

Chapter One

Introduction to Requirements Engineering

What is a requirement?

- The software requirements are description of features and functionalities of the target system.
- Something required, something wanted or needed.
 - Webster's dictionary
- There is a huge difference between *wanted* and *needed* and it should be kept in mind all the time.
- Requirements convey the expectations of users from the software product.
- The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.
- Requirements form the basis of all software engineering projects.

Cont'd...

➤ Requirements are **defined during the early stages of a system development**

- as a specification of what should be implemented or as a constraint of some kind on the system.

□ **Requirements may be:**

- a user-level facility description,
- a detailed specification of expected system behavior,
- a general system property,
- a specific constraint on the system,
- information on how to carry out some computation,
- A constraint on the development of the system.

Cont'd...

Example:

- “Write a program that will read in a list of 100 positive integers, sort them in ascending order, display the sorted list and display the average of the numbers”

➤ **Requirements:-**

- ✓ Read in a list of 100 positive integers
- ✓ Sort the list of integers in ascending order
- ✓ Display the average of the numbers

Types of requirements

- *User requirements*
- *System requirements*
 - Software specifications – provide more (design) detail

Cont'd...

User requirements

- Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge
- User requirements are defined using natural language, tables, and diagrams

Problems with natural language

- Precision vs. understandability
- Functional vs. non-functional requirements confusion
- Requirements amalgamation

Cont'd...

System requirements

- More detailed specifications of user requirements
- Serve as a basis for designing the system
- May be used as part of the system contract
- System requirements may be **expressed using system models.**

Cont'd...

What happens if the requirements are wrong?

- The system may be delivered late and cost more than originally expected.
- The customer and end-users are not satisfied with the system.
- The system may be unreliable in use with regular system errors and crashes disrupting normal operation.
- If the system continues in use, the costs of maintaining the system is very high.

Example: [A Case.pptx](#)

Requirements Engineering

- Refers to the process of defining, documenting and maintaining requirements in the engineering design process.
 - It is a common role in systems engineering and software engineering.
- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.
- Requirements specify *what the system is supposed to do*, but not *how* the system is to accomplish its task.

Cont'd...

- The **process to gather the software requirements from client, analyse and document them** is known as **requirement engineering**.
- The **goal** of requirement engineering is to develop and maintain refined and descriptive ‘**System Requirements Specification**’ document.
- ❖ In practice, requirements engineering isn’t sequential process, it’s an **iterative process** in which activities are interleaved.
 - For example, you iterate first on the user requirements; elicitation, specification, and validation, and repeat the same steps for the system requirements.
- Each software development process goes through the phase of requirements engineering
- The use of the term ‘**engineering**’ implies that systematic and repeatable techniques should be used to ensure that system requirements are complete, consistent, relevant, etc.

Cont'd...

Why is requirements engineering difficult?

- Businesses operate in a **rapidly changing environment** so their requirements for system support are constantly changing.
- Multiple **stakeholders with different goals and priorities** are involved in the requirements engineering process.
- System **stakeholders do not have clear ideas** about the system support that they need.
- Requirements are often influenced by political and organisational factors.

Functional and Non-functional Requirements

Functional Requirements

- Functional requirements define what a system is supposed to *do*.
- Functional requirements are usually in the form of "**system shall do <requirement>**", an individual action or part of the system.
- They define functions and functionality within and from the software system.
- Functional requirement **captures the behavioral aspects / function of the proposed automated system.**
- Functional requirements are the back bone of all software products.

Cont'd...

❑ Requirements are categorized logically as:

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

❖ While developing software, ‘**Must have**’ **must be implemented**, ‘**Should have**’ is a matter of debate with stakeholders, whereas ‘**could have**’ and ‘**wish list**’ can be kept for software updates.

Cont'd...

❖ Examples of functional requirement:

• Requirement #1:

- ✓ The system shall solve a quadratic equation using the following formula.
- ✓ $X = \frac{-b \pm \sqrt{b^2 - 4 * a * c}}{2 * a}$

• Requirement #2:

- The user shall be able to search either the entire database of patients or select a subset from it (admitted patients or patients with asthma, etc)

• Requirement #3:

- The system shall provide appropriate viewers for the user to read documents in the document store.

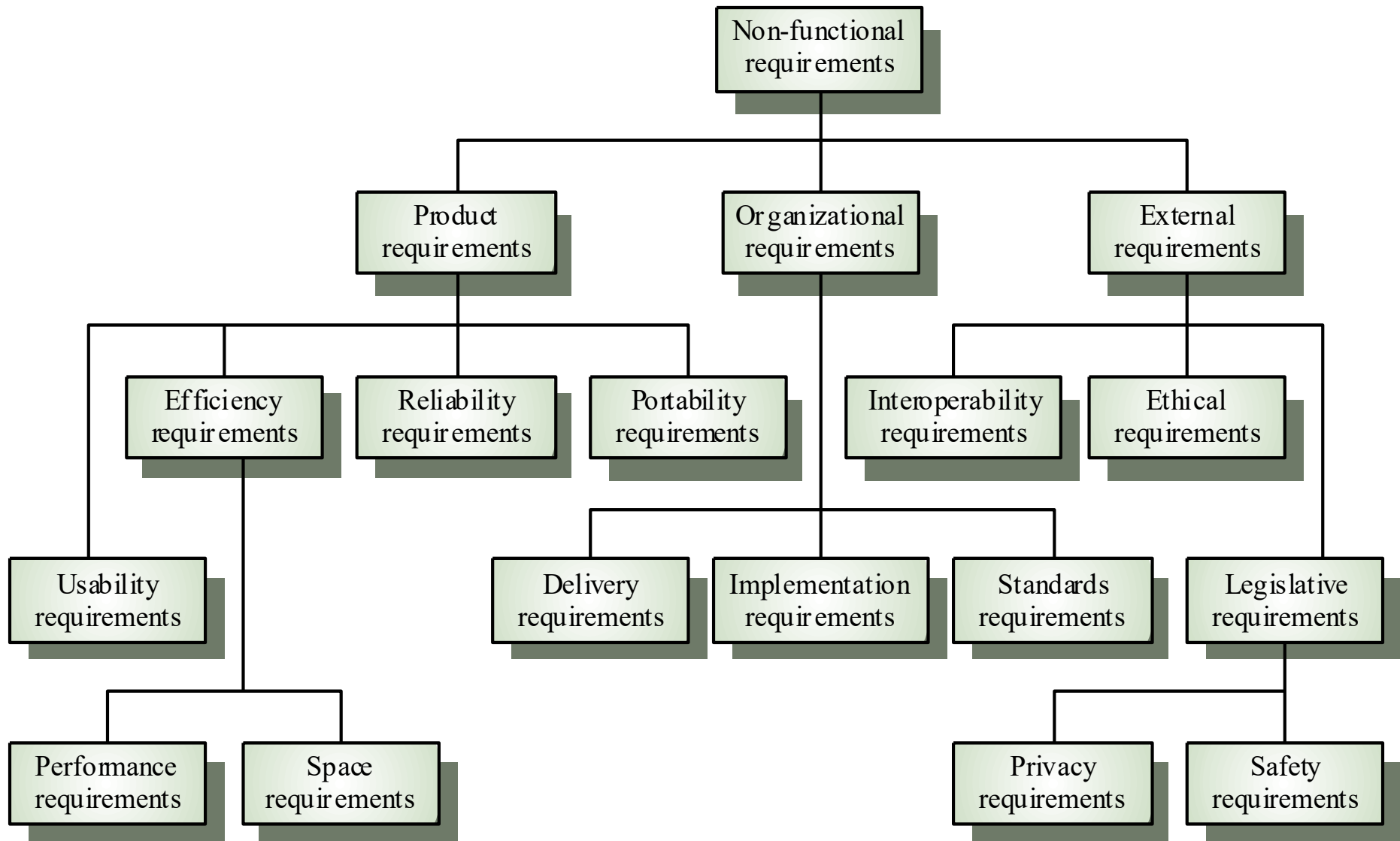
Non-Functional Requirements

- Non-functional requirements define **how a system is supposed to *be***.
- Non-functional requirements are in the form of "**system shall be <requirement>**", **an overall property of the system as a whole** or of a particular aspect and not a specific function.
- Non-functional requirements are often called "**quality attributes**" of a system.
- Other terms for non-functional requirements are "qualities", "quality goals", "quality of service requirements", "constraints" and "non-behavioral requirements".
- Informally these are sometimes called the "**ilities**", from attributes like stability and portability.

Cont'd...

- ❖ Qualities that is non-functional requirements can be divided into two main categories:
 - **Execution qualities**, such as safety, security and usability, which are observable during operation (**at run time**).
 - **Evolution qualities**, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the system.

Types of Non-Functional Requirements (NFRs)



Product-oriented attributes

- ✓ **Performance** : (a) response time, (b) throughput (number of operations performed per second)
- ✓ **Usability**: effort required to learn, use, provide input and interpret results of a program
- ✓ **Efficiency**: minimal use of resources (memory, processor, disk, network...)
- ✓ **Reliability**: of computations, precision
- ✓ **Security**: resistance to unauthorized attempts
- ✓ **Robustness**: in the presence of faults, stress, invalid inputs...
- ✓ **Adaptability**: to other environments or problems
- ✓ **Scalability**: for large number of users or quantities of data
- ✓ **Cost**: total cost of ownership (TCO) for acquisition, installation.
- **Portability**: does it work for several platforms
- **Modifiability**: addition of new functionalities
- **Reusability**: of components, code, designs, and even requirements in other systems

Process-oriented attributes

- **Maintainability:** changes to functionalities, repairs
- **Readability:** of code, documents
- **Testability:** ease of testing and error reporting
- **Understandability:** of design, architecture, code
- **Integrability:** ability to integrate components
- **Complexity:** degree of dependency and interaction between components

Performance Measures

- **Lots of measures**
 - Response time, number of events processed/denied in some interval of time, throughput, capacity, usage ratio, loss of information, latency...
 - Usually with probability and confidence interval.

Cont'd...

- **Examples of performance requirements**
 - The system shall be able to process 100 payment transactions per second in peak load.
 - In standard workload, the CPU usage shall be less than 50%, leaving 50% for background jobs.
 - Production of a simple report shall take less than 20 seconds for 95% of the cases.
 - Scrolling one page up or down in a 200 page document shall take at most 1 second.

Reliability Measures

- **Measure degree to which the system performs as required**
 - Includes resistance to failure
 - Ability to perform a required function under stated conditions for a specified period of time
 - Very important for critical systems
- **Can be measured using**
 - Probability that system will perform its required function for a specified interval under stated conditions
 - Mean-time to failure
 - Defect rate
 - Degree of precision for computations

Cont'd...

- **Examples of Reliability Measures**
 - The system defect rate shall be less than 1 failure per 1000 hours of operation.
 - No more than 1 per 1000000 transactions shall result in a failure requiring a system restart.

Availability Measures

- Definition: Percentage of time that the system is up and running correctly
- Can be calculated based on Mean-Time Between Failure (MTBF) and Mean-Time to Repair (MTTR)
 - **MTBF** : Length of time between failures
 - **MTTR** : Length of time needed to resume operation after a failure
 - $\text{Availability} = \text{MTBF} / (\text{MTBF} + \text{MTTR})$

Cont'd...

- **Examples**

- The system shall meet or exceed 99.99% uptime.
- The system shall not be unavailable more than 1 hour per 1000 hours of operation.
- Less than 20 seconds shall be needed to restart the system after a failure 95% of the time. (This is a MTTR requirement)

- **Availability**

Downtime

- | | |
|------------|-----------------|
| – 90% | 36.5 days/year |
| – 99% | 3.65 days/year |
| – 99.9% | 8.76 hours/year |
| – 99.99% | 52 minutes/year |
| – 99.999% | 5 minutes/year |
| – 99.9999% | 31 seconds/year |

Security Measures

- There are at least two measures:
 - The ability to resist unauthorized attempts at usage
 - Continue providing service to legitimate users while under denial of service attack (resistance to DoS attacks)
- **Measurement methods:**
 - Success rate in authentication
 - Resistance to known attacks
 - Time/efforts/resources needed to find a key
 - Probability/time/resources to detect an attack
 - Percentage of useful services still available during an attack
 - Percentage of successful attacks
 - Lifespan of a password, of a session
 - Encryption level

Cont'd...

- **Security may lead to architectural requirements**
 - Authentication, authorization, audit
 - Detection mechanisms
 - Firewall, encrypted communication channels
- **Examples of Security requirements**
 - The application shall identify all of its client applications before allowing them to use its capabilities.
 - The application shall ensure that the name of the employee in the official human resource and payroll databases exactly matches the name printed on the employee's social security card.
 - At least 99% of intrusions shall be detected within 10 seconds.

Usability Measures

- In general, concerns **ease of use and of training end users**.
- ❖ The following more specific measures can be identified:
 - **Learnability**
 - Proportion of functionalities or tasks mastered after a given training time.
 - **Efficiency**
 - Acceptable response time
 - Number of tasks performed or problems resolved in a given time
 - Number of mouse clicks needed to get to information or functionality
 - **Memorability**
 - Number (or ratio) of learned tasks that can still be performed after not using the system for a given time period
 - **Error avoidance**
 - Number of error per time period and user class
 - Number of calls to user support

Cont'd...

- **Error handling**
 - Mean time to recover from an error and be able to continue the task
- **User satisfaction**
 - Satisfaction ratio per user class
 - Usage ratio

❖ **Examples**

- Four out of five users shall be able to book a guest within 5 minutes after a 2-hour introduction to the system.
- Novice users shall perform tasks X and Y in 15 minutes.
- Experienced users shall perform tasks X and Y in 2 minutes.
- At least 80% of customers asked after a 3 months usage period shall rate their satisfaction with the system at 7 and more on a scale of 1 to 10.

Maintainability Measures

- **Measures ability to make changes quickly and cost effectively**
 - Extension with new functionality
 - Deleting unwanted capabilities
 - Adaptation to new operating environments (portability)
 - Restructuring (rationalizing, modularizing, optimizing, creating reusable components)
- **Can be measured in terms of**
 - Coupling/cohesion metrics, **cyclomatic complexity**(measure of the number of linearly independent paths through a program's source code)
 - Mean time to fix a defect, mean time to add new functionality
 - Quality/quantity of documentation

Cont'd...

- **Measurement tools**
 - code analysis tools such as IBM Structural Analysis for Java (<http://www.alphaworks.ibm.com/tech/sa4j>)
- **Examples of requirements**
 - Every program module must be assessed for maintainability according to procedure xx. 70% must obtain “highly maintainable” and none “poor”.
 - The cyclomatic complexity of code must not exceed 7.
 - No method in any object may exceed 200 lines of code.
 - Installation of a new version shall leave all database contents and all personal settings unchanged.
 - The product shall provide facilities for tracing any database field to places where it is used.

Testability Measures

- Measures the ability to detect, isolate, and fix defects
 - Time to run tests
 - Time to setup testing environment (development and execution)
 - Probability of visible failure in presence of a defect
 - Test coverage (requirements coverage, code coverage...)
- May lead to architectural requirements
 - Mechanisms for monitoring
 - Access points and additional control
- Examples
 - The delivered system shall include unit tests that ensure 100% branch coverage.
 - Development must use regression tests allowing for full retesting in 12 hours.

Portability Measures

- Measure ability of the system to run under different computing environments
 - Hardware, software, OS, languages, versions, combination of these
- **Can be measured as**
 - Number of targeted platforms (hardware, OS...)
 - Proportion of platform specific components or functionality
 - Mean time to port to a different platform
- **Examples**
 - No more than 5% of the system implementation shall be specific to the operating system.
 - The meantime needed to replace the current Relational Database System with another Relational Database System shall not exceed 2 hours.
 - No data loss should arise.

Integrability and Reusability Measures

- **Integrability**
 - Measures ability to make separated components work together
 - Can be expressed as
 - Mean time to integrate with a new interfacing system
- **Reusability**
 - Measures ability that existing components can be reused in new applications
 - Can be expressed as
 - Percentage of reused requirements, design elements, code, tests...
 - Coupling of components
 - Degree of use of frameworks

Robustness Measures

- Measure ability to cope with the unexpected situations.
 - Percentage of failures on invalid inputs
 - Degree of service degradation
 - Minimum performance under extreme loads
 - Active services in presence of faults
 - Length of time for which system is required to manage stress conditions
- **Examples**
 - The estimated loss of data in case of a disk crash shall be less than 0.01%.
 - The system shall be able to handle up to 10000 concurrent users when satisfying all their requirements and up to 25000 concurrent users with browsing capabilities.

Domain-specific Measures

- The most appropriate quality measures may vary from one application domain to another,

❖ E.g.

- **Performance**

- **Web-based system:**

- Number of requests processed per second

- **Video games:**

- Number of 3D images per second

- **Accessibility**

- Web-based system:

- Compliance with standards for the blind

- Video games:

- Compliance with age/content ratings systems (e.g., no violence)

Thank You!

?

Chapter Two

Requirements Engineering Processes

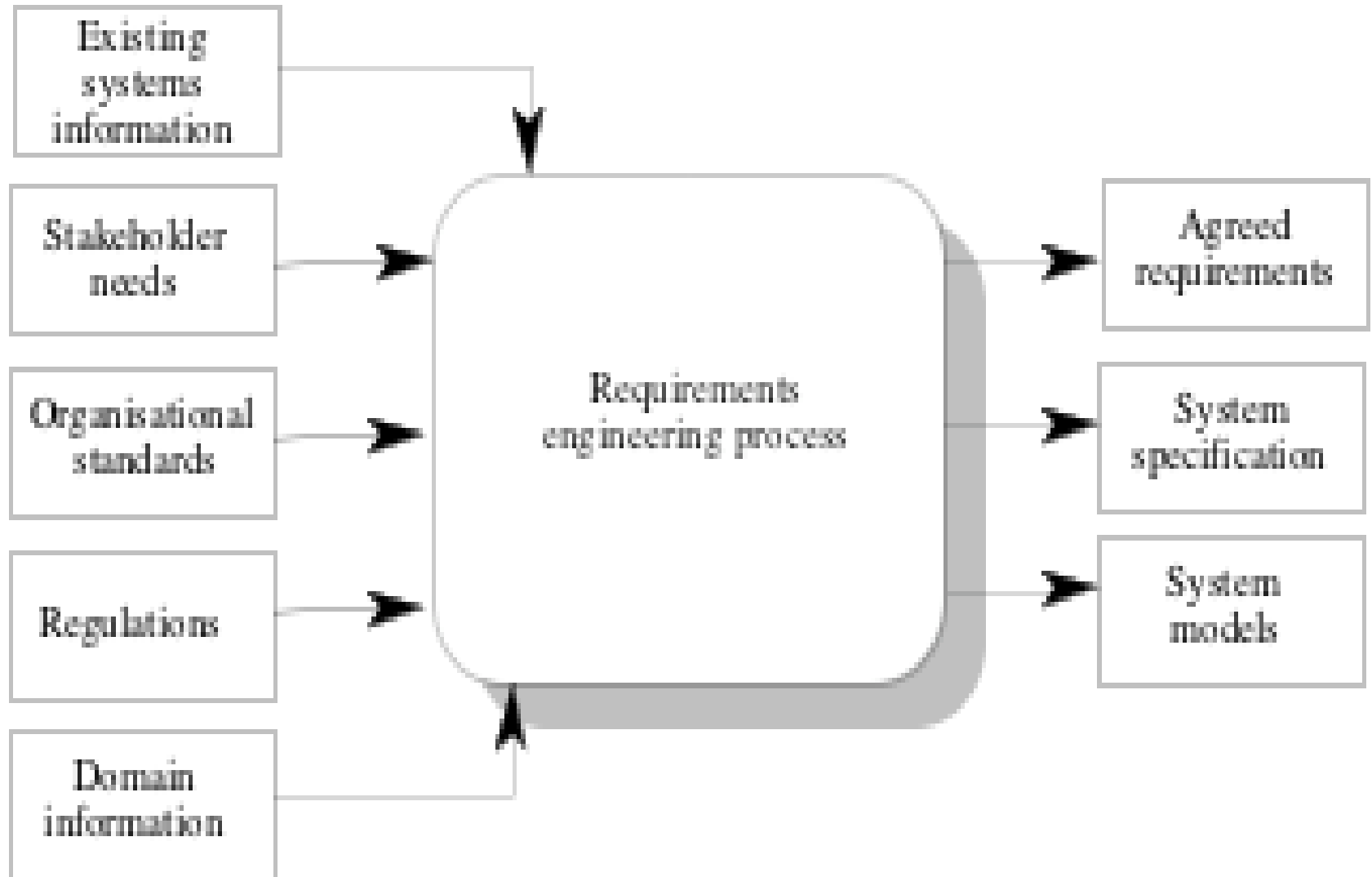
Processes

- ◆ A process is an organized set of activities which transforms inputs to outputs.
- ◆ Process descriptions encapsulate knowledge and allow it to be reused.
- ◆ Examples of process descriptions
 - Instruction manual for a dishwasher
 - Cookery book
 - Procedures manual for a bank
 - Quality manual for software development

Design processes

- ◆ Processes which involve creativity, interactions between a wide range of different people, engineering judgement and background knowledge and experience.
- ◆ Examples of design processes
 - Writing a book
 - Organizing a conference
 - Designing a processor chip
 - Requirements engineering

RE process - inputs and outputs



Input/Output description

Input or output	Type	Description
Existing system information	Input	Information about the functionality of systems to be replaced or other systems which interact with the system being specified
Stakeholder needs	Input	Descriptions of what system stakeholders need from the system to support their work
Organisational standards	Input	Standards used in an organisation regarding system development practice, quality management, etc.
Regulations	Input	External regulations such as health and safety regulations which apply to the system.

Cont'd...

