Software Engineering (MCA-122)
MCA 3rd Sem (2020-21)

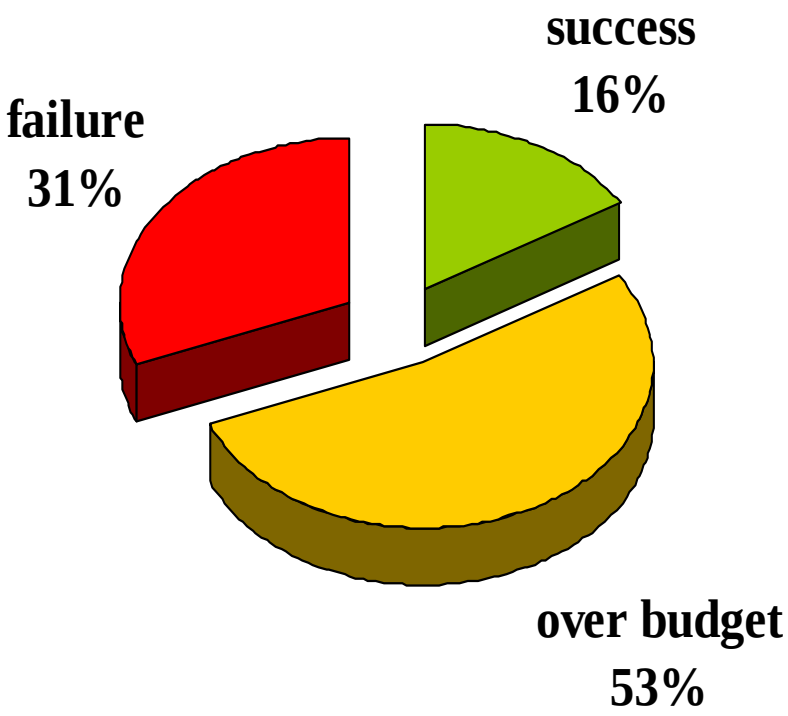
D. S. Singh
Associate Professor
Department of ITCA
MMMUT Gorakhpur

Need of Software Engineering

- Change in nature of software
- Change in complexity of software
- Concept of one “Guru” is over
- Improvements are always required
- Ready for change

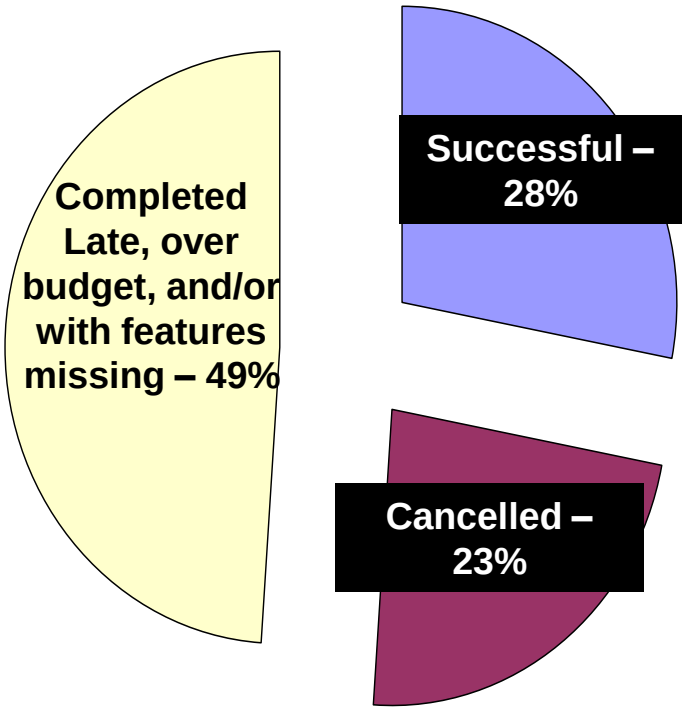
The Evolving Role of Software

- **Software industry is in Crisis!**



The Evolving Role of Software

This is the SORRY state of Software Engineering Today!



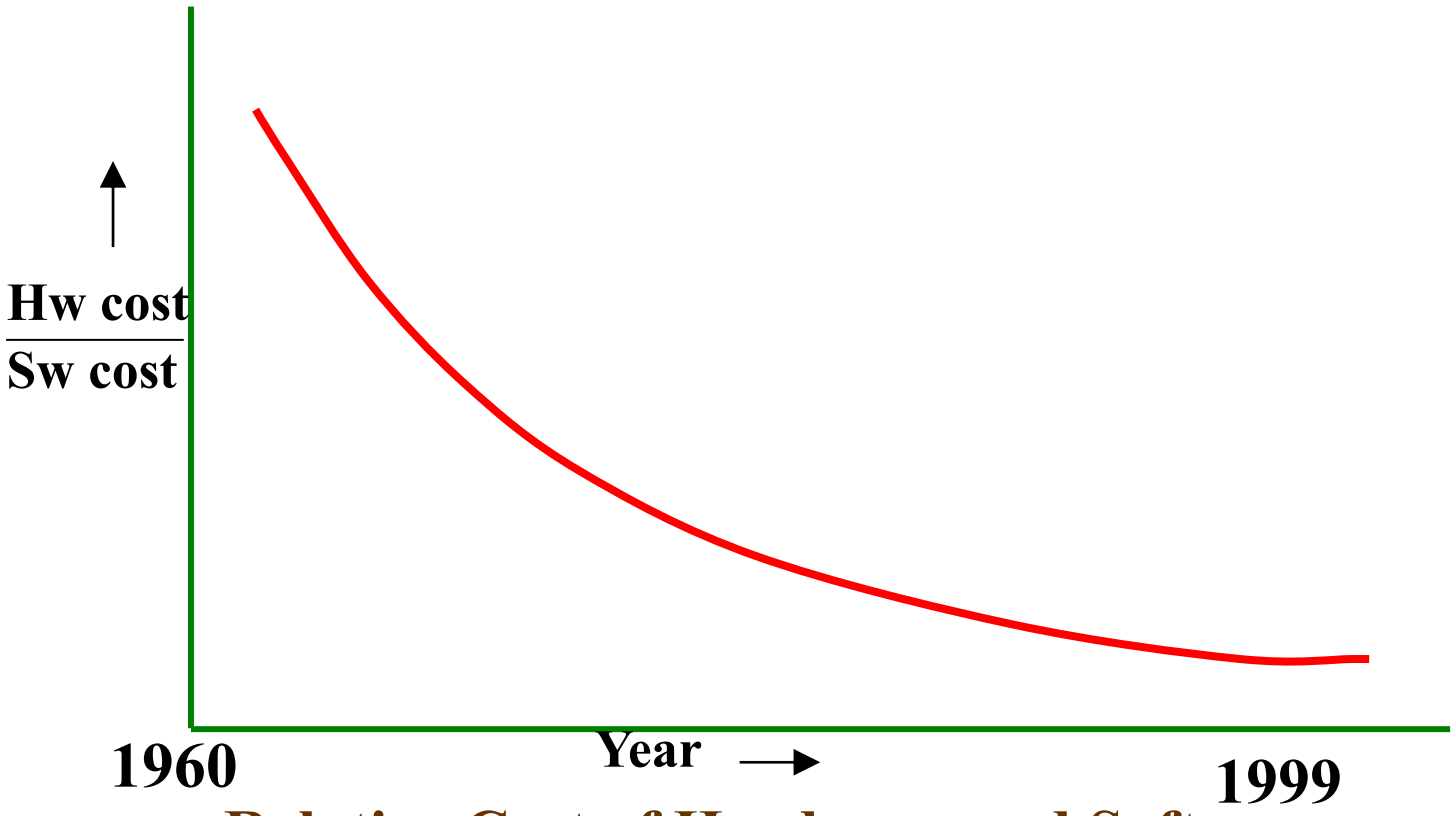
- *Data on 28,000 projects completed in 2000*

The Evolving Role of Software

As per the IBM report

- 31% of the project get cancelled before they are completed
- 53% over-run their cost estimates by an average of 189%
- For every 100 projects, there are 94 restarts

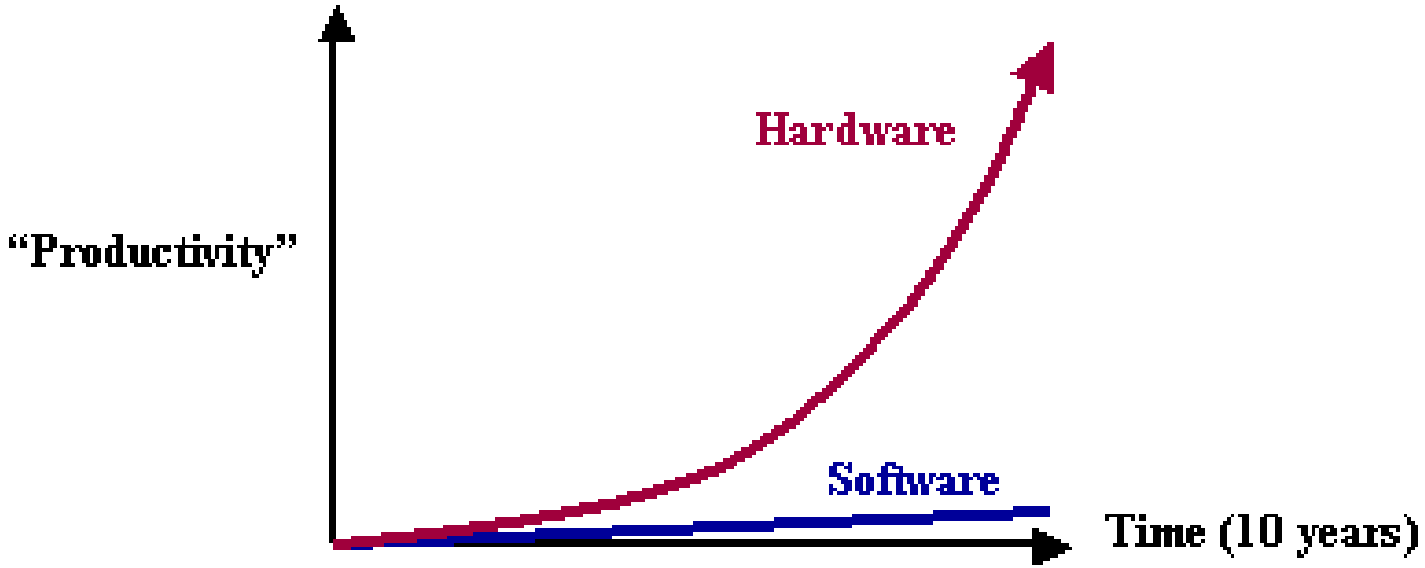
The Evolving Role of Software



Relative Cost of Hardware and Software

The Evolving Role of Software

- Unlike Hardware
  - Moore's law: processor speed/memory capacity doubles every two years



