1. The Requirements Engineering Process

It is the process of establishing what services are required and the constraints on the system's operation and development. It seeks to map out the WHAT in software development.

Requirements engineering is the process of developing a software specification. The requirements engineering process:

- Feasibility study
- Requirements Elicitation and Analysis
- Requirements Specification
- Requirements Validation

Requirements engineering forms the FOUNDATION of the software product. Incorrect requirements might lead to:

- Increased cost of development
- Delayed release of the product

1.1 Feasibility Study

A feasibility study is carried out whenever there is complex problem or opportunity. It is undertaken to determine the possibility or probability of either improving the existing system or developing completely the new system.

It assesses whether the system is useful to the business and aims to answer these questions:

- Does the system contribute to the overall objectives of the organisation?
- Can the system be implemented using current technology and within given cost and scheduling constraints?
- Can the system be integrated with other systems which are already in place?

1.1.1 Economic Feasibility

This first seeks to identify alternatives, and then determines the saving and expected cost of each alternative.

It considers the one-time cost e.g. cost for converting present system to new system, buying software, etc and the recurring costs e.g. maintenance and perform as a Return On Investment (ROI) analysis:

ROI = net earnings/ total investment

; a negative ROI indicated losses incurred and a positive one would mean that one might consider to go ahead with the project

1.1.2 Operational Feasibility

This seeks to answer questions like:

- Will the system be useful after it is implemented?
- Will there be resistance from users?

1.1.3 Technical Feasibility

- Can the work for the project be done with the present equipment, current procedures, existing software technology, and available personnel?
- If new technology is needed, what alternatives will be needed in the present structure and work ethos?
- Is the available technology adequate?
- Is the hardware available adequate?

The result is a **feasibility study report** recommending whether or not it is worth carrying on with the project.

The feasibility report consists of:

- The Statement of the problem
- Details of findings
- Findings and recommendations in concise form

1.2 Requirements Elicitation and Analysis

1.2.1 Requirements Elicitation

Requirements Elicitation involves identifying **WHAT** is needed from the system **NOT HOW** the system will achieve its goals.

It is a process in through which one is able to discover, reveal, articulate, and understand the problem to be solved, the system services, the required performance of the system, hardware constraints, etc.

Software Engineers work with customers and system end users to find out the user's needs; the objectives for the system, what is to be accomplished and how the system fits into the business needs.

1.2.1.1 The Requirements Elicitation Process

- Establish objectives - Check with business goals, problem to be solved, system constraints
- Understand Background - Look at the organisational structure, Application domain, and Existing systems
- Organise knowledge - This entails stakeholder identification, goal prioritisation and domain knowledge filtering
- Collect Requirements -These might be stakeholder requirements, domain requirements, organisation requirements

1.2.1.2 Requirements Elicitation Guidelines

The following are a set of suggested guidelines for requirements elicitation:

- Assess the business and technical feasibility for the proposed system
- Identify the people who will help to specify requirements.
- Define technical environment (os, system architecture, telecommunications system) into which the system will be placed.
- Identify "domain constraints" (characteristics of the business environment specific to the application domain).

- Define one or more requirement elicitation methods (interviews, meetings)
- Solicit participation from many people so that requirements are defined from different points of view.

1.2.1.3 Elicitation Techniques

On identifying requirements sources, the software engineer can start eliciting requirements from them. A number of techniques exist for doing this, the principal ones being:

1. Interviews

This is a "traditional" means of eliciting requirements. System analysts collect information from individuals through face to face interaction with the potential user of the system. It is useful for:

- finding facts,
- clarifying facts,
- verifying facts,
- getting the end user involved,
- identifying requirements,
- soliciting ideas and opinions.

There are two types of interviews:

- **Structured**: agenda is preset; there is a clear interview purpose and scope and questions are determined beforehand.
- **Unstructured**: also called '*open-ended*' there is no preset agenda, notes or questionnaires; might involve a face-to-face dialogue

Interview Question guidelines

1. Establish rapport with the interviewee; put them at ease.
2. Use clear and concise language; preferably day to day language instead of jargon or technical terms
3. Don't include your opinion as part of the question.
4. Avoid biased questions.
5. Avoid long or complex questions.
6. Be patient and courteous.
7. Listen and endeavour to keep the interviewee at ease

Advantages

- They are simple and quick.
- They are versatile and the analyst has a free hand and he can probe in depth and adapt follow up questions hence extracting almost all the information from the concerned people.

