UNIT – 2 REQUIREMENTS ANALYSIS AND SPECFCIFICATION

Software Requirements: Functional and Non-Functional, User requirements, Software Requirements Document – Requirement Engineering Process: Feasibility Studies, Requirements elicitation and analysis, requirements validation, requirements management- Classical analysis: Structured system Analysis, Petri Nets.

IEEE defines Requirement as :

1. A condition or capability needed by a user to solve a problem or achieve an objective
2. A condition or capability that must be met or possessed by a system or a system component to satisfy standard, specification or formally imposed document
3. A documented representation of a condition nor capability as in 1 or 2

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories:

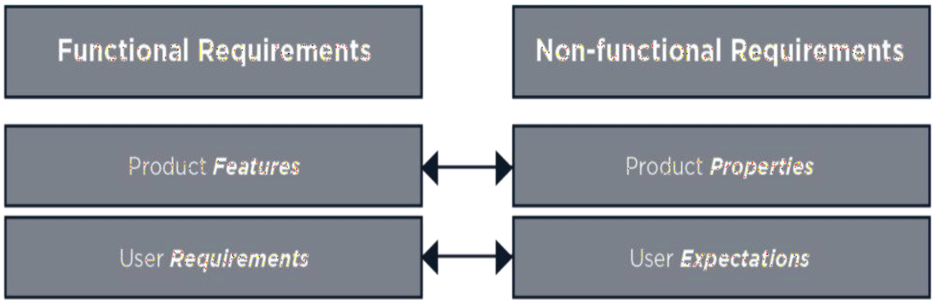
Functional Requirements

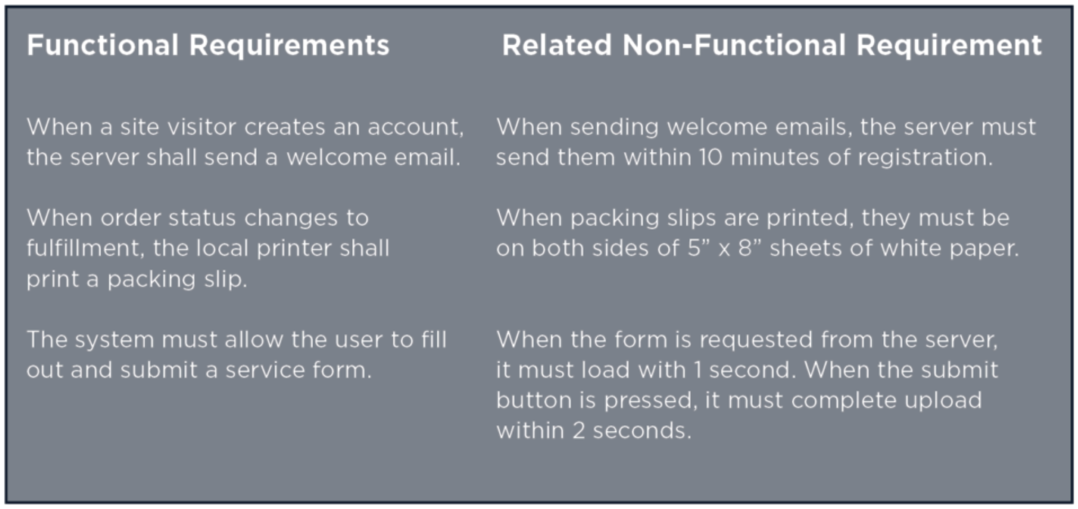
Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

Examples -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should follow business rules and administrative functions.
- Software is developed keeping downward compatibility intact.





Functional Requirements	Related Non-Functional Requirement
When a site visitor creates an account, the server shall send a welcome email.	When sending welcome emails, the server must send them within 10 minutes of registration.
When order status changes to fulfillment, the local printer shall print a packing slip.	When packing slips are printed, they must be on both sides of 5" x 8" sheets of white paper.
The system must allow the user to fill out and submit a service form.	When the form is requested from the server, it must load with 1 second. When the submit button is pressed, it must complete upload within 2 seconds.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