The Evolving Role of Software

Managers and Technical Persons are always asked:

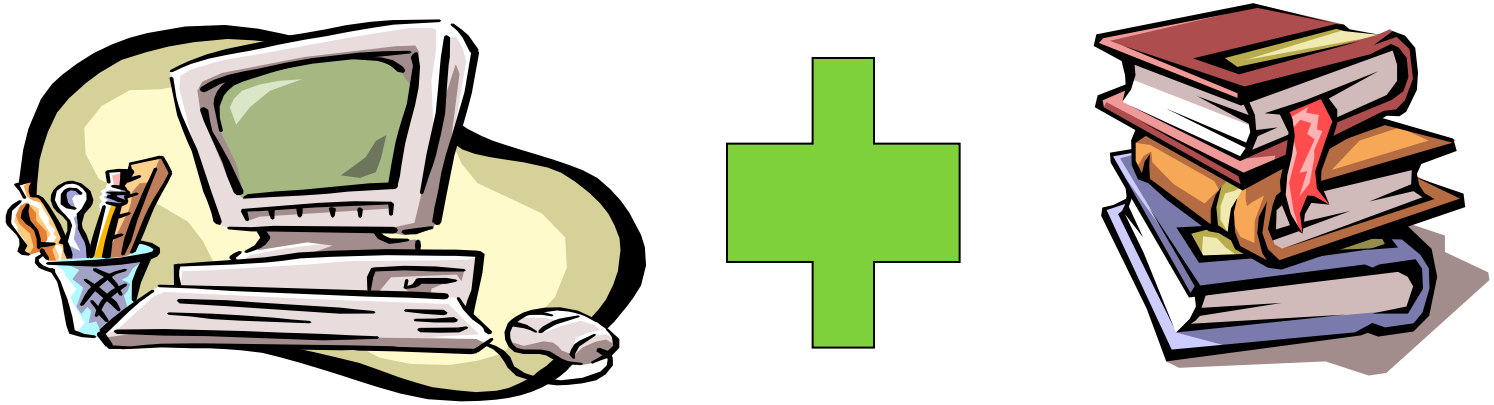
- Why does it take so long to get the program finished?
- Why are costs so high?
- Why can not we find all errors before release of software?
- Why do we have difficulty in measuring progress of software development?

Factors Contributing to the Software Crisis

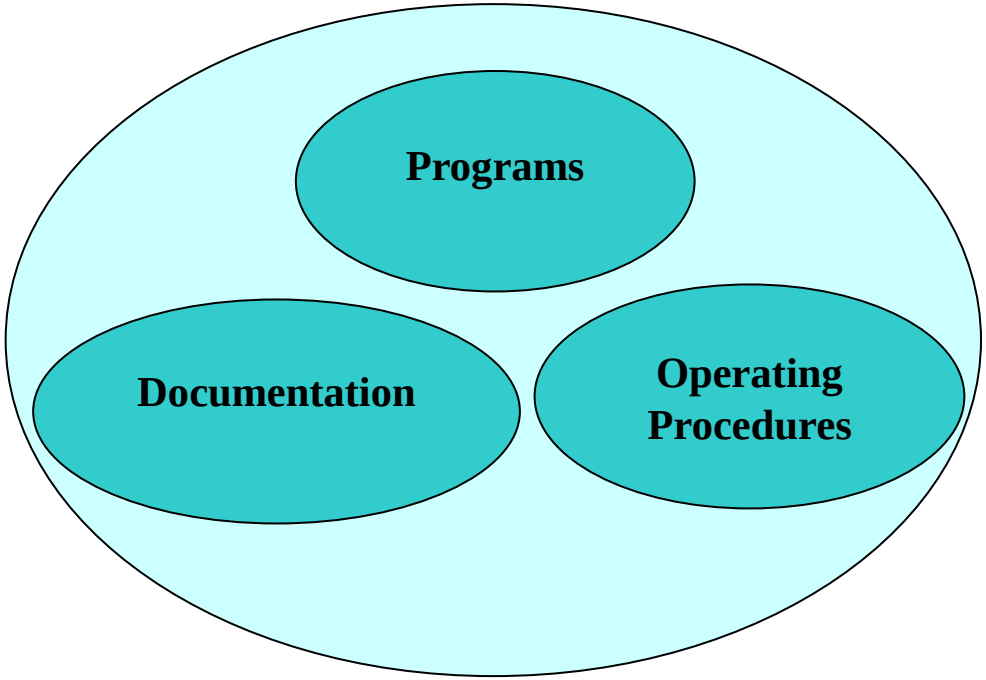
- Larger problems
- Lack of adequate training in software engineering
- Increasing skill shortage
- Low productivity improvements

What is software?

- Computer programs and associated documentation



What is software?



Software= Program + Documentation + Operating Procedures

Components of software

Software Product

- Software products may be developed for a particular customer or may be developed for a general market
- Software products may be
  - Generic - developed to be sold to a range of different customers
  - Bespoke (custom) - developed for a single customer according to their specification

Software Product

Software product is a product designated for delivery to the user

Software Engineering

Software engineering is an engineering discipline which is concerned with all aspects of software production. It should

- adopt a systematic and organised approach to their work
- use appropriate tools and techniques depending on
  - ✓ the problem to be solved,
  - ✓ the development constraints and
  - ✓ use the resources available

Software Engineering

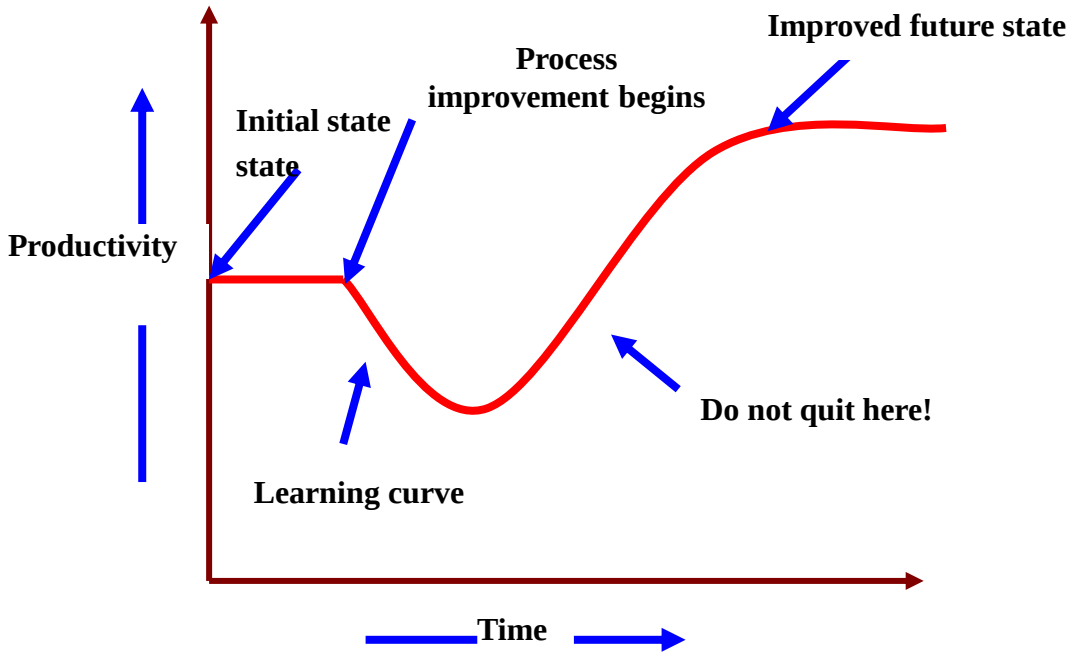
- At the first conference on software engineering in 1968, Fritz Bauer defined software engineering as "The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines".
- Stephen Schacht defined the same as "A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements".
- Both the definitions are popular and acceptable to majority. However, due to increase in cost of maintaining software, objective is now shifting to produce quality software that is maintainable, delivered on time, within budget, and also satisfies its requirements.

Software Process

- The software process is the way in which we produce software
- It is difficult to improve software process because of
  - Not enough time
  - Lack of knowledge

