Chapter 1

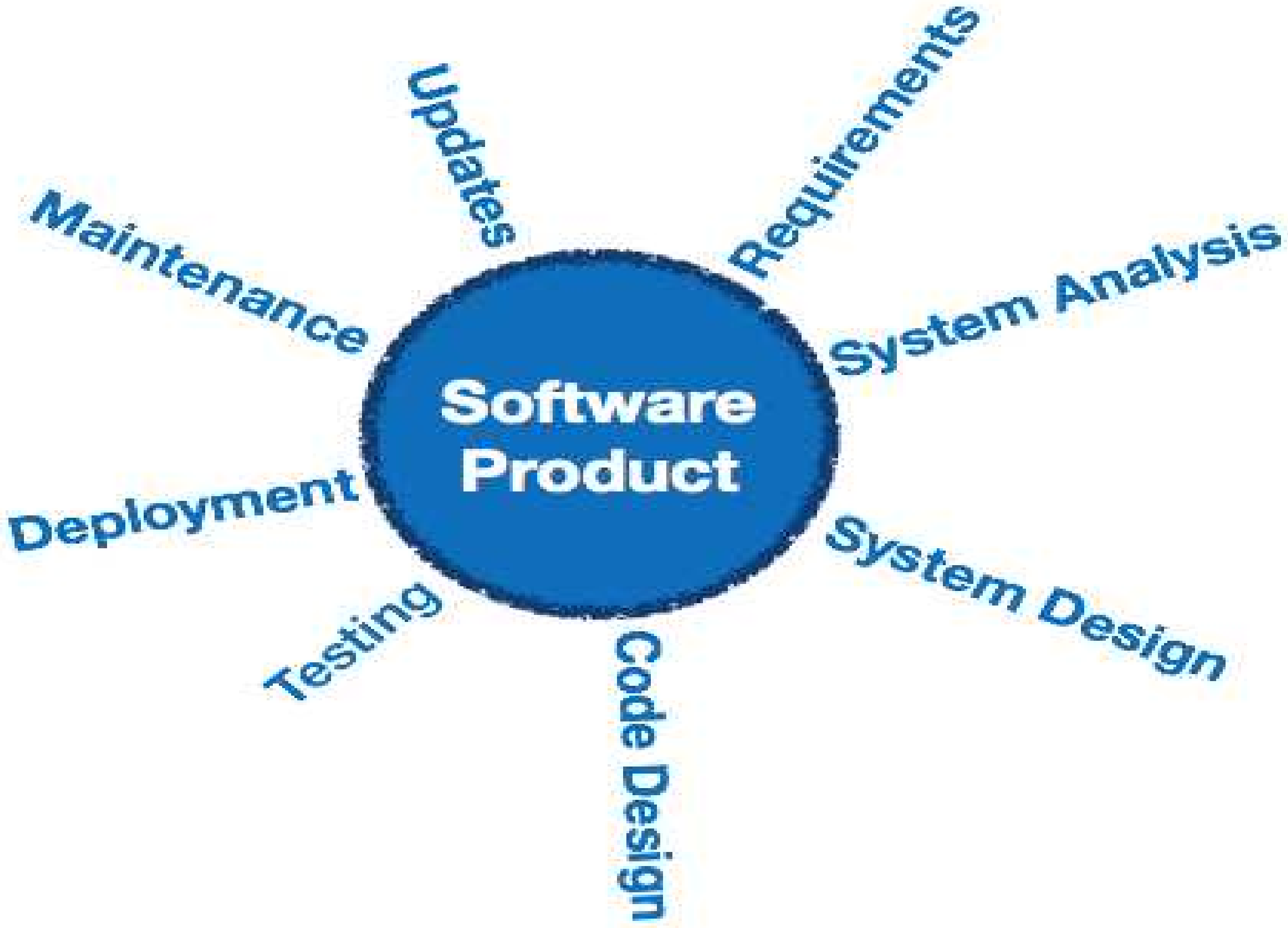
Introduction to Software Engineering

INTRODUCTION TO SOFTWARE ENGINEERING

CHAPTER ONE

WHAT IS SOFTWARE ENGINEERING

- The term software engineering is composed of two words, software and engineering.
- Software is more than just a program code.
- A program is an executable code, which serves some computational purpose.
- Software is considered to be a collection of executable programming code, associated libraries and documentations.
- Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.
- software engineering is an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures.
- The outcome of software engineering is an efficient and reliable software product.
- Without using software engineering principles it would be difficult to develop large programs
- Software engineering principles use two important techniques to reduce problem complexity: abstraction and decomposition.


Software Product
Requirements
System Analysis
System Design
Code Design
Testing
Deployment
Maintenance
Updates

NEED OF SOFTWARE ENGINEERING

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software -** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

Characteristics of Good Software

Operational :-This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional: This aspect is important when the software is moved from one platform to another:

- Portability
- Reusability

Maintenance: how well software has the capabilities to maintain itself in the ever changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

Professional and ethical responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills

- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals

Issues of professional responsibility

- ***Confidentiality***
  - **Engineers should normally respect the confidentiality of their employers or clients.**
- ***Competence***
  - **Engineers should not misrepresent their level of competence. They should not accept work beyond their competence.**

- ***Intellectual property rights***
  - **Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.**
- ***Computer misuse***
  - **Software engineers should not use their technical skills to misuse other people's computers.**
    - **Computer misuse ranges from relatively trivial (game playing on an employer's machine) to extremely serious (dissemination of viruses).**

Software engineering ethics

Some of these are:

1. **Confidentiality** You should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement
has been signed.

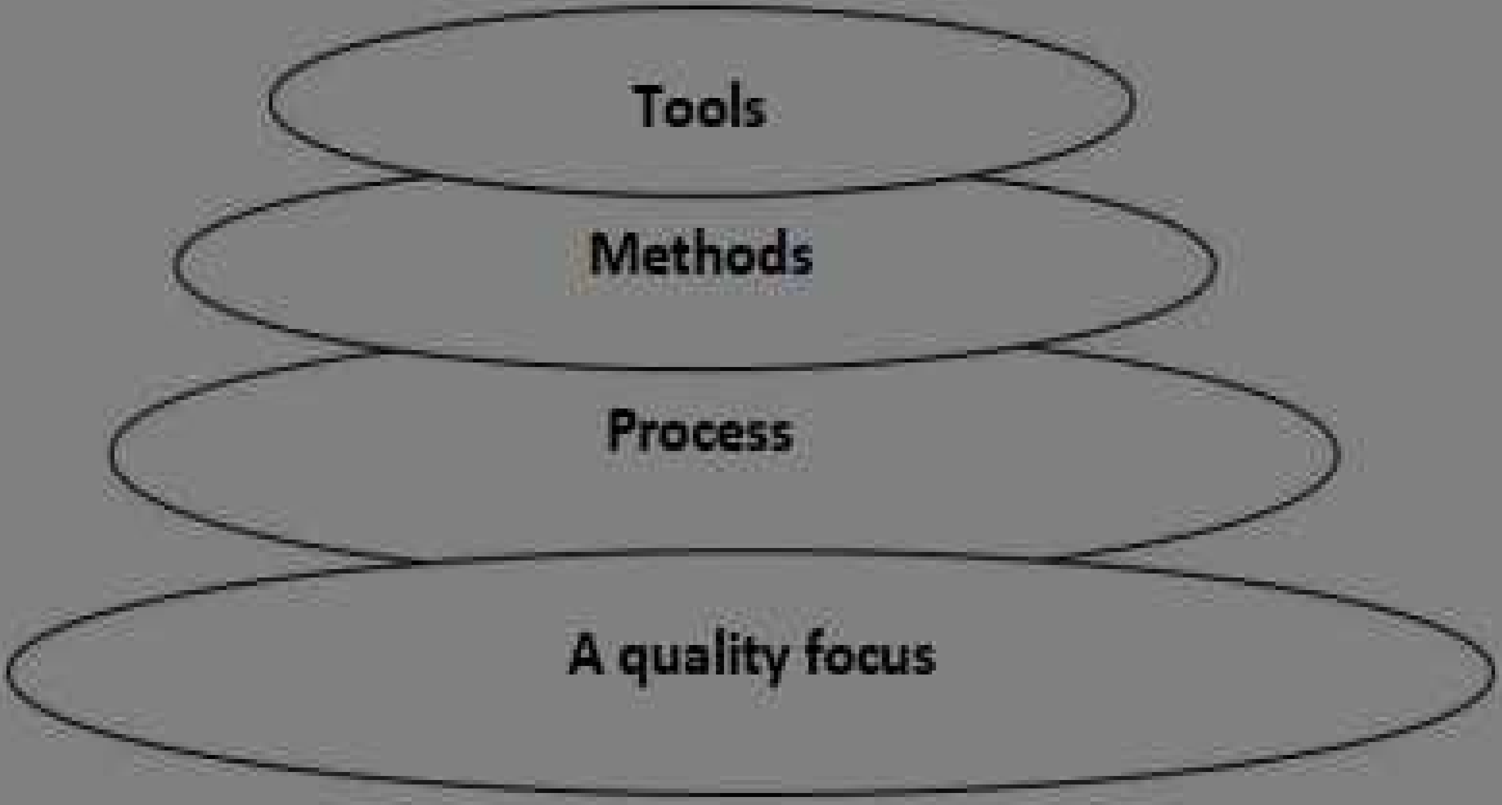
2. **Competence** You should not misrepresent your level of competence.