Input or output	Type	Description
Domain information	Input	General information about the application domain of the system.
Agreed requirements	Output	A description of the system requirements which is understandable by stakeholders and which has been agreed by them.
System specification	Output	This is a more detailed specification of the system functionality which may be produced in some cases.
System models	Output	A set of models such as a data-flow model, an object model, a process model, etc. which describes the system from different perspectives

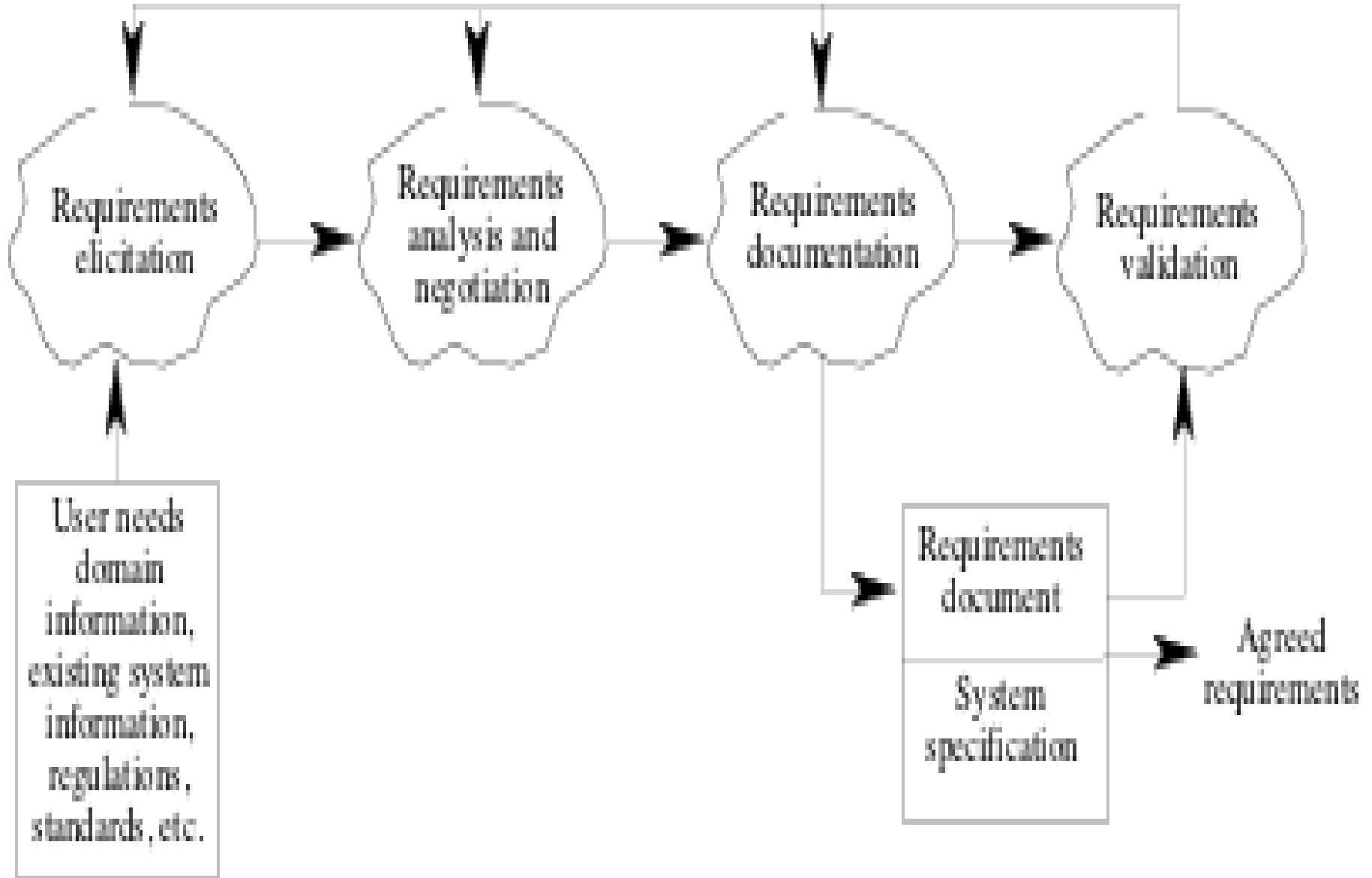
RE process variability

- ◆ RE processes vary radically from one organization to another.
- ◆ Factors contributing to this variability include
 - Technical maturity
 - Disciplinary involvement
 - Organizational culture
 - Application domain
- ◆ There is therefore no ‘ideal’ requirements engineering process

Process models

- ◆ A process model is a simplified description of a process presented from a particular perspective.
- ◆ Types of process model include:
 - Coarse-grain activity models
 - Fine-grain activity models
 - Role-action models
 - Entity-relation models

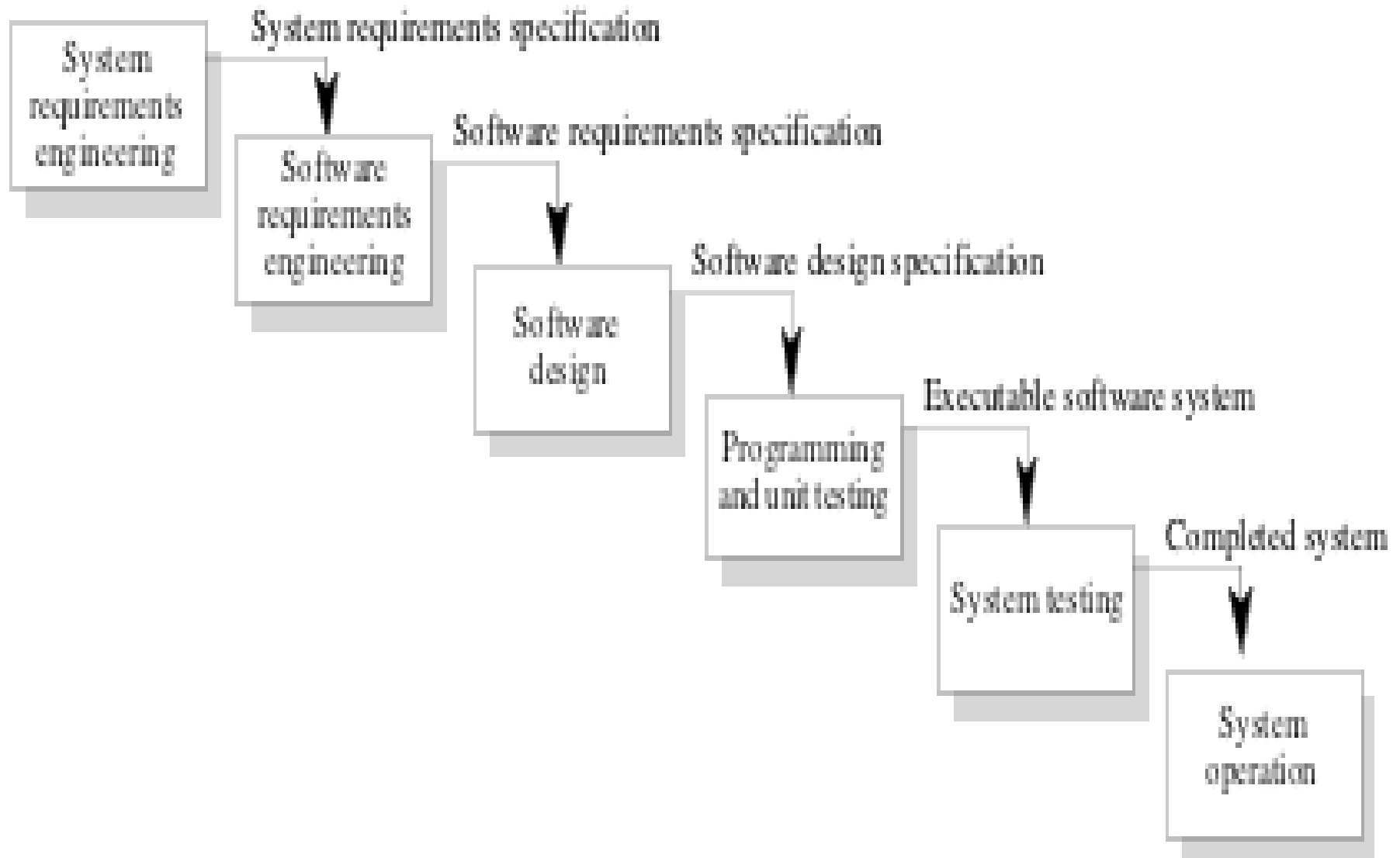
Coarse-grain activity model of RE



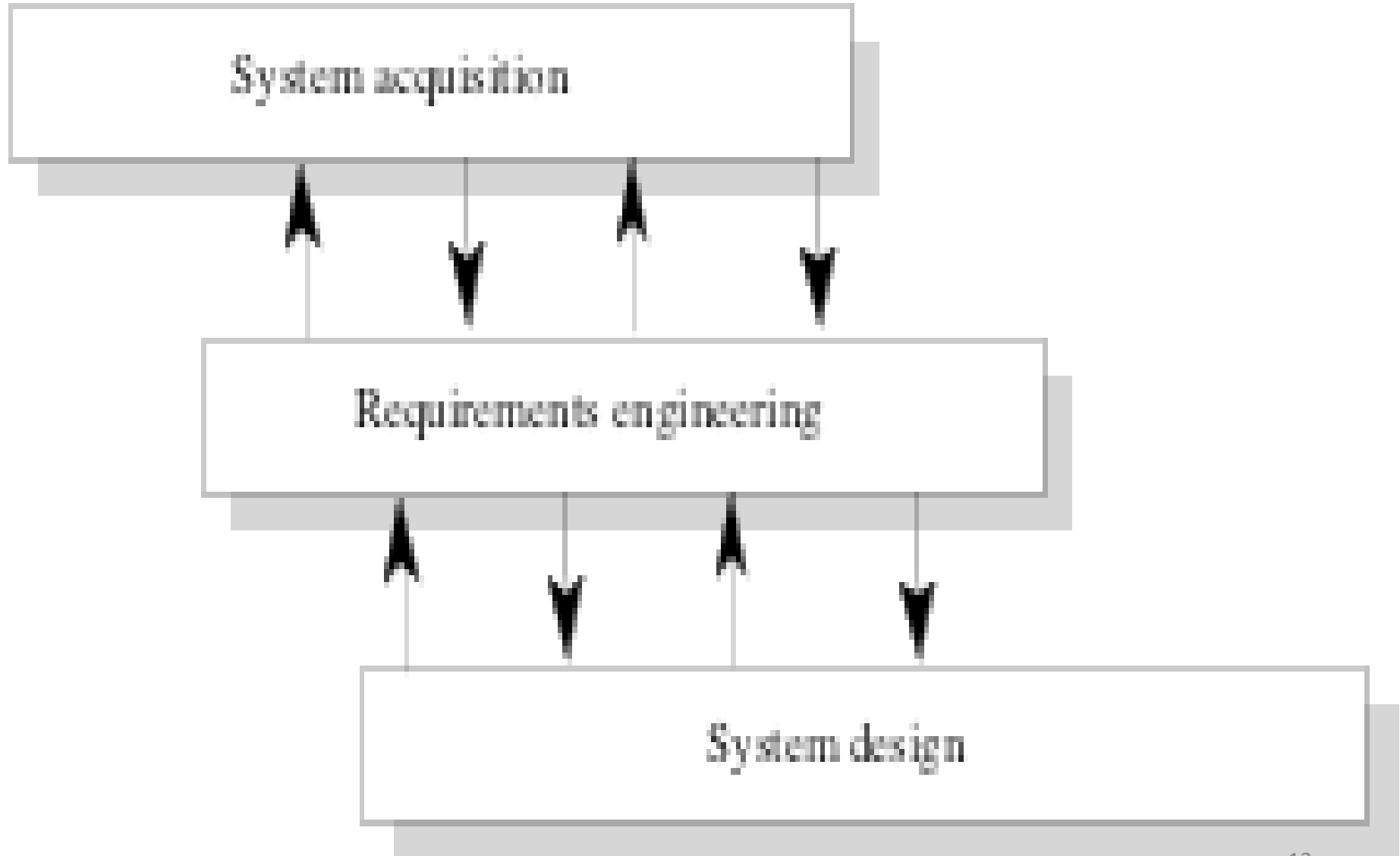
RE process activities

- ◆ Requirements elicitation
 - Requirements discovered through consultation with stakeholders.
- ◆ Requirements analysis and negotiation
 - Requirements are analyzed and conflicts resolved through negotiation
- ◆ Requirements documentation
 - A requirements document is produced.
- ◆ Requirements validation
 - The requirements document is checked for consistency and completeness.

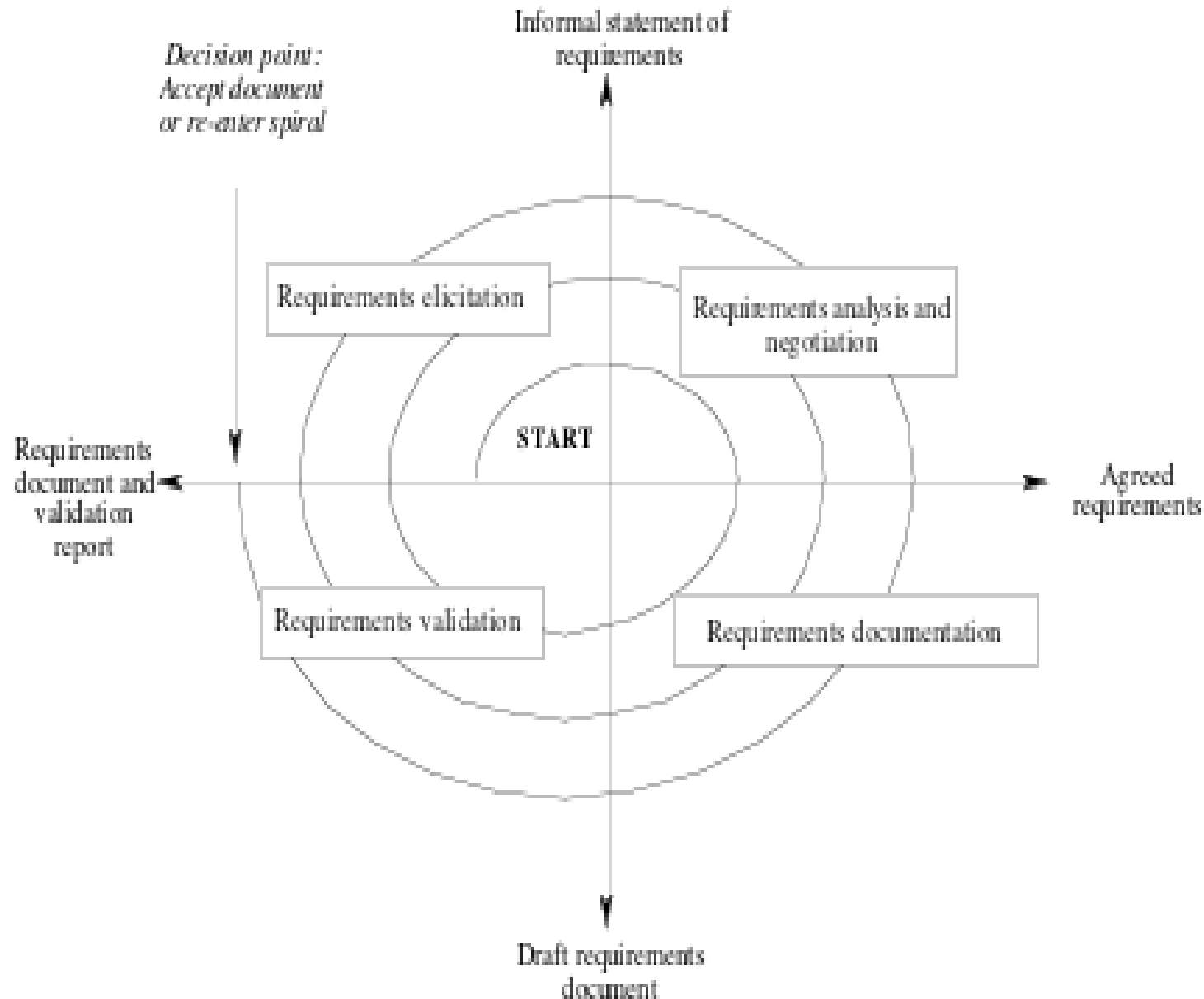
Waterfall model of the software process



Context of the RE process



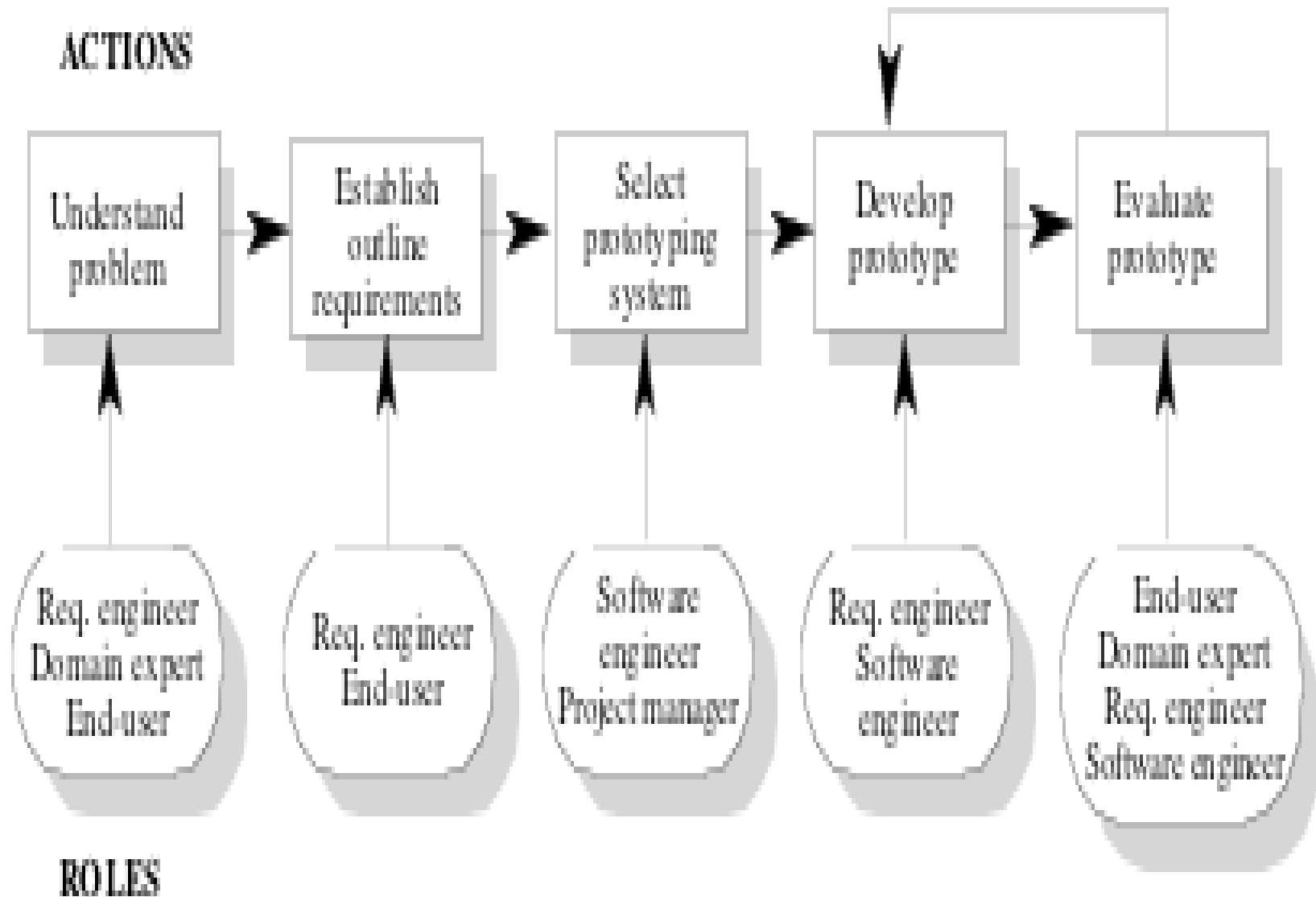
Spiral model of the RE process



Actors in the RE process

- ◆ Actors in a process are the people involved in the execution of that process.
- ◆ Actors are normally identified by their roles rather than individually.
- ◆ Requirements engineering involves actors who are primarily interested in the problem to be solved (end-users, etc.) as well actors interested in the solution (system designers, etc.)
- ◆ **Role-action diagrams** document which actors are involved in different activities.

RAD for software prototyping



Role descriptions

Role	Description
Domain Expert	Responsible for providing information about the application domain and the specific problem in that domain which is to be solved.
System end-user	Responsible for using the system after delivery.
Requirements Engineer	Responsible for eliciting and specifying the system requirements.
Software Engineer	Responsible for developing the prototype software system.
Project Manager	Responsible for planning and estimating the prototyping project.

Human and social factors

- ◆ Requirements engineering processes are dominated by human, social and organizational factors because they always involve a range of stakeholders from different backgrounds and with different individual and organizational goals.
- ◆ System stakeholders may come from a range of technical and non-technical background and from different disciplines.

Types of stakeholder

- ◆ **Software engineers** responsible for system development.
- ◆ **System end-users** who will use the system after it has been delivered.
- ◆ **Managers of system end-users** who are responsible for their work.
- ◆ **External regulators** who check that the system meets its legal requirements.
- ◆ **Domain experts** who give essential background information about the system application domain.

Factors influencing requirements

- ◆ Personality and status of stakeholders
- ◆ The personal goals of individuals within an organization
- ◆ The degree of political influence of stakeholders within an organization.

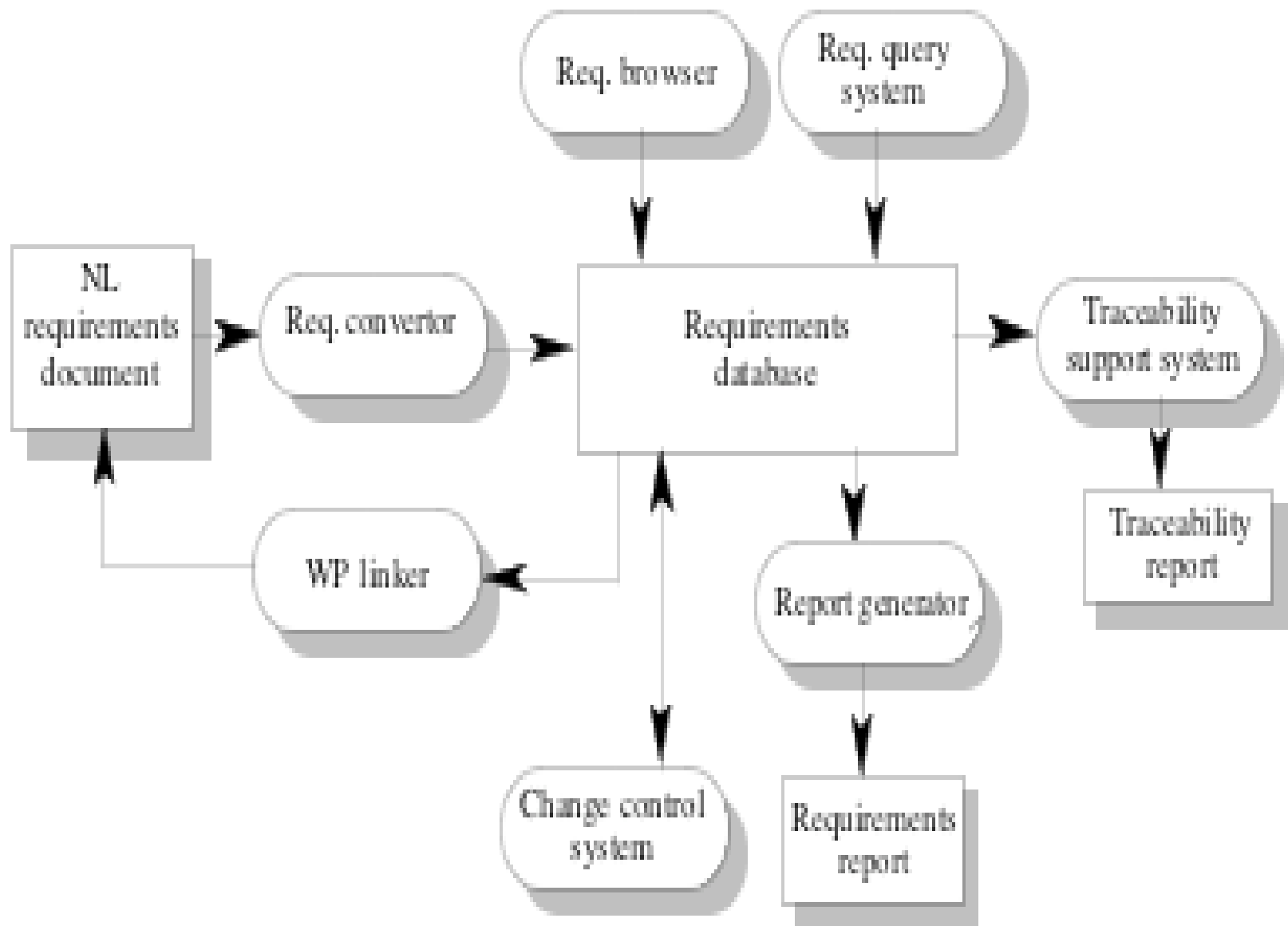
Process support

- ◆ CASE tools provide automated support for software engineering processes.
- ◆ The most **mature CASE tools** support well-understood activities such as programming and testing and the use of structured methods.
- ◆ Support for requirements engineering is still limited because of the informality and the variability of the process.

CASE tools for RE

- ◆ Modelling and validation tools support the development of system models which can be used to specify the system and the checking of these models for completeness and consistency.
- ◆ Management tools help manage a database of requirements and support the management of changes to these requirements.

A requirements management system



Requirements management tools

- ◆ Requirements browser
- ◆ Requirements query system
- ◆ Traceability support system
- ◆ Report generator
- ◆ Requirements converter and word processor linker
- ◆ Change control system

Process improvement

- ◆ Process improvement is concerned with modifying processes in order to meet some improvement objectives.
- ◆ Improvement objectives
 - Quality improvement
 - Schedule reduction
 - Resource reduction

Planning process improvement

- ◆ What are the problems with current processes?
- ◆ What are the improvement goals?
- ◆ How can process improvement be introduced to achieve these goals?
- ◆ How should process improvements be controlled and managed?