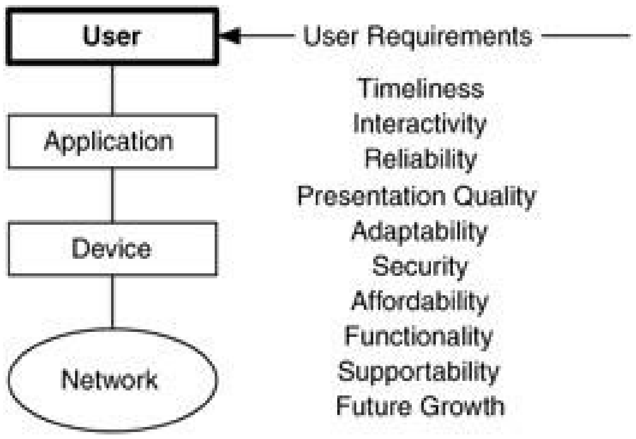
Non-functional requirements include -

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

User Interface requirements

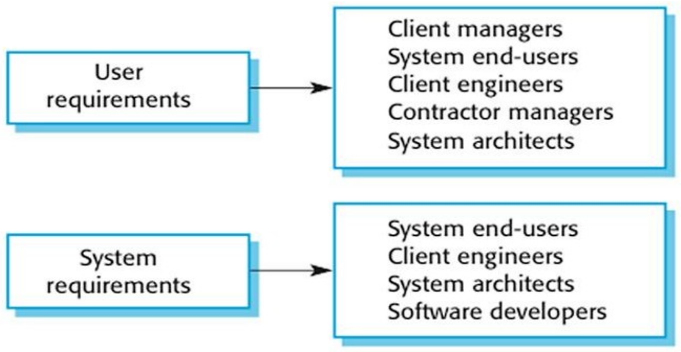
UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- 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 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.



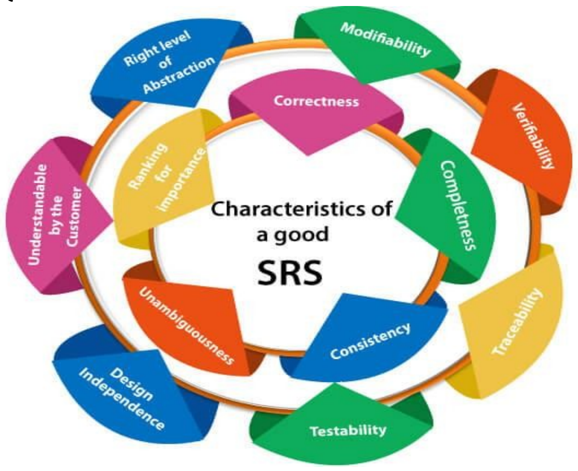
Software Requirements Document

What is Software Requirement Specification - [SRS]?

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

In other words, the software requirements document (SRD) describes the business or organization's understanding of the end user's needs and dependencies as well as any constraints on the system.

Qualities of SRS:



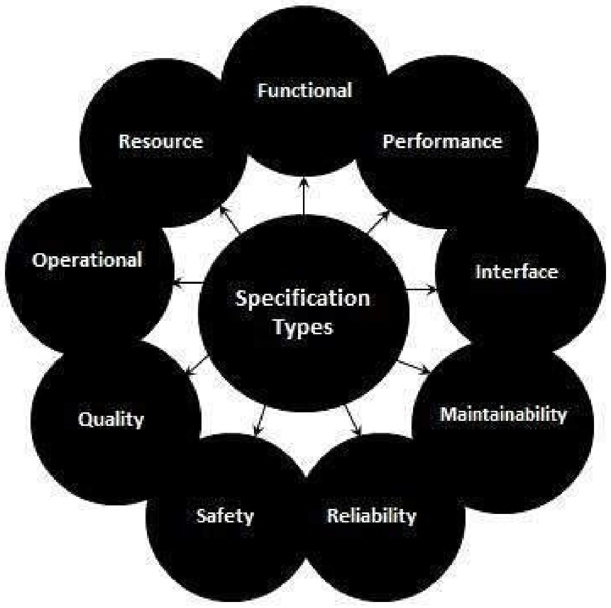
- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Benefits of SRS:

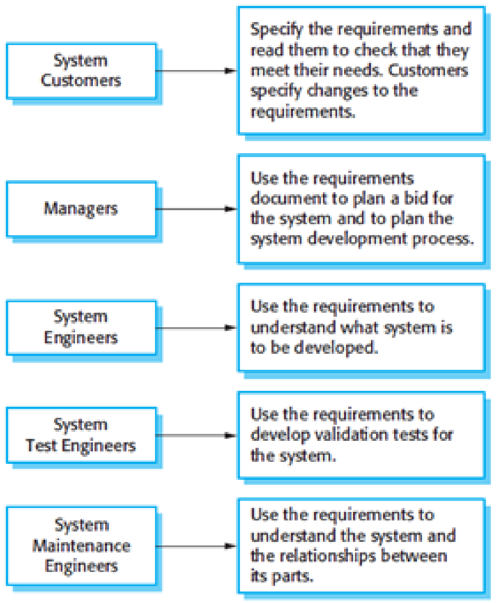
- Establish the basis for agreement between the customers and suppliers on what the software product is to do.
- Complete description of functions
- Reduce the development effort
- Provide a basis for estimating cost and schedule
- Serve as a basis for enhancement

Types of Requirements:

The below diagram depicts the various types of requirements that are captured during SRS.



SRS Users



Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.

Use the requirements document to plan a bid for the system and to plan the system development process.

Use the requirements to understand what system is to be developed.

Use the requirements to develop validation tests for the system.

Use the requirements to understand the system and the relationships between its parts.

What is included in an SRD?

While the SRD functions as a blueprint for managing the scope of a project, it ultimately only defines functional and nonfunctional requirements for a system. The document does not outline design or technology solutions. Those decisions are made later by the developers.