Disadvantages

- They are time consuming for both the interviewer and interviewee.
- They are inefficient where large quantities of repetitive data are required.
- Maybe inadequate where information is highly complex and/or technical.

- They should be planned well in advance and require skill.
- Large amounts of data can be hard to analyse.

Sample interview questions

- Who is the user?
- Who is the customer?
- Are their needs different?
- Where else can a solution to this problem be found?

2. Documentation Review

The purpose of documentation review is to understand what needs to be done and it is used as basis for further study, business process definition, interviews, etc

Examples of sources: company reports, organisation charts, policy manuals, job descriptions, documentation of existing systems, Request for Proposal, statement of work, contract/task order, proposal, and standards.

Documentation review Guidelines

- Determine what is right and wrong with the current business process or products
- Identify opportunities to reuse existing products or information
- Extract requirements from many diverse sources into a consolidated requirements list

Advantages

- It helps the analyst to get an unbiased understanding of the organization before meeting the people who work there.
- It is a good ground on which to prepare for other types of fact finding e.g. by being aware of the business objectives of the organization.
- It can give you detailed insight into requirements for the current system.

Disadvantages

- Written documents rarely exactly match up to reality.
- Can be long-winded with much irrelevant detail.
- May feel intrusive to the stakeholders.
- May need to be confirmed by reference to primary/secondary sources

Appropriate

- Whenever one is unfamiliar with the organization being investigated.

3. Questionnaires (and Surveys)

These special-purpose documents are used by the analyst to collect information and opinions from respondents. There are two types of questionnaires:

A. Open Ended questionnaire - Designed to offer the respondent greater flexibility in answering. It provides spaces after question for respondent to record the answer.

B. Closed ended questionnaire – Questions are presented such that they require selecting an answer from pre-defined available responses. This type eliminates subjectivity of open ended questionnaires.

Advantages

- These are suitable where similar data is required from a vast and diverse number of sources.
- They are relatively cheap
- They provide flexibility in that the respondent can fill them it at their own leisure.
- They provide a written record
- Because they can be answered anonymously, the possibility of getting honest responses is high.
- They can be administered remotely
- One is able to quickly collect information from large numbers of people.

Disadvantages

- Questionnaires may be hard to interpret and lead to false responses.
- They have a low reliability
- They might be rigid, providing no room for users to convey their real needs.

When using this technique one needs to be mindful of the following:

- Possible bias in the sample selection
- Possible bias in self selecting respondents
- Small sample sizes that will have no statistical significance
- Open ended questionnaires posing challenges in analysis
- Ambiguous questions

4. Brainstorming

This is group technique for generating ideas and involves both idea generation and idea reduction. It allows people to suggest and explore ideas in an atmosphere free of criticism and judgment and overcomes cognitive limitations and communication barriers.

The group sessions should be made of 4 to 10 people including a leader whose role is to get the session started.

Brainstorming sessions have two phases:

- Generation phase – As many ideas as possible are offered BUT there is no discussion of their merits. '*a.k.a. idea generation.'*
- Consolidation phase – The ideas previously offered are discussed, revised, and organized. '*a.k.a. idea reduction'*

Brain storming Guidelines

- Use in a facilitated workshop as it will require order.
- All ideas are good; do not evaluate, debate or criticize during the idea generation phase.
- Do not be bounded by what is possible; keep an open mind
- Attempt to produce lots of ideas, novel ideas
- Piggyback on others' ideas

Brainstorming Session Participant Roles

- **Leader/Facilitator**-This person needs to be patient and a good listener. He/she is the moderator of the session.

- **Scribe**-This person is in charge of writing down EVERY idea clearly and where everyone in the group can see them and
- **team member**- participants

Advantages

- It provides a more natural reaction than formal interviews.
- It stimulates imaginative thinking and 'out-of-the-box' thinking
- It helps to avoid tendency to focus too narrowly too soon
- It provides more comfortable social setting
- It is easy to learn; and gives very little overhead
- It encourages participation by all parties present.
- It allows participants to "piggyback" on one another's ideas.
- It has high bandwidth. Many ideas can be generated in a short period of time.