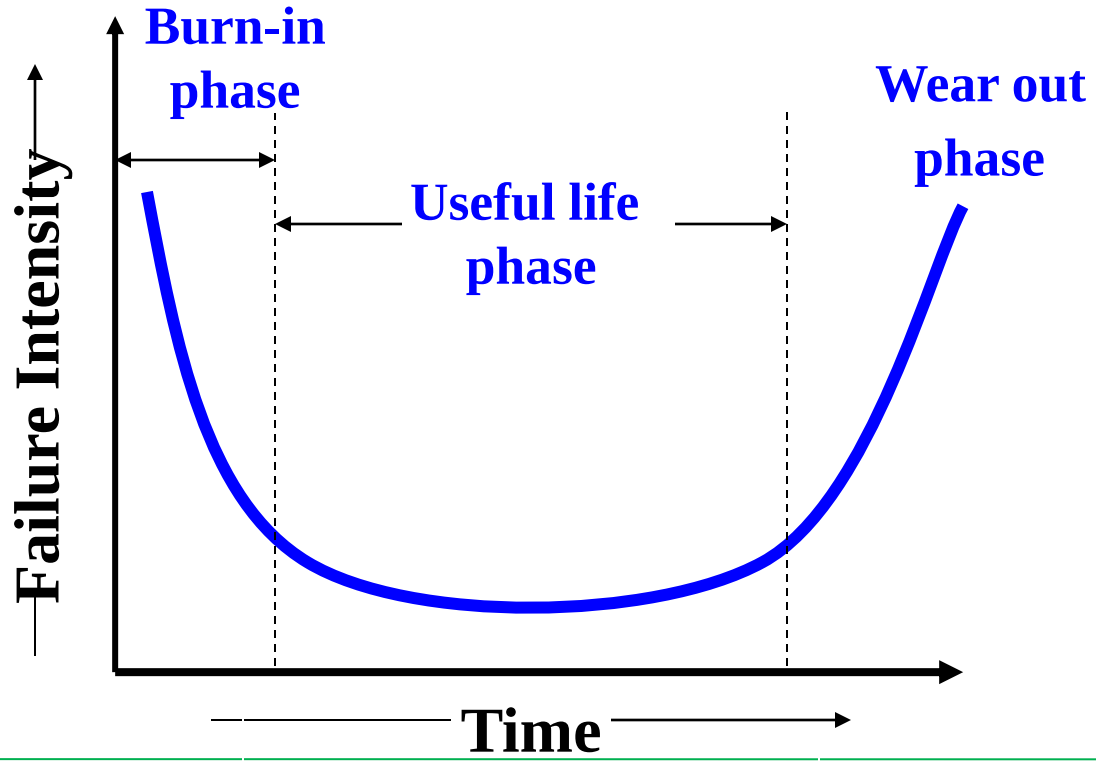
Software Process

- Wrong motivations
- Insufficient commitment



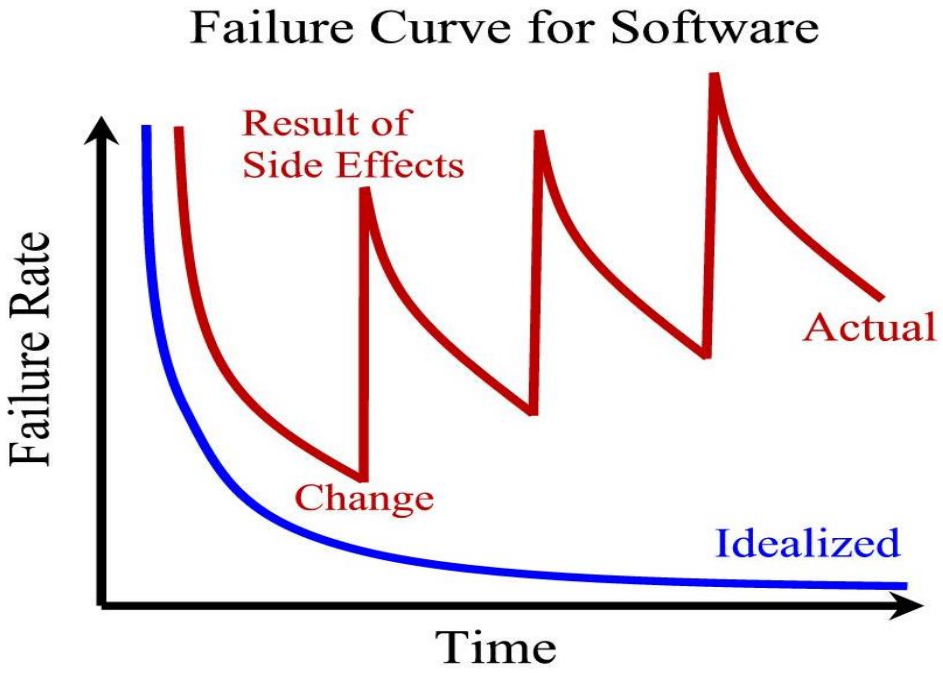
Software Characteristics

- Software does not wear out



Software Characteristics

- Software is not manufactured
- Reusability of components
- Software is flexible

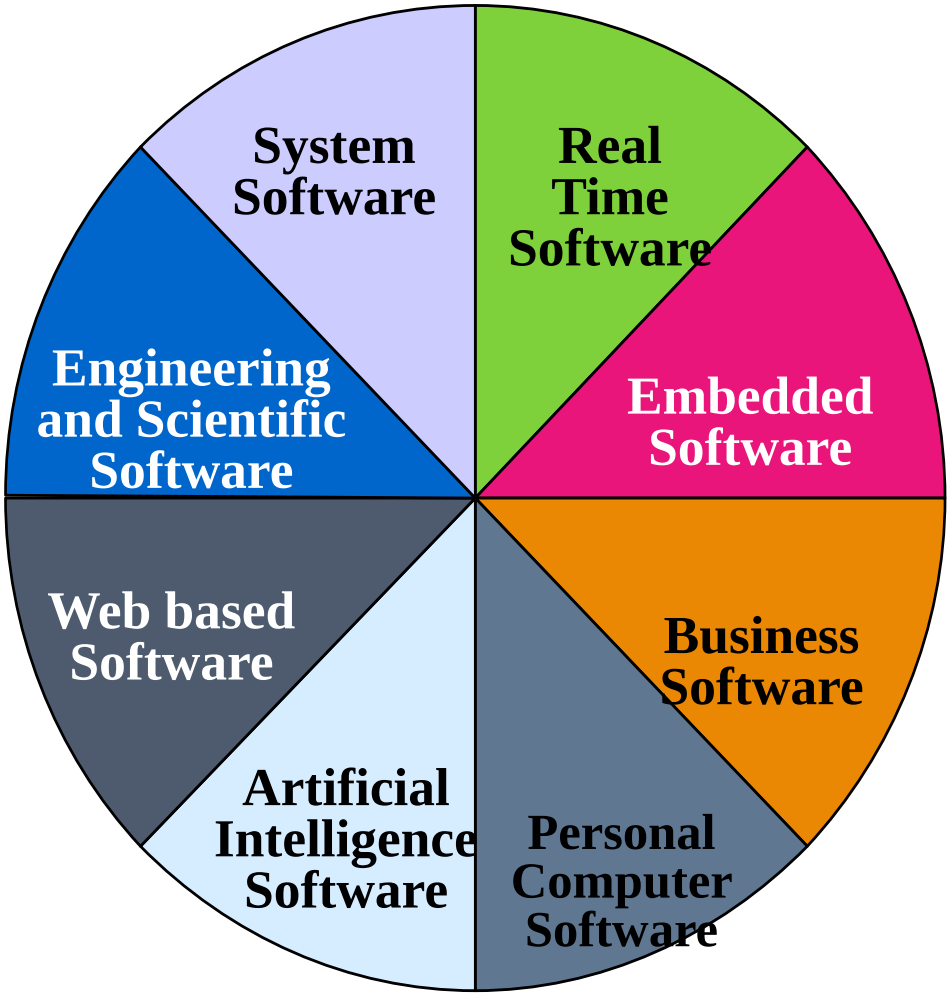


Software Characteristics

Comparison of constructing a bridge vis-à-vis writing a program

Sr.No	Constructing a bridge	Writing a program
1.	The problem is well understood	Only some parts of the problem are understood, others are not
2.	There are many existing bridges	Every program is different and designed for special applications.
3.	The requirement for a bridge typically do not change much during construction	Requirements typically change during all phases of development.
4.	The strength and stability of a bridge can be calculated with reasonable precision	Not possible to calculate correctness of a program with existing methods.
5.	When a bridge collapses, there is a detailed investigation and report	When a program fails, reasons are often unavailable or even deliberately concealed.
6.	Engineers have been constructing bridges for thousands of years	Developers have been writing programs for 50 years or so.
7.	Materials (wood, stone,iron, steel) and techniques (making joints in wood, carving stone, casting iron) change slowly.	Hardware and software changes rapidly.

The Changing Nature of Software



The Changing Nature of Software

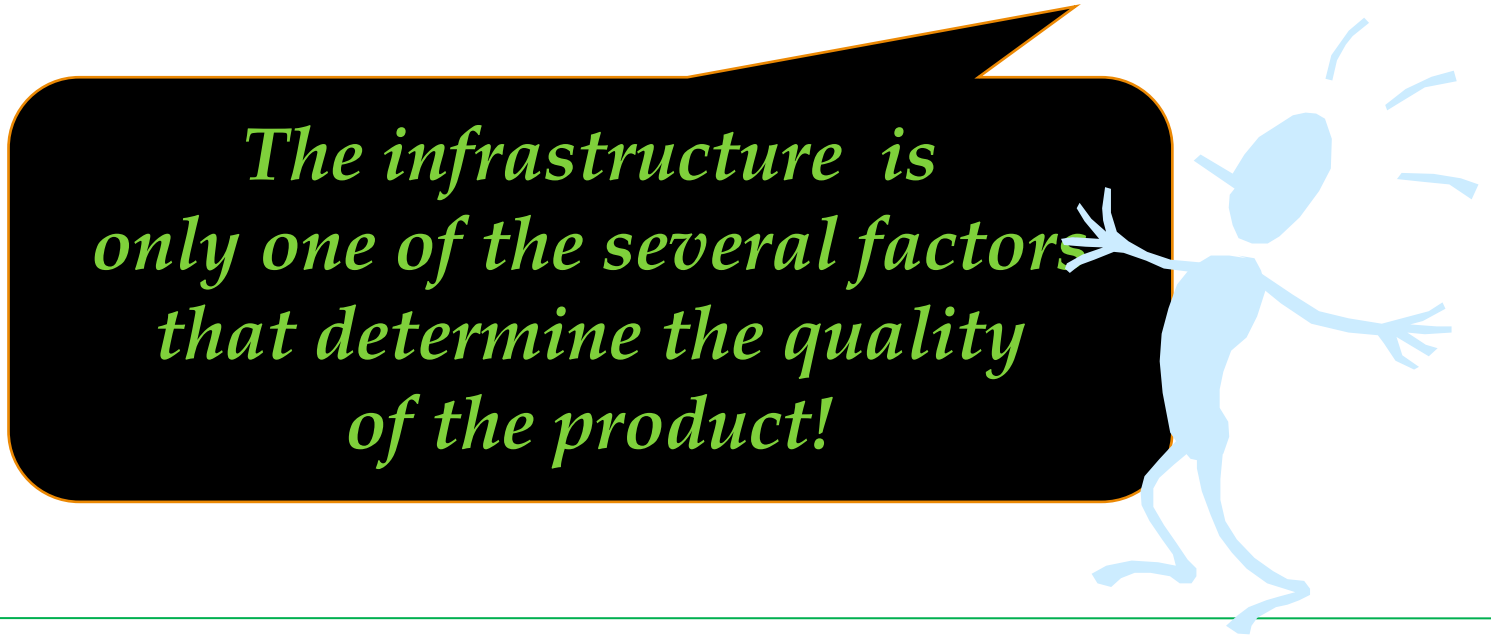
- Trend has emerged to provide source code to the customer and organizations.
- Software where source codes are available are known as open source software.
- Examples: Open source software: LINUX, MySQL, PHP, Open office, Apache webserver etc.

Software Myths (Management Perspectives)

- Management may be confident about good standards and clear procedures of the company.

Software Myths (Management Perspectives)

- Company has latest computers and state-of-the-art software tools, so we shouldn't worry about the quality of the product.



Software Myths (Management Perspectives)

- Addition of more software specialists, those with higher skills and longer experience may bring the schedule back on the track!

Unfortunately,
that may further delay the schedule!

Software Myths (Management Perspectives)

- Software is easy to change

The reality is totally different.

Software Myths (Management Perspectives)

- Computers provide greater reliability than the devices they replace

This is not always true.

Software Myths (Customer Perspectives)

- A general statement of objectives is sufficient to get started with the development of software. Missing/vague requirements can easily be incorporated/detailed out as they get concretized.

If we do so, we are heading towards a disaster.