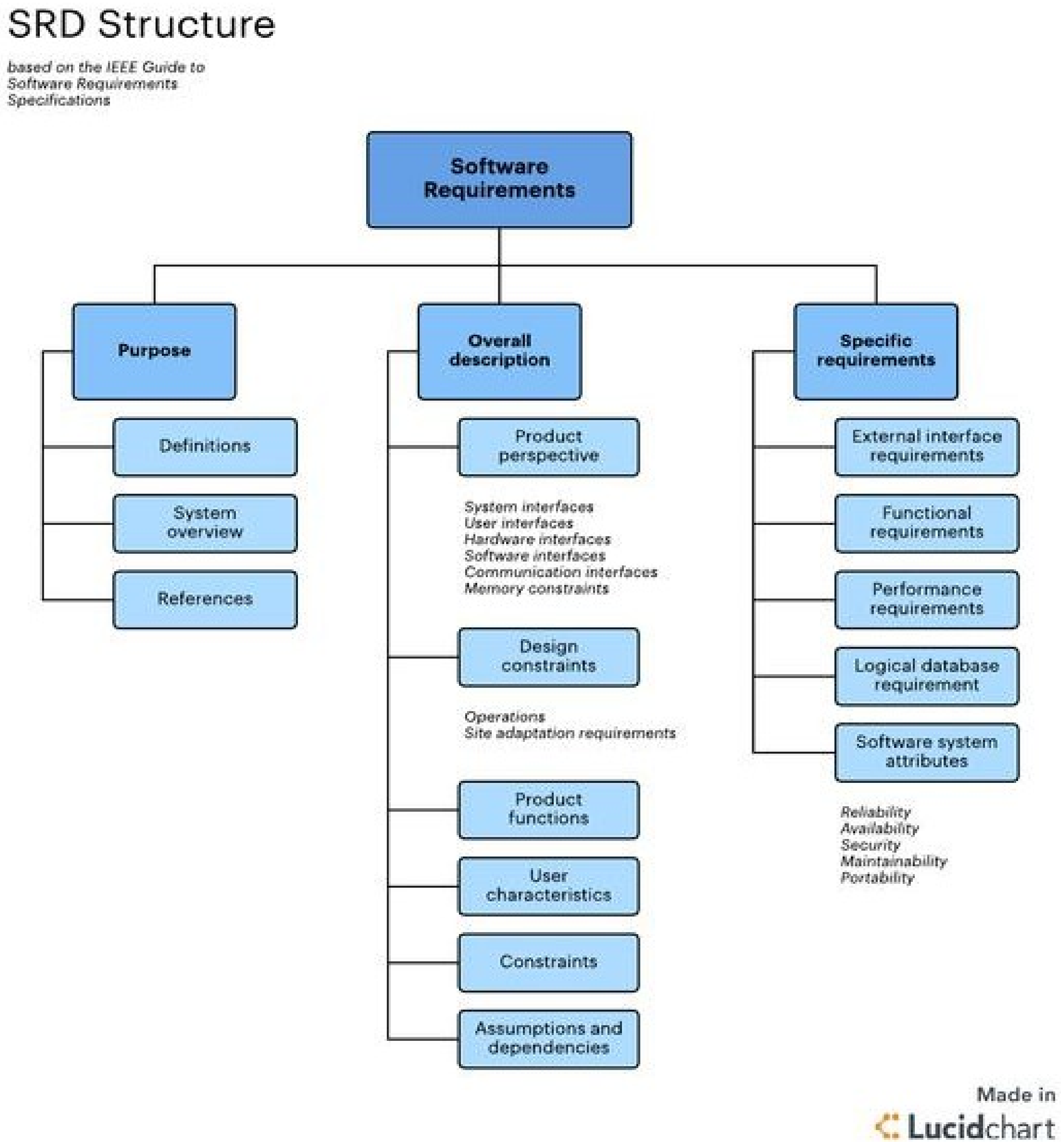
A well-written SRD should:

- Break the problem into manageable parts.
- Serve as a reference for testing and validation.
- Inform the design specifications (i.e., the SRD needs to include sufficient information on the requirements of the software in order to render an effective design).
- Provide feedback to the client (end user).

The SRD demonstrates to the client that your organization understands the issue they want to be solved and how to address those problems through software solutions. Because clients are often direct stakeholders, it is especially important to draft the documentation clearly in simple terms.

Again, how you write your SRD will depend on the approach and methodology your team or organization subscribes to. However, the IEEE standards organization recommends typical SRDs should cover the following topics:

- Interfaces
- Functional Capabilities
- Performance Levels
- Data Structures/Elements
- Safety
- Reliability
- Security/Privacy
- Quality
- Constraints and Limitations


SRD Structure
based on the IEEE Guide to Software Requirements Specifications
Software Requirements
Purpose
Definitions
System overview
References
Overall description
Product perspective
System interfaces
User interfaces
Hardware interfaces
Software interfaces
Communication interfaces
Memory constraints
Design constraints
Operations
Site adaptation requirements
Product functions
User characteristics
Constraints
Assumptions and dependencies
Specific requirements
External interface requirements
Functional requirements
Performance requirements
Logical database requirement
Software system attributes
Reliability
Availability
Security
Maintainability
Portability
Made in Lucidchart

IEEE requirements standard

Defines a generic structure for a requirements document that must be instantiated for each specific system.