Disadvantages

- They may be uncomfortable for participants and therefore create unnatural groups.
- It may not produce the same quality and level of detail as some other processes since it is un-facilitated and relatively unstructured
- There is a danger of peer influence
- Responses to technical are more likely to be superficial
- Would require a highly trained facilitator.

When using this technique, one should watch for:

- Sample bias and try as much as possible to neutralize the group
- Dominance and submission tendencies in some participants
- Adequacy in the number of ideas generated

5. Storyboarding

A Storyboard is a logical and conceptual description of system functionality for a specific scenario, including the interaction required between the system users and the system.

It is a series of illustrations and images displayed in sequence for the purpose of pre-visualizing requirements. It "tells a specific story".

The purpose of storyboarding is to elicit early "Yes, But" reactions.

They identify the players, explain what happens to them, and describes how it happens. It is therefore wise to storyboard early and often on projects with new or innovative content.

A storyboard should be sketchy, easy to modify, and un-shippable.

Types of storyboards

a) Passive storyboards

   These tell a story to the user and consist of sketches, pictures, screen shots, PowerPoint presentations, or sample application outputs. They walk the user through, with a "When you do this, this happens" explanation. Consist of sketches, screen shots and pictures

b) Active storyboards

These try to make the user see “a movie that hasn't actually been produced yet” providing an automated description of the way the system behaves in a typical usage or operational scenario.

c) Interactive storyboards

Let the user experience the system in as realistic a manner as practical and thus require participation by the user.

Advantages

- This technique enables one to understand how requirements impact implementation and testing.
- It describes how a system or part of a system is intended to work “before the fact”
- It documents system functionality
- It help to communicate and verify functionality with relevant stakeholders who may not be technical or programming savvy.
- GUI storyboards help business users to identify issues and gaps early, minimizing the cost of rework.
- They go a long way generating UI specs, functional specs and detailed test scripts saves hundreds of man hours.
- Because storyboards exist independently of the software system they describe, they have many advantages over regular prototypes e.g. do not give the false impression that the system is already developed, feedback is easier to accommodate, they can not crash etc

Disadvantages

- One of the biggest problems with storyboards is that they can become outdated very quickly. User interfaces originally defined often change over time, and that creates a maintenance burden.

6. JAD Sessions

Developed at IBM in the late 1970’s JAD (Joint Application Development) is a technique for promoting co-operation, understanding and teamwork among clients and developers. It facilitates the creation of a shared vision of what the system should be.

The following principles are used along with this technique:

- Group Dynamics – workshops are preferable compared to interviews. However participants should be chosen carefully.
- Visual Aids – Use lots of visualization media, e.g. wall charts, large monitors, graphical interfaces as a stimulus for response.
- Organized, Rational Process – encompass techniques like brainstorming and top-down analysis
- WYSIWYG Documentation Approach- each JAD session results in a document which is easy to understand and is created and agreed upon during the session

JAD Steps

A. Project definition

The JAD facilitator interviews managers and clients to determine objectives and scope and forms a team of users, clients and developers: all stakeholders are represented; participants are able to make binding decisions

B. Research
The JAD facilitator interviews present and future users, gathers domain info and describes work flows. He also starts a list of issues to be addressed during the session

C. Preparation
The working document is created and this is the first draft of the final document; the session agenda is made along with overheads, flip charts etc. to represent information gathered during session

D. *Research session*
The teams are guided in creating the system specification and then they also define and agree on work flow, data elements, screens and reports.

E. *Final document*
The JAD facilitator prepares the Final Document from the draft and decisions made during step 4; the document is then distributed to session participants for review and they meet to discuss the reviews and finalize the document.

7. Scenarios
This is a valuable means for providing context to the elicitation and clarification of user requirements. They allow the software engineer to provide a framework for questions about user tasks by permitting "what if" and "how is this done" questions to be asked. The most common type of scenario is the use case.
Scenarios describe how a user interacts with a system and discovering scenarios exposes possible system interactions and reveals system facilities which may be required.
For complex systems, a fairly large number of scenarios will usually be required.

a. Use Cases
A use case in software engineering and systems engineering is a description of a system's behaviour as it responds to a request that originates from outside of that system. Use Cases, like storyboards, identify the 'who and what' of system behaviour and describe the interactions between a user and a system, focusing on what they system "does" for the user.