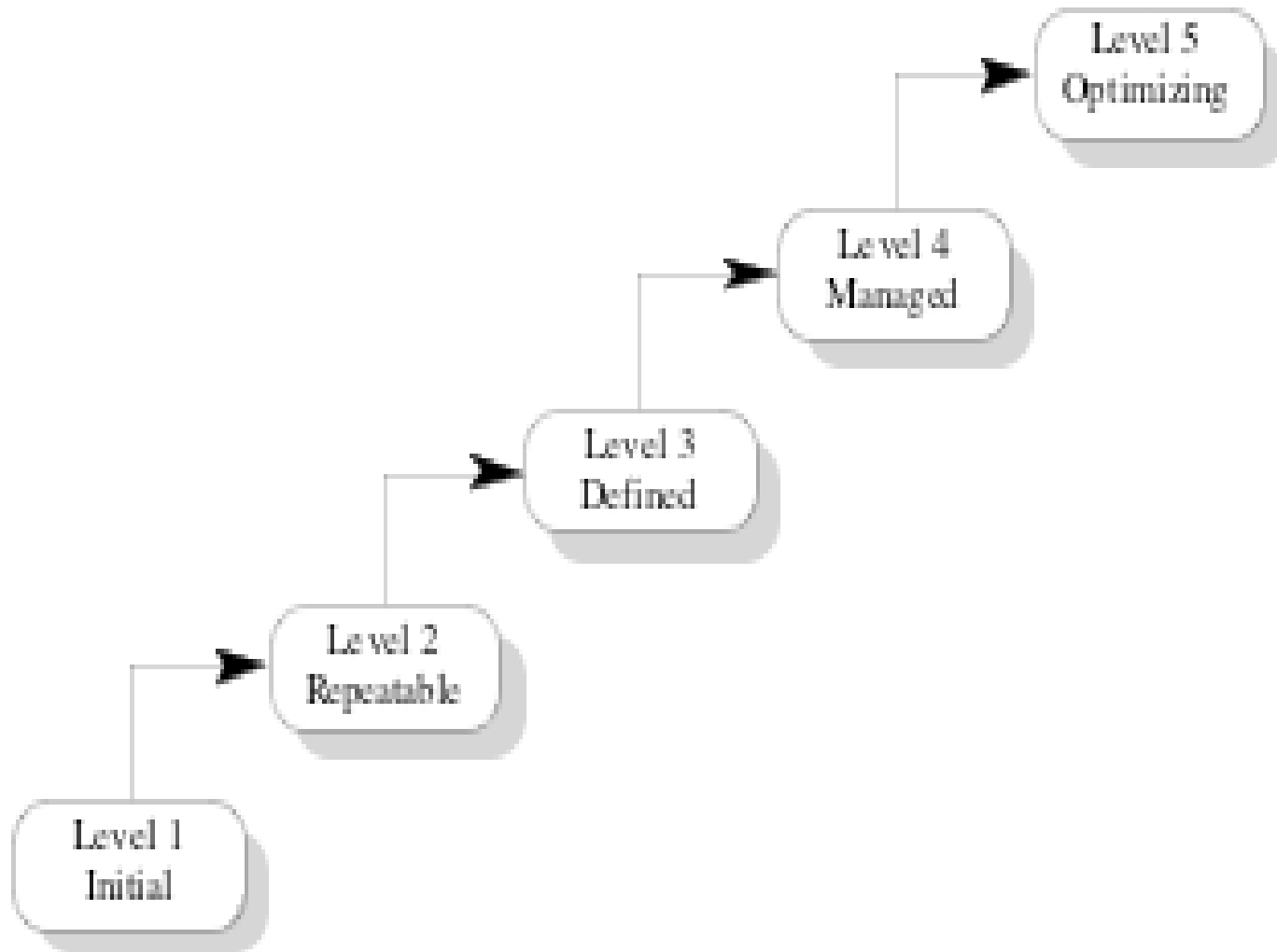
RE process problems

- ◆ Lack of stakeholder involvement
- ◆ Business needs not considered
- ◆ Lack of requirements management
- ◆ Lack of defined responsibilities
- ◆ Stakeholder communication problems
- ◆ Over-long schedules and poor quality requirements documents

Process maturity

- ◆ Process maturity can be thought of as the extent that an organization has defined its processes, actively controls these processes and provides systematic human and computer-based support for them.
- ◆ The SEI's Capability Maturity Model is a framework for assessing software process maturity in development organizations

Capability maturity model



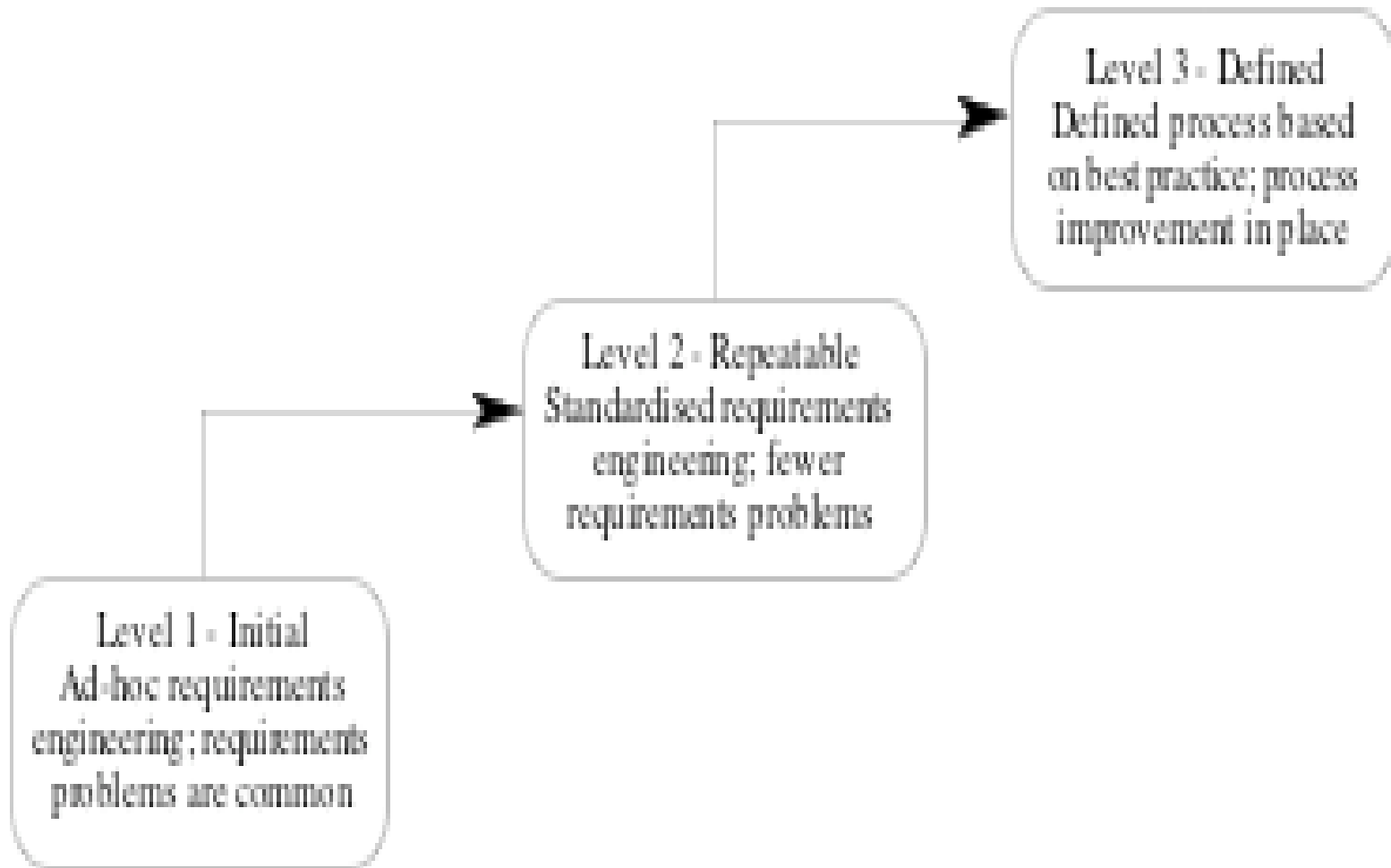
Maturity levels

- ◆ Initial level
 - Organizations have an undisciplined process and it is left to individuals how to manage the process and which development techniques to use.
- ◆ Repeatable level
 - Organizations have basic cost and schedule management procedures in place. They are likely to be able to make consistent budget and schedule predictions for projects in the same application area.
- ◆ Defined level
 - The software process for both management and engineering activities is documented, standardized and integrated into a standard software process for the organization.

Cont'd...

- ◆ Managed level
 - Detailed measurements of both process and product quality are collected and used to control the process.
- ◆ Optimizing level
 - The organization has a continuous process improvement strategy, based on objective measurements, in place.

RE process maturity model



RE process maturity levels

- ◆ Initial level
 - No defined RE process.
 - Suffer from requirements problems such as requirements volatility, unsatisfied stakeholders and high rework costs.
 - Dependent on individual skills and experience.
- ◆ Repeatable level
 - Defined standards for requirements documents and policies and procedures for requirements management.
- ◆ Defined level
 - Defined RE process based on good practices and techniques.
 - Active process improvement process in place.

Good practice for RE process improvement

- ◆ RE processes can be improved by the systematic introduction of good requirements engineering practice.
- ◆ Each improvement cycle identifies good practice guidelines and works to introduce them in an organization.

Examples of good practice guidelines

- ◆ Define a standard document structure
- ◆ Uniquely identify each requirement
- ◆ Define policies for requirements management
- ◆ Use checklists for requirements analysis
- ◆ Use scenarios to elicit requirements
- ◆ Specify requirements quantitatively
- ◆ Use prototyping to animate requirements
- ◆ Reuse requirements

Key points

- ◆ The requirements engineering process is a structured set of activities which lead to the production of a requirements document.
- ◆ Inputs to the requirements engineering process are information about existing systems, stakeholder needs, organizational standards, regulations and domain information.
- ◆ Requirements engineering processes vary radically from one organization to another. Most processes include requirements elicitation, requirements analysis and negotiation and requirements validation.

Cont'd...

- ◆ Requirements engineering process models are simplified process description which are presented from a particular perspective.
- ◆ Human, social and organizational factors are important influences on requirements engineering processes.
- ◆ Requirements engineering process improvement is difficult and is best tackled in an incremental way.
- ◆ Requirements engineering processes can be classified according to their degree of maturity.

Thank You!

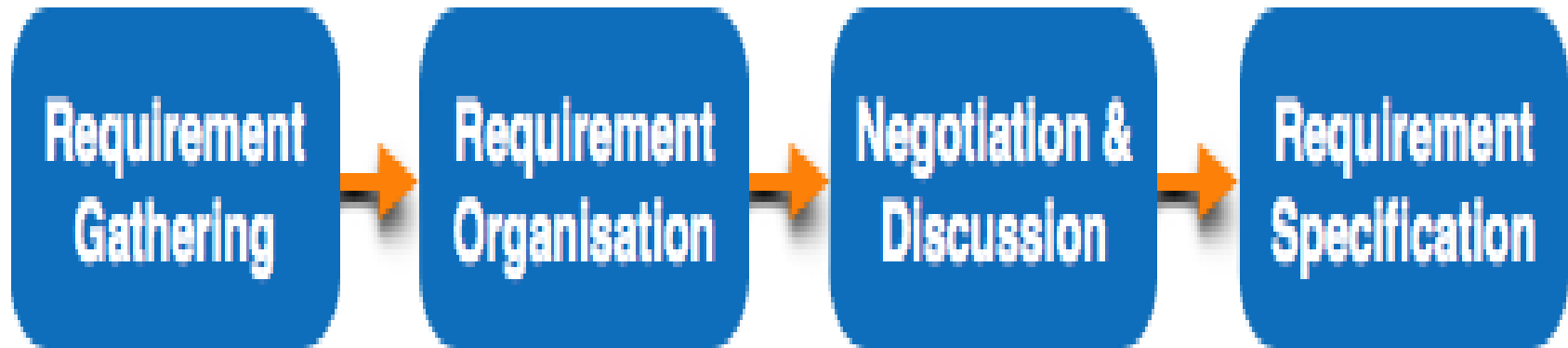
?

Chapter Three

Requirement Elicitation and Analysis

Requirements Elicitation and Analysis

- Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.
- Requirement elicitation process can be depicted using the following diagram:



Cont'd...

- **Requirements gathering** - The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** - If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders.
 - Requirements may then be prioritized and reasonably compromised.
- ❑ The requirements come from various stakeholders.
 - To remove the ambiguity and conflicts, they are discussed for clarity and correctness.
 - **Unrealistic requirements** are compromised reasonably.
- **Documentation** - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

- **There are various ways to discover requirements.**
 - ✓ **Interviews**
 - ✓ **Surveys**
 - ✓ **Questioners**
 - ✓ **Task analysis**
 - ✓ **Domain analysis**
 - ✓ **Scenarios**
 - ✓ **Brainstorming**
 - ✓ **Prototyping**

Cont'd...

1. Interviews

- Interviews are strong medium to collect requirements.
- Organization may conduct several types of interviews such as:
 - **Structured (closed) interviews**, the requirements engineer looks for answers to a pre-defined set of questions.
 - **Non-structured (open) interviews**, where there is no pre-defined agenda and the requirements engineer discusses, in an open-ended way, what stakeholders want from the system.
 - more flexible and less biased.
 - **Oral interviews**
 - **Written interviews**
 - **One-to-one interviews** which are held between two persons across the table.
 - **Group interviews** which are held between groups of participants.
 - They help to uncover any missing requirement as numerous people are involved.

Cont'd...

2. Surveys

- Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

3. Questionnaires

- A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.
- A **shortcoming** of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Cont'd...

4. Task analysis

- Team of engineers and developers may **analyze the operation** for which the new system is required.
- If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

5. Domain Analysis

- Every software falls into some domain category.
- The **expert people in the domain** can be a great help to analyze general and specific requirements.

Cont'd...

6. Scenarios

- Bridging the gap between user and developer:
 - *Scenarios*: Example of the use of the system in terms of a series of interactions with between the user and the system
 - *Use cases*: Abstraction that describes a class of scenarios

7. Brainstorming

- An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Cont'd...

8. Observation

- Team of experts visit the client's organization or workplace.
- They observe the actual working of the existing installed systems.
- They observe the workflow at client's end and how execution problems are dealt.
- The team itself draws some conclusions which aid to form requirements expected from the software.
- ❖ **Ethnography** is a technique from the social sciences which has proved to be valuable in **understanding actual work** processes.
 - involves an observer spending an extended period in a society or culture, making detailed observation of all their practices.

Cont'd...

9. Prototyping

- A prototype is **a demonstration system** which shows end-users and system stakeholders what facilities the system can provide.
- If you have vague or poorly understood requirements, you should consider developing a prototype system which **simulates the behavior of system software and hardware.**
- End-users can experiment with this to refine their ideas about the system requirements.

Benefits of prototype

- I. Prototyping makes the real meaning of requirements easier to understand.
 - If stakeholders do not fully understand the system requirements, they may agree to a requirements specification which does not reflect their real needs.
 - The ultimate system which is developed is likely to be unsuitable.
- II. Prototyping is the only effective way of developing system user interfaces.
 - If a prototype has been developed as part of the requirements process, this can reduce later development costs for the system.

Cont'd...

III. The prototype system may help establish the overall feasibility and usefulness of the system before high development costs are incurred.

IV. For some types of system, which are primarily software,

- the **final system** can sometimes be developed by modifying and adding facilities to the prototype.
- This means that a limited but usable system may be available relatively quickly.

Implementation of prototypes

- There are three possible approaches to system prototyping:
 - I. Paper prototyping** where a mock-up of the system is developed and used for system experiments.
 - II. 'Wizard of Oz' prototyping** where a person simulates the responses of the system in response to some user inputs
 - III. Automated prototyping** where a fourth generation language or other rapid development environment is used to develop an executable prototype.

Paper prototyping

- Is a **cheap** and surprisingly **effective** approach to prototype development.
- It is **most suitable** for establishing end-user requirements **for software systems**.
- Paper versions of the screens which might be presented to the end-user are drawn and various usage scenarios are planned.
- Analysts and end-users work through these scenarios to find users reactions to the system, the information they require and how they would normally interact with the system.

'Wizard of Oz' prototyping

- Is also **relatively cheap** as it does not require much software to be developed.
- The user interacts with what appears to be the system but his or her inputs are actually channeled to a person who simulates the responses of the system.
- **This approach is particularly useful** when a new system has to be developed based on an existing interface.
- Users are familiar with the interface and can see the interactions between it and the system functionality simulated by the 'Wizard of Oz'.

Automated Prototyping

- ❑ Developing an executable prototype of the system is a more expensive option.
 - It involves writing software to simulate the functionality of the system to be delivered.
 - It is very important that the prototype should be developed quickly and this means that you should use very high level languages and support environments for prototype development.

Cont'd...

- ❑ Depending on the class of system you might use any of the following.

1. Fourth generation languages based around database systems.

- These are good for prototyping applications which involve **information management**.
- However, they do have restrictions which are inherent in the interaction facilities which they provide.
- You must develop your prototype within these limitations and this may mean that some facilities (e.g. navigation around a database by using a graphical visualization of the data) may be impossible.

Cont'd...

2. Very high-level languages such as Visual Basic or Smalltalk.

- These are general purpose programming language which usually come with a powerful development environment and access to a range of reusable objects.
- These **allow applications to be developed quickly** and with more flexibility than is usually provided with 4GLs.

Cont'd...

3. Internet-based prototyping solutions based on World-Wide-Web browsers and languages such as Java.

- Here, you have a ready-made user interface.
- You add functionality to it by associating segments of Java programs with the information to be displayed.
- These segments (called **applets**) are executed automatically when the page is loaded into the browser.
- This approach is a fast way of developing user interface prototypes but you must accept the inherent restrictions imposed by the browser.
- At the time of writing, this approach seems to have a lot of potential but is still immature.

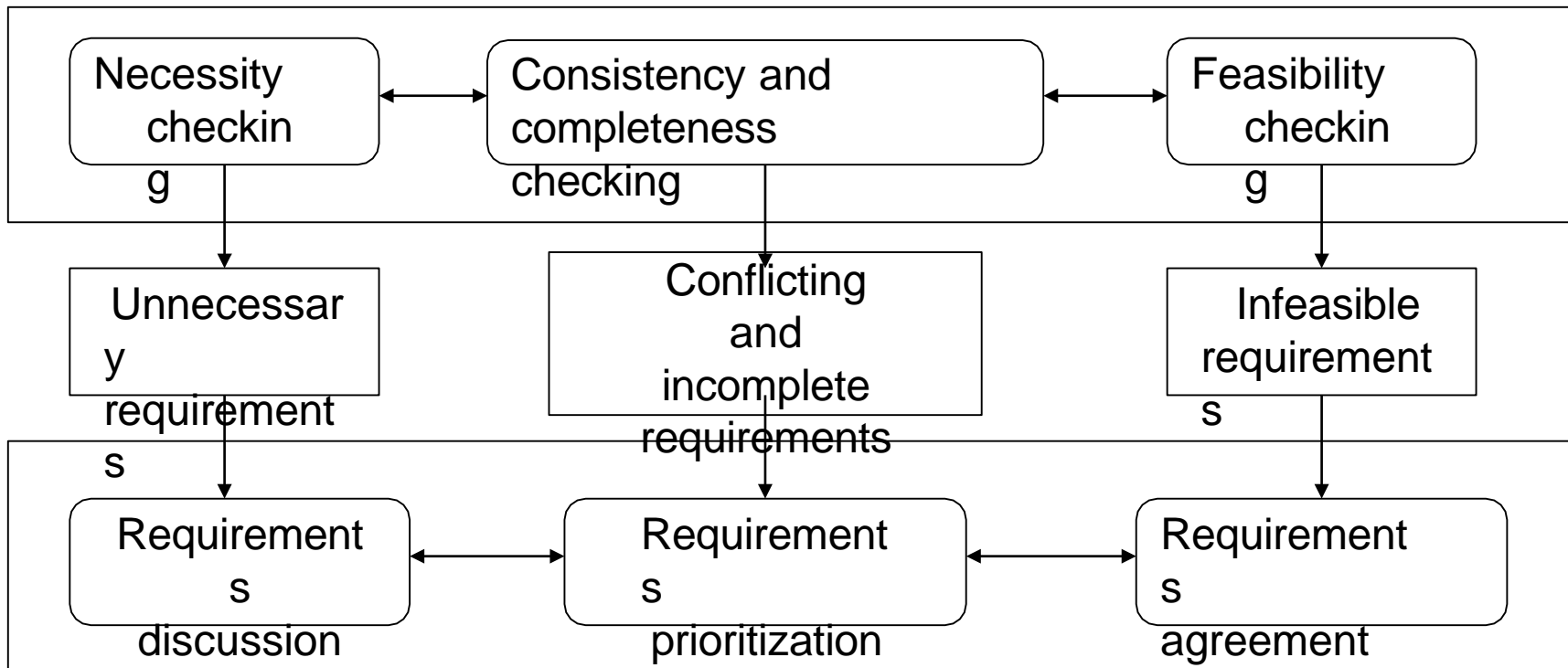
Cont'd...

- ❖ In general, because the performance, efficiency, reliability, etc., of a prototype is quite different from a finished system, you can only use prototypes to discover the functional requirements for a system.

Requirements Analysis and Negotiation

- Requirements analysis and negotiation are inter-leaved activities and join to form a major activity of the requirements engineering process.

Requirements Analysis

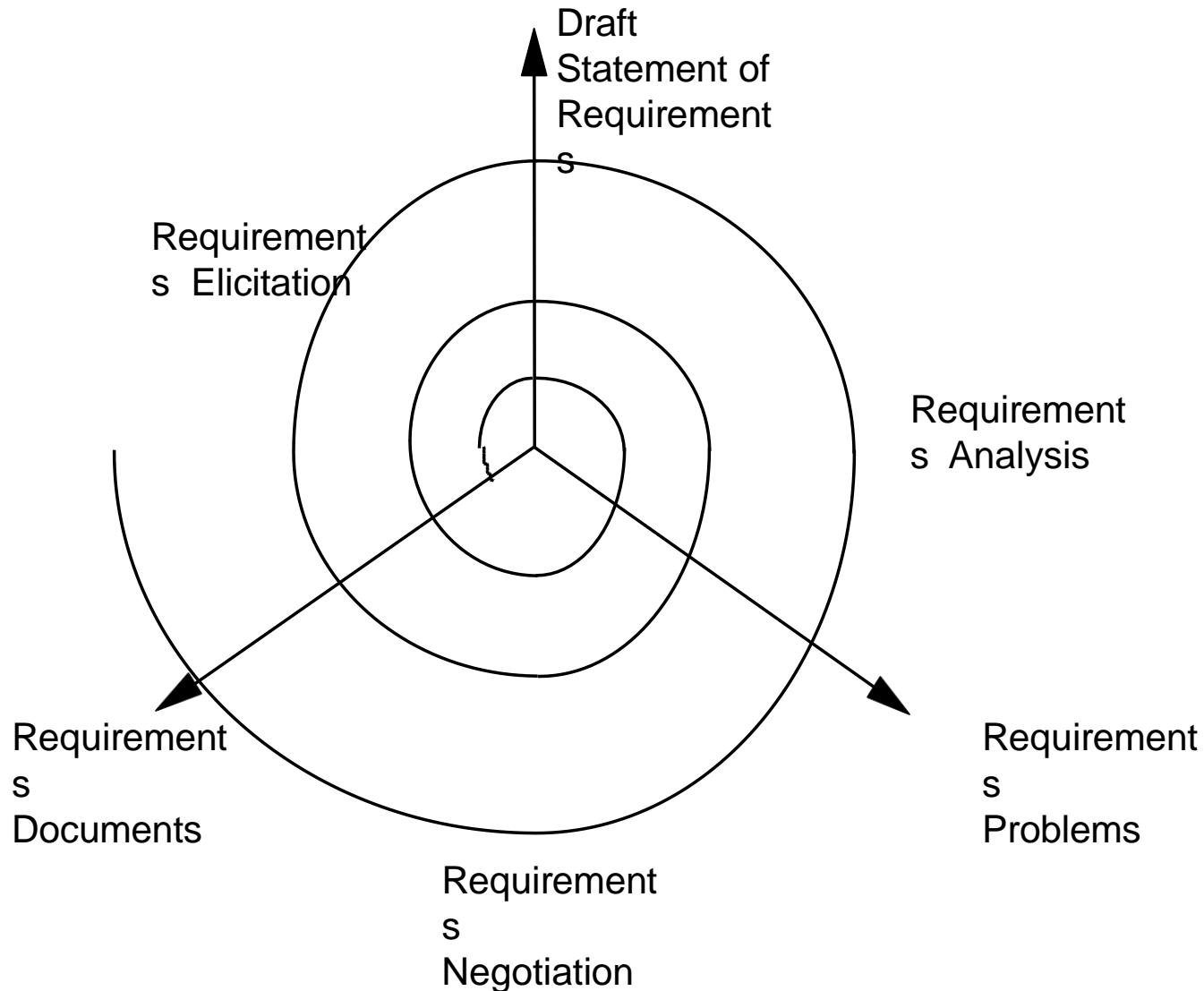


Requirements Negotiation

Requirements Analysis

- The **aim** of requirements analysis is to **discover problems with the system requirements**, especially incompleteness and inconsistencies.
- Some analysis is inter-leaved with requirements elicitation as problems are sometimes obvious as soon as a requirement is expressed.
- ❑ Detailed analysis usually takes place after the initial draft of the requirements document is produced.
- **Analysis is concerned with incomplete set of requirements, which has not been discussed by stakeholders.**

Iterative Aspects of Elicitation, Analysis, and Negotiation



Comments on Requirements Analysis

- **Analysts** read the requirements, highlight problems, and discuss them in requirements review meetings.
 - This is a time-consuming and expensive activity.
- Analysts have to think about implications of the draft statements of requirements.
- ❖ People do not think in the same way and different analysts tackle the process in different ways.
- It is not possible to make this activity a structured and systematic process.
 - It depends on the judgment and experience of process participants.