Software Myths (Customer Perspectives)

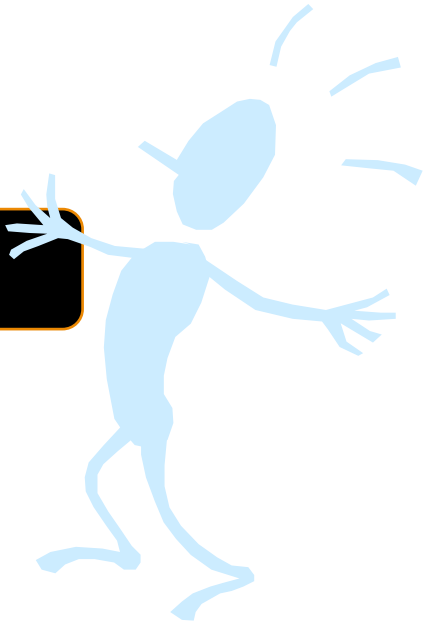
- Software with more features is better software
- Software can work right the first time

Both are only myths!

Software Myths (Developer Perspectives)

- Once the software is demonstrated, the job is done.

Usually, the problems just begin!



Software Myths (Developer Perspectives)

- Software quality can not be assessed before testing.

However, quality assessment techniques should be used through out the software development life cycle.

Software Myths (Developer Perspectives)

- The only deliverable for a software development project is the tested code.

Software Myths (Developer Perspectives)

- Aim is to develop working programs

Those days are over. Now objective is to develop good quality maintainable programs!

Some Terminologies

❖ Deliverables and Milestones

- Different deliverables are generated during software development. The examples are source code, user manuals, operating procedure manuals etc.

- The milestones are the events that are used to ascertain the status of the project. Finalization of specification is a milestone. Completion of design documentation is another milestone. The milestones are essential for project planning and management.

Some Terminologies

❖ Product and Process

- Product: What is delivered to the customer, is called a product. It may include source code, specification document, manuals, documentation etc. Basically, it is nothing but a set of deliverables only.

- Process: Process is the way in which we produce software. It is the collection of activities that leads to (a part of) a product. An efficient process is required to produce good quality products.

If the process is weak, the end product will undoubtedly suffer, but an obsessive over reliance on process is also dangerous.

Some Terminologies

Measures, Metrics and Measurement

- A measure provides a quantitative indication of the extent, dimension, size, capacity, efficiency, productivity or reliability of some attributes of a product or process.
- Measurement is the act of evaluating a measure.
- A metric is a quantitative measure of the degree to which a system, component or process possesses a given attribute.

Some Terminologies

Software Process and Product Metrics

- Process metrics quantify the attributes of software development process and environment; whereas product metrics are measures for the software product

- Examples of Process metrics: Productivity, Quality, Efficiency etc. and Product metrics: Size, Reliability, Complexity etc.

Some Terminologies

Productivity and Effort

- Productivity is defined as the rate of output, or production per unit of effort, i.e. the output achieved with regard to the time taken but irrespective of the cost incurred.

- Hence most appropriate unit of effort is Person Months (PMs), meaning thereby number of persons involved for specified months. So, productivity may be measured as LOC/PM (lines of code produced/person month)

Some Terminologies

Module and Software Components

- There are many definitions of the term module. They range from “a module is a FORTRAN subroutine” to “a module is an Ada Package”, to “Procedures and functions of PASCAL and C”, to “C++ Java classes” to “Java packages” to “a module is a work assignment for an individual developer”. All these definition are correct. The term subprogram is also used sometimes in place of module.

Some Terminologies

- An independently deliverable piece of functionality providing access to its services through interfaces

- A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces".

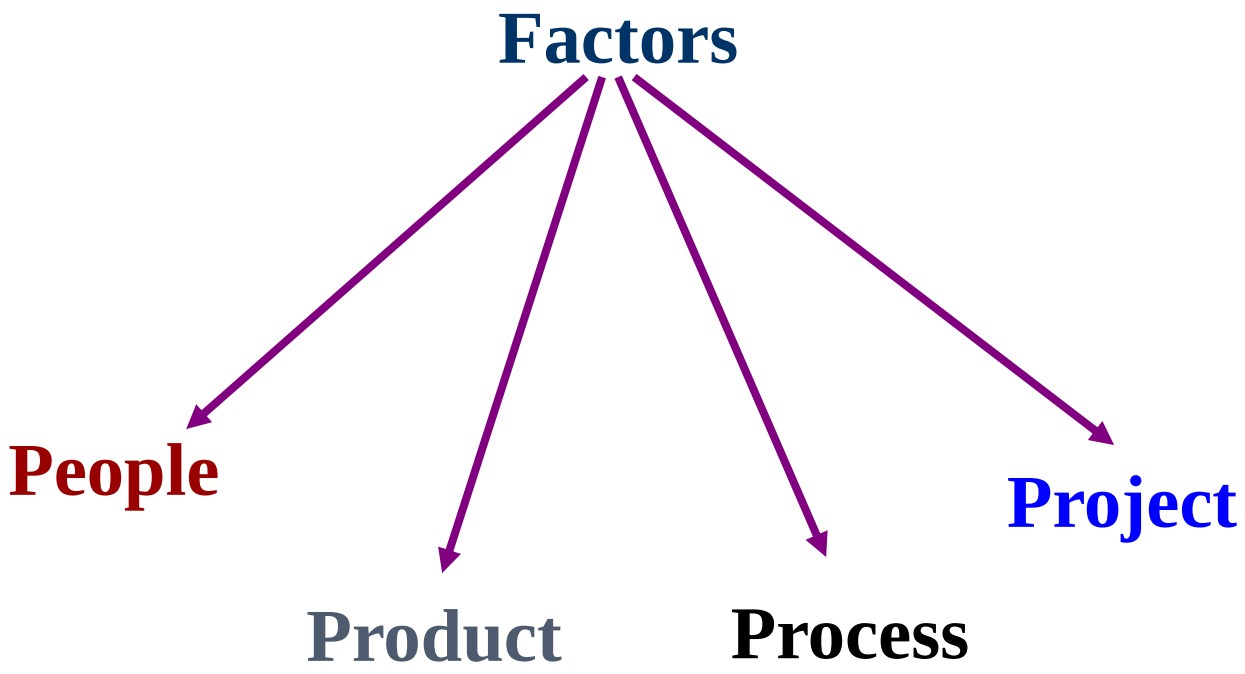
Some Terminologies

Generic and Customized Software Products

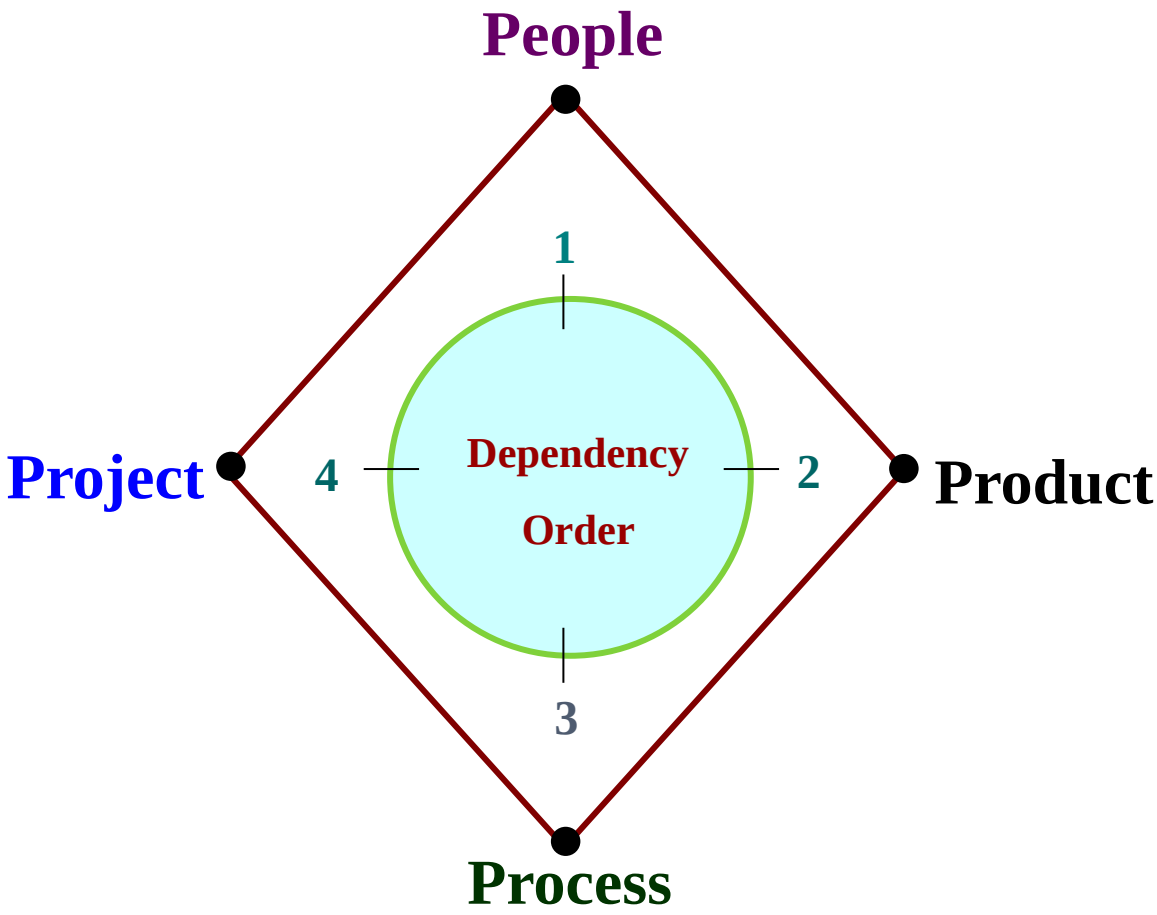
- Generic products are developed for anonymous customers. The target is generally the entire world and many copies are expected to be sold. Infrastructure software like operating system, compilers, analyzers, word processors, CASE tools etc. are covered in this category.

- The customized products are developed for particular customers. The specific product is designed and developed as per customer requirements. Most of the development projects (say about 80%)come under this category.

Role of Management in Software Development



Role of Management in Software Development



Software Life Cycle Models (SDLC)

- The goal of Software Engineering is to provide models and processes that lead to the production of well-documented maintainable software in a manner that is predictable.
- The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a requirement phase, design phase, Coding phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase, and sometimes retirement phase.