Use Case Diagram: A use case diagram provides an overview over the use cases of a system and who is using the functionality.

Detailed Use Case Description: A detailed use case description describes the interaction between the user and the system as a set of scenarios

Strengths
- The Use Case model describes the totality of the systems functional behaviour.
- Help manage complexity of a system as functionally is broken down.
- Focus on real user needs
- Provides groundwork for generating user manual and test cases

Weaknesses
- Use cases leave out a system's non-functional requirements and quality attributes.
- They do not define user interfaces

- Use cases leave out application architecture

b. User Stories

User stories have been introduced with Extreme Programming. They are very close to the concept of use cases, but also are different. User stories fulfil the same purpose as use cases, i.e. keeping track of the requirements of the system. However, user stories differ from use cases, as they focus on one feature of the software. A feature can be a functional requirement of the system, but also a non-functional requirement.

Note: With use cases, the focus is on the functionality of the system and not on the non-functional aspects. Thus a use case mainly represents functional requirements, while a user story can represent both, a functional and a non-functional requirement.

How the user story works, is by providing a story of how a user uses the system.

E.g.

1. "As a customer, I would like to book and plan a single flight from Copenhagen to Paris". – Functional requirement
2. "As a customer, within five seconds I would like have a list of all flights from Copenhagen to Paris that start on a given date." – Non-functional requirement

User story guidelines

- Approximately 3 sentence description of what a software feature should do
- Written in the customer's language often on an index card
- Should only provide enough detail to make a low-risk time estimate (a few days).
- They typically take Role-Goal-Benefit form:

  "As a <ROLE>, I want to <GOAL> in order to <BENEFIT>"
  E.g. As a **student**, I need to **login** to Piazza in order **to finish the assignment**

- The user story may also:
  - Specify acceptance or completion criteria that define what is meant for this feature to be DONE
  - Provides estimate of the required effort (story points)

Examples of User Stories

- As a user, I want to search for contacts so I can message them.
- As a customer, I want to search for product items so I can buy them.
- As an employer, I want to post a job on the website so people can apply for it.

NOT User Stories

- Implement contact list view ContactListView.java
- Define the product table database schema
- Automate the job posting algorithm

More Detailed examples of user stories

US1
As a job company, I can use my credit card so I can pay for postings.

Note: Accept Visa, MasterCard, American Express.

Test: (on the back of the story card)
Test with Visa, MasterCard and American Express (pass)
Test with Diner's Club (fail)
Test with good, bad and missing card ID numbers.
Test with expired cards.
Test with over $100 and under $100.

US2
As a Creator, I want to upload a video from my local machine so that any users can view it.

Note: N/A

Test:
Click the "Upload" button.
Specify a video file to upload.
Check that .flv, .mov, .mp4, .avi, and .mpg extensions are supported.
Check that other filetypes aren't able to be uploaded.
Check that files larger than 100MB results in an error.
Check that movies longer than10 mins result in an error.
Click "Upload Video".
Check that progress is displayed in real time.

INVEST – The making of a good user story
Independent, **N**egotiable, **V**aluable to Users or Purchasers, **E**stimable, **S**mall, **T**estable.

Independent
- Little dependence between stories
- Keeps development flexible
- Makes estimation easier

Negotiable
- Details are negotiated between developers and users
- Cards become reminders of what has been negotiated

Valuable to Users or Purchasers
Customer:
  - Purchaser: person who pays for the software. An administrator can determine how many people used the software in the last week.
  - User: person who uses the software A user can search for people by profile attributes

Makes sure customer can estimate value of a story. (Stories should not be valuable to the

Developers) E.g. The backend database will be MySQL (no value to user; just restricts your options)

Estimable

- A developer should be able to estimate how long a story should take to complete
- Helps in planning

Problems with estimation

- Lack of domain knowledge - ask customer
- Lack of technical knowledge - brush up on tech.
- Story is too big - break it up

Small

Stories should be "just the right size". As a rule-of-thumb: half a day to several days to implement.