Requirements Analysis Stages

- I. Necessity checking
- II. Consistency and completeness checking
- III. Feasibility checking

Necessity Checking

- The need for the requirement is analyzed.
- In some cases, requirements may be proposed which don't contribute to the business goals of the organization or to the specific problem to be addressed by the system.

Cont'd...

Consistency and Completeness Checking

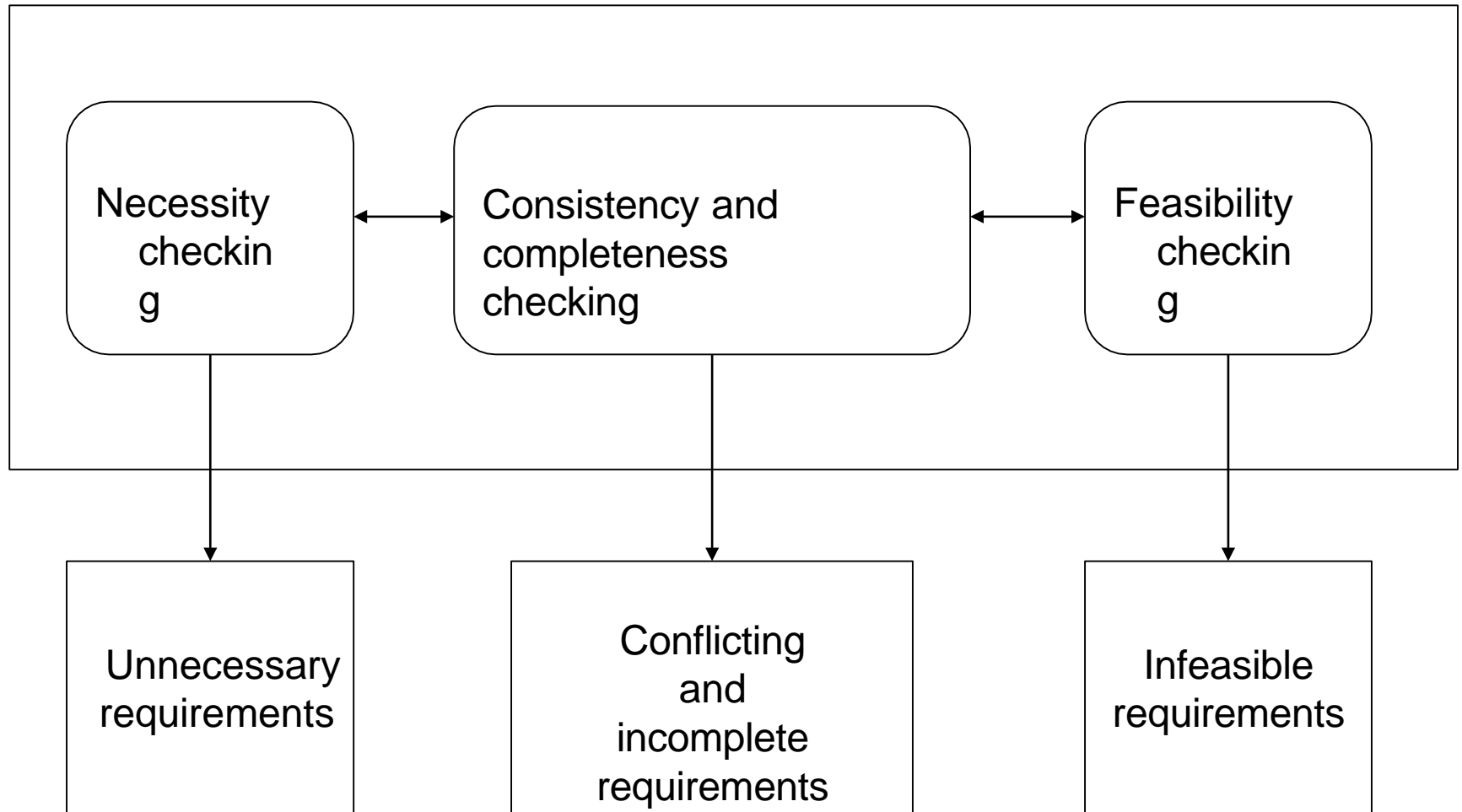
- The requirements are cross-checked for consistency and completeness.
- **Consistency** means that **no requirements should be contradictory**
- **Completeness** means that **no services** or constraints which are needed have been **missed out**.

Feasibility Checking

- The requirements are checked to ensure that they are feasible in the context of the **budget** and **schedule** available for the system development.

Requirements Analysis Process

Requirements Analysis



Analysis Techniques

- **Analysis checklists**
 - A checklist is a list of questions which analysts may use to assess each requirement.
- **Interaction matrices**
 - Interaction matrices are used to **discover** interactions between requirements and to highlight **conflicts** and **overlaps**.

Cont'd...

Analysis Checklists

- Each requirement may be assessed against the checklist
- When potential problems are discovered, these should be noted carefully
- They can be implemented as a spreadsheet, where the **rows** are labeled with the **requirements identifiers** and **columns** are the **checklist items**
- They are useful as they provide a reminder of what to look for and reduce the chances that you will forget some requirements checks.

Cont'd...

- They must evolve with the experience of the requirements analysis process.
- ❑ The questions should be general, rather than restrictive, which can be irrelevant for most systems.
- ✓ Checklists should not include more than ten items, because people forget items on long checklists reading through a document.
- ❖ The following list shows a possible requirement analysis checklist:

Checklist Items and their Description

- **Premature design**
 - Does the requirement include premature design or implementation information?
- **Combined requirements**
 - Does the description of a requirement describe a single requirement or could it be broken down into several different requirements?
- **Unnecessary requirements**
 - Is the requirement ‘gold plating’? That is, is the requirement a cosmetic addition to the system which is not really necessary?

Cont'd...

■ **Use of non-standard hardware**

- Does the requirement mean that non-standard hardware or software must be used?
 - To make this decision, you need to know the computer platform requirements.

■ **Conformance with business goals**

- Is the requirement consistent with the business goals defined in the introduction to the requirements document?

Cont'd...

■ **Requirements ambiguity**

- Is the requirement ambiguous i.e., could it be read in different ways by different people? What are the possible interpretations of the requirement?

■ **Requirements realism**

- Is the requirement realistic given the technology which will be used to implement the system?

■ **Requirements testability**

- Is the requirement testable, that is, is it stated in such a way that **test engineers can derive a test which can show if the system meets that requirement?**

Requirements Interactions

- A very important objective of requirements analysis is to discover the interactions between requirements and to **highlight requirements conflicts and overlaps.**
- A **requirements interaction matrix** shows how requirements interact with each other, which can be constructed using a spreadsheet.
 - Each requirement is compared with other requirements, and the matrix is filled as follows:
 - For requirements which **conflict**, fill in a **1**
 - For requirements which **overlap**, fill in a **1000**
 - For requirements which are **independent**, fill in a **0**
- **Consider the following example**

An Interaction Matrix

Requirement	R1	R2	R3	R4	R5	R6
R1	0	0	1000	0	1	1
R2	0	0	0	0	0	0
R3	1000	0	0	1000	0	1000
R4	0	0	1000	0	1	1
R5	1	0	0	1	0	0
R6	1	0	1000	1	0	0

Comments on Interaction Matrices

- If you can't decide whether requirements conflict, you should assume that a conflict exists.
- If an error is made it is usually fairly cheap to fix; it can be much more expensive to resolve undetected conflicts.
- In the example, we are considering, we can see that R1 overlaps with R3 and conflicts with R5 and R6.
- R2 is an independent requirement.
- R3 overlaps with R1, R4, and R6.
- The advantage of using numeric values for conflicts and overlaps is that you can sum each row and column to find the number of conflicts and the number of overlaps.
- ❖ Requirements which have high values for one or both of these figures should be carefully examined.

Cont'd...

- A large number of conflicts or overlaps means that any changes to that requirement will probably have a major impact on the rest of the requirements
- ❖ Interaction matrices work only when there is relatively small number of requirements, as each requirement is compared with every other requirement
 - The upper limit should be about **200** requirements.
- These overlaps and conflicts have to be discussed and resolved during **requirements negotiation**, which we'll discuss next.

Requirements Negotiation

- Disagreements about requirements are inevitable when a system has many stakeholders.
- **Conflicts** are not ‘failures’ but reflect **different stakeholder needs and priorities**.
- **Requirements negotiation** is the process of **discussing requirements conflicts** and **reaching a compromise that all stakeholders can agree to**.
 - In planning a requirements engineering process, it is important to leave enough time for negotiation.
 - Finding an acceptable compromise can be time-consuming.
- The final requirements will always be a compromise which is governed by the needs of the organization in general, the specific requirements of different stakeholders, design and implementation constraints, and the budget and schedule for the system development.

Requirements Negotiation Stages

1. Requirements Discussion

- Requirements which have been highlighted as problematic are discussed and the stakeholders involved present their views about the requirements.

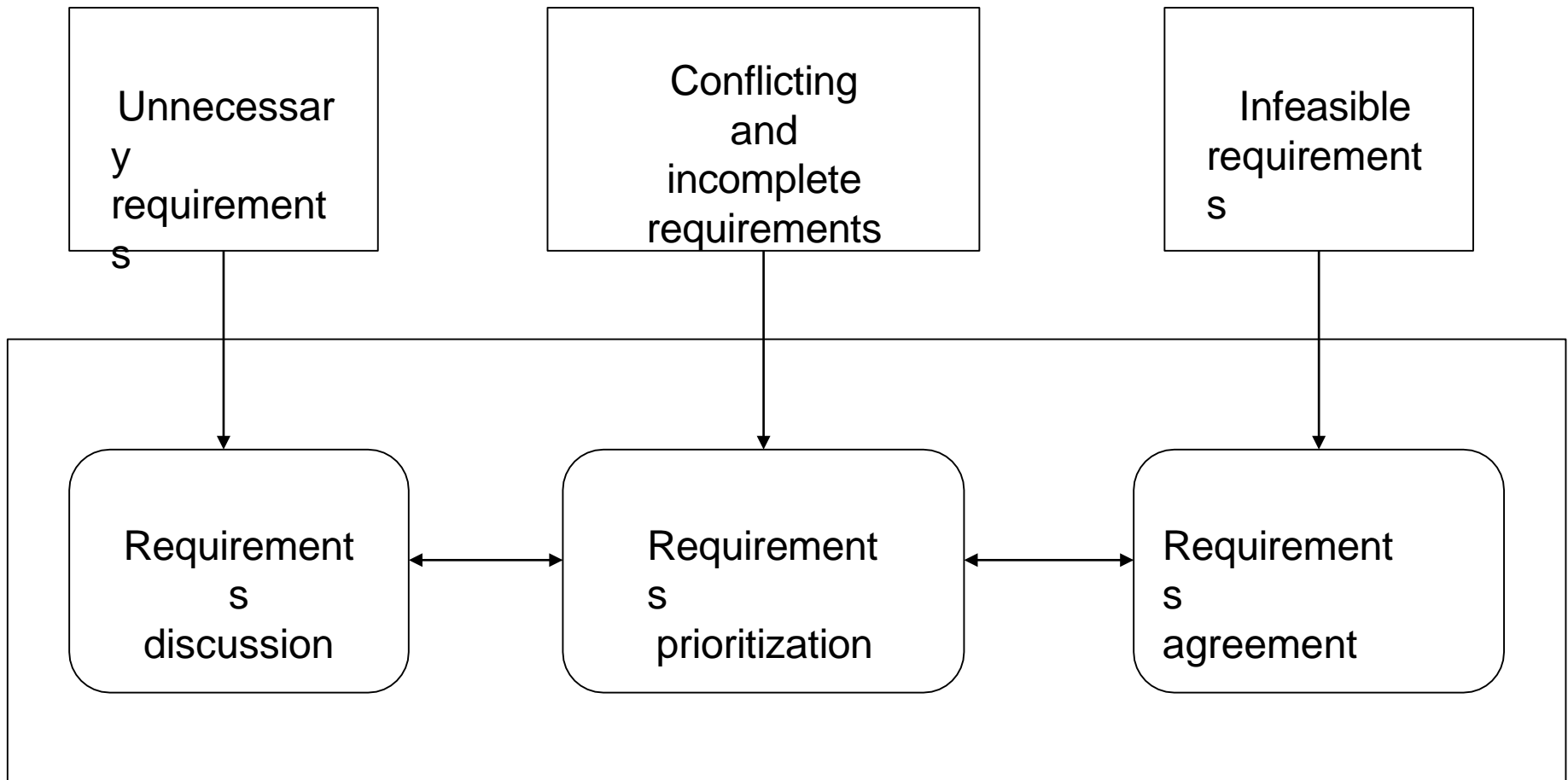
2. Requirements Prioritization

- Disputed/unclear requirements are prioritized to **identify critical requirements** and to help the decision making process.

3. Requirements Agreement

- Solutions to the requirements problems are identified and a compromised set of requirements are reached.
- Generally, this will **involve making changes to some of the requirements.**

Requirements Negotiation Process



Requirements
Negotiation

Comments on Requirements Negotiation

- ❑ In principle, requirements negotiation should be an objective process.
- The judgments should be the requirements for the system and should be based on technical and organizational needs.
- Reality is, however, often different.
- ❑ Negotiations are rarely conducted using only logical and technical arguments.
- They are influenced by organizational and political considerations, and the personalities of the people involved.
- A strong personality may force their priorities on other stakeholders.
- Requirements may be accepted or rejected because they strengthen the political influence in the organization of some stakeholders.
- End-users may be resistant to change and may block requirements,⁴¹ etc.

Cont'd...

- The majority of time in requirements negotiation is usually spent resolving requirements conflicts.
- A requirement conflicts with other requirements if they ask for different things.
- Conflicts should not be viewed as ‘failures’, they are natural and inevitable – rather healthy.
- **Meetings are the most effective way to negotiate requirements and resolve requirements conflicts.**
- All requirements which are in conflict should be discussed individually.
- ❑ **Negotiation meetings should be conducted in three stages.**

Stages of Negotiation Meetings

1. Information Stage

- The nature of the problems associated with a requirement is explained.

2. Discussion Stage

- A discussion stage where the stakeholders involved discuss how these problems might be resolved
 - All stakeholders with an interest in the requirement should be given the opportunity to comment.
 - Priorities may be assigned to requirements at this stage.

Cont'd...

3. Resolution Stage

- A resolution stage where actions concerning the requirement are agreed
 - These actions might be to delete the requirement, to suggest specific modifications to the requirement or to elicit further information about the requirement.

Thank You!
?

Chapter Five

**SDLC and Software Development Model
Approaches: Use-case-oriented modeling**

Introduction

- Software development life cycle (SDLC) is a series of phases that provide a common understanding of the software building process.
 - The good software engineer should have enough knowledge on how to choose the SDLC model based on the project context and the business requirements.
- Therefore, it may be required to **choose the right SDLC model** according to the **specific concerns and requirements** of the project **to ensure its success.**

Types of Software developing Model Approaches

- Waterfall model
- V-Model
- Incremental Model
- Agile Model
- Iterative Model and many more.

Waterfall model

- This model has five phases:
 - ✓ Requirements analysis and specification
 - ✓ design,
 - ✓ implementation, and unit testing,
 - ✓ integration and system testing, and
 - ✓ operation and maintenance.
- The steps always follow in this order and do not overlap.
- The developer must complete every phase before the next phase begins.
- This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.

Cont'd...



Cont'd...

A. Requirements analysis and specification phase:

- The aim of this phase is to understand the exact requirements of the customer and to document them properly.
- In this phase, a large document called **SRS** document is created which contained a detailed description of what the system will do in the common language.

B. Design Phase:

- This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language.
- It defines the overall software architecture together with high level and detailed design.
- All this work is documented as a **Software Design Document (SDD)**.

Cont'd...

C. Implementation and unit testing:

- During this phase, **design is implemented**.
- If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.
- **During testing, the code is thoroughly examined and modified.**
- Small modules are tested in isolation initially.
- After that these modules are tested by writing some code to check the interaction between these modules and the flow of intermediate output.

Cont'd...

D. Integration and System Testing:

- This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out.
- The better output will lead to satisfied customers, lower maintenance costs, and accurate results.
- In this phase, the modules are tested for their **interactions** with each other and with the system.

E. Operation and maintenance phase

- Maintenance is the task performed once the software has been delivered to the customer, installed, and operational.

Cont'd...

When to use SDLC Waterfall Model?

- Some Circumstances where the use of the Waterfall model is most suited are:
 - When the requirements are constant and not changed regularly.
 - A project is short
 - The situation is calm
 - Where the tools and technology used is consistent and is not changing
 - When resources are well prepared and are available to use.

Cont'd...

Advantages of Waterfall model

- This model is simple to implement also the number of resources that are required for it is minimal.
- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
- The start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development.
- It gives easy to control and clarity for the customer due to a strict reporting system.

Cont'd...

Disadvantages of Waterfall model

- In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
- This model cannot accept the changes in requirements during development.
- It becomes tough to go back to the phase.
 - For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
- Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

V-Model

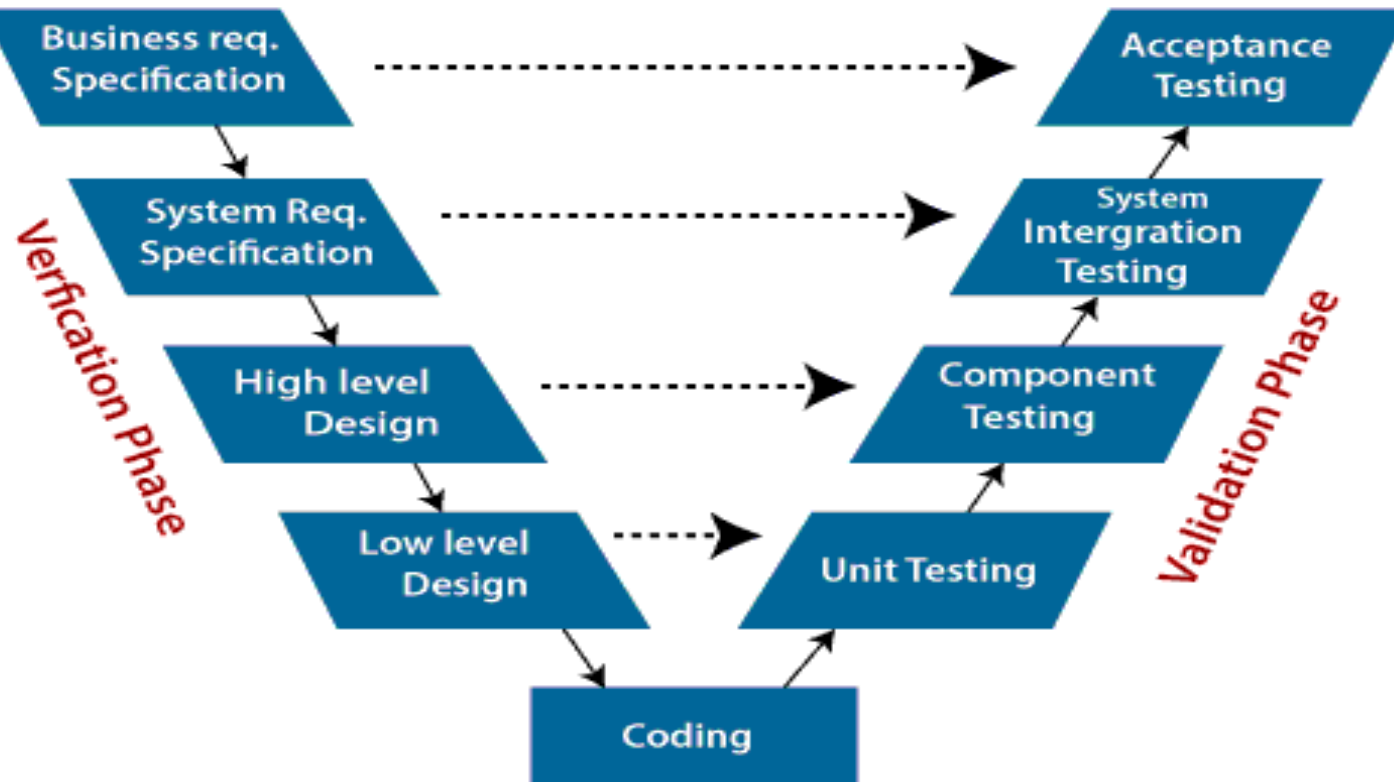
- It is also referred to as the Verification and Validation Model.
- Each phase of SDLC must complete before the next phase starts.
- It follows a sequential design process same as the waterfall model.
- Testing is planned in parallel with a corresponding stage of development.

Cont'd...

V- Model

Developer's life Cycle

Tester's Life Cycle



Cont'd...

Verification:

- It involves a static analysis method (review) done without executing code.
- It is the process of **evaluation of the product development process to find whether specified requirements meet.**

Validation:

- It involves dynamic analysis method (functional, non-functional), testing is done by executing code.
- Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.
- Verification and Validation process is joined by coding phase in V-shape.
- Thus it is known as V-Model.

Cont'd...

- **There are the various phases of Verification Phase of V-model:**

A. Business requirement analysis:

- This is the first step where product requirements understood from the customer's side.
- It contains detailed communication to understand customer's expectations and exact requirements.

B. System Design:

- In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.

C. Architecture Design:

- The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc.
- **The integration testing** model is carried out in a particular phase.

Cont'd...

D. Module Design:

- the system breaks down into small modules.
- The detailed design of the modules is specified, which is known as Low-Level Design

E. Coding Phase:

- After designing, the coding phase is started.
- Based on the requirements, a suitable programming language is decided.
- There are some guidelines and standards for coding.
- Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

Cont'd...

- **There are the various phases of Validation Phase of V-model:**

A. Unit Testing:

- Unit Test Plans (UTPs) are developed during the module design phase.
- These UTPs are executed to eliminate errors at code level or unit level.
- A unit is the smallest entity which can independently exist, e.g., a program module.
- **Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.**

B. Integration Testing:

- Integration Test Plans are developed during the Architectural Design Phase.
- These tests verify that groups created and tested independently can coexist and communicate among themselves.

Cont'd...

C. System Testing:

- System Tests Plans are developed during System Design Phase.
- Unlike Unit and Integration Test Plans, System Tests Plans are composed by the clients business team.
- System Test ensures that expectations from an application developer are met.

D. Acceptance Testing:

- Acceptance testing is related to the business requirement analysis part.
- It includes testing the software product in user atmosphere.
- Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere.

Cont'd...

When to use V-Model?

- When the requirement is well defined and not ambiguous.
- The V-shaped model should be used for **small to medium-sized projects** where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantage of V-Model:

- Easy to Understand.
- **Testing Methods like planning, test designing happens well before coding.**
- This saves a lot of time. Hence a higher chance of success over the waterfall model.
- **Avoids the downward flow of the defects.**
- Works well for small plans where requirements are easily understood.

Cont'd...

Disadvantage of V-Model:

- Very rigid and least flexible.
- Not a good for a complex project.
- Software is developed during the implementation stage, so no early prototypes of the software are produced.
- If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

Incremental Model

- It is a process of software development where requirements divided into multiple standalone modules of the software development cycle.
- Each module goes through the requirements, design, implementation and testing phases.
- Every subsequent release of the module adds function to the previous release.
- The process continues until the complete system achieved.