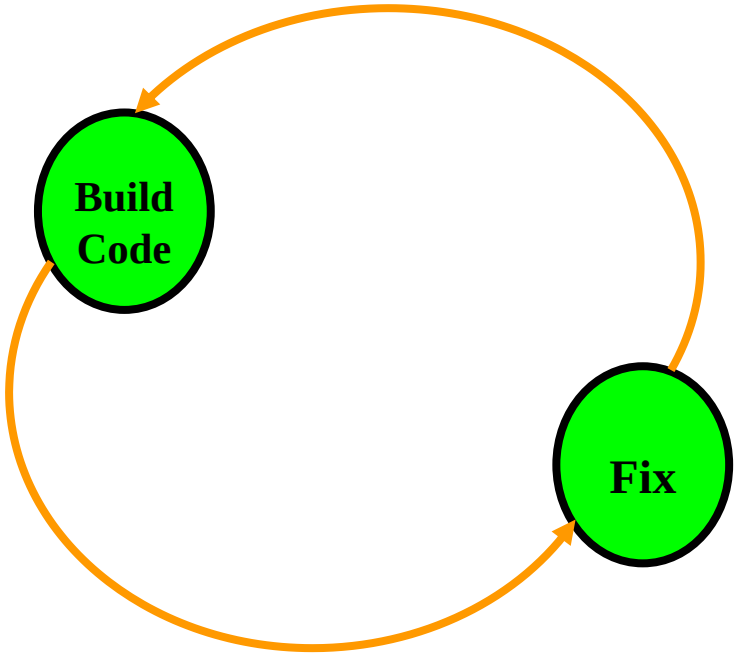
Software Life Cycle Models

Types of Models

- Build & Fix Model
- Waterfall Model
- Incremental Process Model
- Iterative Enhancement Model
- Rapid Application Development Model
- Evolutionary Process Modell
- Prototyping Model
- Spiral Model

Build & Fix Model

- Product is constructed without specifications or any attempt at design
- Ad-hoc approach and not well defined
- Simple two phase model

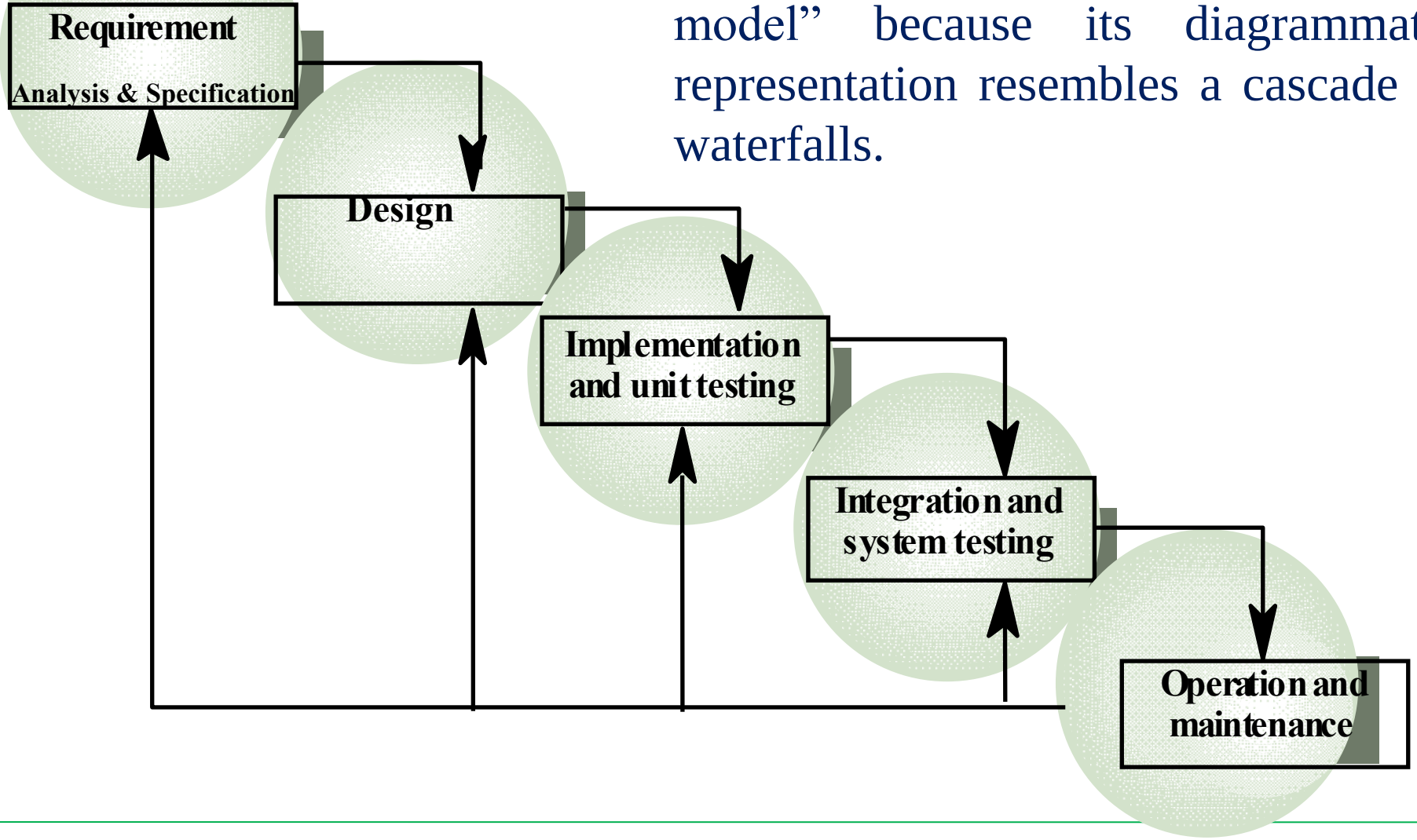


Build & Fix Model

- Suitable for small programming exercises of 100 or 200 lines
- Unsatisfactory for software for any reasonable size
- Code soon becomes unfixable & unenhanceable
- No room for structured design
- Maintenance is practically not possible

Waterfall Model

This model is named "waterfall model" because its diagrammatic representation resembles a cascade of waterfalls.



Waterfall Model

- This model is easy to understand and reinforces the notion of “define before design” and “design before code.

- The model expects complete & accurate requirements early in the process, which is unrealistic.

Waterfall Model

Problems of waterfall model

- It is difficult to define all requirements at the beginning of a project
- This model is not suitable for accommodating any change
- A working version of the system is not seen until late in the project's life
- It does not scale up well to large projects
- Real projects are rarely sequential.

Incremental Process Model

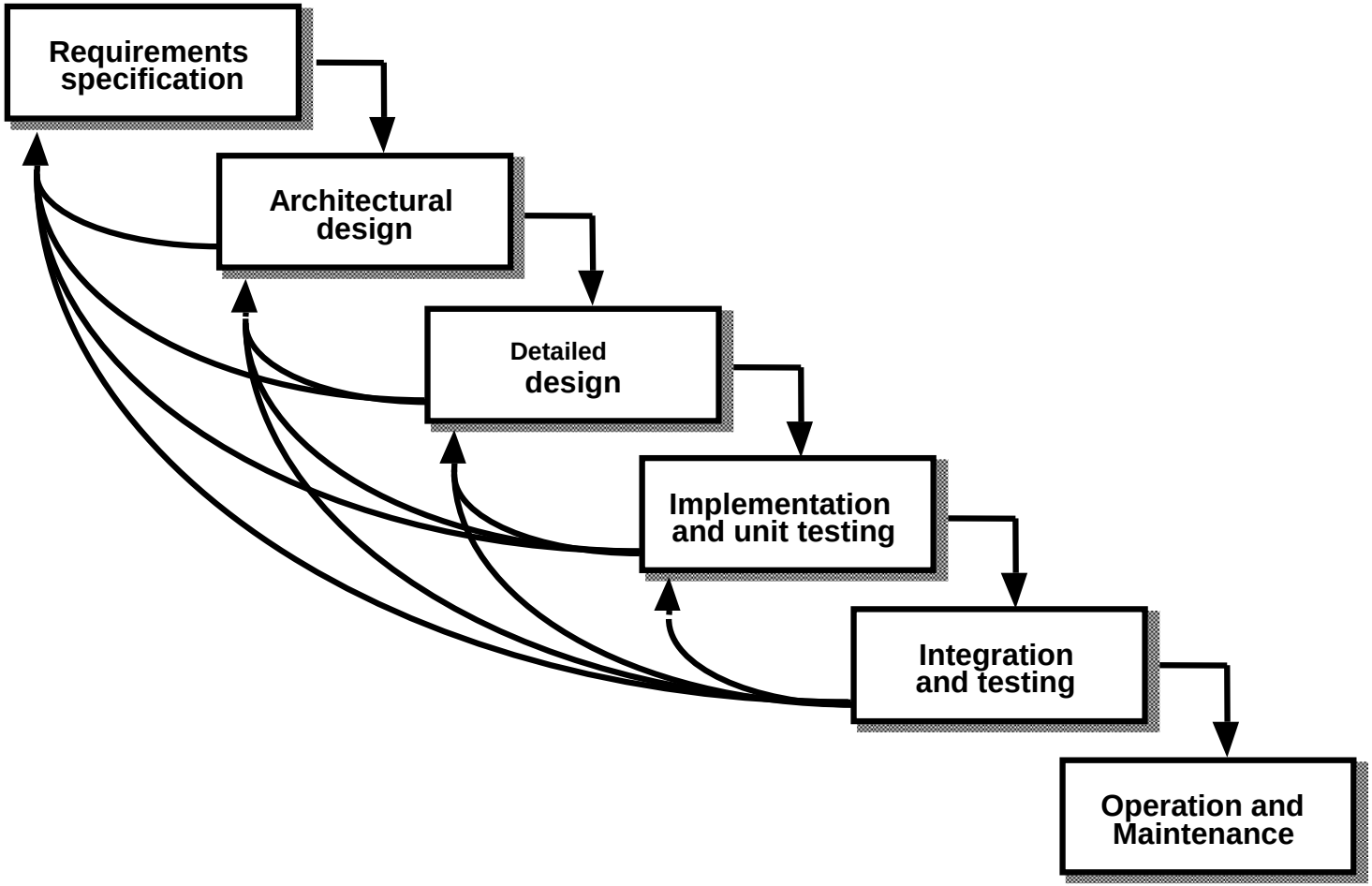
- They are effective in the situations where requirements are defined precisely and there is no confusion about the functionality of the final product.
- After every cycle a useable product is given to the customer.
- Popular particularly when we have to quickly deliver a limited functionality system.

Iterative Enhancement Model

- This model has the same phases as the waterfall model, but with fewer restrictions. Generally the phases occur in the same order as in the waterfall model, but they may be conducted in several cycles. Useable product is released at the end of the each cycle, with each release providing additional functionality.