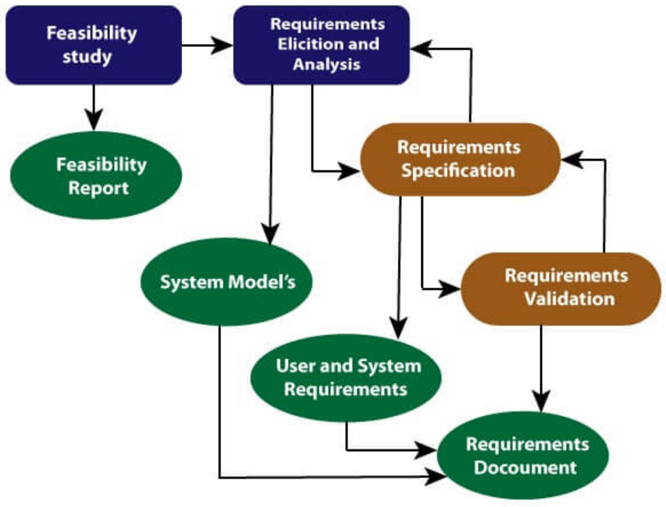
– Introduction.
– General description.
– Specific requirements.
– Appendices.
– Index.

IEEE requirements standard

1.Introduction
1.1 Purpose
1.2 Scope
1.3 Definitions, Acronyms and Abbreviations
1.4 References
1.5 Overview
2. General description
2.1 Product perspective
2.2 Product function summary
2.3 User characteristics
2.4 General constraints
2.5 Assumptions and dependencies
3. Specific Requirements
Functional requirement
-External interface requirements
- Performance requirements
- Design constraints
- Attributes eg. security, availability, maintainability, transferability/conversion

- Other requirements
• Appendices
• Index

Requirement Engineering Process



Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Elicitation and Analysis
- Requirement Validation
- Requirement Management

Feasibility study

Starting point of the requirements engineering process

• Input: Set of preliminary business requirements, an outline description of the system and how the system is intended to support business processes

• Output: Feasibility report that recommends whether or not it is worth carrying out further Feasibility report answers a number of questions:

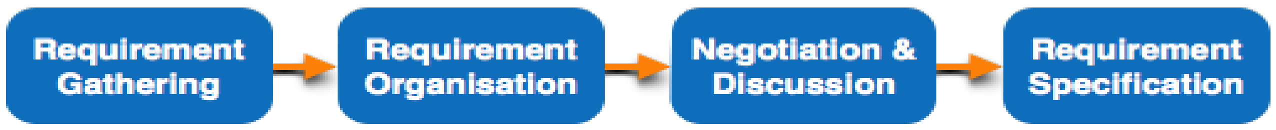
1. Does the system contribute to the overall objective
2. Can the system be implemented using the current technology and within given cost and schedule
3. Can the system be integrated with other system which are already in place.

Requirement Elicitation and Analysis

This is also known as the gathering of requirements. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc.

Requirement elicitation process can be depicted using the following diagram:



- 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.

Requirements validation

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.
In the requirements validation process, we perform a different type of test to check the requirements mentioned in the Software Requirements Specification (SRS), these checks include:

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

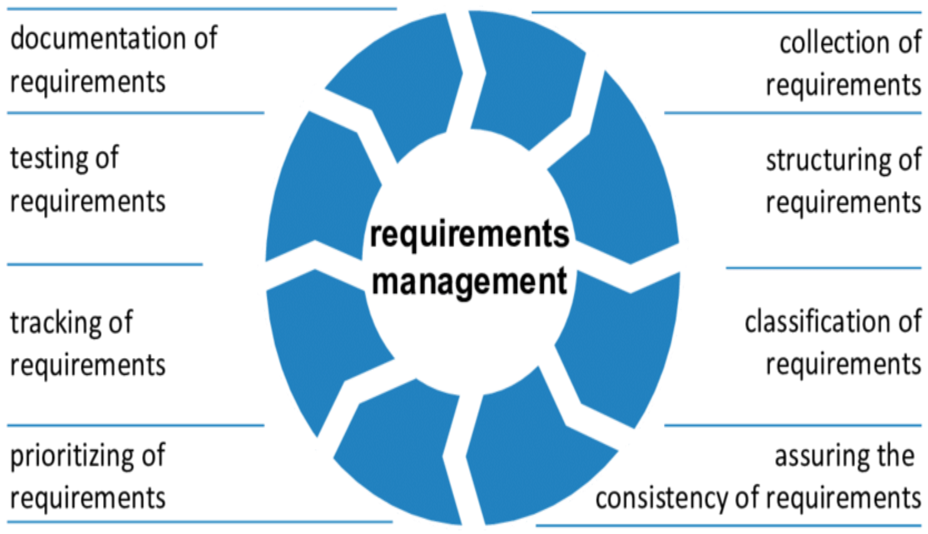
Requirements can be checked against following conditions -