- Too big a story might mean inaccurate estimate and no value may be delivered until story is complete. The solution here is to break it up or compound it
- Too small a story means writing down the story may take longer than implementing it! Combine (staple) the stories together.

Testable

The story should have a test that goes with it to demonstrate that the story is implemented
Two types

- Automated e.g. => JUnit test
- Manual e.g. => An untrained user should be able to complete the wizard in less than two minutes

8. Observation

People often find it hard to describe what they do because it is so natural to them. Therefore the best way to understand it is to observe them at work. Ethnography is a technique from the social sciences which has proved to be valuable in understanding actual work processes:

a. Actual work processes often differ from formal, prescribed processes.

b. It involves an observer spending an extended period in an organisation, making detailed observation of all their practice, and building up a picture of how work is done.

Observation Guidelines

- Assume that people are good at doing their job and look for non-standard ways of working.
- Spend time getting to know the people and establish a trust relationship.
- Keep detailed notes of all work practices. Analyze them and draw conclusions from them.
- Combine observation with open-ended interviewing.
- Organize regular de-briefing session where the ethnographer talks with people outside the process.

- Combine ethnography with other elicitation techniques.

Advantages

- Observation has surface unarticulated procedures and criteria
- The technique is not biased by opinion
- It is a good way to understand the problem domain

Disadvantages

- The technique is time consuming
- It is likely that one's behaviour will change due to observation consciousness
- The technique is not representative of time period.
- The technique might be perceived as intrusive by the stakeholders.

9. Prototyping

They are a valuable tool for clarifying unclear requirements and obtaining feedback from stakeholders. They can act in a similar way to scenarios by providing users with a context within which they can better understand what information they need to provide.

A software requirements prototype is a partial implementation of a software system, built to help developers, users, and customers better understand system requirements.

Prototypes can be flat diagrams (often referred to as wireframes) or worsking applications using synthesized functionality. This helps to prevent confusion over the final visual look and feel of the application

Prototyping Styles

- Evolutionary or operational prototyping – The initial prototype is produced and refined through a number of stages to the final system. The last prototype is thus used as the starting point for the design of the production system.
- Throw-away or Exploratory prototyping – It is a practical implementation of the system and help to discover requirement problems. After the system requirements have been finalized, the prototype is discarded.
- Incremental prototyping - used to build the final product as a separate prototype.

Prototyping Objectives

- To deliver a working system to end-users.
  The development starts with those requirements which are best understood.
- To validate or derive the system requirements.
  The prototyping process starts with those requirements which are poorly understood.
- To merge the separate prototypes in the overall design.

Uses of System Prototypes

- They help customers and developers understand the requirements for the system.
  - Requirements elicitation
    - To see how the system supports their work.
  - Requirements validation
    - To reveal errors and omissions in the requirements.

- It is a risk reduction activity which reduces requirements risks as a ‘version’ of the product can be experienced early in development.

Advantages

- Prototypes help users get an idea of what the system will look like, and make it easier for users to make design decisions without waiting for the system to be built.
- They help narrow the communication gap between clients and developers. Misunderstandings between software users and developers are exposed.
- Missing services may be detected and confusing services may be identified.
- A working system is available early in the process.
- The prototype may serve as a basis for deriving a system specification.
- The system can support user training and system testing.

Disadvantages

- Managers, once they see a prototype, may have a hard time understanding that the finished design will not be produced for some time.
- Designers often feel compelled to use patched together prototype code in the real system, because they are afraid to 'waste time' starting again.
- Prototypes principally help with design decisions and user interface design. However, they can’t tell you what the requirements originally were.
- Designers and end users might focus too much on user interface design and too little on producing a system that serves the business process.
- Prototypes work well for user interfaces; screen layout and screen flow but are not so useful for batch or asynchronous processes which may involve complex database updates and/or calculations.

Application

- Systems where the user requirements are unclear, ambiguous or incomplete. Prototyping helps clarify the requirements.
- Systems with a major user interface and/or a lot of user interaction.
- Prototyping is a form a risk sharing between the user and the developer
- Where there is the much required strong commitment from the user(s) to evaluate and improve the prototype.