- You should not knowingly accept work that is outside your competence.

3. **Intellectual property rights** You should be aware of local laws governing the use of intellectual property such as patents and copyright.

- You should be careful to ensure that the intellectual property of employers and clients is protected.
  4. **Computer misuse** You should not use your technical skills to misuse other people's computers.
- Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses or other malware).

SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY:



Software Engineering Layers

The following are the set of Umbrella Activities.

- **Software project tracking and control – allows the software team to assess progress against**
- the project plan and take necessary action to maintain schedule.
- **Risk Management - assesses risks that may effect the outcome of the project or the quality of**
- the product.
- **Software Quality Assurance - defines and conducts the activities required to ensure**
- software quality.
- **Formal Technical Reviews - assesses software engineering work products in an effort to**
- uncover and remove errors before they are propagated to the next action or activity.

- **Measurement - define and collects process, project and product measures that assist the team in**
- delivering software that needs customer‘s needs, can be used in conjunction with all other framework and umbrella activities.
- **Software configuration management - manages the effects of change throughout the software** process.
- **Reusability management - defines criteria for work product reuse and establishes mechanisms** to achieve reusable components.
- **Work Product preparation and production - encompasses the activities required to create** work products such as models, document, logs, forms and lists.

Chapter 2

Software Processes

“You’ve got to be very careful if you don’t know where you’re going, because you might not get there.”

Yogi Berra

Learning outcomes:

- What is software process?
- Understand that organizations and their members are systems and that analysts need to take a systems perspective.
- Depict systems graphically using context-level data flow diagrams, entity-relationship models, and use cases and use case scenarios.
- Comprehend that organizational culture impacts the design of information systems

Software processes

- There are many different software processes but all must include four activities that are fundamental to software engineering:
  1. **Software specification** The functionality of the software and constraints on its operation must be defined.
  2. **Software design and implementation** The software to meet the specification must be produced.
  3. **Software validation** The software must be validated to ensure that it does what the customer wants.
  4. **Software evolution** The software must evolve to meet changing customer needs

Software processes

- A structured set of activities required to develop a software system
- A software process model is an abstract representation of a process.
- It presents a description of a process from some particular perspective

Software process descriptions

When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities

Process descriptions may also include:

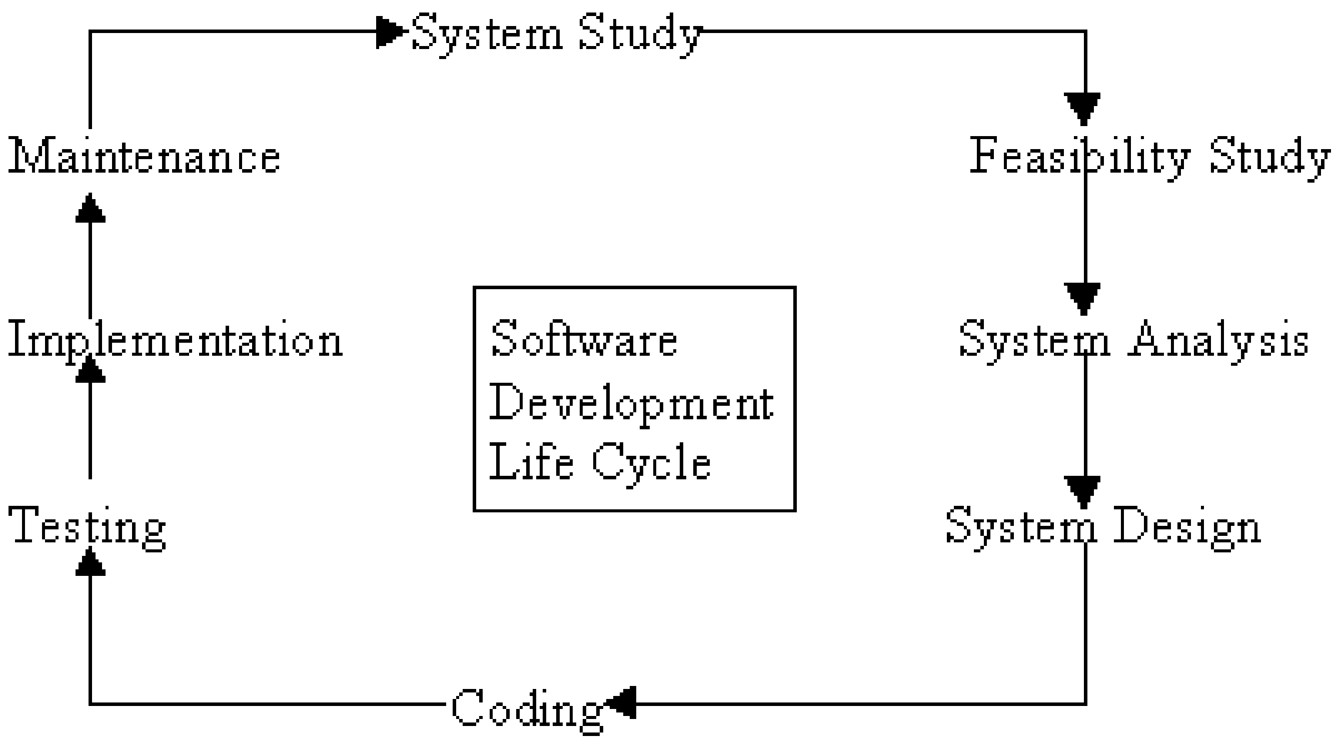
- **Products**, which are the outcomes of a process activity;
- **Roles**, which reflect the responsibilities of the people involved in the process;
- **Pre- and post-conditions**, which are statements that are true before and after a process activity has been enacted or a product produced.

Software processes types

- Software processes may be categorized as either plan-driven or agile processes
- **Plan-driven processes** are processes where all of the process activities are planned in advance and progress is measured against this plan
- **agile processes,** planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches
- There are no right or wrong software processes

SDLC Model

A framework that describes the activities performed at each stage of a software development project.