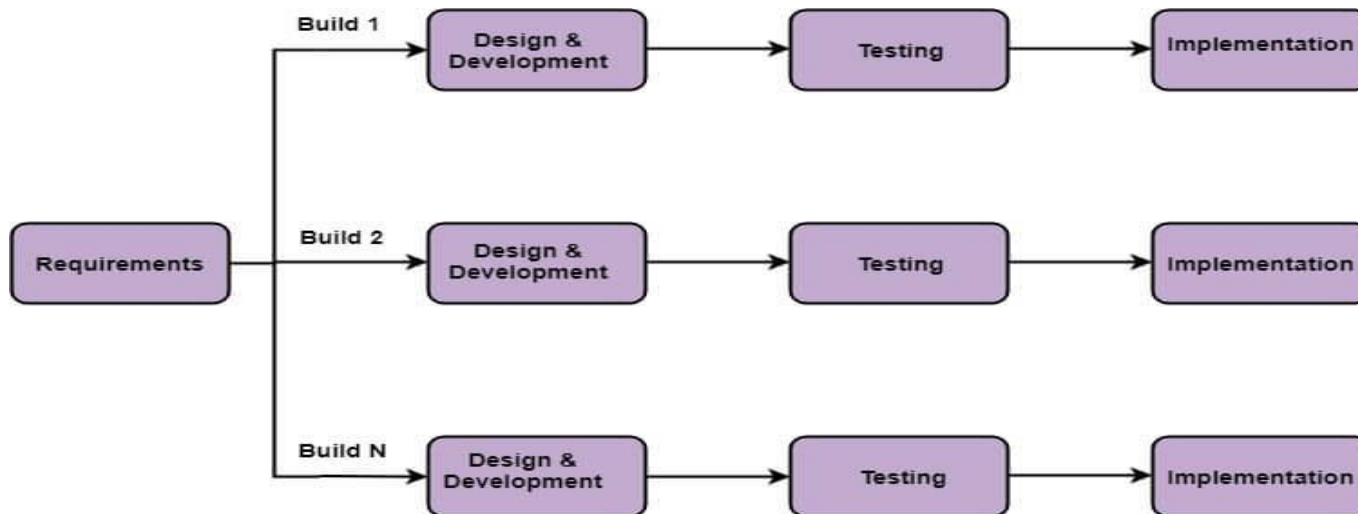


Fig: Incremental Model

Cont'd...

The various phases of incremental model are as follows:

1. Requirement analysis:

- In the first phase of the incremental model, the product analysis expertise identifies the requirements.
- And the system functional requirements are understood by the requirement analysis team.
- To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development:

- In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success.

Cont'd...

3. Testing:

- the testing phase checks the performance of each existing function as well as additional functionality.
- the various methods are used to test the behavior of each task.

4. Implementation:

- Implementation phase enables the coding phase of the development system.
- It involves the final coding.
- After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

Cont'd...

When we use the Incremental Model?

- When the requirements are more.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Cont'd...

Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

Agile Model

- The meaning of Agile is swift or versatile.“
- **Agile process model"** refers to a software development approach based on iterative development.
- Agile methods **break tasks into smaller iterations**, or parts do not directly involve long term planning.
- The project scope and requirements are laid down at the beginning of the development process.
- **Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.**

Cont'd...

- Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks.
- The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements.
- Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

Cont'd...

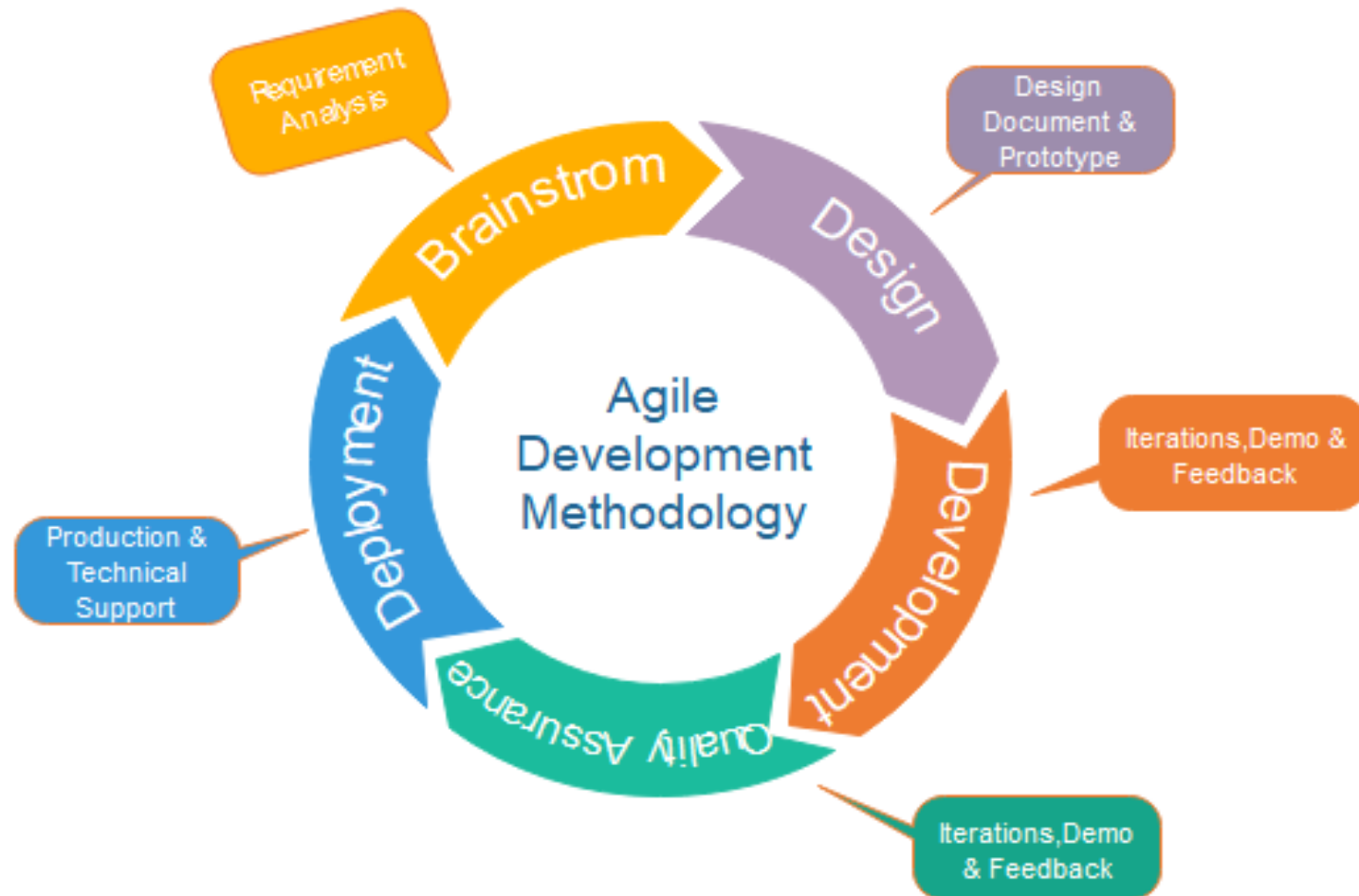


Fig. Agile Model

Cont'd...

Phases of Agile Model

1. Requirements gathering:

- In this phase, you must define the requirements.
- You should explain business opportunities and plan the time and effort needed to build the project.
- Based on this information, you can evaluate technical and economic feasibility.

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements.

- You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

Cont'd...

- 3. Construction/ iteration:** When the team defines the requirements, the work begins.
 - Designers and developers start working on their project, which aims to deploy a working product.
 - The product will undergo various stages of improvement, so it includes simple, minimal functionality.
- 4. Testing:** In this phase, the Quality Assurance team examines the product's performance and looks for the bug.
- 5. Deployment:** In this phase, the team issues a product for the user's work environment.
- 6. Feedback:** After releasing the product, the last step is feedback.
 - In this, the team receives feedback about the product and works through the feedback.

Cont'd...

When to use the Agile Model?

- When frequent changes are required.
- When a highly qualified and experienced team is available.
- When a customer is ready to have a meeting with a software team all the time.
- When project size is small.

Advantage(Pros) of Agile Method:

- Frequent Delivery
- Face-to-Face Communication with clients.
- Efficient design and fulfils the business requirement.
- Anytime changes are acceptable.
- It reduces total development time.

Cont'd...

Disadvantages(Cons) of Agile Model:

- Due to the shortage of formal documents, it **creates confusion** and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
- **Due to the lack of proper documentation**, once the project completes and the developers allotted to another project, **maintenance of the finished project can become a difficulty.**

Iterative Model

- In this Model, you can start with some of the software specifications and develop the first version of the software.
- After the first version if there is a need to change the software, then a new version of the software is created with a new iteration.
- Every release of the Iterative Model finishes in an exact and fixed period that is called iteration.
- The Iterative Model allows the accessing of earlier phases, in which the variations made respectively.
- The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.

Cont'd...

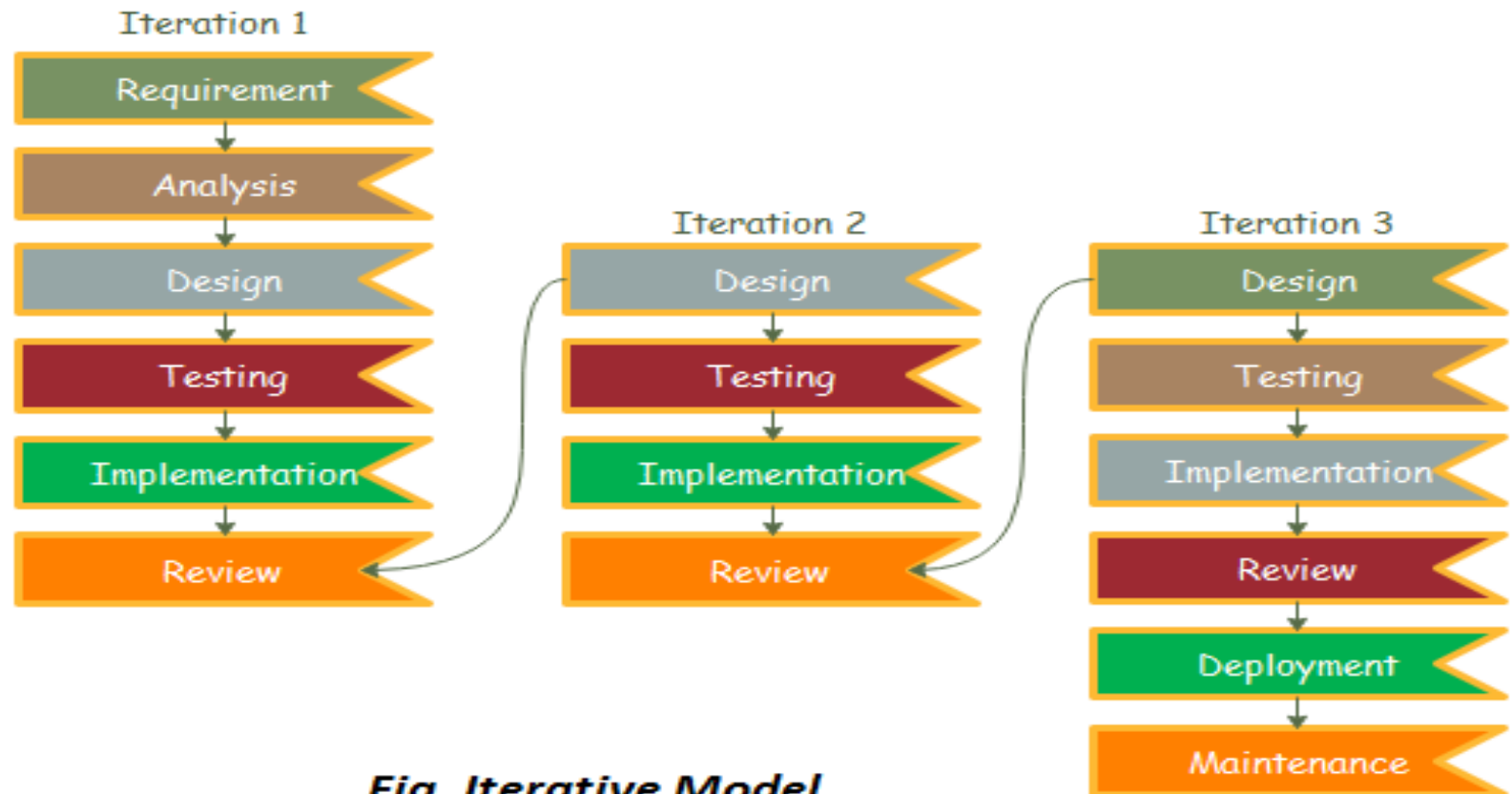


Fig. Iterative Model

Cont'd...

The various phases of Iterative model are as follows:

1. Requirement gathering & analysis:

- In this phase, requirements are gathered from customers and check by an analyst whether requirements will fulfil or not.
- Analyst checks that need will achieve within budget or not.
- After all of this, the software team skips to the next phase.

2. Design:

- In the design phase, team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.

3. Implementation:

- In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.

Cont'd...

4. **Testing:** After completing the coding phase, software testing starts using different test methods.
5. **Deployment:** After completing all the phases, software is deployed to its work environment.
6. **Review:** In this phase, after the product deployment, review phase is performed **to check the behaviour and validity of the developed product.**
 - And if there are any error found then the process starts again from the requirement gathering.
7. **Maintenance:** after deployment of the software in the working environment, there may be some bugs, some errors or new updates are required.
 - Maintenance involves **debugging and new addition options.**

Cont'd...

When to use the Iterative Model?

- When requirements are defined clearly and easy to understand.
- When the software application is large.
- When there is **a requirement of changes in future.**

Advantage of Iterative Model:

- Testing and debugging during smaller iteration is easy.
- A Parallel development.
- It is easily acceptable to ever-changing needs of the project.
- Risks are identified and resolved during iteration.
- Limited time spent on documentation and extra time on designing.

Cont'd...

Disadvantage of Iterative Model:

- It is not suitable for smaller projects.
- More Resources may be required.
- Design can be changed again and again because of imperfect requirements.
- Requirement changes can cause over budget.
- Project completion date not confirmed because of changing requirements.

Reading Assignment

- ✓ **RAD Model**
- ✓ **Big Bang Model**
- ✓ **Prototype Model**
- ✓ **Evolutionary Process Model**
- **Agile Testing Methods**

Thank You!

?

Chapter Four

Requirements Specification

Introduction

- A software requirements specification (SRS) is a **description of a software system to be developed**.
 - its defined after business requirements specification (CONOPS):
 - a document describing the characteristics of a proposed system from the viewpoint of an individual who will use that system.
 - also called **stakeholder requirements specification** (StRS).
 - other document related is the **system requirements specification** (SyRS).
- The SRS lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

Cont'd...

❑ Main aim of requirements specification:

- systematically organize the requirements arrived during requirements analysis.
- document requirements properly.

❑ The SRS document is useful in various contexts:

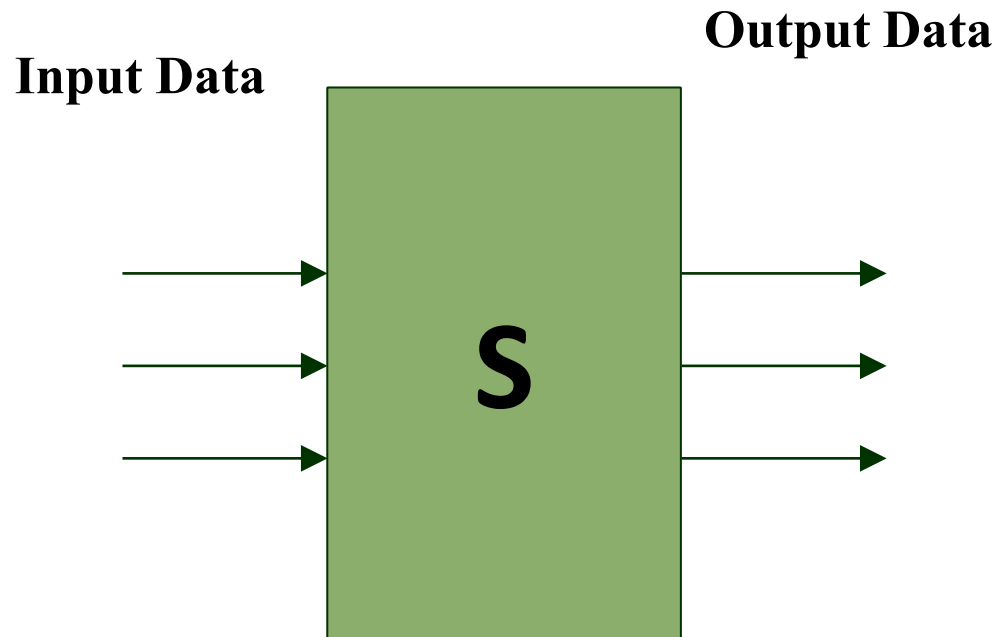
- statement of user needs
- contract document
- reference document
- definition for implementation

Software Requirements Specification : **A Contract Document**

- Requirements document is a reference document.
 - SRS document is a contract between the development team and the customer.
 - Once the SRS document is approved by the customer,
 - any subsequent controversies are settled by referring the SRS document.
 - Once customer agrees to the SRS document:
 - development team starts to develop the product according to the requirements recorded in the SRS document.
- The final product will be acceptable to the customer:
- as long as it satisfies all the requirements recorded in the SRS document.

Cont'd...

- The SRS document is known as black-box specification:
 - the system is considered as a black box whose internal details are not known.
 - only its visible external (i.e. input/output) behaviour is documented.



Cont'd...

- SRS document concentrates on:
 - what needs to be done.
 - carefully avoids the solution (“how to do”) aspects.
- The SRS document serves as a contract
 - between development team and the customer.
 - **Should be carefully written.**
- The requirements at this stage:
 - written using end-user terminology.
- If necessary:
 - later a formal requirement specification may be developed from it.

Cont'd...

- SRS document, normally contains three important parts:
 - Functional requirements,
 - Non-functional requirements,
 - Constraints on the system.

Organization of SRS Document

- Introduction
- Functional Requirements
- Non-functional Requirements
 - External interface requirements
 - Performance requirements
- Constraints

Requirements smells/Code Smells

- **Requirements smells** has been proposed to describe issues in requirements specification where **the requirement is not necessarily wrong but could be problematic.**
- ❖ In particular, the requirements smell:
 - is an indicator for a quality problem of a requirements artifact.
 - does not necessarily lead to a defect and, thus, has to be judged by the context.
 - has a concrete location in the requirements artifact itself, e.g. a word or a sequence.
 - has a concrete detection mechanism (which can be automatic or manual and more or less accurate).

Cont'd...

- Examples of requirements smells are :
 - Subjective Language (feel, believe, think) , Ambiguous Adverbs (Quickly, gently) and Adjectives (Small, sharp) , Superlatives and Negative Statements.
- ❖ In computer programming, **code smell**, (software smell or bad smell) is **any symptom in the source code of a program that possibly indicates a deeper problem.**
- One way to look at smells is with respect to principles and quality:
 - "smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality".

Cont'd...

- **Code smells** are usually not bugs—they are not technically incorrect and do not currently prevent the program from functioning.
 - Instead, they indicate **weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future.**

Techniques to write high-quality requirements

- Many software requirements specifications (SRS) are filled with badly written requirements.
 - Because **the quality of any product depends on the quality of the raw materials fed into it,**
 - poor requirements cannot lead to excellent software.
 - Sadly, few software developers have been educated about how to elicit, analyze, document, and verify the quality of requirements.
- ❑ No matter how much you scrub, analyze, review, and refine the requirements, **they will never be perfect.**
- However, if you keep the following characteristics in mind, you will produce better requirements documents and you will build better products

Characteristics of Quality Requirement Statements

- How can we distinguish good software requirements from those that have problems?

❖ Six characteristics individual requirement statements should exhibit:

- 1) Correct
- 2) Feasible
- 3) Necessary
- 4) Prioritized
- 5) Unambiguous
- 6) verifiable

Cont'd...

I. Correct

- Each requirement must accurately describe the functionality to be delivered.
- **The reference for correctness** is the source of the requirement, such as an actual customer or a higher-level system requirements specification.
- Only user representatives can determine the correctness of user requirements, which is why it is essential to include them, or their close surrogates, in inspections of the requirements.
- Requirements inspections that do not involve users can lead to developers saying, **“That doesn’t make sense.”**
 - This is probably what they meant.” This is also known as “guessing.”

Cont'd...

II. Feasible

- It must be possible to implement each requirement within the known capabilities and limitations of the system and its environment.
 - To avoid infeasible requirements, have a developer work with the requirements analysts or marketing personnel throughout the elicitation process.

III. Necessary

- Each requirement should document something the customers really need.

Cont'd...

IV. Prioritized

- Assign an implementation priority to each requirement, feature, or use case to indicate **how essential it is to include it in a particular product release.**
- **Three levels of priority:**
 - **High priority:-** means the requirement **must be incorporated in the next product release.**
 - **Medium priority:-** means the requirement is necessary but it can be deferred to a **later release if necessary.**
 - **Low priority:-** means it would be nice to have, but we realize it might have to be dropped **if we have insufficient time or resources.**

Cont'd...

V. Unambiguous

- The reader of a requirement statement should be able to draw **only one interpretation** of it.
 - Also, **multiple readers** of a requirement should arrive at the **same interpretation**.
 - Natural language is highly prone to ambiguity.
 - Avoid subjective words like user friendly, easy, simple, rapid, efficient, several, state-of-the-art, improved, maximize, and minimize.
- Words that are clear to the SRS author may not be clear to readers.

Cont'd...

- Write each requirement in concise, simple, straightforward language of the user domain, not in computers.
- **Effective ways to reveal ambiguity include:**
 - **Formal inspections** of the requirements specifications,
 - **Writing test cases** from requirements and
 - **Creating user scenarios** that illustrate the expected behavior of a specific portion of the product.

Cont'd...

VI. Verifiable

- See whether you can devise **tests** or use other verification approaches, such as inspection or demonstration, to **determine whether each requirement is properly implemented in the product.**
- If a requirement is not verifiable, determining whether it was correctly implemented is a matter of opinion.
- ❖ Requirements that are not consistent, feasible, or unambiguous also are not verifiable.
- Any requirement that says the product shall “**support**” something is not verifiable.

Characteristics of Quality Requirements Specifications

Characteristics of a high quality SRS:

A. Complete

- No requirements or necessary information should be missing.
- Completeness is also a desired characteristic of an individual requirement.

B. Consistent

- Consistent requirements do not conflict with other software requirements.
- Disagreements among requirements must be resolved before development can proceed.
- ❑ Be careful when modifying the requirements, as inconsistencies can slip in undetected if you review only the specific change and not any related requirements.

Cont'd...

C. Modifiable

- You must be able to revise the SRS when necessary and maintain a history of changes made to each requirement.
- You can make an SRS more modifiable by organizing it so that related requirements are grouped together, and by creating a table of contents, index, and cross-reference listing.

D. Traceable

- You should be able to link each software requirement to its source, which could be a higher-level system requirement, a use case, or a voice-of-the-customer statement.

Modelling

- **System modeling** is the process of developing abstract models of a system,
 - with each model presenting a different view or perspective of that system.
- It is about representing a system using some kind of graphical notation, which is now almost always based on notations in the **Unified Modeling Language (UML)**.
- Models help the analyst to understand the functionality of the system;
 - they are used to communicate with customers.

Cont'd...

□ Models can explain the system from **different perspectives**:

- An **external** perspective, where you model the context or environment of the system.
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural** perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events.

Cont'd...

- System models describe a particular aspect of a system, such as:
 - the way the system is decomposed into subsystems,
 - the way that data is processed, the structure of the data, etc.
- They are used in a requirements document:
 - ✓ to add information to natural language descriptions of the system requirements.
- They may be developed during the requirements elicitation and analysis to help understand the requirements.
- Sometimes, they can be considered as a detailed system specification of what is required.

Cont'd...

- System models supplement natural language descriptions of requirements.
- ❑ The requirements description and the associated system models are usually developed at the same time.
 - The development of one helps with the development of the other as exploring aspects of the requirements through **modelling reveals inconsistencies and incompleteness.**
- ❑ In some organizations, it is normal practice to develop very detailed and complete system models as part of the system specification.

Cont'd...

- The models which you need for your system depend on the **type of system** and the **people** who will use the models.
- However, it is recommend that two high-level models should always be included in a requirements specification.
- These are as follows.
1. A model which shows the **environment** in which the system being specified will operate.
 2. An **architectural model** which shows how the system is decomposed into sub-systems.

Assignment-#1 (10%)

- Read about the following UML Diagrams that are the most useful for system modeling and prepare a short note on these models with examples.

1. Activity diagram
2. Use case diagram
3. Sequence diagram
4. Class diagram
5. State diagram

Writing requirement documents

- **General requirements of all software documentation:**
 - ✓ Should provide for communication among team members
 - ✓ Should act as an information repository to be used by maintenance engineers
 - ✓ Should provide enough information to management to allow them to perform all program management related activities
 - ✓ Should describe to users how to operate and administer the system
 - ✓ In all software projects, some amount of documentation should be created prior to any code being written.
 - Design docs, etc.
 - ✓ Documentation should continue after the code has been completed
 - User's manuals, etc.

Cont'd...

- The two main types of documentation created are *Process* and *Product* documents

❖ Process Documentation

- Used to record and track the development process.
 - Planning documentation, Cost, Schedule, Fund tracking, Standards etc.
- This documentation is created to allow for successful management of a software product and has a relatively short lifespan.
- Only important to internal development process except in cases where the customer requires a view into this data.

❑ Some items, such as papers that describe design decisions should be extracted and moved into the *product documentation* category when they become implemented

Cont'd...