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

Requirements Validation Techniques

- o Requirements reviews/inspections: systematic manual analysis of the requirements.
- o Prototyping: Using an executable model of the system to check requirements.
- o Test-case generation: Developing tests for requirements to check testability.
- o Automated consistency analysis: checking for the consistency of structured requirements descriptions.

Requirements management


documentation of requirements
collection of requirements
testing of requirements
structuring of requirements
requirements management
tracking of requirements
classification of requirements
prioritizing of requirements
assuring the consistency of requirements

Requirements are likely to change for large software systems and as such requirements management process is required to handle changes.

Reasons for requirements changes
(a) Diverse Users community where users have different requirements and priorities
(b) System customers and end users are different
(c) Change in the business and technical environment after installation

Two classes of requirements
(a) Enduring requirements: Relatively stable requirements
(b) Volatile requirements: Likely to change during system development process or during operation

Requirements management planning
An essential first stage in requirement management process. Planning process consists of the following
1. Requirements identification -- Each requirement must have unique tag for cross reference and traceability
2. Change management process -- Set of activities that assess the impact and cost of changes
3. Traceability policy -- A matrix showing links between requirements and other elements of software development
4. CASE tool support --Automatic tool to improve efficiency of change management process. Automated tools are required for requirements storage,change management and traceability management

Traceability

Maintains three types of traceability information.

1. Source traceability--Links the requirements to the stakeholders
2. Requirements traceability--Links dependent requirements within the requirements document
3. Design traceability-- Links from the requirements to the design module

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	

Requirements change management Consists of

three principal stages:

1. Problem analysis and change specification-- Process starts with a specific change proposal and analysed to verify that it is valid
2. Change analysis and costing--Impact analysis in terms of cost, time and risks
3. Change implementation--Carrying out the changes in requirements document, system design and its implementation

CLASSICAL ANALYSIS:

Products of Classical Analysis Workflow

- Specification document
- Software Project Management Plan

Specification document-detailed description of what system will do
Software Project Management Plan-will be the answers for What? How much? When?

Software Specification Methods

- Informal-enough for clients to understand
- Semi –Formal
  - Structured System Analysis
  - ER diagrams
- Formal-enough for developers and be contractually binding

Informal:

Informal Specifications Written in natural language. Easy for client to understand ,Easy to read and write. Informal specifications also have problems, Hard to be precise & detailed in natural languages, Makes finding incomplete or contradictory specifications difficult.

Structured Systems Analysis

Three graphical specification methods

- DeMarco
- Gane and Sarsen
- Yourdon

All are equivalent & equally good, Often used for commercial products

Gane and Sarsen's 9-step method. Many parts include stepwise refinement. This method among the most commonly used.

Sally's Software Shop Mini Case Study

> Sally's Software Shop buys software from various suppliers and sells it to the public. Popular software packages are kept in stock, but the rest must be ordered as required.Institutions and corporations are given credit facilities, as are some members of the public. Sally's Software Shop is doing well, with a monthly turnover of 300 packages at an average retail cost of $250 each. Despite her business success, Sally has been advised to computerize. \