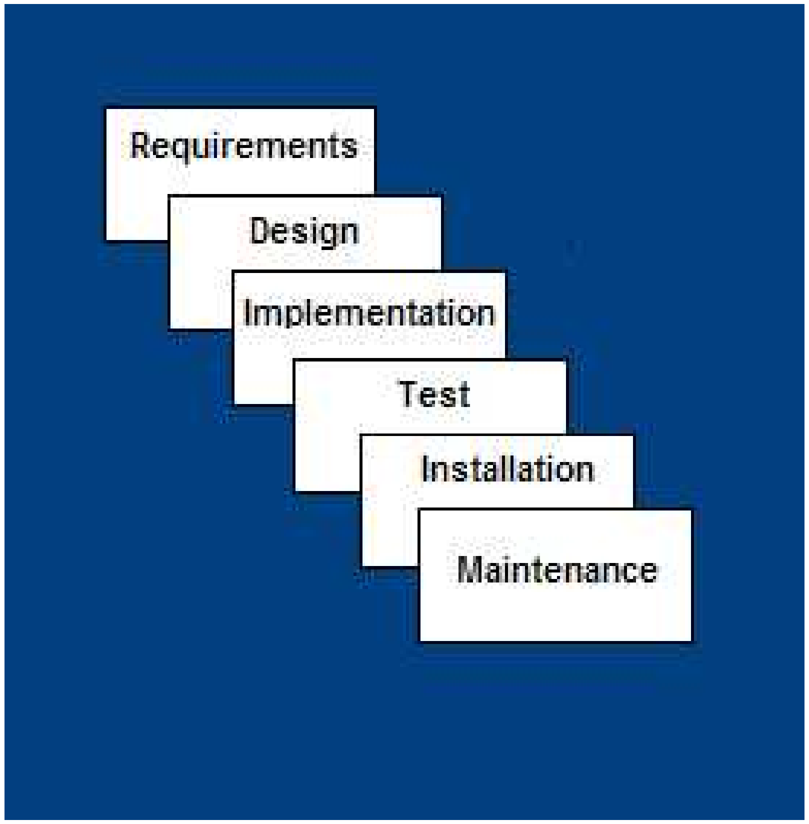


Systems Development Methodologies

- In the continuing effort to improve the systems analysis and design process, several different methodologies have been developed. Some of the popular and widely used methodologies are:

i. Waterfall Model
ii. Prototyping Model
iii. Iterative Enhancement Model
iv. Spiral Model
v. Rapid Application Model
vi. Big Bang Model

Waterfall Model



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation, testing.

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

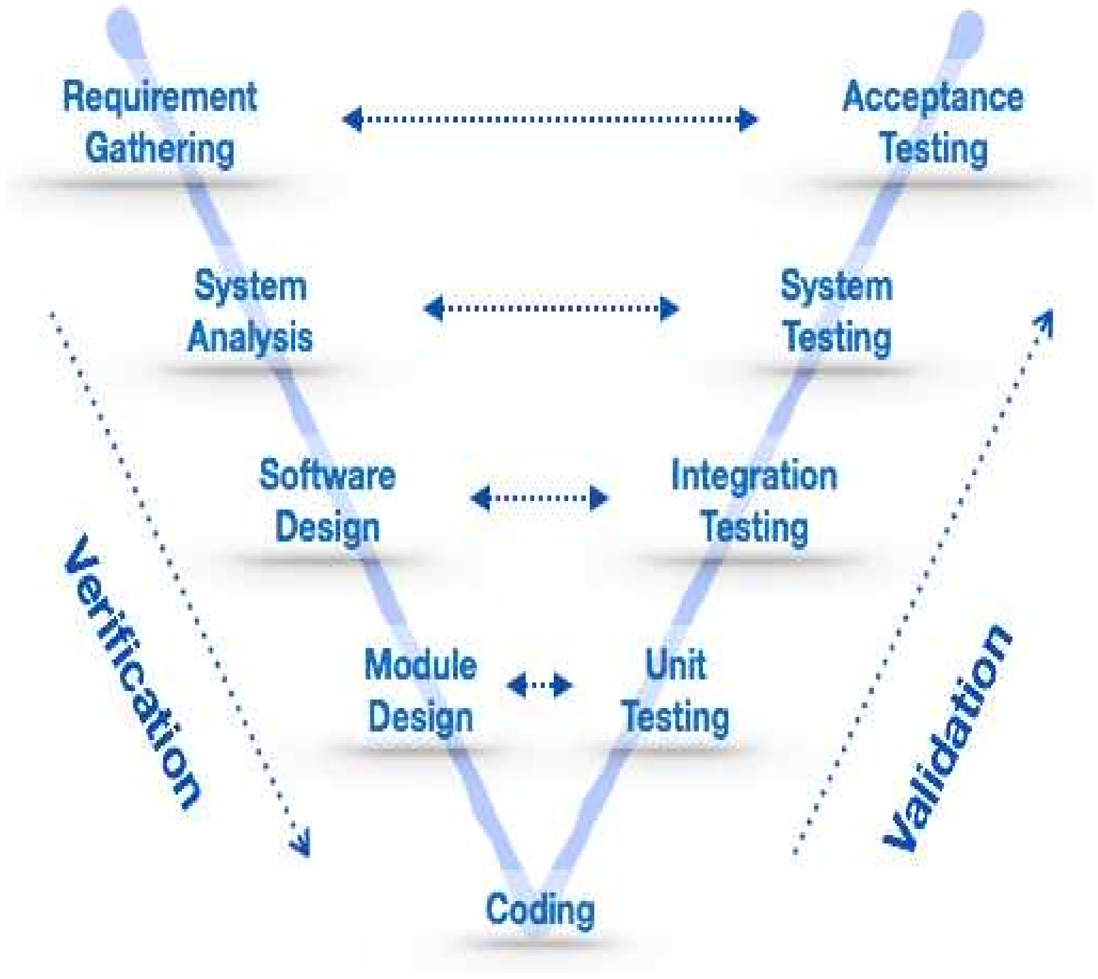
Waterfall Deficiencies

- All requirements must be known upfront
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use the Waterfall Model

- Requirements are very well known
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

V-Shaped Model



- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system
- Architecture or High-Level Design – defines how software functions fulfill the design
- Detailed Design – develop algorithms for each architectural component
- Production, operation and maintenance – provide for enhancement and corrections
- System and acceptance testing – check the entire software system in its environment
- Integration and Testing – check that modules interconnect correctly
- Unit testing – check that each module acts as expected
- Coding – transform algorithms into software