Software Documentation Standards

- Documentation standards in a software project are important
 - because documents are the only tangible way of representing the software and the software process.
- Standardized documents have a consistent appearance, structure and quality,
 - and should therefore be easier to read and understand.

Cont'd...

2. Product Documentation

- Describes the **delivered** product
- Must evolve with the development of the software product
- **Two main categories:**
 - System Documentation
 - User Documentation

Cont'd...

❖ System Documentation

- Describes how the system works, but not how to operate it.

❑ Examples:

- Requirements Spec
- Architectural Design: framework for sub-system control and communication.
- Detailed Design :implementation details, modules and their implementation.
- Commented Source Code: including output such as JavaDoc
- Test Plan: including test cases
- V&V plan and results
- List of Known Bugs

Cont'd...

- **User Documentation:** has two main types
 1. End User
 2. System Administrator
- There are five important areas that should be documented for a **formal release of a software application**
 - a) Functional Description of the Software
 - b) Installation Instructions
 - c) Introductory Manual
 - d) Reference Manual
 - e) System Administrator's Guide

Cont'd...

❖ Document Structure

- All documents for a given product should have a similar structure
 - A good reason for *product standards*
- The IEEE Standard for **User Documentation** lists such a structure
 - The authors “best practices” are:
 1. Put a cover page on all documents
 2. Divide documents into chapters with sections and subsections
 3. Add an index if there is lots of reference information
 4. Add a glossary to define ambiguous terms

Cont'd...

❖ Standards

- Standards play an important role in the development, maintenance and usefulness of documentation.
- Standards **can act as a basis for quality documentation** but are not good enough on their own.
- Usually define high level content and organization

1. Process standards

- **Define the approach that is to be used when creating the documentation**
- Don't actually define any of the content of the documents

Cont'd...

2. Product standards

- Goal is to have all documents created for a specific product attain a **consistent structure and appearance**
 - Can be based on organizational or contractually required standards
- **Four main types:**
 1. Documentation Identification Standards
 2. Document Structure Standards
 3. Document Presentation Standards
 4. Document Update Standards

Cont'd...

3. Interchange standards

- Deals with the creation of documents in a format that allows others to effectively use.
- PDF may be good for end users who don't need to edit
- Word may be good for text editing
- Specialized CASE tools need to be considered.


Cont'd...

❖ Other standards

➤ IEEE

- Has a published standard for user documentation
- Provides a structure and superset of content areas
- Many organizations probably won't create documents that completely match the standard.

➤ Writing Style

- Ten “best practices” when writing are provided
- Author proposes that 
 - group edits of important documents should occur in a similar fashion to software walkthroughs.



Ten best practices

10 Best Practices for Writing and Editing Technical Documents

1. Know Your Audience and Write Exclusively for Them and to Them.
2. Organize and Outline Your Technical Writing Before You Write.
3. Consider Each Document's Layout Before You Write.
4. Always Insert Images, Videos, Data Visualizations, and Other Visual Aids.
5. Be Concise and Use Plain Language.
6. Remain Consistent.
7. Remember That You're a Human Writing for Other Humans
8. Always Have Others Edit Your Work and Provide Feedback
9. Stay Focused on the Purpose of Each Technical Document
10. Test the Practical Usefulness of Each Technical Document

Reviewing Requirements for Quality

- What do good requirements really look like?
 - Following are several requirements adapted from actual projects.
- ❑ **Example #1:** *“The product shall provide status messages at regular intervals **not less than every** 60 seconds.”*
- This requirement is incomplete:
 - what are the status messages and how are they supposed to be displayed to the user?
 - The requirement contains several ambiguities.
 - What part of “the product” are we talking about?
 - Is the interval between status messages really supposed to be at least 60 seconds, so showing a new message every 10 years is okay? Perhaps the intent is to have no more than 60 seconds elapse between messages; would 1 millisecond be too short? The word “**every**” just confuses the issue.
- As a result of these problems, the requirement is not verifiable.

Cont'd...

- Here is one way we could rewrite the requirement to address those shortcomings:
- **“1. Status Messages.**
 - 1.1. The Background Task Manager shall display status messages in a designated area of the user interface at intervals of 60 plus or minus 10 seconds.
 - 1.2. If background task processing is progressing normally, the percentage of the background task processing that has been completed shall be displayed.
 - 1.3. A message shall be displayed when the background task is completed.
 - 1.4. An error message shall be displayed if the background task has halted.”

Guidelines for Writing Quality Requirements

- **There is no formulaic way to write excellent requirements.**
- It is largely a matter of experience and learning from the requirements problems you have encountered in the past.
- ❖ **Here are a few guidelines to keep in mind as you document software requirements.**
 - Keep sentences and paragraphs short.
 - Use the active voice.
 - Use proper grammar, spelling, and punctuation.
 - Use terms consistently and define them in a glossary or data dictionary.
 - Avoid long narrative paragraphs that contain multiple requirements.
 - Never use “**and/or**” in a requirement statement.
 - Write requirements at a consistent level of detail throughout the document.
 - Avoid stating requirements redundantly in the SRS.

Thank You!

?

Chapter Five

Requirements Validation

Introduction

- Requirements validation is concerned with checking the requirements document for consistency, completeness and accuracy.
- It is the final stage of requirements engineering.
- **Aim:** to “validate” the requirements,
 - i.e., check the requirements to certify that they **represent an acceptable description of the system** which is to be implemented.
- The process involves:
 - ✓ system stakeholders,
 - ✓ requirements engineers and
 - ✓ system designers who analyze the requirements for problems, omissions and ambiguities.

Requirements Analysis Vs Requirements Validation

- These activities have much in common.
 - They involve:
 - ✓ Analyzing the requirements,
 - ✓ Judging if they are an appropriate description of stakeholder needs and
 - ✓ Checking for requirements problems.
- However, there are important differences between these activities so that it makes sense to consider them separately.

Requirements Analysis

- ✓ Is **concerned with 'raw' requirements** as elicited from system stakeholders.
- ✓ The requirements are usually incomplete and are expressed in an informal and unstructured way.
- ✓ **Focus** on making sure that the requirements meet stakeholder needs rather than the details of the requirements description.
- ✓ It should mostly be **concerned with** answering the question 'have we got the right requirements?'

Requirements Validation

- ✓ Is **concerned with checking a final draft of a requirements document**, which includes all system requirements and where known incompleteness and inconsistency has been removed.
- ✓ The document and the requirements should follow defined quality standards.
- ✓ The validation process should be **more concerned with the way in which the requirements are described**.
- ✓ It should mostly (but not exclusively) be concerned with answering the question 'have we got the requirements right?'

Cont'd...

➤ Some stakeholder-related problems are inevitably discovered during requirements validation and these must be corrected.

❑ **Examples of requirements problems discovered during validation might be:**

- lack of conformance to quality standards.
- poorly worded requirements which are ambiguous.
- errors in models of the system or the problem to be solved.
- requirements conflicts which were not detected during the analysis process.

➤ These problems should be solved before the requirements document is approved and used as a basis for system development.

What kind of problems are these and how to fix them?

On the next slide 

Cont'd...

- ✓ In some cases, the problems will simply be **documentation problems** and
 - can be **fixed by improving the requirements description**.
- ✓ In other cases, however, the problems result from **flaws in the requirements elicitation, analysis and modelling**.
 - You may have to **re-enter some of the earlier requirements engineering process** activities.
- The main- problem of requirements validation is that **there is no existing document** which can be a basis for the validation.
- A design or a program may be validated against the specification.
 - However, there is no way to demonstrate that a requirements specification is 'correct' with respect to some other system representation.

Cont'd...

❑ **Specification validation**, therefore, really means **ensuring that the requirements document represents a clear description of the system** for design and implementation and is a final check that the requirements meet stakeholder needs.

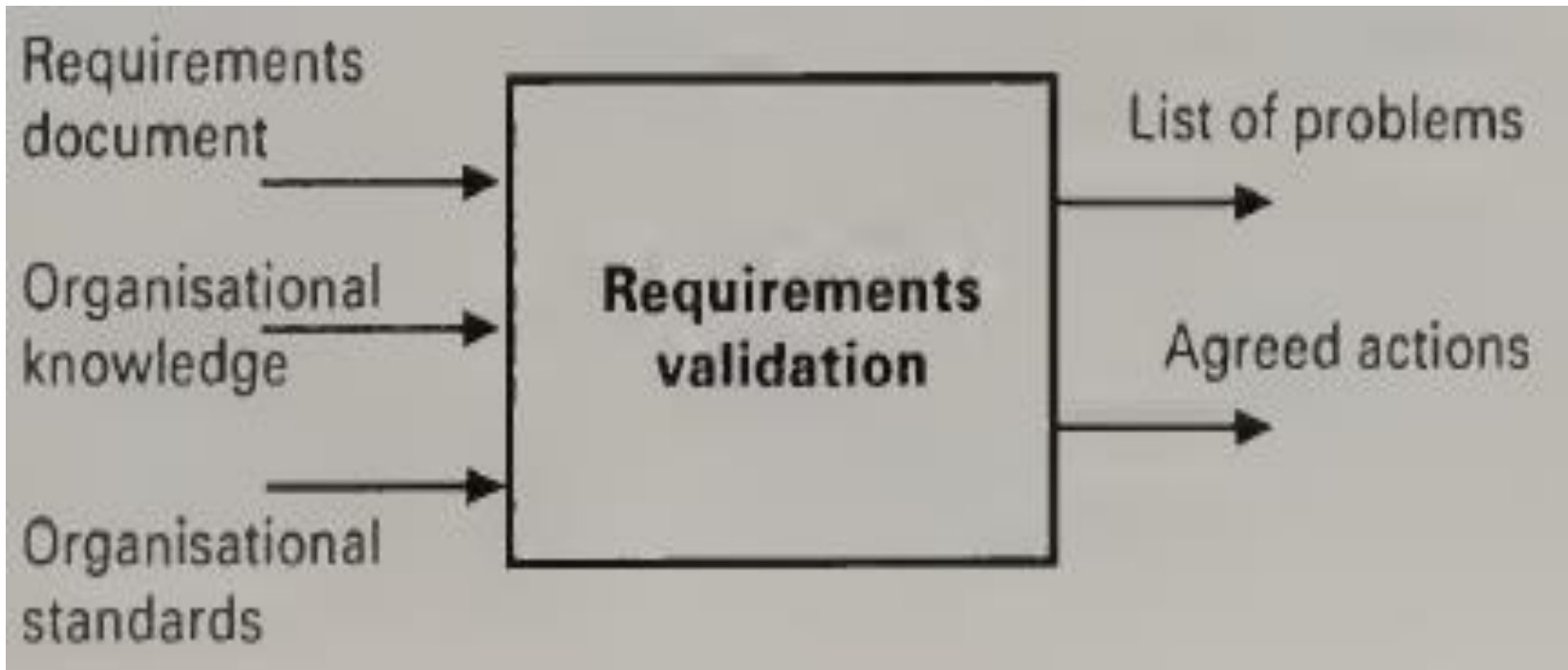


Figure 5.1 Requirements validation - inputs and outputs.

Cont'd...

- The inputs to the requirements validation process are as follows.

1. The requirements document

- The complete version of the requirement document which is formatted and organized according to organizational standards.

2. Organizational standards

- The requirements validation process should check conformance with company standards.
- standards are relevant for the requirements document should be an input to the validation process.

3. Organizational knowledge

- The people involved in requirements validation may know the organization, its particular terminology and its practices and the skills of the people involved in developing and using the system.
- This implicit knowledge is very important as the same requirements may be closely linked to organizational structure, standards and culture.

Cont'd...

- The process outputs are as follows.

1. A problem list

- This is a list of reported problems with the requirements document.
 - e.g., ambiguity, incompleteness, etc.
- In practice, however, it is often difficult to classify problems in this way.

2. Agreed actions

- This is a list of actions in response to requirements problems which have been agreed by those involved in the validation process.
- There is not necessarily a 1:1 correspondence between problems and actions.
 - ✓ Some problems may result in several corrective actions; others may simply be noted but with no associated corrective action.

Cont'd...

- ❑ Requirements validation is a **prolonged process** as it involves people reading and thinking about a lengthy document.
- Meetings may have to be arranged and experiments carried out with prototype systems.
- It can take several weeks or sometimes months to validate the requirements for a complex system.
- This is particularly likely when stakeholders from different organizations are involved.

Techniques for Requirements Validation

- ✓ Requirements reviews
 - Pre-review checking
 - Requirements checklist
- ✓ Prototyping
- ✓ User manual development
 - Rewriting the requirement

Requirements Reviews

- Requirements reviews are the **most widely used technique of requirements validation**.
- They involve a group of people who read and analyze the requirements, look for problems, meet to discuss the problems and agree on a set of actions to address the identified problems.

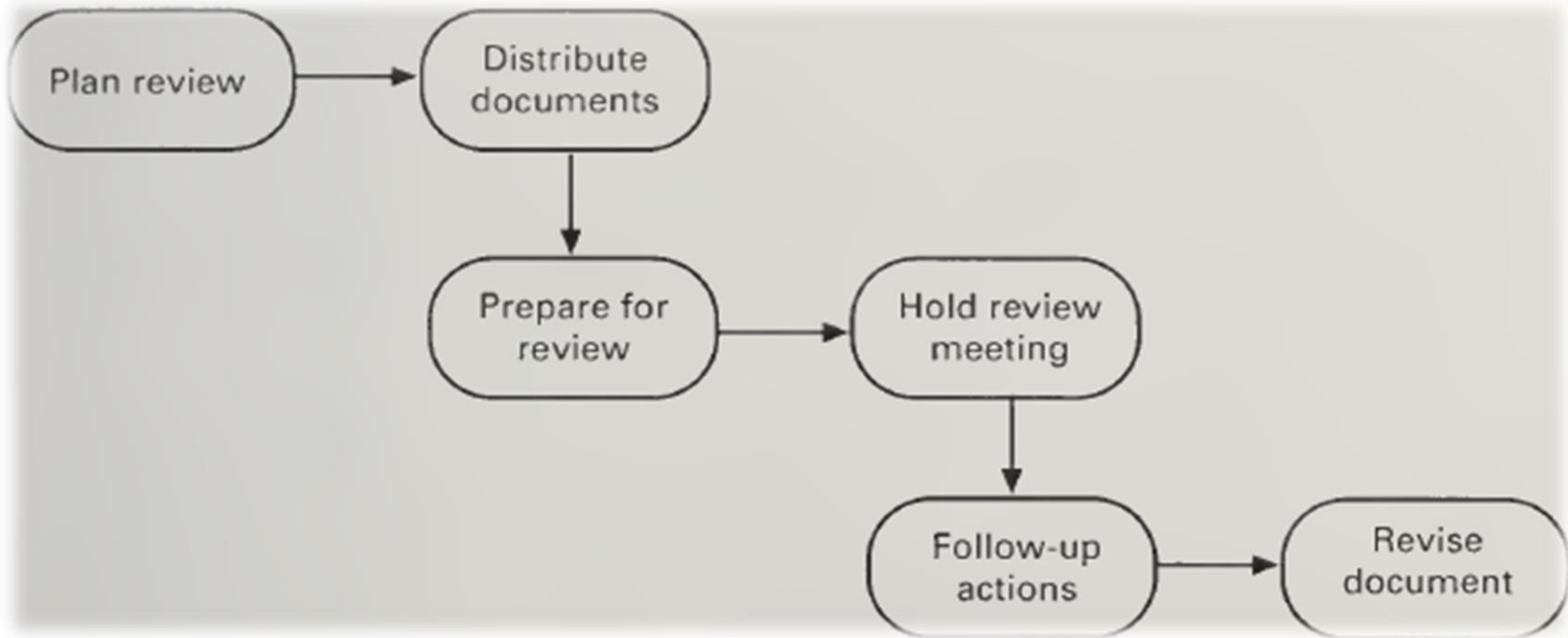


Figure 5.2 process model for the requirements review process

Cont'd...

❑ The principal stages in the review process are as follows.

1. Plan review

- The review team is selected and a time and place for the review meeting is chosen.

2. Distribute documents

- The requirements document and any other relevant documents are distributed to the review team members.

3. Prepare for review

- The individual reviewers read the requirements document to identify conflicts, omissions, inconsistencies, deviations from standards and any other problems.

Cont'd...

4. Hold review meeting

- The individual comments and problems are discussed and a set of actions to address the problems is agreed.

5. Follow-up actions

- The chair of the review checks that the agreed actions have been carried out.

6. Revise document

- The requirements document is revised to reflect the agreed actions.
- At this stage, it may be accepted or it may be re-reviewed.

Cont'd...

- ❑ The requirements review is a formal meeting.
- It should be **chaired by someone** who has not been involved in producing the requirements which are being validated.
- During the meeting, a **requirements engineer** presents each requirement in turn for comment by the group and identified problems are recorded for later discussion.
- One member of the group should be assigned the role of **scribe** to note the identified requirements problems.

Cont'd...

- In program inspections, it is normal practice to report errors to the program author for correction.
- ❑ However, in requirements reviews, the review group make decisions on actions to be taken to correct the identified problems.
- Unlike programming errors, the problems usually require discussion and negotiation to agree on a possible solution.

Actions which might be decided for each problem are as follows.

1. Requirements clarification

- The requirement may be badly expressed or may have accidentally omitted information which has been collected during requirements elicitation.
- The author should improve the requirement by rewriting it.

Cont'd...

2. Missing information

- Some information is missing from the requirements document.
- It is the responsibility of the requirements engineers who are **revising the document** to discover this information from system stakeholders or other requirements sources.

3. Requirements conflict

- There is a significant conflict between requirements.
- The stakeholders involved must **negotiate** to resolve the conflict.

4. Unrealistic requirement

- The requirement does not appear to be implementable with the technology available or given other constraints on the system.
- Stakeholders must be consulted to decide whether **the requirement should be deleted or modified** to make it more realistic.

Pre-review checking

- Reviews involve a lot of time and expense so it makes sense to minimize the work of reviewers.
 - Before distributing the requirements document for general review, one person should **carry out quick standards check** to ensure that the document structure and the defined requirements are consistent with whatever standards have been defined.
 - They should also process the document with whatever automatic checkers are available to remove spelling mistakes, cross-reference errors, etc.
 - This is a quick and cheap way to check standards conformance as only one person is needed to carry out this type of check.
- Furthermore, checking against a standard can quickly reveal problems with the requirements.

Cont'd...

- An analyst or an engineer who is familiar with the requirements standards but who has not been involved in the system requirements specification should be responsible for this initial document check.
- It is not necessary for the document checker to understand the requirements in detail.
 - ✓ The checker should compare the structure of the requirement document to the defined standard and should highlight missing or incomplete sections.
- Individual requirements should also be checked for standards conformance.

Cont'd...

- This initial check should also check that:
 - all pages in the document are numbered,
 - all diagrams and figures are labelled,
 - there are no requirements which are unfinished or labelled 'To be completed' and that all required appendices in the document have been included.

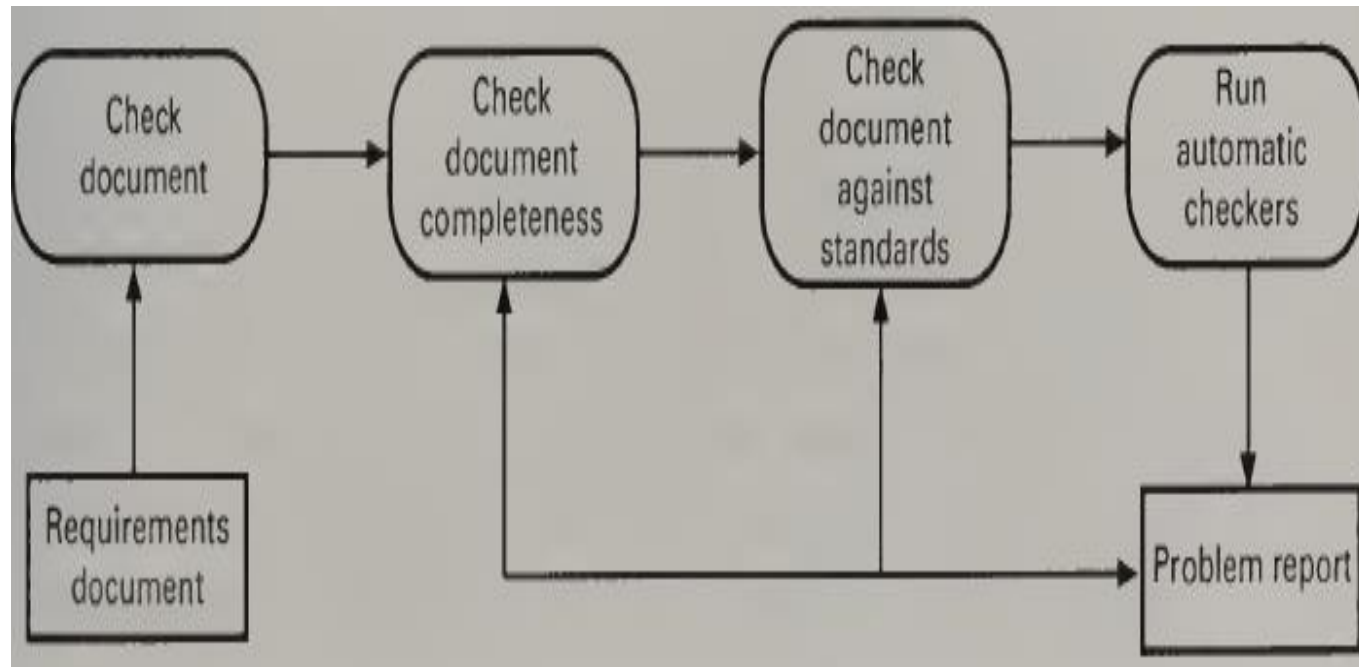


Figure 5.3 Pre-review checking

Cont'd...

❑ After the initial checking process is complete, there are two possible options if deviations from the standard are found.

I. Return the document to the requirements engineering team to correct deviations from the standards.

- This option should be chosen if there is enough time to allow for a re-issue of the document.

II. Note the deviations from the standards and distribute this to document reviewers.

- This saves the time and cost of creating a new version of the requirements document.
- However, the deviations from standards may make it harder for reviewers to understand the requirements.

Review team membership

- Selecting the right membership for the requirements review team is important.
- Ideally, the requirements document should be reviewed by a **multi-disciplinary team** drawn from people with different backgrounds.
- If possible, the team should include:
 - a system end-user or end-user representative,
 - a customer representative,
 - one or more domain experts,
 - engineers who will be responsible for system design and implementation and
 - requirements engineers.

Cont'd...

❑ The **advantages** of involving stakeholders from different disciplines are as follows.

- I. People from different backgrounds bring different skills, domain knowledge and experience to the review.
 - It is therefore more probable that requirements problems will be discovered.
- II. If system stakeholders from different backgrounds are involved in the review process,
 - they feel involved in the requirements engineering process and develop an understanding of the needs of other stakeholders.
 - They are therefore more likely to understand why changes to requirements which they have proposed are necessary.

Review checklists

- Requirements review checklists should be more general.
 - They should not be concerned with individual requirements but with the quality properties of the requirements document as a whole and with the relationships between requirements.
- The following table 5.1 lists some **general quality properties of requirements and requirements documents** which may be used to derive requirements checklists.

Cont'd...

Table 5.1 Requirements quality attributes

Review check	Description
Understandability	<ul style="list-style-type: none">• Can readers of the document understand what the requirements mean?• This is probably the most important attribute of a requirements document - if it can't be understood, the requirements can't be validated.
Redundancy	<ul style="list-style-type: none">• Is information unnecessarily repeated in the requirements document?• Sometimes, of course, repeating information adds to understandability.• There must be a balance struck between removing all redundancy and making the document harder to understand.

Cont'd...

Review check	Description
Completeness	Does the checker know of any missing requirements or is there any information missing from individual requirement descriptions?
Ambiguity	<ul style="list-style-type: none">• Are the requirements expressed using terms which are clearly defined?• Could readers from different backgrounds make different interpretations of the requirements?
Consistency	<p>Do the descriptions of different requirements include contradictions?</p> <p>Are there contradictions between individual requirements and overall system requirements?</p>

Cont'd...

Review check	Description
Organization	<ul style="list-style-type: none">• Is the document structured in a sensible way?• Are the descriptions of requirements organized so that related requirements are grouped?• Would an alternative structure be easier to understand?
Conformance to standards	<ul style="list-style-type: none">• Does the requirements document and individual requirements conform to defined standards?• If there is a departure from the standards, is it justified?
Traceability	<ul style="list-style-type: none">• Are requirements unambiguously identified, include links to related requirements and to the reasons why these requirements have been included?• Is there a clear link between software requirements and more general systems engineering requirements?