Gane and Sarsen's method

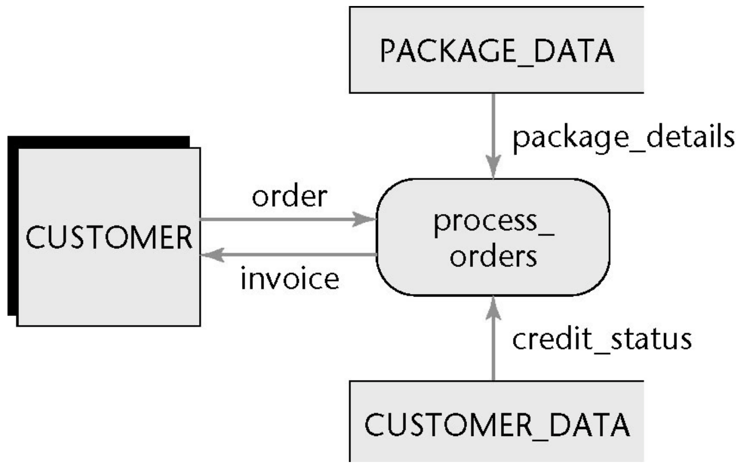
Nine-step method

Stepwise refinement is used in many steps

Step 1: Draw the DFD

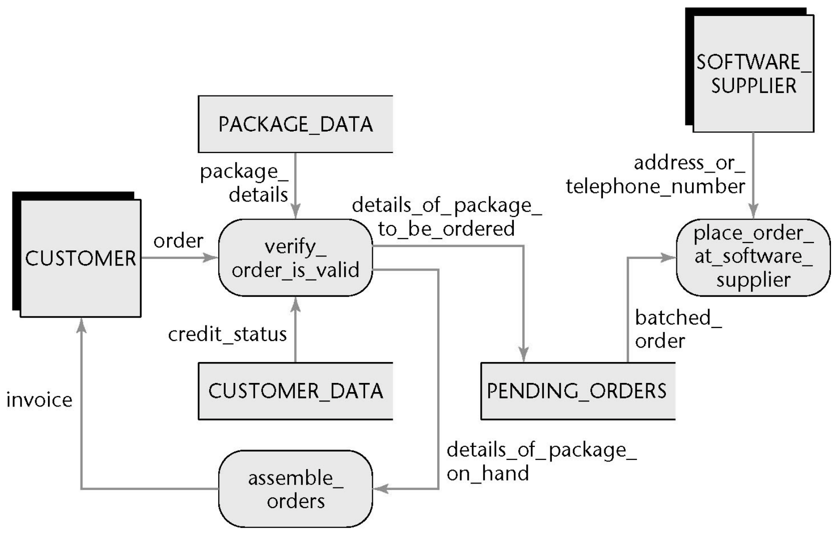
First refinement:

Infinite number of possible interpretations

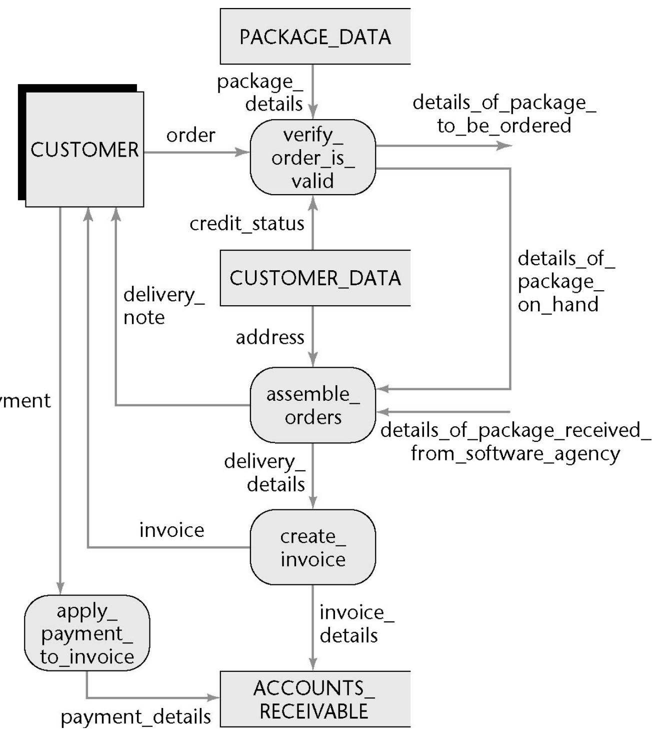


Second refinement:

PENDING ORDERS is scanned daily



Portion of third refinement:



The final DFD is larger

But it is easily understood by the client

Step 2: Decide What Parts to Computerize and How

- It depends on how much client is prepared to spend
- Cost/benefit analysis
- Determine Solution Strategy(How)

Step 3: Determine the Details of the Data Flows

- Determine the data items for each data flow
- Refine each flow stepwise
- We need a data dictionary for larger products

Step 4: Define the Logic of the Processes

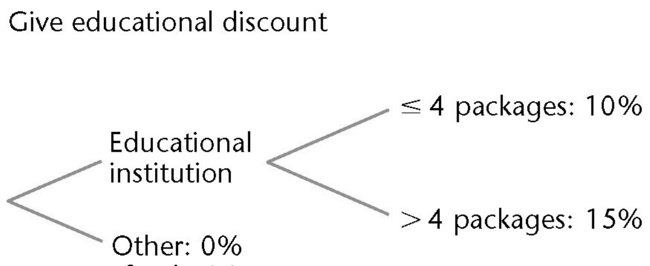
We have process give educational discount

Sally must explain the discount she gives to educational institutions

10% on up to 4 packages

15% on 5 or more

Translate this into a decision tree



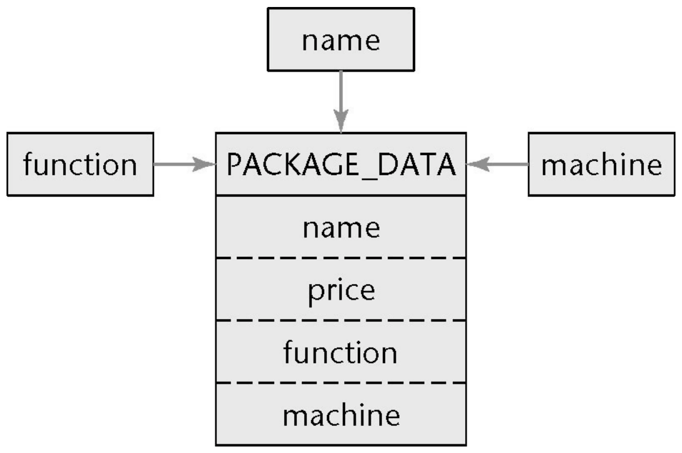
The advantage of a decision tree

-Missing items are quickly apparent

Step 5: Define the Data Stores

- Define the exact contents and representation (format)
- Specify where immediate access is required