V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

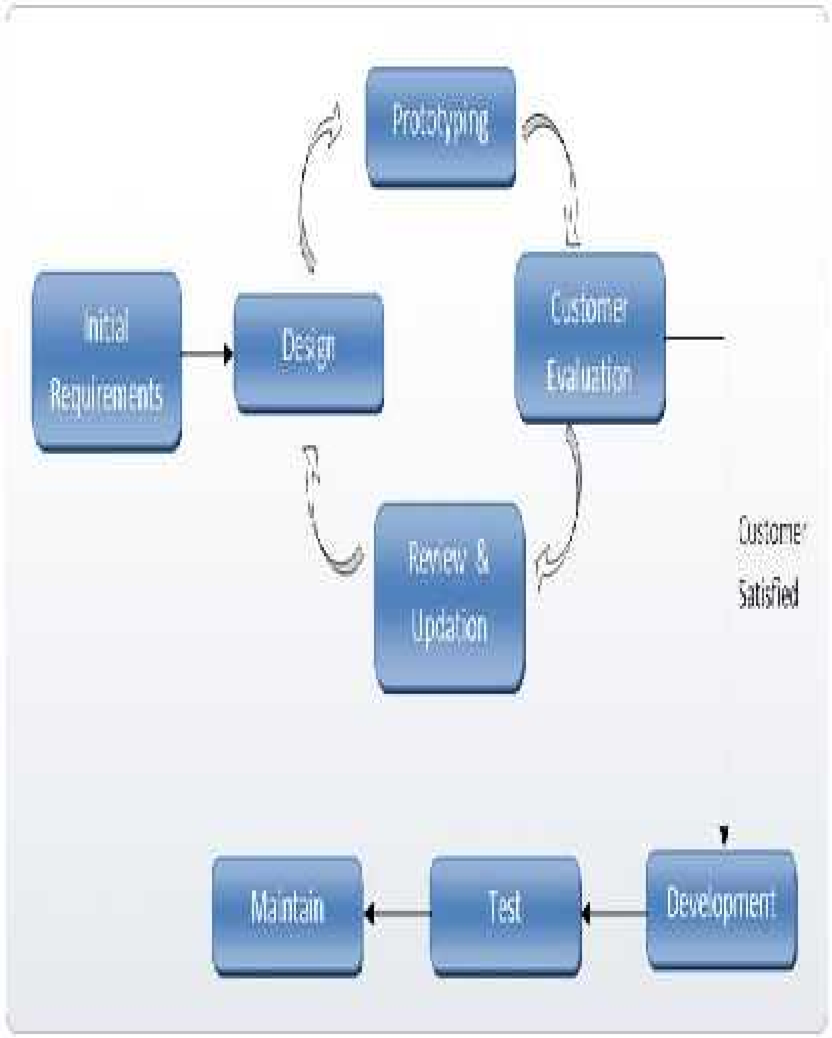
V-Shaped Weaknesses

- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

When to use the V-Shaped Model

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When it can be modified to handle changing requirements beyond analysis phase
- Solution and technology are known

Prototyping Model



- Developers build a prototype during the requirements phase
- Prototype is evaluated by end users
- Users give corrective feedback
- Developers further refine the prototype
- When the user is satisfied, the prototype code is brought up to the standards needed for a final product.

Structured Evolutionary Prototyping Steps

- A preliminary project plan is developed
- Partial high-level paper model is created
- The model is source for a partial requirements specification
- A prototype is built with basic and critical attributes
- The designer builds
  - the database
  - user interface
  - algorithmic functions
- The designer demonstrates the prototype, the user evaluates for problems and suggests improvements.
- This loop continues until the user is satisfied

Structured Evolutionary Prototyping Strengths

- Customers can “see” the system requirements as they are being gathered
- Developers learn from customers
- A more accurate end product
- Unexpected requirements accommodated
- Allows for flexible design and development
- Steady, visible signs of progress produced
- Interaction with the prototype stimulates awareness of additional needed functionality

S.E Prototyping Weaknesses

- Tendency to abandon structured program development for "code-and-fix" development
- Bad reputation for "quick-and-dirty" methods
- Overall maintainability may be overlooked
- The customer may want the prototype delivered.
- Process may continue forever (scope creep)

When to use Structured Evolutionary Prototyping

- Requirements are unstable or have to be clarified
- As the requirements clarification stage of a waterfall model
- Develop user interfaces
- Short-lived demonstrations
- New, original development
- With the analysis and design portions of object-oriented development.

Rapid Application Model (RAD)

- Requirements planning phase (a workshop utilizing structured discussion of business problems)
- User description phase – automated tools capture information from users
- Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. (“Do until done”)
- Cutover phase -- installation of the system, user acceptance testing and user training

RAD Strengths

- Reduced cycle time and improved productivity with fewer people means lower costs
- Time-box approach mitigates cost and schedule risk
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs

RAD Weaknesses

- Accelerated development process must give quick responses to the user
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

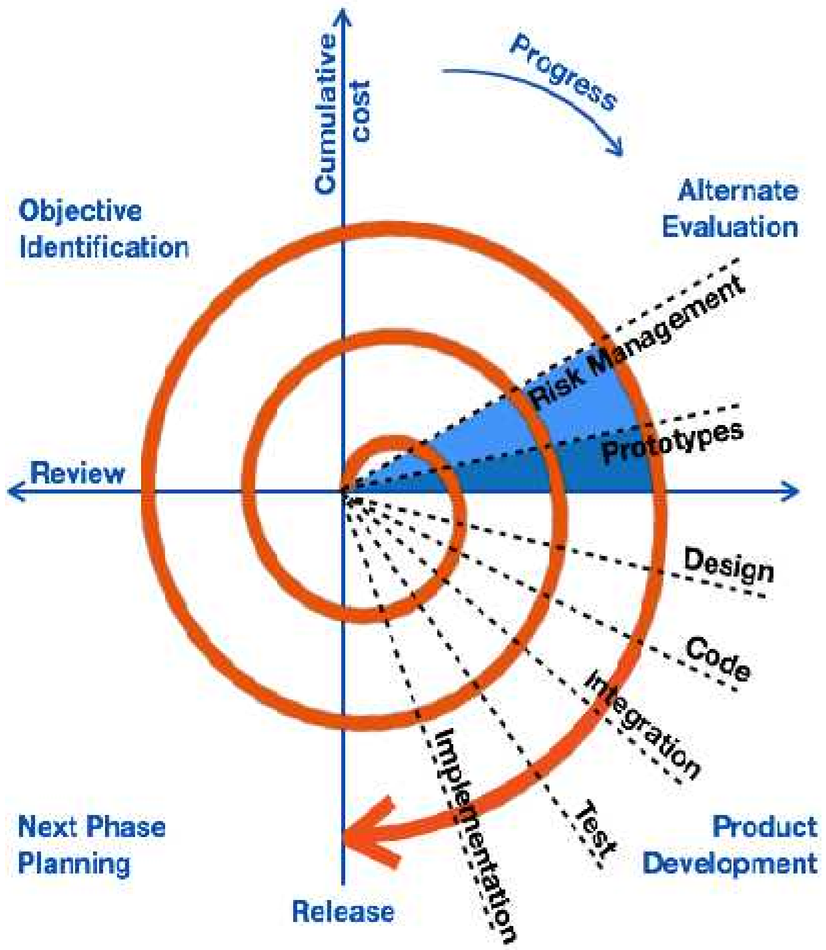
When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

Spiral Model

- This model considers risk, which often goes un-noticed by most other models.
- The model starts with determining objectives and constraints of the software at the start of one iteration.
- Next phase is of prototyping the software. This includes risk analysis.
- In the fourth phase of the plan of next iteration is prepared.

Prototyping Model



- This model considers risk, which often goes un-noticed by most other models.
- The model starts with determining objectives and constraints of the software at the start of one iteration.
- Next phase is of prototyping the software. This includes risk analysis.
- In the fourth phase of the plan of next iteration is prepared. .