1.2.1.4 Challenges in Elicitation

- The boundary of the system may be ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.
- Ambiguous understanding of processes; Customers/users may not be completely sure of what is needed, and thus have trouble communicating needs.
- It is highly probable that there will be inconsistency within a single process by multiple users.
- Insufficient input from stakeholders as the clients might omit information that is believed to be “obvious”.
- Conflicting stakeholder interests might also lead to inconsistent requirements.
- There is a high possibility of changes in requirements after project has begun.

- People find it hard to describe knowledge they regularly use.
- The process requires collaborations of people with different backgrounds. This implies that given a variety of stakeholders and a number of requirements elicitation techniques, analysts should identify a set of appropriate techniques to elicit requirements proactively.

1.3 Requirements Specification

Everyone knew exactly what had to be done until someone wrote it down!!!???!!!

The goal of requirements specification is to provide a representation of the software for the customer's review and approval. The specification is thus developed as a joint effort between the developer and the customer and serves as a basis for review for both customer and developer.
It is a culmination of requirements analysis and it describes WHAT is desired rather than HOW it is to be realized.

1.3.1 Requirements Specification Principles

1. Functionality should be separated from implementation. Requirements should be expressed in WHAT form rather than HOW.
2. In dynamic systems, use a process oriented specification to specify the model of desired behavior in terms of functional responses to various environmental stimuli
3. If the system is only a component of a larger system, specification MUST encompass the system of which the software is a component.
4. The environment in which the system operates and interacts with must also be included in the specification.
5. System must be described as perceived by the user community, rather than a design or implementation model.
6. Specification must be complete & formal enough to determine if the proposed implementation satisfies the specification
7. No specification can ever be 100% complete and thus it should be able to adapt to change.

There are different ways of **specifying requirements** for **different readers**:

Requirements Definition

- A statement in natural language plus simple diagrams of the services the system provides and its operational constraints. It is an informal outline of requirements.
- Complete listing of everything the customer expects the proposed system to do
- Primarily for customer's understanding (Written for customers)

Requirement Specification

- A structured document setting out detailed descriptions of the system services.
- Restates the requirements in technical terms appropriate for development of system design
- Written **as a contract** between client and contractor, for system architects, and software developers

1.3.2 Software Specification

The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an e-Commerce Web site, and so on) must provide, as well as any required constraints by which the system must abide.
It also functions as a blueprint for completing a project with as little cost growth as possible.

Note: The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

The SRS contains **functional and nonfunctional requirements only**; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behaviour necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.
- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. As mentioned previously, the SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.
- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

1. **Introduction**
   - 1.1 Purpose of the requirements document
   - 1.2 Scope of the product
   - 1.3 Definitions, acronyms and abbreviations
   - 1.4 References
   - 1.5 Overview of the remainder of the document
2. **General description**
   - 2.1 Product perspective
   - 2.2 Product functions
   - 2.3 User characteristics
   - 2.4 General constraints
   - 2.5 Assumptions and dependencies
3. **Specific requirements** cover functional, non-functional and interface requirements. This is obviously the most substantial part of the document but because of the wide variability in organisational practice, it is not appropriate to define a standard structure for this section. The requirements may document external interfaces, describe system functionality and performance, specify logical database requirements, design constraints, emergent system properties and quality characteristics.
4. **Appendices**
5. **Index**

1.3.3 Formal specification

Formal specification is part of a more general collection of techniques that are known as 'formal methods'. Formal specification techniques are techniques for the unambiguous specification of software.

Note: Formal methods have limited practical applicability but their principal benefits are in *reducing the number of errors in systems*. For this reason their main area of applicability is critical systems where the use of formal methods is most likely to be cost-effective.

They are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.

Types of formal specification techniques/approaches include:

1. Algebraic approach
   The system is specified in terms of its operations and their relationships.
2. Model-based approach
   The system is specified in terms of a state model that is constructed using mathematical constructs such as sets and sequences. Operations are defined by modifications to the system's state.