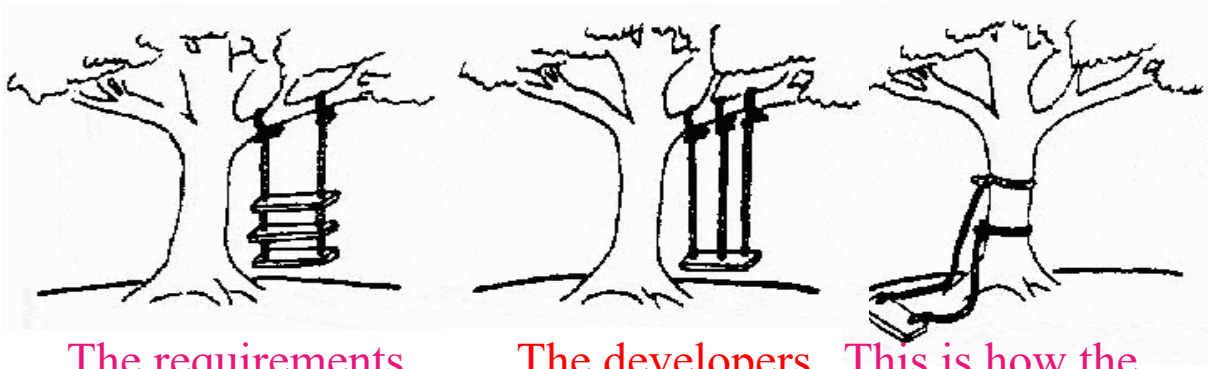
  - Customers and developers specify as many requirements as possible and prepare a SRS document.
  - Developers and customers then prioritize these requirements
  - Developers implement the specified requirements in one or more cycles of design, implementation and test based on the defined priorities.

Iterative Enhancement Model



Rapid Application Development (RAD) Model

- Developed by IBM in 1980
- User participation is essential



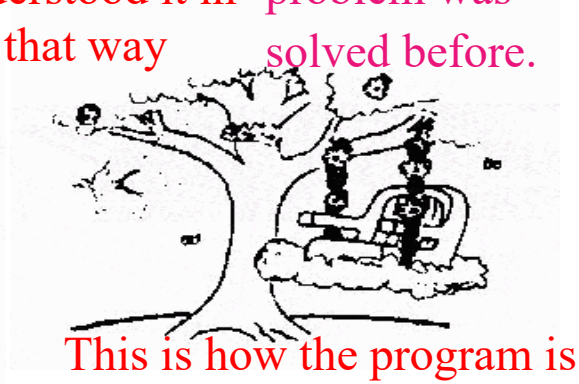
The requirements specification was defined like this

The developers understood it in that way

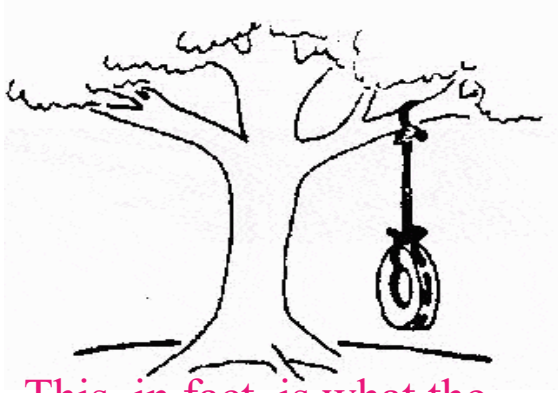
This is how the problem was solved before.

This is how the problem is solved now

That is the program after debugging



This is how the program is described by marketing department

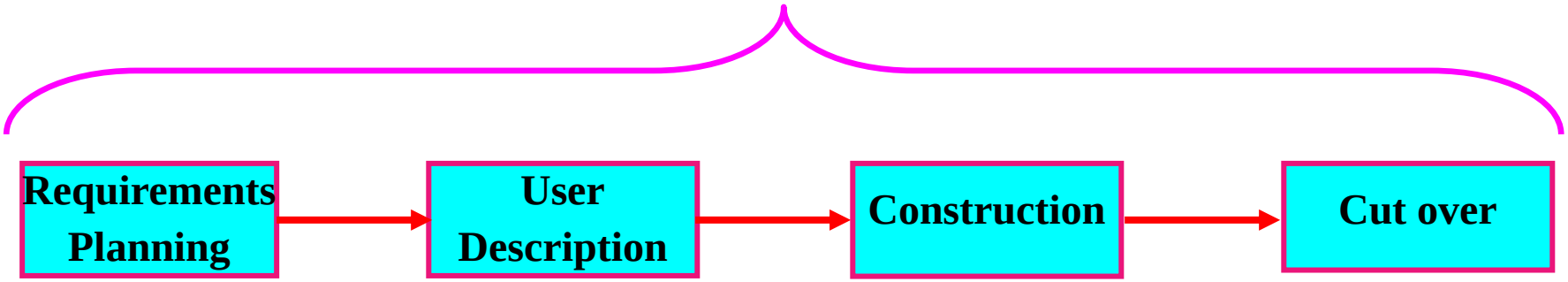


This, in fact, is what the customer wanted …

Rapid Application Development (RAD) Model

- Build a rapid prototype
- Give it to user for evaluation & obtain feedback
- Prototype is refined

With active participation of users



Rapid Application Development (RAD) Model

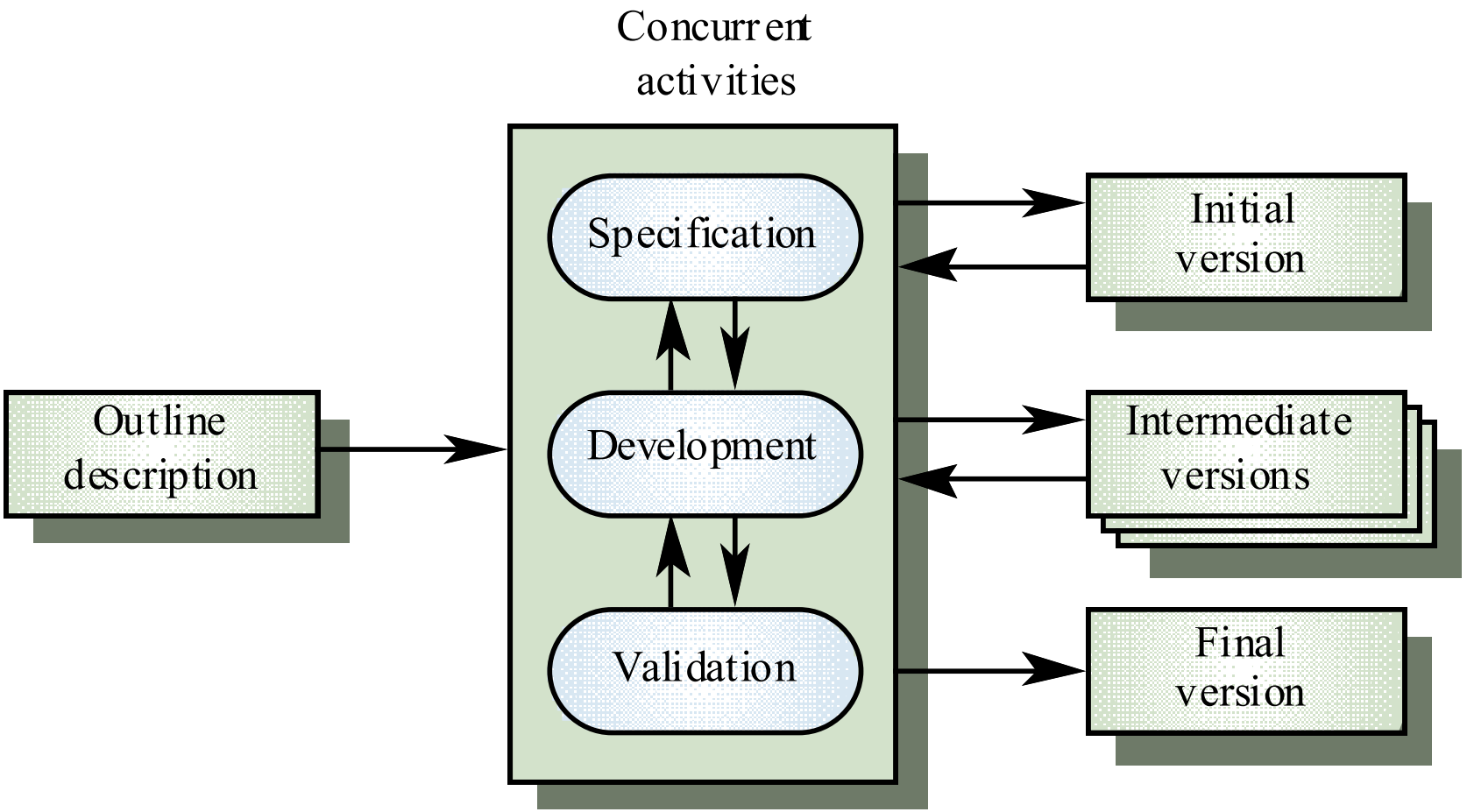
- Not an appropriate model in the absence of user participation.
- Reusable components are required to reduce development time.
- Highly specialized & skilled developers are required and such developers are not easily available.

Evolutionary Process Models

- Evolutionary process model resembles iterative enhancement model. The same phases as defined for the waterfall model occur here in a cyclical fashion. This model differs from iterative enhancement model in the sense that this does not require a useable product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

- This model is useful for projects using new technology that is not well understood. This is also used for complex projects where all functionality must be delivered at one time, but the requirements are unstable or not well understood at the beginning.