Advantages

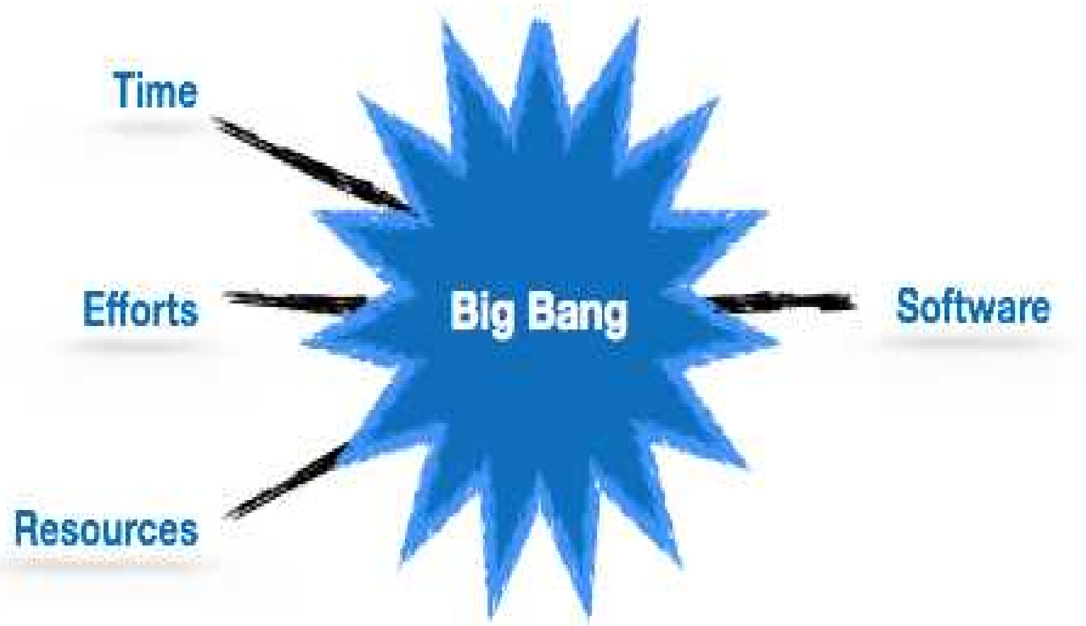
- It provides the potential for rapid development of increasingly more complete versions of the software.
- The spiral model is a realistic approach to the development of large-scale systems and software.
- The spiral model uses prototyping as a risk reduction mechanism but, more importantly enables the developer to apply the prototyping approach at any stage in the evolution of the product.

Draw Backs

- The spiral model is not a panacea.
- It may be difficult to convince customers that the evolutionary approach is controllable.
- It demands considerable risk assessment expertise and relies on this expertise for success.
- If a major risk is not uncovered and managed, problems will undoubtedly occur.

Big Bang Model

- This model is the simplest model in its form. It requires little planning, lots of
- programming and lots of funds. This model is conceptualized around the big bang
- of universe. As scientists say that after big bang lots of galaxies, planets, and
- stars evolved just as an event. Likewise, if we put together lots of programming
- and funds, you may achieve the best software product.


Time
Efforts
Resources
Big Bang
Software

Iterative Model

- The iterative enhancement life cycle model counters the limitations of the waterfall model and tries to combine the benefits of both prototyping and the waterfall model

Advantages of Iterative Model

i. i. This approach can result in better testing

ii. ii. The increments provides feedback to the client which is useful for determining the final

Fundamental of Software Engineering

Chapter 3: Requirement Engineering

Topics covered

- Feasibility studies
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

Objectives

- To describe the principal requirements engineering activities
- To introduce techniques for requirements elicitation and analysis
- To describe requirements validation
- To discuss the role of requirements management in support of other requirements engineering processes

Requirements engineering (RE)

- The process to gather the software requirements from client, analyze, and document them is known as requirement engineering. The goal of requirement engineering is to develop and maintain sophisticated and descriptive ‘System Requirements Specification’ document.

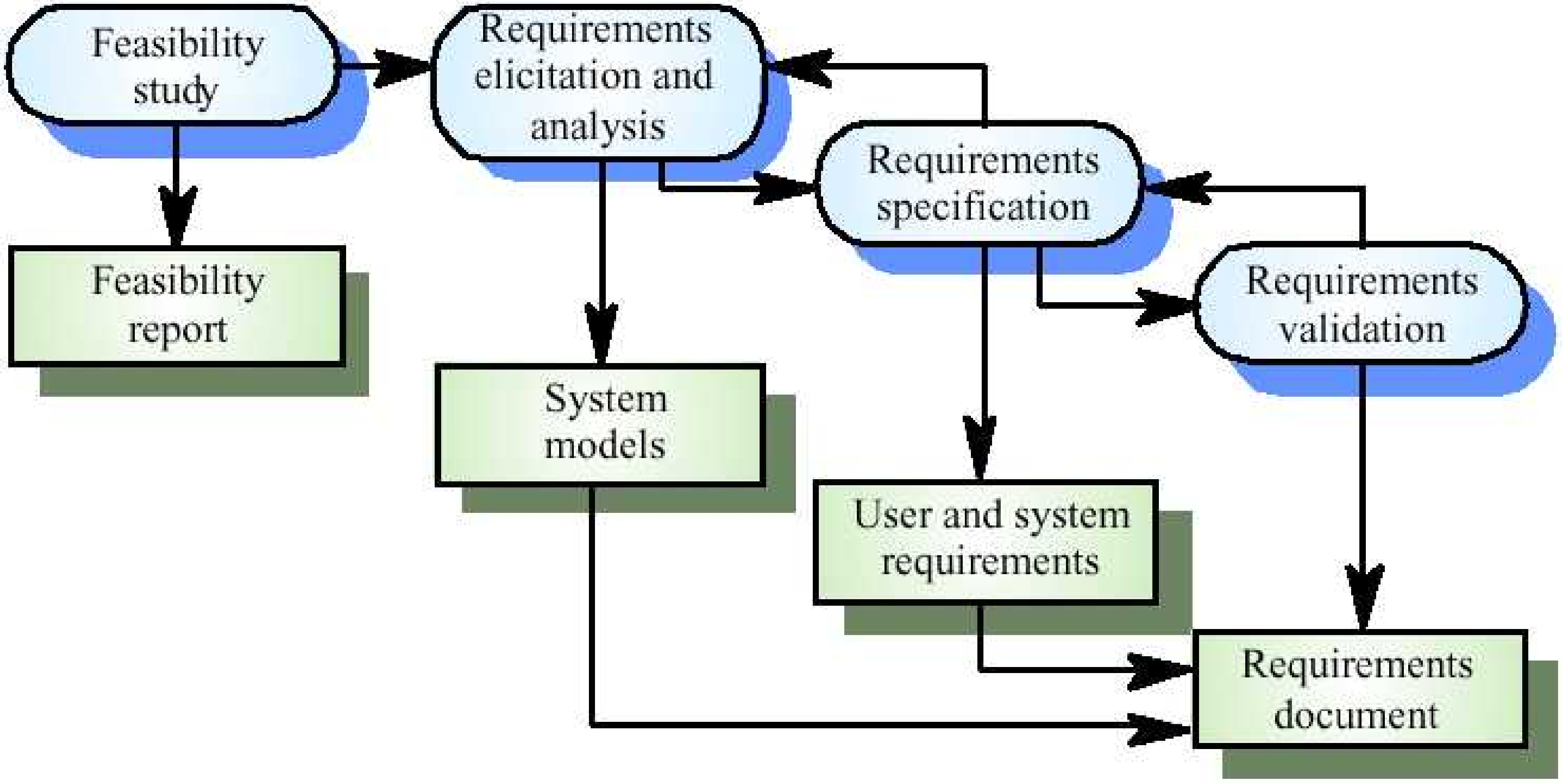
Requirements engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organization developing the requirements
- However, there are a number of generic activities common to all processes

1. Feasibility Study
2. Requirements elicitation/ Requirement Gathering
3. Requirements Specification / Requirement analysis
4. Requirements validation
5. Requirements management

- In practice, RE is an iterative activity in which these processes are interleaved

The requirements engineering process



Feasibility studies

- A feasibility study decides whether or not the proposed system is worthwhile.
- A short focused study that checks
  - If the system contributes to organisational objectives;
  - If the system can be engineered using current technology and within budget;
  - If the system can be integrated with other systems that are used.

Feasibility study implementation

- Based on information assessment (what is required), information collection and report writing.
- Questions for people in the organisation
  - What if the system wasn’t implemented?
  - What are current process problems?
  - How will the proposed system help?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed system?