Table 1: Formal Specification Languages

	Sequential	**Concurrent**
Algebraic	Larch (Guttag, Horning et al., 1985; Guttag, Horning et al., 1993), OBJ (Futatsugi, Goguen et al., 1985)	Lotos (Bolognesi and Brinksma, 1987),
Model-based	Z (Spivey, 1992) VDM (Jones, 1980) B (Wordsworth, 1996)	CSP (Hoare, 1985) Petri Nets (Peterson, 1981)

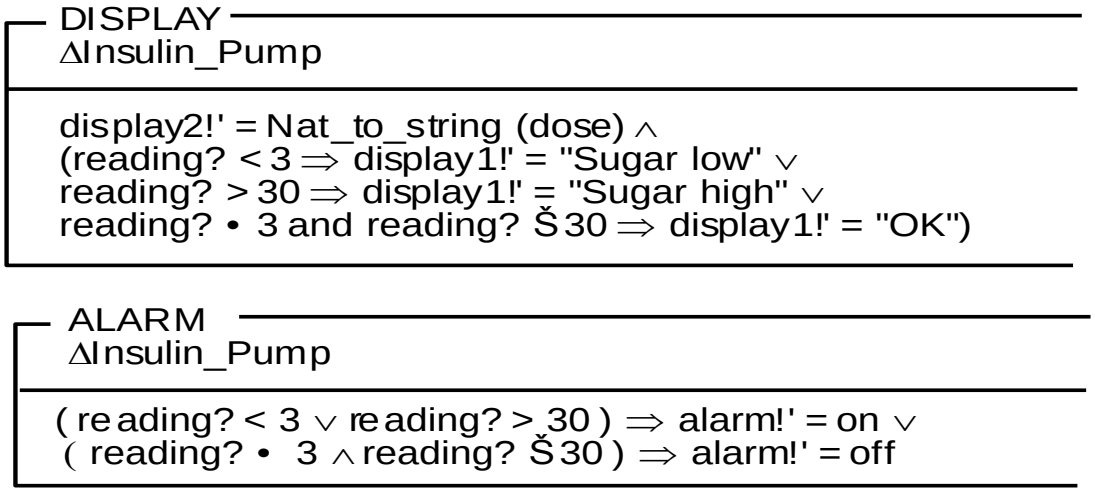

Figure 1: Z schema for an Insuline Pump (Output schema)

Benefits of Formal specification

- Formal specification involves investing more effort in the early phases of software development. This reduces requirements errors as it forces a detailed analysis of the requirements.
- Incompleteness and inconsistencies can be discovered and resolved.
- Savings as made as the amount of rework due to requirements problems is reduced.

Weaknesses of Formal Methods

- **Expense:** Because of the rigor involved, formal methods are always going to be more expensive than traditional approaches to engineering. In general, formal methods involve a large initial cost followed by less consumption as the project progresses; this is a reverse from the normal cost model for software development.
- **Limits of Computational Models:** While not a universal problem, most formal methods introduce some form of computational model, usually hamstringing the operations allowed in order to make the notation elegant and the system provable.
- **Usability:** While an all-encompassing formal description is attractive from a theoretical perspective, it invariably involved developing an incredibly complex and nuanced description language, which returns to the difficulties of natural language. Case studies of full formal methods often acknowledge the need for a less all-encompassing approach.

1.4 Requirements Validation

This activity is concerned with demonstrating that the requirements define the system that the customer really wants. It draws its importance from the fact that error costs (fixing requirements error after product delivery) can be so high.

Requirements validation examines the specification to ensure that all system requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product.

The primary method for requirements validation is reviews involving client and contractor staff. Prototyping is also important technique of requirements validation.

1.4.1 Requirements Validation Criteria

Correctness

The requirements should represent the client's view. Does the system provide the functions which best support the customer's needs?

Completeness

All possible scenarios, in which the system can be used, are described, including exceptional behavior by the user or the system. Are all functions required by the customer included?

Consistency

The functional or nonfunctional requirements should not contradict each other. Are there any requirements conflicts?

Realism

Requirements should be implementable and deliverable. Can the requirements be implemented given available budget and technology?

Traceability

Each system function should be traced to a corresponding set of functional requirements.

Modular: Changes occur inevitably. Can the requirements accommodate change?

1.5 Requirements Evolution

There is need to plan for change because:

- Requirements validation may lead to evolution kicking in to accommodate any changes that might have been resolved.
- Requirements also evolve often as better understanding of user needs is devolved and as organization objectives change.