applicable to the requirements for large systems engineering projects.
- 

Requirements specification



- The process of writing the user and system requirements in a **requirements document**.
 - User requirements have to be **understandable** by end-users and customers who **do not have a technical background**.
 - System requirements are more **detailed** requirements and may include more **technical information**.
 - The requirements may be part of a **contract** for the system development
 - It is therefore important that these are as **complete** as possible.
- 

Ways of writing a system requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language . Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template . Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.

Ways of writing a system requirements specification

Notation	Description
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Requirements and design

- In principle, **requirements** should state **what the system should do** and the **design** should describe **how it does this**.
- In practice, requirements and design are **inseparable**
 - A system architecture may be designed to structure the requirements
 - The use of a specific architecture to satisfy non-functional requirements may be a domain requirement


Natural Language Specification

- Should describe **functional** and **non-functional** requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- Usually defined using natural language, tables and diagrams which can be understood by all users.

No software jargon, formal notations, or implementation details

Problems with natural language



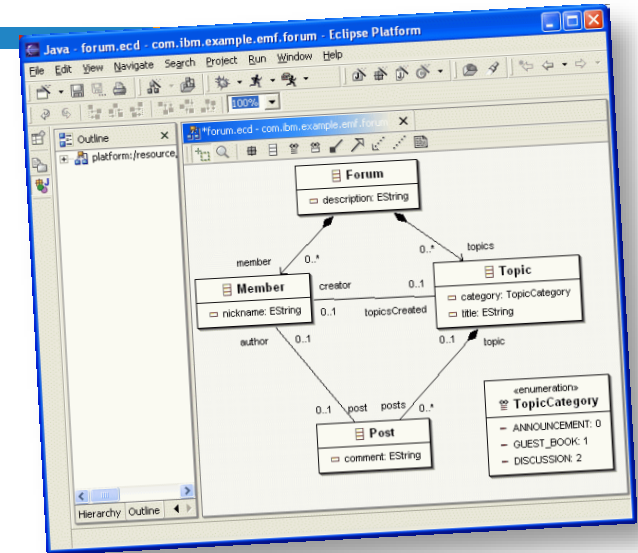
- Lack of **clarity**
 - Precision is difficult without making the document difficult to read.
 - Requirements **confusion**
 - Functional and non-functional requirements tend to be mixed-up.
 - Requirements **amalgamation**
 - Several different requirements may be expressed together.
- 

Lack of clarity

- Problems arise when requirements are not precisely stated.
(Ambiguous Requirements)
- For example, what does the following mean?
“With your meal you have a choice of soup or salad and rice”
- For example, the term ‘appropriate viewers’
 - User intention - special purpose viewer for each different document type
 - Developer interpretation - Provide a text viewer that shows the contents of the document.



Requirements amalgamation



2.6 Grid facilities To assist in the positioning of entities on a diagram, the user may turn on a grid in either centimetres or inches, via an option on the control panel. Initially, the grid is off. The grid may be turned on and off at any time during an editing session and can be toggled between inches and centimetres at any time. A grid option will be provided on the reduce-to-fit view but the number of grid lines shown will be reduced to avoid filling the smaller diagram with grid lines.

How to make requirements clear?

- Associate a **rationale** with each user requirement
- Use a standard format
- Use language consistently
 - “Shall” → mandatory (indicates a requirement that is contractually binding, meaning it must be implemented)
 - “Should” → desirable/recommended
- Use text highlighting (**bold**, *italic* or colour)

2.6.1 Grid facilities

The editor shall provide a grid facility where a matrix of horizontal and vertical lines provide a background to the editor window. This grid shall be a passive grid where the alignment of entities is the user's responsibility.

Rationale: A grid helps the user to create a tidy diagram with well-spaced entities. Although an active grid, where entities 'snap-to' grid lines can be useful, the positioning is imprecise. The user is the best person to decide where entities should be positioned.

Specification: ECLIPSE/WS/Tools/DE/FS Section 5.6

Source: Ray Wilson, Glasgow Office

Requirements completeness and consistency



■ Complete

- They should include descriptions of all facilities required.

■ Consistent

- There should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, it is impossible to produce a complete and consistent requirements document.