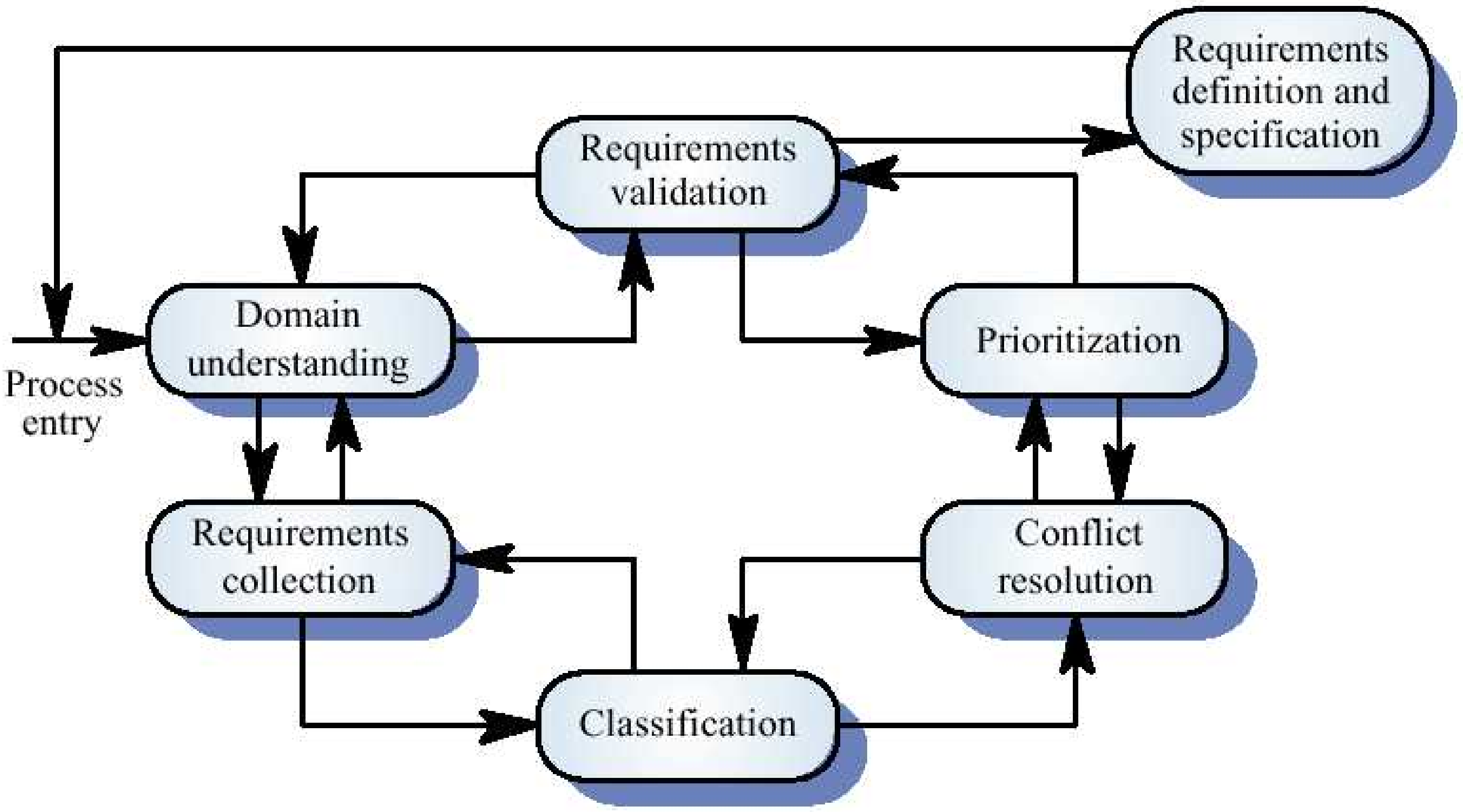
Elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system’s operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders*.

Problems of requirements analysis

- Stakeholders don’t know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment change.

The requirements analysis process



Requirement Gathering

- If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user.
- Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification (SRS)

- SRS is a document created by system analyst after the requirements are collected from various stakeholders.
- SRS defines how the intended software will interact with hardware, external
  interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.
- The requirements received from client are written in natural language.
- It is the responsibility of the system analyst to document the requirements in technical language so that they can be comprehended and used by the software development team.

SRS should come up with the following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

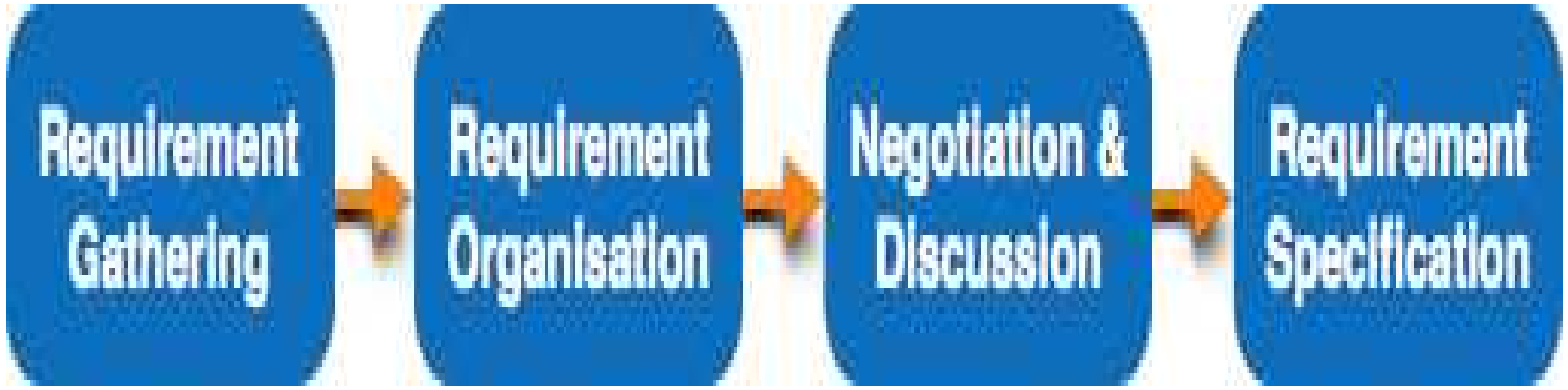
- After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or
experts may interpret the requirements inaccurately.
- This results in huge increase in cost if not nipped in the bud.

Requirements can be checked against following conditions

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

- Requirement elicitation process can be depicted using the following diagram:



- ✓ **Requirements discovery**: - The developers discuss with the client and end users and know their expectations from the software. Domain requirements are also discovered at this stage
- ✓ **Requirements classification and organisation**:-Groups related requirements and organizes them into coherent clusters
- - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- - Prioritizing requirements and resolving requirements conflicts
- ✓ **Negotiation & discussion** - If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, it is then negotiated and discussed with the stakeholders. Requirements may then be prioritized and Unrealistic requirements are compromised reasonably.
- • The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness.
- ✓ **Requirements specification**:-Documentation - All formal and informal, functional and non-functional requirements are documented and made available for next phase processing.

Problems of requirements elicitation

- Stakeholders don’t know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements. Organizational and political factors may influence the system requirements.
- The requirements change during the analysis process.
- New stakeholders may emerge and the business environment change

Requirements validation techniques

- **Requirements reviews**;- Systematic manual analysis of the requirements
- **Prototyping**:- Using an executable model of the system to check requirements
- **Test-case generation**:- Developing tests for requirements to check testability

Requirement Elicitation Techniques

There are various ways to discover requirements. Some of them are explained below:

- ✓ Interviews
- ✓ Surveys
- ✓ Questionnaires
- ✓ Task analysis
- ✓ Domain Analysis
- ✓ Prototyping
- ✓ Brainstorming
- ✓ Observation

Interviews

- Interviews are strong medium to collect requirements. Organization may conduct
- several types of interviews such as:
- • Structured (closed) interviews, where every single information to gather is
- decided in advance, they follow pattern and matter of discussion firmly.
- • Non-structured (open) interviews, where information to gather is not decided in advance, 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.