Evolutionary Process Model

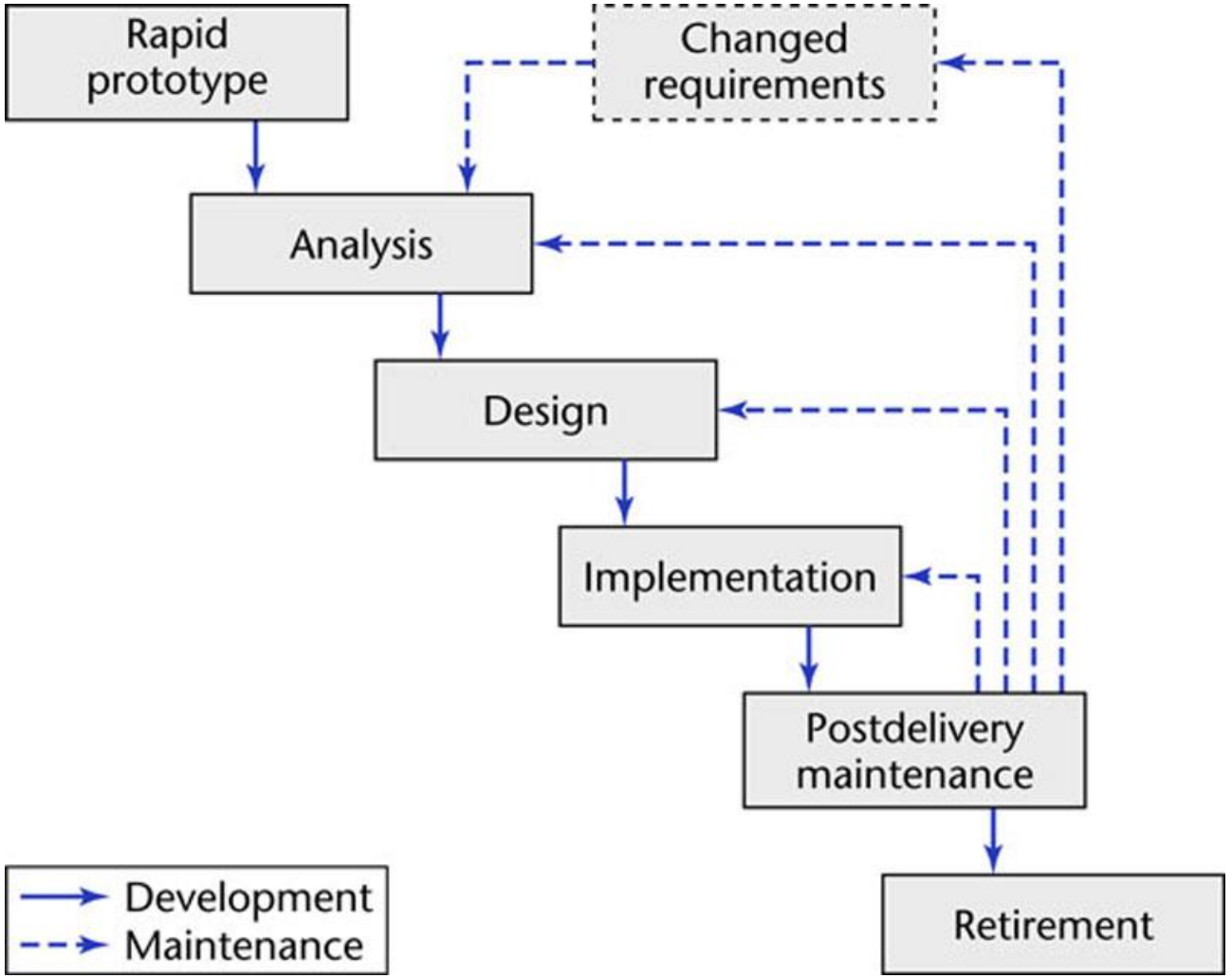


Prototyping Model

- The prototype may be a usable program but is not suitable as the final software product.
  - The code for the prototype is thrown away. However experience gathered helps in developing the actual system.
  - The development of a prototype might involve extra cost, but overall cost might turnout to be lower than that of an equivalent system developed using the waterfall model.

Prototyping Model

- Linear mode
- "Rapid"

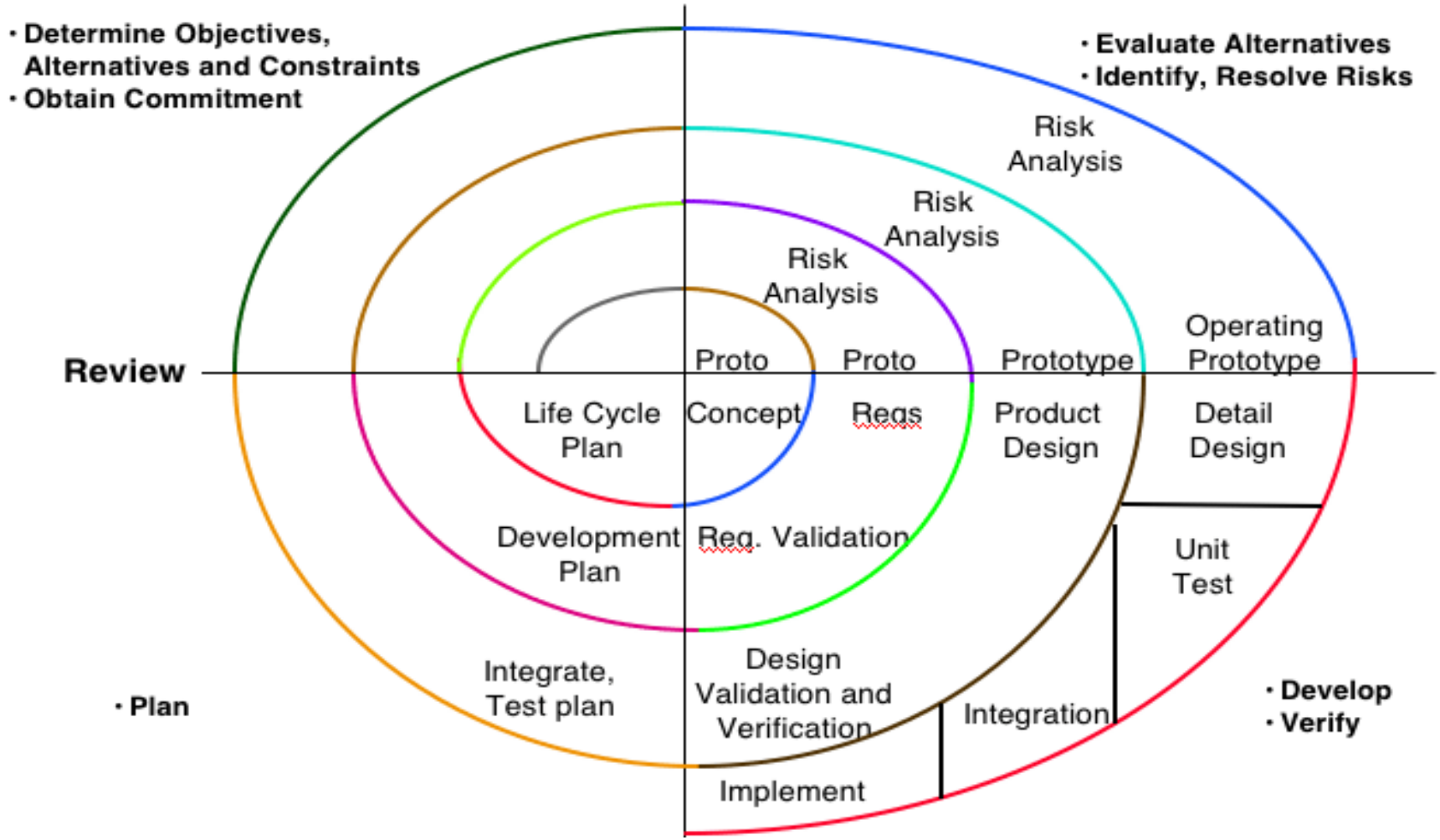


Prototyping Model

- Models do not deal with uncertainly which is inherent to software projects.
- Important software projects have failed because project risks were neglected & nobody was prepared when something unforeseen happened.
- Barry Boehm recognized this and tired to incorporate the "project risk" factor into a life cycle model.

The result is the spiral model, which was presented in 1986.

Spiral Model



Spiral Model

- The radial dimension of the model represents the cumulative costs. Each path around the spiral is indicative of increased costs. The angular dimension represents the progress made in completing each cycle. Each loop of the spiral from X-axis clockwise through 360^{o} represents one phase. One phase is split roughly into four sectors of major activities.
- Planning: Determination of objectives, alternatives & constraints.
- Risk Analysis: Analyze alternatives and attempts to identify and resolve the risks involved.
- Development: Product development and testing product.
- Assessment: Customer evaluation

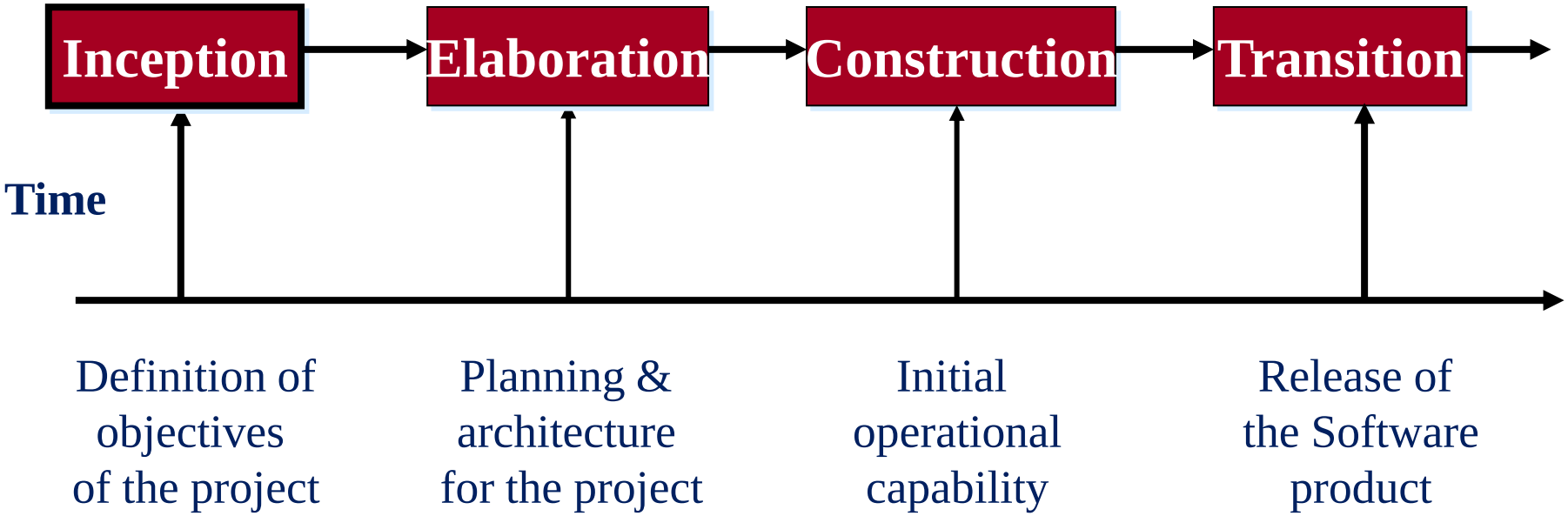
Spiral Model

- An important feature of the spiral model is that each phase is completed with a review by the people concerned with the project (designers and programmers)
- The advantage of this model is the wide range of options to accommodate the good features of other life cycle models.
- It becomes equivalent to another life cycle model in appropriate situations.
- The spiral model has some difficulties that need to be resolved before it can be a universally applied life cycle model. These difficulties include lack of explicit process guidance in determining objectives, constraints, alternatives; relying on risk assessment expertise; and provides more flexibility than required for many applications.

The Unified Process

- Developed by I.Jacobson, G.Booch and J.Rumbaugh.
- Software engineering process with the goal of producing good quality maintainable software within specified time and budget.
- Developed through a series of fixed length mini projects called iterations.
- Maintained and enhanced by Rational Software Corporation and thus referred to as Rational Unified Process (RUP).