Cont'd...

- Organizations should derive their own set of requirements review questions based on their experience and their local standards.
- These may include more specialized questions which are tailored to the types of system which they develop.
- ❑ The following Table 5.2 gives some examples of possible questions which might be associated with the quality attributes described in Figure 5.1.
- Checklists should be expressed in a fairly general way and should be understandable by people such as end-users who are not system experts.
- ❑ As a general rule, checklists should not be too long.
 - If checklists have more than 10 items, checkers cannot remember all items.

Cont'd...

- Checklists can be distributed and used to remind people what to look for when reading the requirements document.
- Alternatively, they can be used more systematically where, for each requirement, an indication is given that the checklist item has been considered.
- This may be done on paper or you can manage this type of checklist completion by using a simple database or a spreadsheet.

Prototyping

- People find it very difficult to visualize how a written statement of requirements will translate into an executable software system.
- ❑ If you develop a prototype system to demonstrate requirements,
 - stakeholders and other end-users find it easier to discover problems and suggest how the requirements may be improved.
- If a prototype has been developed for requirements elicitation,
 - ✓ it makes sense to use this later in the requirements engineering process for validation.
- However, **if a prototype system is not already available,**
 - ✓ **it is not likely to be cost-effective** to develop a prototype system only for requirements validation.

Cont'd...

Elicitation Prototypes Vs Validation Prototypes

- Prototypes for validation must be more complete than elicitation prototypes.
- **Elicitation prototypes:**
 - Can simply include those requirements which are particularly difficult to describe or understand.
 - They may leave out well-understood requirements.
 - Usually have missing functionality and may not include changes agreed during the requirements analysis process.

Cont'd....

- It is therefore usually necessary to continue development of the prototype during the requirements validation process as shown in Figure 5.3.
- This figure also shows the other process activities which should go on in parallel with prototype development.

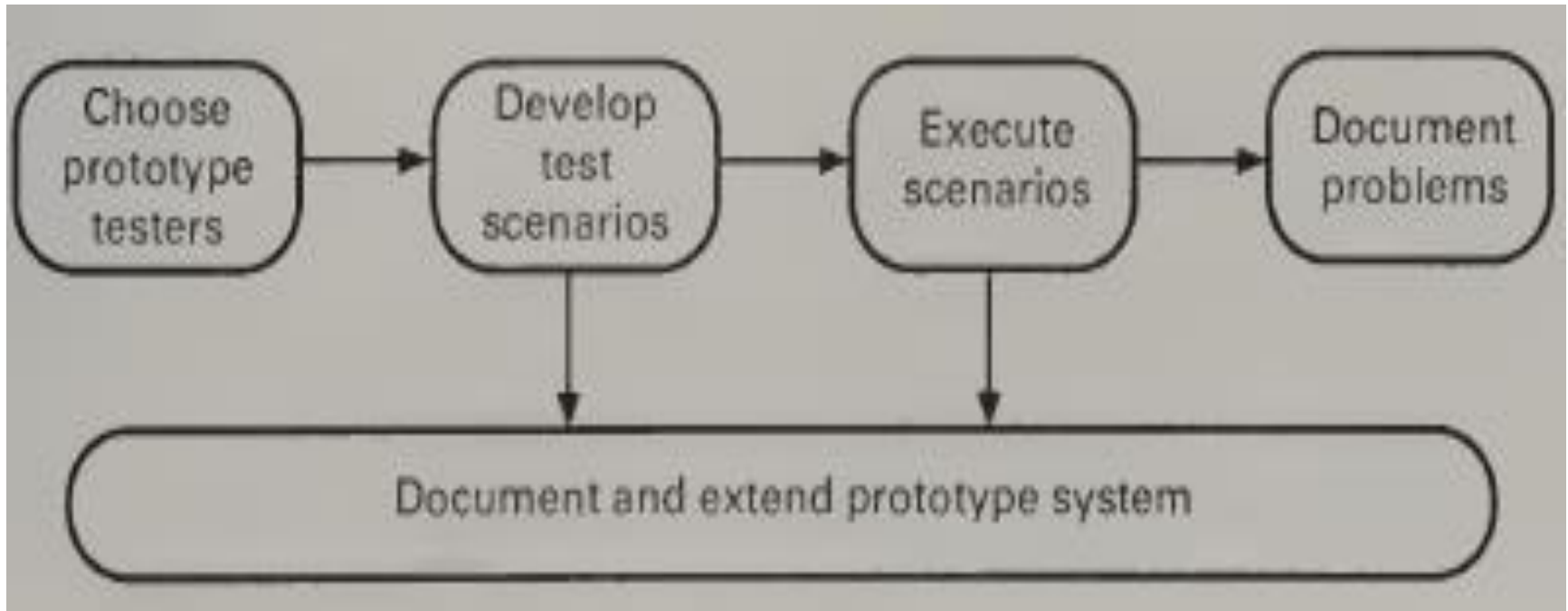


Figure 5.3 Prototyping for requirements validation.

Cont'd...

1. Choose prototype testers

- Choosing the right people to act as prototype testers is very important.
- The best testers are users who are fairly experienced and who are openminded about the use of new systems.
- If possible, end-users who do different jobs should be involved so that different areas of system functionality will be covered.

2. Develop test scenarios

- This means that careful planning is required to draw up a set of test scenarios which provide broad coverage of the requirements.

Cont'd...

3. Execute scenarios

- Requirements engineers should spend some time observing how end-users make use of the system.
- This can reveal particular problem areas and the coping strategies which users develop for dealing with system features which they find useful but awkward to use.

4. Document problems

- To be effective, problems which users encounter must be carefully documented.
- It's usually best to **define some kind of electronic or paper problem report form** which users fill in when they encounter a problem or want to suggest some change to the system.

User manual development

- Rewriting the requirements in a different way is a very effective validation technique.
- To rewrite the requirements you must understand the requirements and the relationships between them.
- Developing this understanding reveals conflicts, omissions and inconsistencies.
- One possible alternative form of the requirements which may be produced using the validation process is a draft of the end-user documentation for the system.

Cont'd...

❑ **The user manuals should include the following information.**

1. A **description of the functionality which has been implemented** and how to access that functionality through the user interface.
2. It should be made clear **which parts of the system are not implemented**.
 - Manual writers should not leave users to find this out when problems arise.
3. **A description of how to recover from difficulties.**
 - Users are working with an experimental system and it is inevitable that things will go wrong.
 - Information about how they can get back to a known system state and restart use of the system should be included.

Cont'd...

4. If users have to install the prototype themselves, **installation instructions** should be provided.
 - If there is insufficient time or resources to build a prototype system for validation, some of the benefits of this approach can be gained by writing a draft user manual.
 - The manual should be written by systematically translating the functionality described in the requirements into descriptions, written in end-user terms, of how to use them.
 - If it is difficult to explain a function to end-users or to explain how to express system functionality, this suggests that there may be a requirements problem.

Cont'd...

- In some cases, the prototype user manual can be a basis for the final user documentation.
- However, this may not always be possible.
- The system development may be cancelled or the system may change to such an extent that a completely new user manual must to be written.
- The cost of writing the draft manual should be included in the requirements validation costs.

Model Validation

- Part of the requirements specification for a system may consist of one or more system models.

➤ The validation of these models has three **objectives**.

1. To demonstrate that each individual model is **self-consistent**.
 - That is, the model should **include all information** which is necessary and there should be **no conflicts between the different parts of the model**.
2. If there are several models of the systems, to demonstrate that these are internally and externally consistent.
 - That is, entities which are referenced in more than one model should be defined to be the same in each model,
 - comparable items should have the same names and the model interfaces should be consistent.

Cont'd...

3. To demonstrate that the models accurately reflect the real requirements of system stakeholders.
 - This is the most difficult model validation task.
 - It involves making convincing arguments that the system defined in the model is the system which stakeholders really need.
- If models are expressed using notations which are supported by CASE tools, some of this checking can be automated.
- CASE tools can check individual models for consistency and can carry out some cross-model checks.
- However, some consistency checks involving several different models cannot be automated but may be considered in model reviews.
 - An example of the type of check which cannot be automated is where an entity defined in one model is referenced in another but the reference is to the wrong entity.

Cont'd...

- Checking that the model reflects the real needs of stakeholders can be difficult.
 - You need to get the stakeholders themselves involved in the model validation process.
 - **Non-technical people** do not intuitively understand data-flow diagrams, event diagrams or object models.
 - They **prefer working with natural language descriptions**.
- ❖ One way to get round this problem is to paraphrase or rewrite the model in natural language.
- Advantage of making this transformation:
 - stakeholders such as end-users, organizational management and regulators can understand and comment on the detailed system specification.
 - It is an effective way to detect errors, inconsistencies and incompleteness in the model.

Cont'd...

❑ For example, in a data-flow diagram, you might use a template with the following fields to describe each transformation:

1. transformation name

2. transformation inputs and input sources

- Gives the name of each input to the transformation and lists where that input comes from.

3. transformation function

- Explain what the transformation is supposed to do to convert inputs to outputs.

4. transformation outputs

- Gives the name of each output and lists where the output goes to.

5. control any exception or control information which is included in the model.

Cont'd...

- ❑ Only a few people in an organization can understand formal models so a **translation** to natural language is essential to validate that they reflect the real requirements of the system.

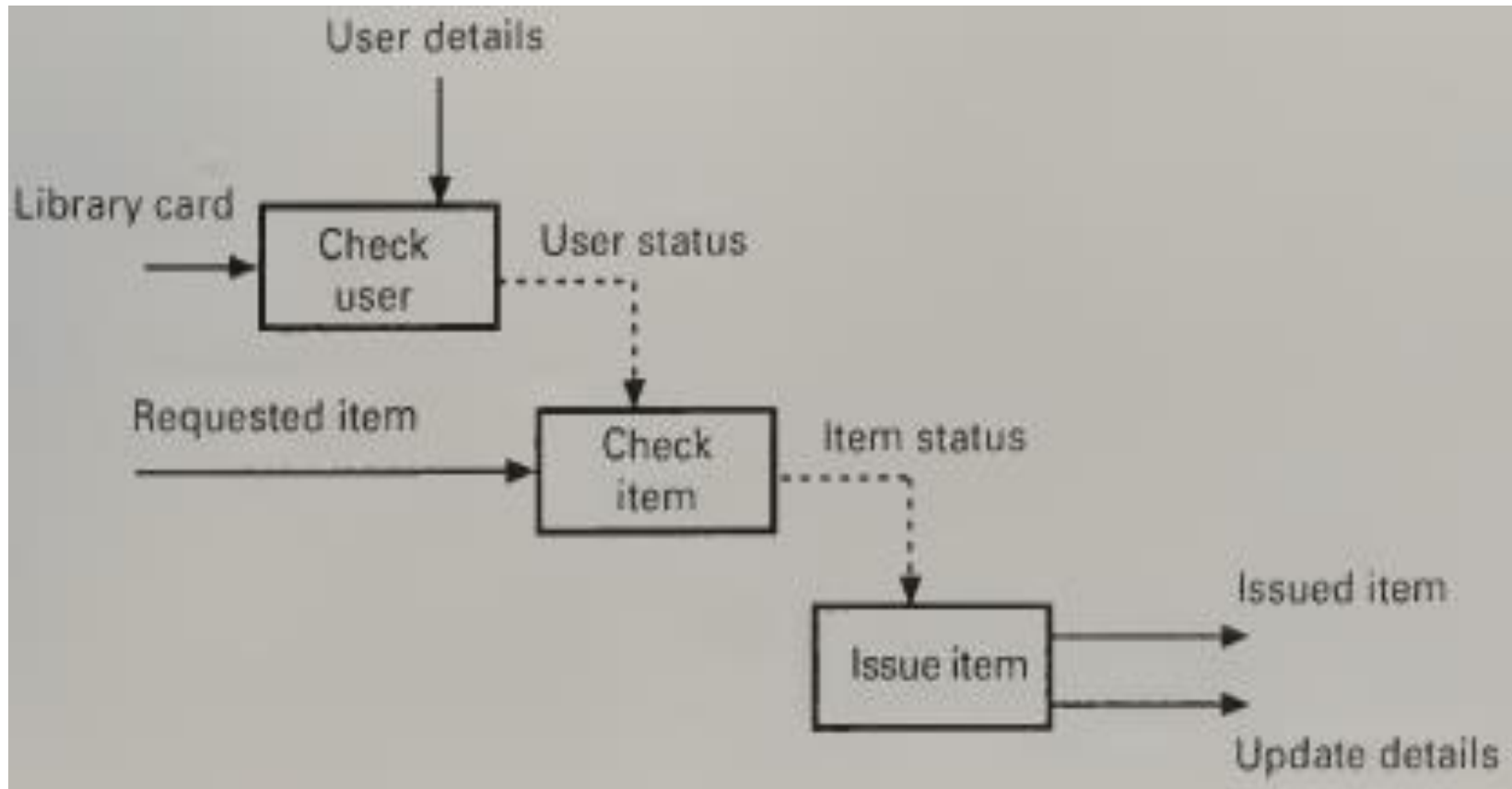


Figure 5.4 Data-flow diagram for the issue function.

Cont'd...

Check user

Inputs and sources	User's library card from end-user
Transformation function	Checks that the user is a valid library user
Transformation outputs	The user's status
Control information	User details from the database

Check item

Inputs and sources	The requested item from the end-user
Transformation function	Checks if an item is available for issue
Transformation outputs	The item's status
Control information	The user's status from Check user

Issue item

Inputs and sources	None
Transformation function	Issues an item to the library user. Items are stamped with a return date.
Transformation outputs	The item issued to the end user Database update details
Control information	Item status – items only issued if available

Figure 5.5 Paraphrased description of 'Issue' data-flow diagram.

Requirements Testing

- A desirable attribute of requirement is that it should be testable.
 - That is, it should be possible to define one or more tests that may be carried out in the finished system which will clearly demonstrate that the requirement has been met.
 - The execution of the implemented requirement may be simulated using these tests.
 - While the actual tests of a system are carried out after implementation, proposing possible tests is an effective way of revealing requirements problems such as incompleteness and ambiguity.
- ❑ If there are **difficulties in deriving test cases for a requirement**, this **implies** that there is some kind of **requirement problem**.
- There may be missing information in the requirement or the requirement description may not make clear exactly what is required.

Cont'd...

- Each functional requirement in the requirements document should be analyzed and a test should be defined which can objectively check if the system satisfies the requirement.
 - The objective of proposing test cases for requirements is to validate the requirement not the system.
- ❑ To define test cases for a requirement, you can ask the following questions about that requirement.
1. What usage scenario might be used to check the requirement?
 - This should define the context in which the test should be applied.

Cont'd...

2. Does the requirement, on its own, include enough information to allow a test to be defined?
 2. If not, what other requirements must be examined to find this information?
 - If you need to look at other requirements, you should record these.
 - There may be dependencies between requirements which are important for traceability.
3. Is it possible to check the requirement using a single test or are multiple test cases required?
 - If you need several tests, it may mean that there is more than one requirement embedded in a single requirement description.
4. Could the requirement be re-stated so that the required test cases are fairly obvious?

Cont'd...

❑ A test record form should be designed and filled in for each requirement which is 'tested'.

- This should include at least the following information.

1. The requirement's identifier

- There should be at least one for each requirement.

2. Related requirements

- These should be referenced as the test may also be relevant to these requirements.

3. Test description

- A brief description of the test which could be applied and why this is an objective requirements test.
- This should include system inputs and the corresponding outputs which are expected.

Cont'd...

4. Requirements problems

- A description of requirements problems which made test definition difficult or impossible.

5. Comments and recommendations

- These are advice on how to solve requirements problems which have been discovered.

Cont'd...

- When designing requirements tests, the test designer need not be concerned with practicalities such as testing costs, avoiding redundant tests, detailed test data definition, etc.
- It isn't necessary to propose real tests which will be applied to the final system.
- The tester can make any assumptions that he wishes about the way in which the system satisfies other requirements and the ways in which the test may actually be carried out.
- However, wherever possible, it makes sense to try and design tests which can be used as system tests.
- These are applied after implementation as part of the system verification and validation process.
- By reusing requirements tests in this way, the overall costs of test planning may be reduced.

Cont'd...

There are problems, however, in designing tests for some types of requirements.

1. System requirements

- These are requirements which apply to the system as a whole.
- In general, these are the most difficult requirements to validate irrespective of the method used as they may be influenced by any of the functional requirements.
- Tests, which are not executed, cannot test for non-functional system-wide characteristics such as usability.

Cont'd...

2. Exclusive requirements

- These are requirements which exclude specific behavior.
- For example, a requirement may state that system failures must never corrupt the system database.
- It is not possible to test such a requirement exhaustively.

3. Some non-functional requirements

- Some non-functional requirements, such as reliability requirements, can only be tested with a large test set.
- Designing this test set does not help with requirements validation.

Thank You !

?

Chapter Six

Requirements Management

Introduction

❑ Requirements management is the **process of managing changes to a system's requirements.**

- New requirements emerge and existing requirements change at all stages of the system development process.
- It is often the case that 50% of a system's requirements will be modified before it is put into service.
 - Clearly, this can cause serious problems for system developers.
 - To minimize difficulties, requirements management is necessary where changes to the requirements are documented and controlled.
- ❑ The impact of proposed changes to requirements must be assessed and, as requirements changes are accepted, system design and implementation must then be modified.
- If changes are not controlled, low priority changes may be implemented before high priority changes and expensive modifications to the system which are not really necessary may be approved.

Cont'd...

- Requirements Management **is carried out in parallel with other requirements engineering processes** and continues after the first version of the requirements document has been delivered.
- The requirements continue to change during system development and these changes must be managed.
- **The principal concerns of requirements management are:**
 1. Managing changes to agreed requirements
 2. Managing the relationships between requirements
 3. Managing the dependencies between the requirements document and other documents produced during software engineering process.

Cont'd...

❑ Changes to system requirements may be:

- due to errors and misunderstandings in the requirements engineering process/ design or implementation problems.
- new requirements may emerge as stakeholders develop a better understanding of the system.
- as a result of changing external circumstances.
- the strategy or priorities of the business buying the system may change as a result of economic changes or new competitors in its market.
- new information about the systems environment may become available.

Cont'd...

❑ Requirements cannot be managed effectively without requirements traceability.

➤ A requirement is traceable if you can discover:

- who suggested the requirement,
- why the requirement exists,
- what requirements are related to it and
- how that requirement relates to other information such as systems designs, implementations and user documentation.

➤ Traceability information is used to find other requirements which might be affected by proposed changes.

Cont'd...

❑ CASE tools for supporting the requirements management process provide facilities such as:

1. a database system for storing requirements
2. document analysis and generation facilities to help construct a requirements database and to help create requirements documents
3. change management facilities which help to ensure that changes are properly assessed and costed
4. traceability facilities which help requirements engineers find dependencies between system requirements.

Stable and Volatile Requirements

- Requirements change is unavoidable and does not imply poor requirements engineering practice.
- Table 6.1 factors leading to requirements change

Change Factor	Description
Requirements errors, conflicts and Inconsistencies	<ul style="list-style-type: none">• As the requirements are analyzed and implemented, errors and inconsistencies emerge and must be corrected.• These problems may be discovered during requirements analysis and validation or later in the development process.

Cont'd...

Change Factor	Description
Evolving customer/end-user knowledge of the system	<ul style="list-style-type: none">• As requirements are developed, customers and end-users develop a better understanding of what they really require from a system.
Technical, schedule or cost problems	<ul style="list-style-type: none">• Problems may be encountered in implementing a requirement.• It may be too expensive or take too long to implement certain requirements.
Changing customer priorities	<ul style="list-style-type: none">• Customer priorities change during system development as a result of a changing business environment, the emergence of new competitors, staff changes, etc.

Cont'd...

Change Factor	Description
Environmental changes	<ul style="list-style-type: none">• The environment in which the system is to be installed may change so that the system requirements have to change to maintain compatibility
Organizational changes	<ul style="list-style-type: none">• The organization which intends to use the system may change its structure and processes resulting in new system requirements

Cont'd...

- ❑ Although change is inevitable, it is usually the case that some requirements are more stable than others.

Stable requirements

- are concerned with the essence of a system and its application domain.
- **They change more slowly than volatile requirements.**

Volatile requirements

- are specific to the instantiation of the system in a particular environment and for a particular customer.

Cont'd...

Example:

❑ Consider a system for managing student records in a university.

➤ Such a system will always have to have **information about students**, the **courses** they have taken and the **assessment** of how well they performed in these courses.

- This are stable features of the system.

➤ The system may also maintain information about the students **attendance** at classes, recommended **course groupings**, **standard letters** sent to students.

- Requirements for these are more volatile.

➤ Courses may be taught remotely over the Internet so that class attendance then means something completely different, course groupings change as a subject evolves and the standard letters also change both with course groupings and with changes to administration.

- These are, therefore, volatile features of the system.

Cont'd...

- There are at least 4 different types of volatile requirement.

1. Mutable requirements

- These are requirements which **change because of changes to the environment** in which the system is operating.
- For example, the requirements for a system which computes tax deductions evolve as the tax laws are changed.

2. Emergent requirements

- These are requirements which **cannot be completely defined when the system is specified** but which **emerge as the system is designed and implemented**.
- For example, it may not be possible to specify, in advance, the details of how information should be displayed.
 - As stakeholders see examples of possible presentations, they may think of new ways of presenting information that would be useful to them.

Cont'd...

3. Consequential requirements

- These are requirements which are **based on assumptions about how the system will be used.**
- When the system is put to use, some of these assumptions will be wrong.
- Users will adapt to the system and find new ways to use its functionality.
- This will result in **demands from users for system changes and modifications.**

4. Compatibility requirements

- These are requirements which **depend on other equipment or processes.**
- As this **equipment changes**, these requirements also evolve.
- For example, an instrument system in a power station control room may have to be modified when a new type of information display is added.

Requirements Identification and Storage

- An essential pre-requisite for requirements management is that every requirement must have some kind of unique identification.
- Consequently, effective requirements management is impossible.
- The commonest approach to requirements identification is based on numbering the requirements according to the chapter and section of the requirements document where the requirement is included.
 - Therefore,
 - the 6th requirement in the 2nd section in Chapter 4 would be number 4.2.6;
 - the 8th requirement in the 3rd section of Chapter 2 would be assigned the number 2.3.8; and so on.

Cont'd...

❑ There are two problems with this style of requirements identification.

1. It isn't possible to assign a unique number until the requirements document has been finished.
 - The chapter and section organization must be stable.
 - When a requirement is elicited, it is unclear where it will appear in the requirements document.
 - It can't be assigned a number and therefore can't be referenced by other requirements.
2. Assigning an identifier based on chapter and section numbers implicitly classifies the requirement.
 - It suggests that the requirement is closely related to other requirements with similar identifiers.
 - Document readers may be misled into thinking that there are no other important relationships between that requirement and other requirements elsewhere in the document.

Cont'd...

- ❑ There are **alternative approaches to requirements identification** which address these problems.
- Some of these are shown in Table 6.2 on the next slide.
- If a requirement has a unique identification number as well as a paragraph number, references to the requirement may use its unique identifier.
- This allows for easy rearrangement of the requirements document but can sometimes confuse requirements readers who mix-up the requirements identifier and the paragraph number.

Cont'd...

Table 6.2 Techniques for requirements identification

Identification method	Description
Dynamic renumbering	<ul style="list-style-type: none">• Some word processing systems allow for automatic renumbering of paragraphs and the inclusion of cross- references.• You can therefore assign a number to a requirement at any time.• As you re-organize your document and add new requirements, the system keeps track of the cross-reference.• It automatically renumbers your requirement depending on its chapter, section and position within the section.• All references to the requirement are also renumbered.

Cont'd...

Identification method	Description
Database record identification	<ul style="list-style-type: none">• When a requirement is identified it is immediately entered in a requirements database and a database record identifier is assigned.• This database identifier is used in all subsequent references to the requirement.
Symbolic identification	<ul style="list-style-type: none">• Requirements can be identified by giving them a symbolic name which is associated with the requirement itself.• For example, EFF-I, EFF-2, EFF-3 may be used for requirements which relate to the efficiency of the system.• The problem with this is that it is sometimes difficult to classify requirements and assign a meaningful mnemonic to them.

Cont'd...

- ❑ The requirements must be maintained in a database with each requirement represented as one or more database entities.
 - ✓ The facilities of the database can be used to link related requirements and it is usually possible to formulate fairly complex database queries to identify requirements groupings.
 - ✓ The database may provide some version control facilities or, at least, provision for these facilities to be implemented.
 - ✓ Databases usually include facilities for browsing and report generation.
 - ✓ Related requirements are linked and simple scripts which scan the database, extract parts of the record for each requirement and generate skeletons of the requirements document may be developed.