  -Data immediate-access diagram (DIAD)



Step 6: Define the Physical Resources

For each file, specify

- -File name
- Organization (sequential, indexed, etc.)
- Storage medium
- Blocking factor
- Records (to field level)
- Table information, if a DBMS is to be used

Step 7: Determine Input/Output Specifications

Specify

- Input forms
- Input screens
- Printed output

Step 8: Determine the Sizing

-Obtain the numerical data needed in Step 9 to determine the hardware requirements

- Volume of input (daily or hourly)
- Size, frequency, deadline of each printed report
- Size, number of records passing between CPU and mass storage
- Size of each file

Step 9: Determine the Hardware Requirements

- Mass storage requirements
- Mass storage for back-up
- Input needs
- Output devices
- Is the existing hardware adequate?

If not, recommend whether to buy or lease additional hardware

The method of Gane and Sarsen/De Marco/ Yourdon has resulted in major improvements in the software industry

Petri Nets

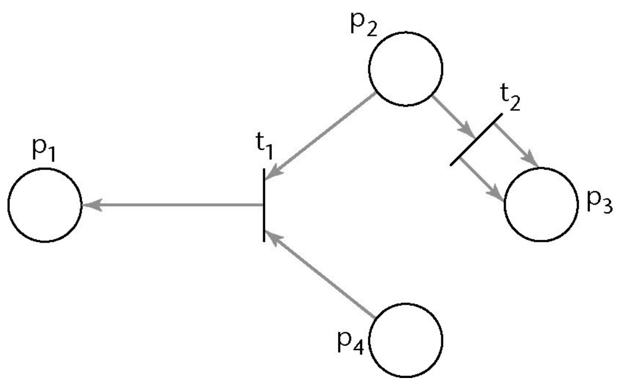
A major difficulty with specifying real-time systems is timing

- Synchronization problems
- Race conditions
- Deadlock
- Often a consequence of poor specifications

Petri nets-A powerful technique for specifying systems that have potential problems with interrelations

A Petri net consists of four parts:

- A set of places P
- A set of transitions T
- An input function I
- An output function O



Set of places P is

{p1, p2, p3, p4}

Set of transitions T

is {t1, t2}

Input functions:

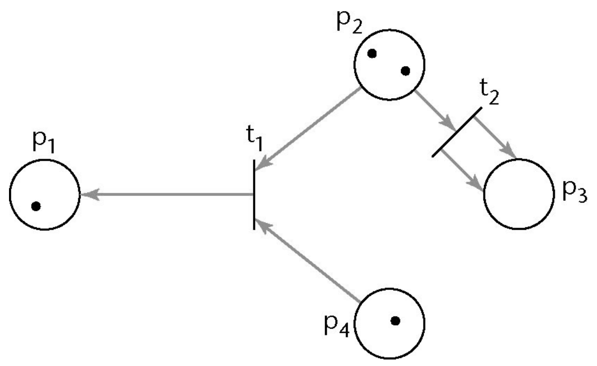
I(t1) = {p2, p4}
I(t2) = {p2}

Output functions:

O(t1) = {p1}
O(t2) = {p3, p3}

More formally, a Petri net is a 4-tuple C = (P, T, I, O)

A marking of a Petri net is an assignment of tokens to that Petri net



Represented by the vector(1,2,0,1)

A transition is enabled if each of its input places has as many tokens in it as there are arcs from the place to that transition

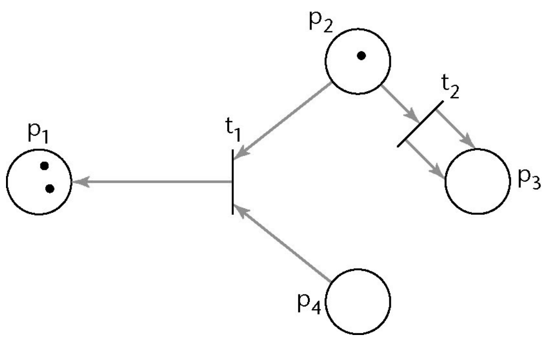
Transition t1 is enabled (ready to fire)

-If t1 fires, one token is removed from p2 and one from p4, and one new token is placed in p1