Surveys

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

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.

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.

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.

Brainstorming

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

Prototyping

- Prototyping is building user interface without adding detail functionality for user
- to interpret the features of intended software product. It helps giving better idea
- of requirements. If there is no software installed at client’s end for developer’s
- reference and the client is not aware of its own requirements, the developer
- creates a prototype based on initially mentioned requirements. The prototype is
- shown to the client and the feedback is noted. The client feedback serves as an
- input for requirement gathering.

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 the
- client’s end and how execution problems are dealt. The team itself draws some
- conclusions which aid to form requirements expected from the software.

What is Requirement

- A requirement is simply a statement of what the system must do or what characteristics it needs to have. During a systems development project.

➢ **Why: Enterprise requirements**

- Context analysis: the reason why the system to be created. Constraints on the environment in which the system is to function

➢ **What: Functional requirements (system)**

- A description of what the system is to do. What information needs to be maintained? What needs to be processes?
- Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform.

➢ **How: Non-functional requirements (system)**

- How the system is to be constructed and function.

Types of requirement

- **Functional requirements:**
  - what the software should do.
- **Business requirements:**
  - requirements will be created that describe what the business needs.
- **User requirements:**
  - what the users need to do.
- **Non-functional requirements:**
  - characteristics the system should have
- **System requirements:**
  - how the system should be built

Software Requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations
- Functional requirements May state what the system should not do
- Broadly software requirements should be categorized in two categories:

1. **Functional Requirements**
2. **Non-Functional Requirements**

Software Requirements Characteristics

- Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct, and well-defined. A complete Software Requirement Specifications must be:
  - Clear
  - Correct
  - Consistent
  - Coherent
  - Comprehensible
  - Modifiable
  - Verifiable
  - Prioritized
  - Unambiguous
  - Traceable
  - Credible source

Functional Requirements

- Requirements, which are related to functional aspect of software fall into this category.
- They define functions and functionality within and from the software system.

EXAMPLES –

- The software automatically validates customers against the ABC Con- tact Management System
- Example
- Only Managerial level employees have the right to view revenue data The software system should be integrated with banking API User should be able to mail any report to management Software is developed keeping downward compatibility intact

Non-Functional Requirements

- They are implicit or expected characteristics of software, which users make assumption of.
- A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system
- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc They are implicit or expected characteristics of software, which users make assumption of
- May be more critical than functional requirements. If these are not met, the system may be useless.
- Non-functional requirements may affect the overall architecture of a system rather than the individual components
- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required

Non-functional classifications

- **Product requirements:-**Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc
- **Organizational requirements:-**Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc
- **External requirements:-** Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc

Non-functional requirements include –

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Metrics for specifying nonfunctional requirements

Property	Measure
Speed	Processed transactions(second User)event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure. Probability of unavailability. Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure. Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

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 and negation, whereas ‘Could have’ and ‘Wish list’ can be kept for software updates.

User Interface requirements

- User Interface (UI) is an important part of any software or hardware or hybrid system. A software is widely accepted if it is –
  - easy to operate
  - quick in response
  - effectively handling operational errors
  - providing simple yet consistent user interface
- User acceptance majorly depends upon how user can use the software.
- UI is the only way for users to perceive the system.
- A well performing software system must also be equipped with attractive, clear, consistent, and responsive user interface.
- Otherwise the functionalities of software system can not be used in convenient way.
- A system is said to be good if it provides means to use it efficiently.

User interface requirements are briefly mentioned below

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategical use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst

- System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly and accurately.
- Role of an analyst starts during Software Analysis Phase of SDLC.
- It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

Needs analysis

1) **Business perspective**: An outline of the current business climate, structure of company and the emerging industry issues that are driving the project.

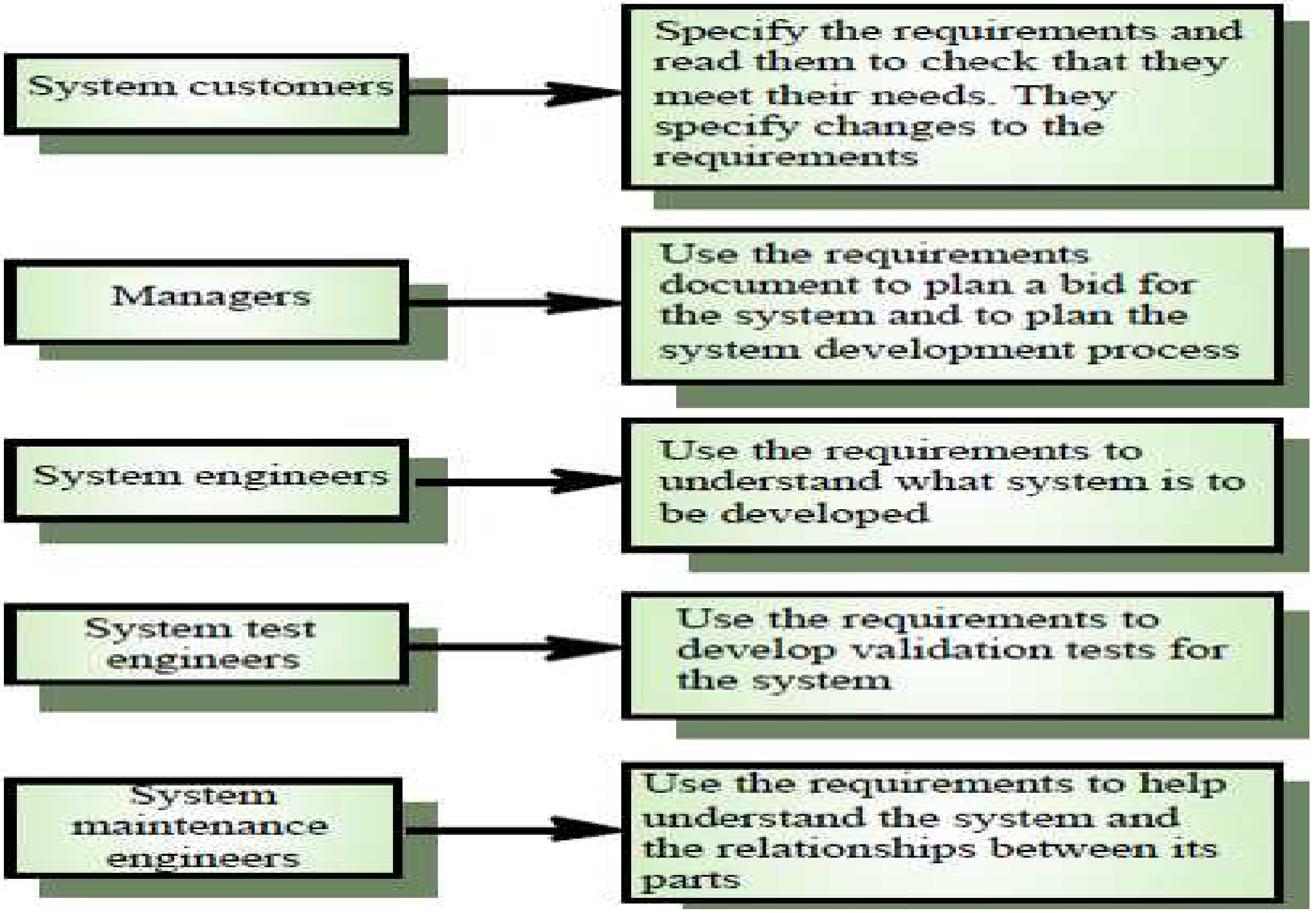
2) **Technical perspective**: An outline of existing IT systems/infrastructure of the company including computer hardware specifications, numbers and locations, details on browsers, operating systems, servers, security policies, networks, bandwidth capacity and so on.