Cont'd...

- **Relational databases** were designed for storing and managing large numbers of records which have the same structure and **minimal links between them.**
- A requirements database, however, may have relatively few records (hundreds rather than hundreds of thousands) each of which includes many links such as links to documents, text files and other requirements.
- Maintaining these links is possible with a relational database but it is inefficient.
- It requires operations on several different tables.
- For very large numbers of requirements, this type of database may be too slow.

Cont'd...

- ❑ **Object-oriented databases** have been developed relatively recently and are structurally **more suited to requirements management**.
- They are better than relational databases:
 - ✓ when there are many different types of entity to be managed and
 - ✓ where **there are direct links between different entities in the database.**
- They allow different types of information to be maintained in different objects and managing links between objects is fairly straightforward.
- The figure on the next slide shows a set of object classes which could be defined in an object- oriented requirements database.

Cont'd...

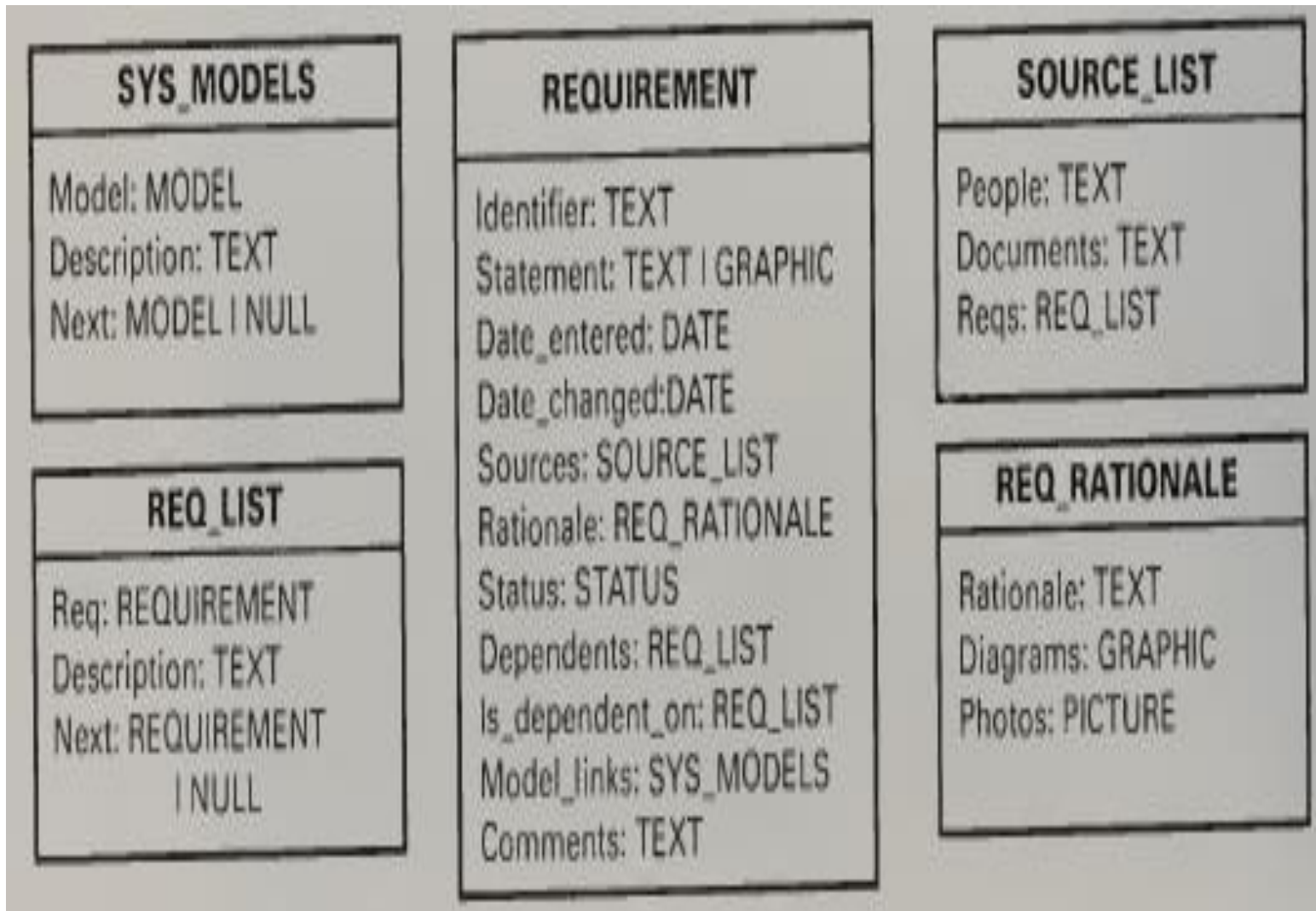


Figure Object classes for a requirements database.

Cont'd...

- The central class is REQUIREMENT which has 11 associated attributes.

1. Identifier

- This is a simple text string which is assigned when a requirement object is created and entered in the database.

2. Statement

- This is a statement of the requirement which may be natural language text or a graphical description of some kind.

3. Date entered

- The date that the requirement was originally entered in the database.

4. Date changed

- The date of the last alteration to the requirement.

Cont'd...

5. Sources

- This is a reference to one or more of the sources of the requirement.
- This helps with analysis when changes to the requirement are proposed.

6. Rationale

- This is a reference to a set of **information** which provides a rationale **explaining why the requirement has been included**.
- The associated information may include text, diagrams or photographs.

7. Status

- This is a variable representing the status of the requirement.
- The status may be '**proposed**', '**under review**', '**accepted**', or '**rejected**'.
- Rejected requirements should be maintained in the database as they may be proposed again in future.
- The analysis of the new proposal is simplified if previous information is available.

Cont'd...

8. Dependents

- This is a list of references to requirements which are dependent on this requirement (i.e. if this requirement is changed, they may also have to be changed)

9. Is_dependent_on

- This is a list of references to requirements on which this requirement depends.
- The Is_dependent_on relationship is therefore the inverse of Dependents.

10. Model links

- This is a link to one or more system models which add detail to the requirement

11. Comments

- This is any other information which may be useful.
- In practice, it is almost impossible to define a schema which covers everything, and having a general description field is often very useful.

Change Management

- Change management is concerned with, the procedures, processes and standards which are used to manage changes to system requirements.
 - Change management ensures that similar information is collected for each proposed change and that overall judgements are made about the costs and benefits of proposed changes.
 - Without formal change management, it is impossible to ensure that proposed changes to the requirements support the fundamental business goals.
- To ensure a consistent approach to change management, organizations may define a set of **change management policies**.
- This cover:
1. The change request process and the information required to process each change request.

Cont'd...

2. The process used to analyze the impact and costs of change and the associated traceability information.
3. The membership of the body which formally considers change requests.
 - It is important to have some 'independent' group who considers change requests as they can make an objective decision about the contribution of the change to the overall goals of the system and the cost-effectiveness of the change.
 - In military projects, this is called the 'Change Request Board' or 'Change Control Board' but in other organizations a less formal group may be used.
4. The software support (if any) for the change control process.

Change Management Processes

- The process of requirements change management consists of a set of activities for documenting, reporting, analyzing, costing and implementing changes to management processes such as change management to delivered software.

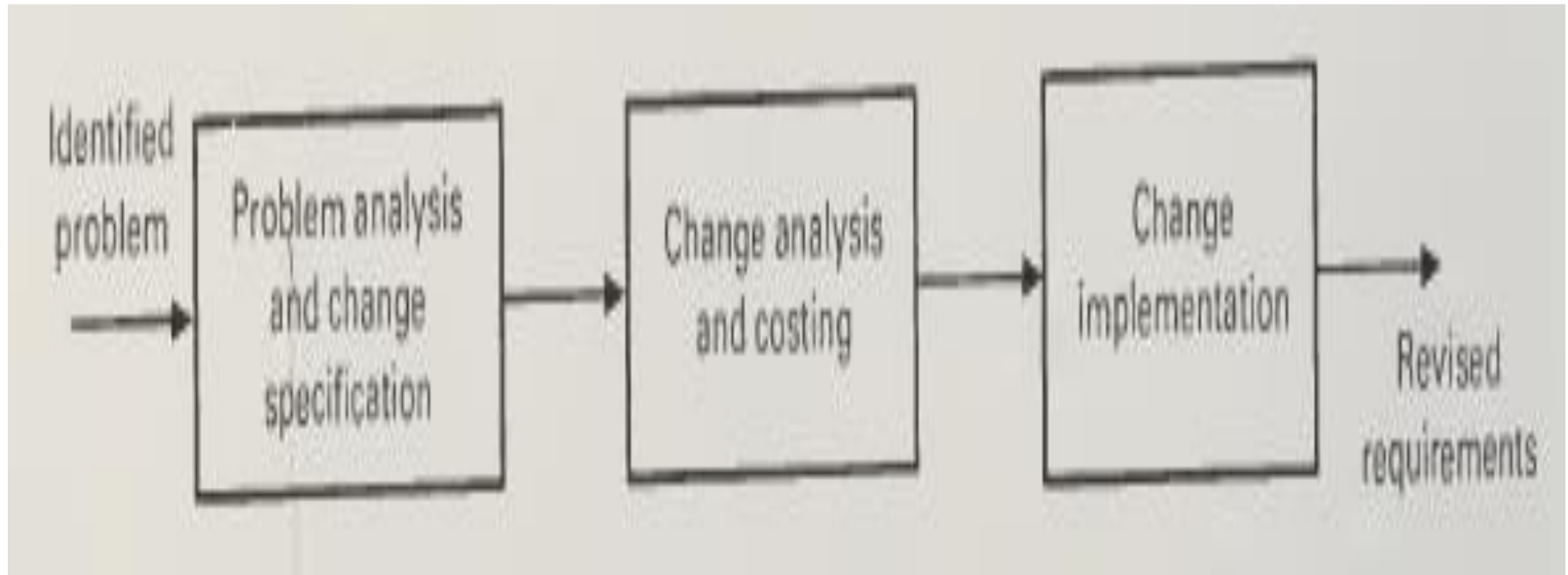


Figure Stages in the change management process

Cont'd...

- The **change management process** can be thought of as a **three-stage process** as shown in the above Figure.
 1. Some requirements problem is identified.
 - The requirements are analyzed using problem information and requirements changes are proposed.
 2. The proposed changes are analyzed to see how many requirements (and, if necessary, system components) are affected by the change and roughly how much it would cost, in both time and money, to make the change.
 3. The change is implemented.
 - A set of amendments to the requirements document or a new document version is produced.

Cont'd...

- The specific processes for problem analysis and change implementation are dependent on:
 - the type of change,
 - the requirements affected and
 - the type of requirements document.
- However, change analysis and costing is a more general process, as shown in the next figure.

Cont'd...

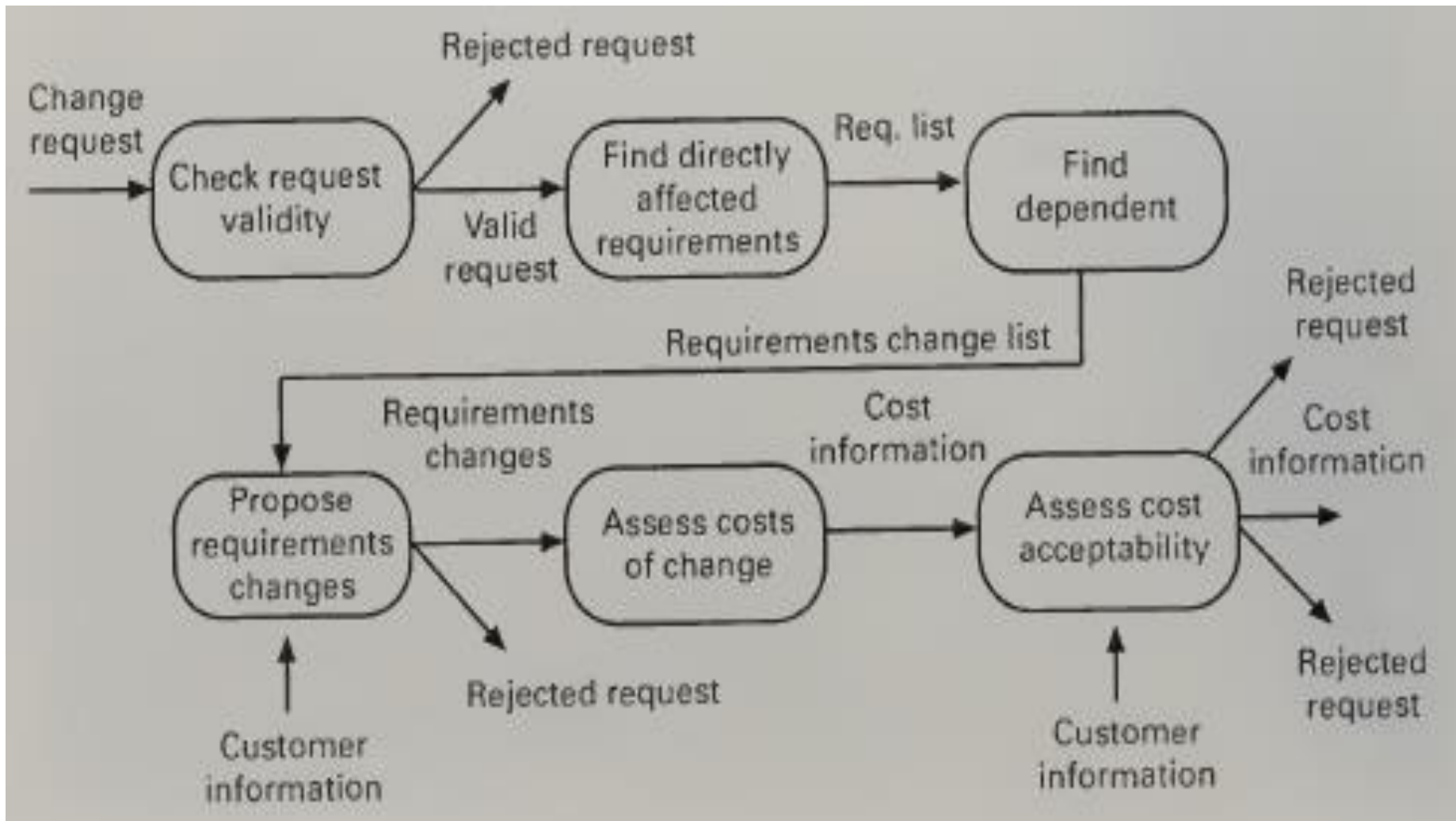


Figure 6.5 The change analysis and costing process.

Cont'd...

- There are six basic activities in the change analysis process:
 1. The change request is checked to see if it is valid.
 - Sometimes, customers misunderstand the requirements and suggest unnecessary changes.
 2. The requirements which are directly affected by the change are discovered.
 3. Traceability information is used to find dependent requirements which may also be affected by the change.
 4. The actual changes which must be made to the requirements are proposed.
 - There may be consultation with customers at this stage to ensure that they are happy with these changes.

Cont'd...

5. The costs of making the changes are estimated.
 - This estimate should include both the effort required to make the change and the amount of calendar time needed.
 - The availability of resources to implement the change must also be considered.
6. Negotiations with customers are held to check if the costs of the proposed changes are acceptable to them.
 - At this stage, it may be necessary to go back to step 4 to propose alternative changes if the customer feels that the change proposal is too expensive.
 - Alternatively, the customer may modify the change request so that the whole process has to be repeated

Cont'd...

- ❑ The **change request may be rejected** at three stages in this process.
- I. If the change request is invalid:
 - this normally arises if a customer has misunderstood something about the requirements and proposed a change which isn't necessary.
 - II. If the change request results in consequential changes which are unacceptable to the user:
 - for example, a change request to decrease the time required to process a transaction may mean that fewer concurrent transactions can be handled.
 - III. If the cost of implementing the change is too high or takes too long.

Tool support for change management

- Change management involves handling large amounts of information and passing it between individuals in an organization.
- It is often necessary to keep track of which changes have been proposed, which have been implemented, which are still under consideration, etc.
- Support for requirements change management may be provided by:
 - specialized requirements management tools or
 - by CASE tools designed to support software configuration management.

Cont'd...

- The capabilities which these tools may provide are:
 1. electronic change request forms which are filled in by different participants in the process
 2. a database to store and manage these forms
 3. a change model which may be instantiated so that people responsible for one stage of the process know who is responsible for the next process activity
 4. electronic transfer of forms between people with different responsibilities and electronic mail notification when activities have been completed
 5. in some cases, direct links to a requirements database.
 - In some cases, this may support the maintenance of multiple versions of a requirement with change information indicating why new requirements versions have been derived.
 - Only the most sophisticated tools provide this functionality.

Cont'd...

- The general problem with these tools is that they all have their own implicit model of the change process.
- Organizations which adopt these tools must conform to that model.
- Special-purpose tools are also fairly expensive and there may be difficulties when integrating these with other CASE tools used in an organization.
- For these reasons, specialized change support tools are mostly used in large organizations such as aerospace companies who are involved in very large projects.
- **General purpose tools** such as **word processors**, **spreadsheets**, and **electronic mail systems** may be used to implement a more limited change management system.

Traceability

- A critical part of the requirements change management process is the assessment of the impact of a change on the rest of the system.
- To carry out impact assessment, information about requirements dependencies, requirements rationale and the implementation of requirements should be maintained to supplement the information in the requirements document.
 - This is usually called traceability information.
- Change impact assessments depend on this traceability information to find out which requirements are affected by a proposed change.

Cont'd...

- Davis (1993) has classified traceability information into four types.

1. Backward-from traceability

- **links requirements to their sources** in other documents or people.

2. Forward-from traceability

- **links requirements to the design and implementation components.**

3. Backward-to traceability

- **links design and implementation components back to requirements.**

4. Forward-to traceability

- **links other documents** (which may have preceded the requirements document) **to relevant requirements.**

Cont'd...

- In practice, it is impossibly expensive to collect and manage all types of traceability information.
- Project managers should define traceability policies setting out what essential traceability information must be maintained.
- Davis's classifications are a useful way to understand the concept of traceability as the traceability information can be visualized as arrows going forwards and backwards from different documents.
- This is illustrated in the following figure which shows how a statement of requirements can include traceability links to and from a design specification and a business plan.

Cont'd...

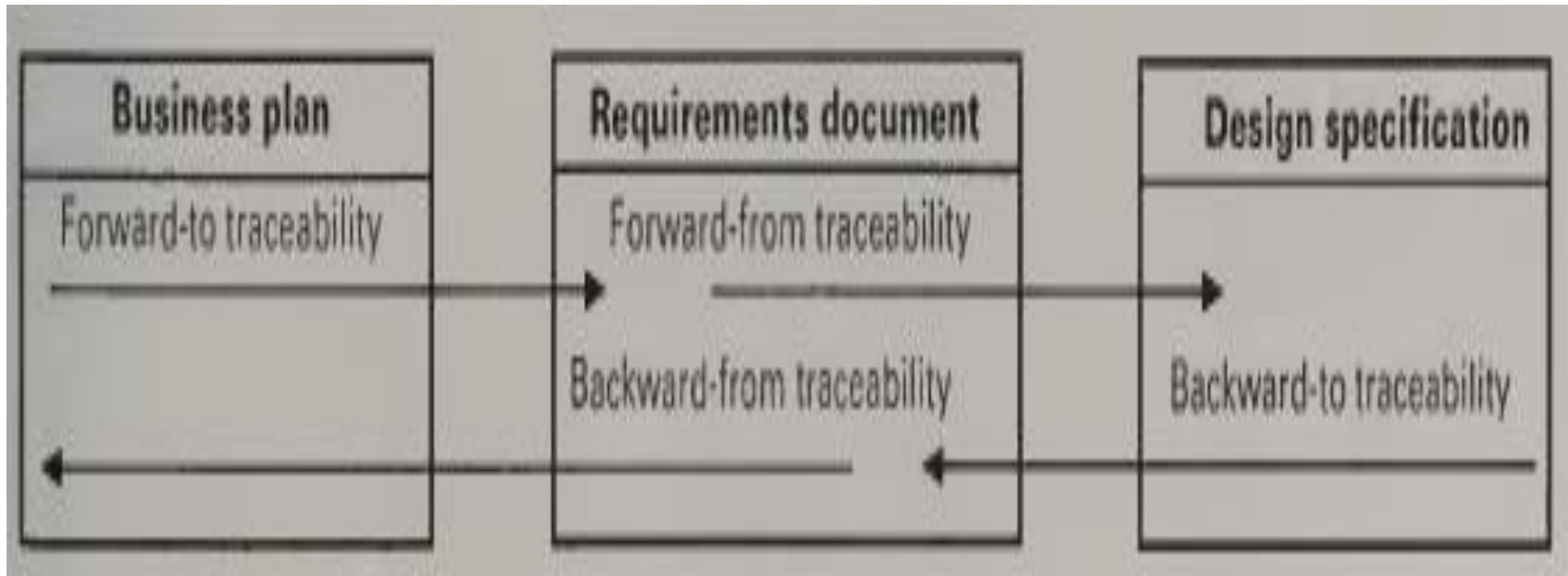


Figure: Backwards and forwards traceability.

- Table 6.1 shows how these different types of traceability can be instantiated more concretely by links between specific information in the requirements document and other system documents.
- In practice, the traceability information which is most commonly maintained during requirements management is **requirements-requirements** traceability and **requirements-design** traceability.

Cont'd...

Table 6.1 Types of traceability

Traceability Type	Description
Requirements-sources traceability	<ul style="list-style-type: none">• Links the requirement and the people or documents which specified the requirement.
Requirements-rationale traceability	<ul style="list-style-type: none">• Links the requirement with a description of why that requirement has been specified.• This can be a distillation of information from several sources.
Requirements-requirements traceability	<ul style="list-style-type: none">• Links requirements with other requirements which are, in some way, dependent on them.• This should be a two- way link (dependants and is-dependent on).

Cont'd...

Traceability Type	Description
Requirements-architecture traceability	<ul style="list-style-type: none">• Links requirements with the sub-systems where these requirements are implemented.• This is particularly important where sub-systems are being developed by different sub-contractors.
Requirements-design traceability	<ul style="list-style-type: none">• Links requirements with specific hardware or software components in the system which are used to implement the requirement.
Requirements-interface traceability	<ul style="list-style-type: none">• Links requirements with the interfaces of external systems which are used in the provision of the requirements.

Traceability Tables

- Traceability tables show the relationships between requirements or between requirements and design components.
- Requirements are listed along the horizontal and vertical axes and relationships between requirements are marked in the table cells.
- A requirements database is not necessary, although recording traceability information in a database can make it much easier to navigate between dependent requirements.

Cont'd...

- Traceability tables for showing requirements dependencies should be defined with requirement numbers used to label the rows and columns of the table.
- In the simplest form of traceability table, you simply put some mark, such as an **asterisk**, in the table cell where there is some kind of dependency relationship between the requirements in the cell row and column.
- That is, if the requirement in row X (say) depends on the requirement in columns P, Q, and R, you should mark table cells (X, P), (X, Q), and (X, R).
- By reading down a column, you see all requirements which depend on a requirement; by reading across a row, you see all requirements which the requirement in that row depends on.

Cont'd...

Depends-on						
	R1	R2	R3	R4	R5	R6
R1			*	*		
R2					*	*
R3				*	*	
R4		*				
R5						*
R6						

Figure : A simple traceability table

- Each row in the table shows dependencies, so that R1 is dependent on R3 and R4, R2 is dependent on R5 and R6, etc.
- Therefore, if a change to R4 is proposed, you can see by reading down the R4 column that requirements R1 and R3 are dependent requirements.
- The impact on R1 and R3 of the proposed change to R4 can therefore be assessed.

Cont'd...

- As the number of requirements grows and matrices become unmanageable, a simplified form of traceability table may be used where, along with each requirement description, one or more lists of the identifiers of related requirements are maintained.
- Traceability lists are simple lists of relationships which can be implemented as text or as simple tables.
- The table on the next slide shows a traceability list for the dependencies shown in the previous figure.

Requirement	Depends-on
R1	R3, R4
R2	R5, R6
R3	R4, R5
R4	R2
R5	R6

Table 6.2 A traceability list

Cont'd...

➤ Traceability lists are more compact than traceability tables and do not become as unmanageable with large numbers of requirements.

✓ They are therefore less prone to error than traceability tables.

- The disadvantage of these lists compared to traceability tables is that there is no easy way to assess the inverse of a relationship.
- You can easily see that R1 is dependent on R3 and R4 but, given R4, you must look through the whole table to see which requirements depend on it.
- If you wish to maintain this 'backward-to' information, you need to construct another table showing these relationships.

Reading Assignment

- Traceability policies

Thank You!

?