Requirements interaction

- Conflicts between different requirements are common in complex systems.
- Example
 - **R1:** After three continues failed login attempts, the account would be locked by the system.
 - **R2:** Once the account is locked, the system sends an account lock notification email to the account's owner.
 - **R3:** Once an account is locked, the system would also send a SMS message to the account's owner to notify him about the situation owner.
 - **R4:** If a user has already received a notification via email, he will not receive the same notification via SMS.

There is a conflict between R2, R3 and R4.

Structured specifications



- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- This works well for some types of requirements, e.g., requirements for embedded control system but is sometimes too rigid for writing business system requirements.

Form-based specifications



1. Definition of the function or entity.
2. Description of inputs and where they come from.
3. Description of outputs and where they go to.
4. Information about the information needed for the computation and other entities used.
5. Description of the action to be taken.
6. Pre and post conditions (if appropriate).
7. The side effects (if any) of the function.

A structured specification of a requirement for an insulin pump

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

A structured specification of a requirement for an insulin pump

Action

CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requirements

Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition

The insulin reservoir contains at least the maximum allowed single dose of insulin.

Post-condition r_0 is replaced by r_1 then r_1 is replaced by r_2 .

Side effects None.

Tabular specification



- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Tabular specification of computation for an insulin pump

Condition	Action
Sugar level falling ($r2 < r1$)	CompDose = 0
Sugar level stable ($r2 = r1$)	CompDose = 0
Sugar level increasing and rate of increase decreasing ($(r2 - r1) < (r1 - r0)$)	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ($(r2 - r1) \geq (r1 - r0)$)	CompDose = $\text{round}((r2 - r1)/4)$ If rounded result = 0 then CompDose = MinimumDose

Key Points



- The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.

- Requirements can be written using different methods
 - Natural language
 - Structured specification

Requirements document structure (1/2)

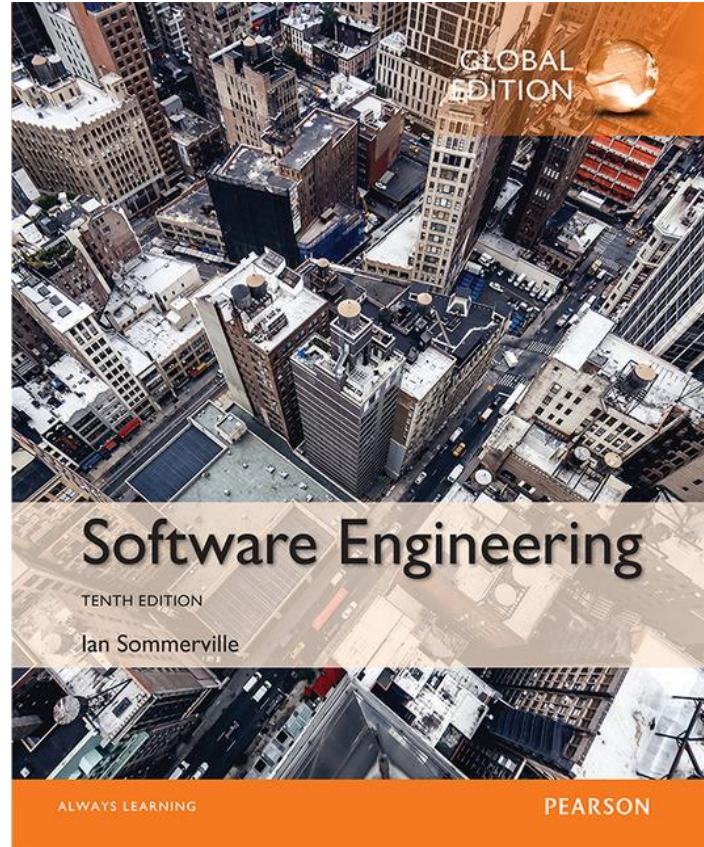
Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

Requirements document structure (2/2)

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Read

Chapter 4



References



- Ian Sommerville, “Software Engineering”, 10th Edition, Addison-Wesley, 2015.
 - Timothy C. Lethbridge and Robert Laganière, “Object-Oriented Software Engineering: Practical Software Development using UML and Java”, 2nd Edition, McGraw Hill, 2001.
 - R. S. Pressman, Software Engineering: A Practitioner’s Approach, 10th Edition, McGraw-Hill, 2005.
- 