Transition t2 is also enabled

Petri nets are indeterminate

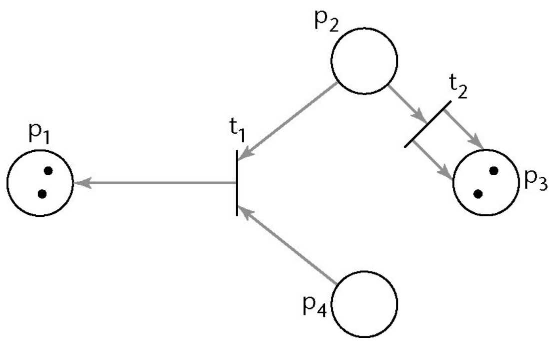
Suppose t1 fires



The resulting marking is (2,1,0,0)

Now only t2 is enabled

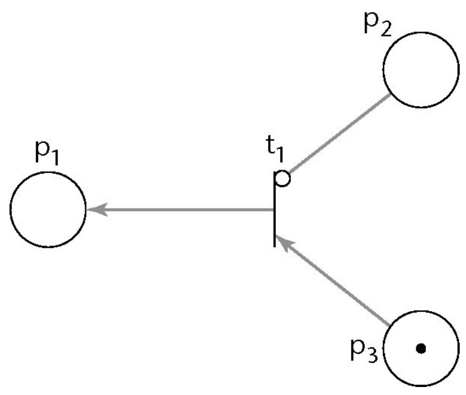
-It fires



The marking is now (2,0,2,0)

A marked Petri net is then a 5-tuple (P, T, I, O, M)

Inhibitor arcs-An inhibitor arc is marked by a small circle, not an arrowhead



Transition t1 is enabled

In general, a transition is enabled if there is at least one token on each (normal) input arc, and no tokens on any inhibitor input arcs.

Petri Nets: The Elevator Problem Case Study

- A product is to be installed to control n elevators in a building with m floors
- Each floor is represented by a place Ff, 1

  f   m

- An elevator is represented by a token
- A token in Ff denotes that an elevator is at floor Ff

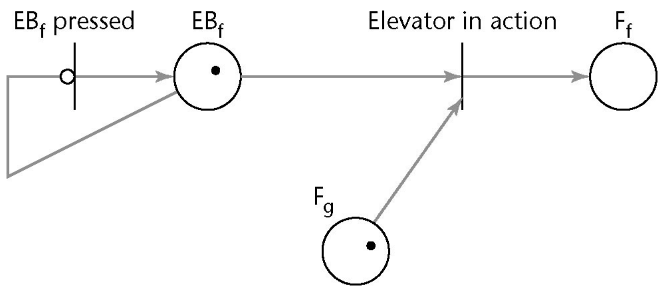
First constraint:

Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by an elevator

The elevator button for floor f is represented by place EBf,

A token in EBf denotes that the elevator button for floor f is illuminated

A button must be illuminated the first time the button is pressed and subsequent button presses must be ignored



If button EBf is not illuminated, no token is in place and transition EBf pressed is enabled. The transition fires, a new token is placed in EBf

Now, no matter how many times the button is pressed, transition EBf pressed cannot be enabled

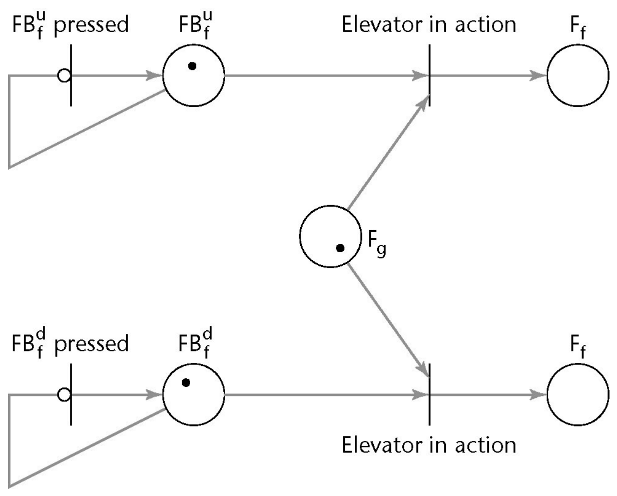
When the elevator reaches floor g, A token is in place Fg. Transition Elevator in action is enabled, and then fires.The tokens in EBf and Fg are removed. This turns off the light in button EBf.

A new token appears in Ff. This brings the elevator from floor g to floor f. Motion from floor g to floor f cannot take place instantaneously. We need timed Petri nets

Second constraint:

Each floor, except the first and the top floor, has two buttons, one to request an up-elevator, one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when the elevator visits the floor, and then moves in desired direction

Floor buttons are represented by places FBuf and FBdf



The Petri net in the previous slide models the situation when an elevator reaches floor f from floor g with one or both buttons illuminated. If both buttons are illuminated, only one is turned off

A more complex model is needed to ensure that the correct light is turned off.

Third constraint:

If an elevator has no requests, it remains at its current floor with its doors closed. If there are no requests, no Elevator in action transition is enabled.

Petri nets can also be used for design

Petri nets possess the expressive power necessary for specifying synchronization aspects of real-time systems