3) **Human perspective**: An outline of the motivation of staff to use new IT systems. It may also cover such considerations as PC literacy, industrial relations issues for staff, legalities and even language issues for users.

System Analysts have the following responsibilities

- ✓ Analyzing and understanding requirements of intended software
- ✓ Understanding how the project will contribute to the organizational objectives
- ✓ Identify sources of requirement
- ✓ Validation of requirement
- ✓ Develop and implement requirement management plan
- ✓ Documentation of business, technical, process, and product requirements
- ✓ Coordination with clients to prioritize requirements and remove ambiguity
- ✓ Finalizing acceptance criteria with client and other stakeholders

Users of a requirements document



System customers
Specify the requirements and read them to check that they meet their needs. They specify changes to the requirements
Managers
Use the requirements document to plan a bid for the system and to plan the system development process
System engineers
Use the requirements to understand what system is to be developed
System test engineers
Use the requirements to develop validation tests for the system
System maintenance engineers
Use the requirements to help understand the system and the relationships between its parts

Requirements specification

- The process of writing the user and system requirements in a requirements document
- User requirements have to be understandable by end-users and customers who do not have a technical background System requirements are more detailed requirements and may include more technical information
- The requirements may be part of a contract for the system development

Ways of writing a system requirements specification

Notation	Description
Natural lan- Gauge	The requirements are written using numbered sentences in natural language. Each sentence should express one require- ment
Structured natural language	The requirements are written in natural language on a stan- dard form or template. Each field provides information about an aspect of the requirement
Design de- \scription languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This ap- proach is now rarely used although it can be useful for in- terface specifications
Graphical Notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are used

Requirements and design

- In principle, requirements should state what the system should do and the design should describe how it does this
- In practice, requirements and design are inseparable
- A system architecture may be designed to structure the requirements The system may inter-operate with other systems that generate design requirements
- The use of a specific architecture to satisfy non-functional requirements may be a domain requirement
- This may be the consequence of a regulatory requirement

Guidelines for writing requirements

- Invent a standard format and use it for all requirements
- Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements
- Use text highlighting to identify key parts of the requirement
- Avoid the use of computer jargon
- Include an explanation (rationale) of why a requirement is necessary

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers

Problems with natural language

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

Structured specifications

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

Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews. Reviews may be formal (with completed documents) or informal.
- Good communications between developers, customers and users can resolve problems at an early stage.

Review checks

- **Verifiability**: Is the requirement realistically testable?
- **Comprehensibility**: Is the requirement properly understood?
- **Traceability**: Is the origin of the requirement clearly stated?
- **Adaptability**: Can the requirement be changed without a large impact on other requirements?

Requirements checking

- **Validity**. Does the system provide the functions which best support the customer's needs?
- **Consistency**. Are there any requirements conflicts?
- **Completeness**. Are all functions required by the customer included?
- **Realism**. Can the requirements be implemented given available budget and technology
- **Verifiability**. Can the requirements be checked?

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements

Changing requirements

- The business and technical environment of the system always changes after installation
- The people who pay for a system and the users of that system are rarely the same people
- System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

Changing requirements

- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory
- The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed

Requirements management planning

- Establishes the level of requirements management detail that is required
- Requirements management decisions:
- Requirements identification Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
- A change management process This is the set of activities that assess the impact and cost of changes.
- Traceability policies These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
- Tool support Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems

Requirements management planning

- During the requirements engineering process, you have to plan:
  - Requirements identification
    - How requirements are individually identified;
  - A change management process
    - The process followed when analysing a requirements change;
  - Traceability policies
    - The amount of information about requirements relationships that is maintained;
  - CASE tool support
    - The tool support required to help manage requirements change;

Traceability

- Traceability is concerned with the relationships between requirements, their sources and the system design
- Source traceability
  - Links from requirements to stakeholders who proposed these requirements;
- Requirements traceability
  - Links between dependent requirements;
- Design traceability
  - Links from the requirements to the design;

A traceability matrix

Req. id	**1.1**	**1.2**	**1.3**	**2.1**	**2.2**	**2.3**	**3.1**	**3.2**
1.1		D	R					
1.2			D			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

CASE tool support

- Requirements storage
  - Requirements should be managed in a secure, managed data store.
- Change management
  - The process of change management is a workflow process whose stages can be defined and information flow between these stages partially automated.
- Traceability management
  - Automated retrieval of the links between requirements.

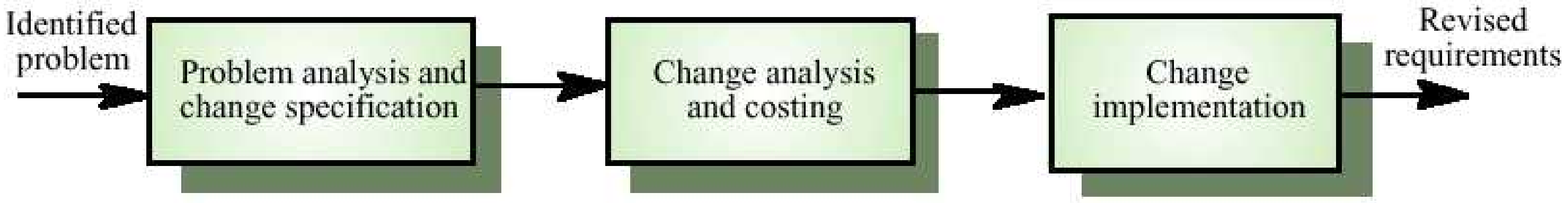
Requirements change management

- Should apply to all proposed changes to the requirements.
- Principal stages
  - Problem analysis. Discuss requirements problem and propose change;
  - Change analysis and costing. Assess effects of change on other requirements;
  - Change implementation. Modify requirements document and other documents to reflect change.

Requirements change management

- Deciding if a requirements change should be accepted
- Problem analysis and change specification: During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
- Change analysis and costing: The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change. Change implementation:
- The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented

Requirements change management


Identified problem
Problem analysis and change specification
Change analysis and costing
Change implementation
Revised requirements

Key points

- The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification and requirements management.
- Requirements elicitation and analysis is iterative involving domain understanding, requirements collection, classification, structuring, prioritisation and validation.
- Systems have multiple stakeholders with different requirements.

Key points

- Social and organisation factors influence system requirements.
- Requirements validation is concerned with checks for validity, consistency, completeness, realism and verifiability.
- Business changes inevitably lead to changing requirements.
- Requirements management includes planning and change management.

Software Metrics and Measures

- Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.
- Software Metrics provide measures for various aspects of software process and software product.
- Software measures are fundamental requirements of software engineering.
- They not only help to control the software development process but also aid to keep the quality of ultimate product excellent.
- According to Tom DeMarco, a (Software Engineer), “You cannot control what you cannot measure.” By his saying, it is very clear how important software measures are.

some software metrics

- **Size Metrics -** Lines of Code (LOC) (), mostly calculated in thousands of
- delivered source code lines, denoted as KLOC.
- • Function Point Count is measure of the functionality provided by the software.
- Function Point count defines the size of functional aspect of the software.
- **Complexity Metrics -** McCabe’s Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph

- **Quality Metrics -** Defects, their types and causes, consequence, intensity of severity and their implications define the quality of the product.
- The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client end, define quality Of the product.
- **Process Metrics -** In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.
- **Resource Metrics -** Effort, time, and various resources used, represents metrics for resource measurement.

Summery

Requirements engineering is the process of developing a software specification

- Requirements engineering process:

1. **Feasibility study**: Is it technically and financially feasible to build the system?
2. **Requirements elicitation and analysis**: what do the system stakeholders require or expect from the system?
3. **Requirements specification:** defining the requirements in detail
4. **Requirements validation**

Software Project Management

Chapter 3