Phases of the Unified Process



Phases of the Unified Process

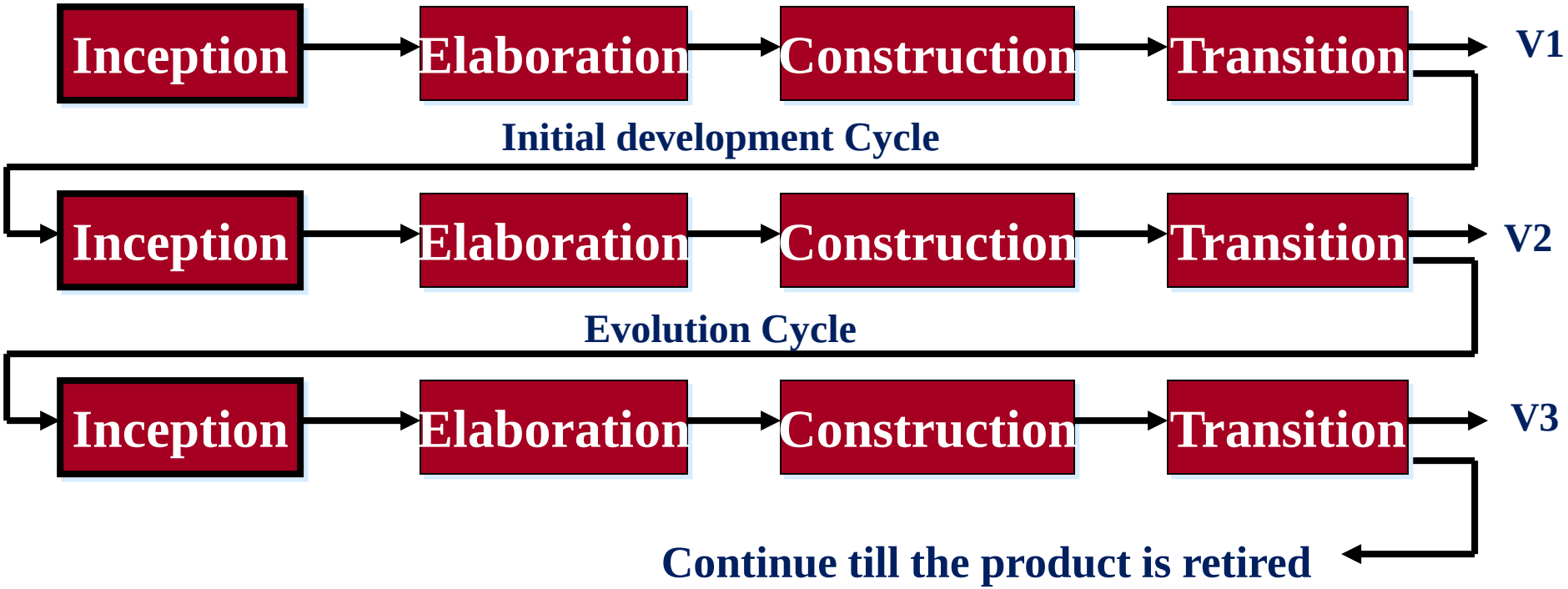
- **Inception:** defines scope of the project.
- **Elaboration**

  - How do we plan & design the project?

  - What resources are required?

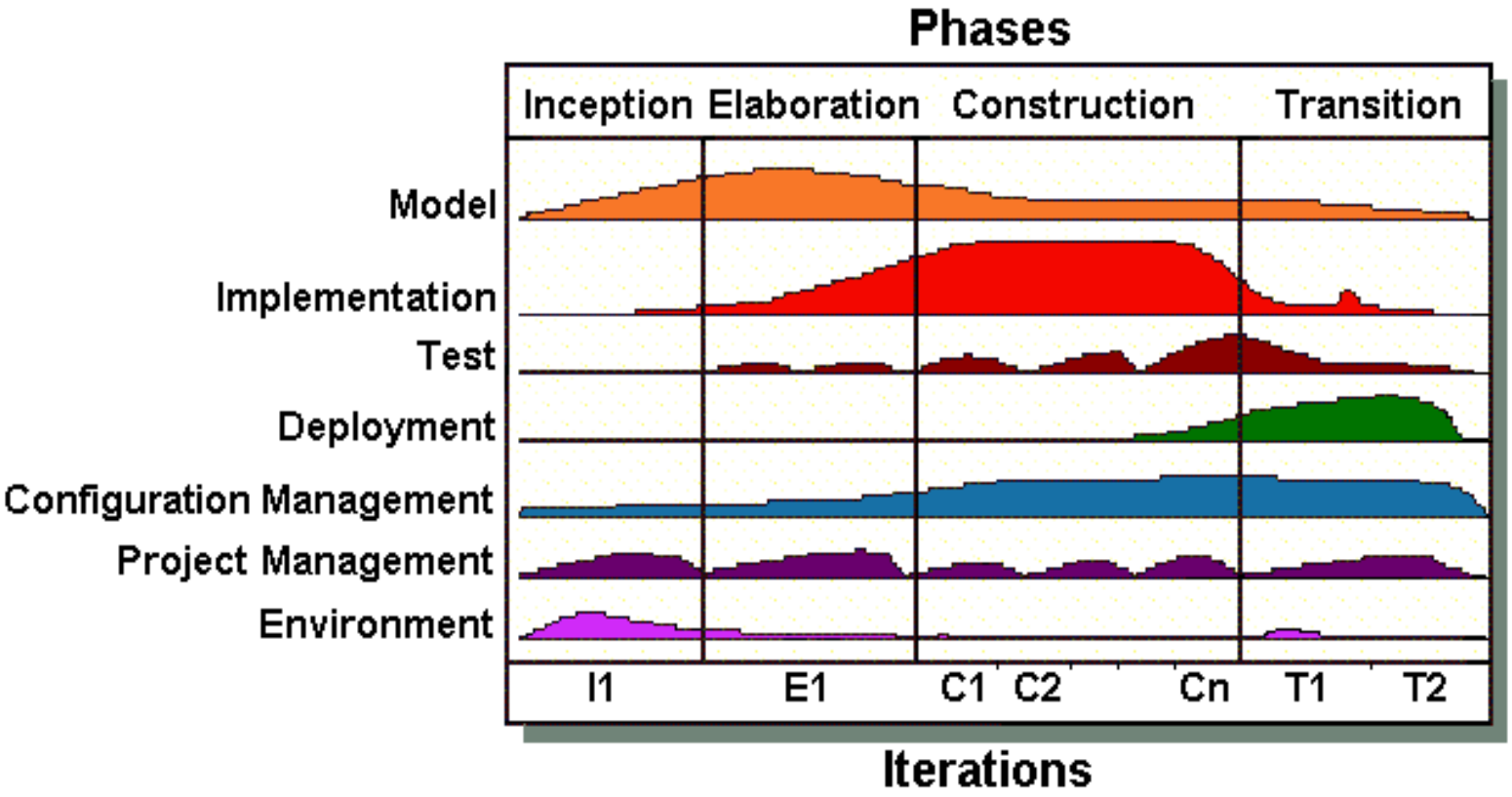
  - What type of architecture may be suitable?
- **Construction:** the objectives are translated in design & architecture documents.
- **Transition :** involves many activities like delivering, training, supporting, and maintaining the product.

Initial development & Evolution Cycles



V1=version1, V2 =version2, V3=version3

Iterations & Workflow of Unified Process

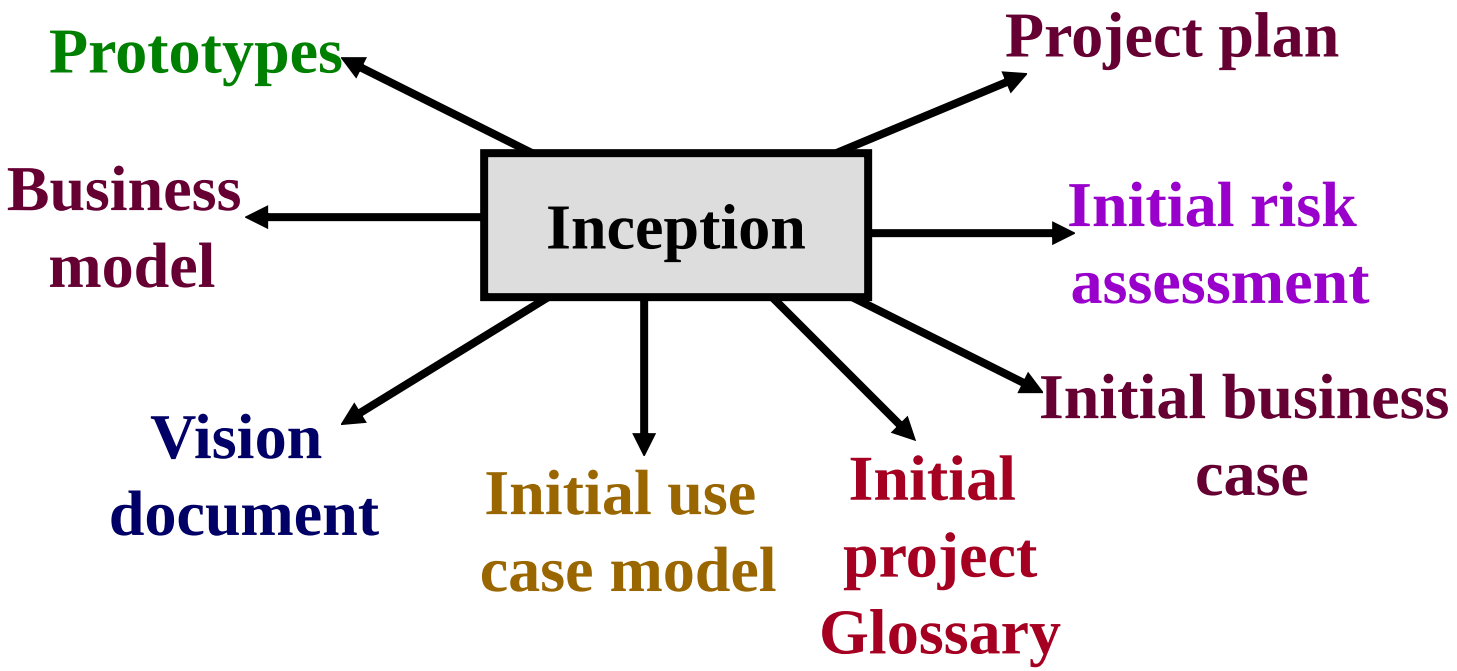


Inception Phase

The inception phase has the following objectives:

- Gathering and analyzing the requirements.
- Planning and preparing a business case and evaluating alternatives for risk management, scheduling resources etc.
- Estimating the overall cost and schedule for the project.
- Studying the feasibility and calculating profitability of the project.

Outcomes of Inception Phase

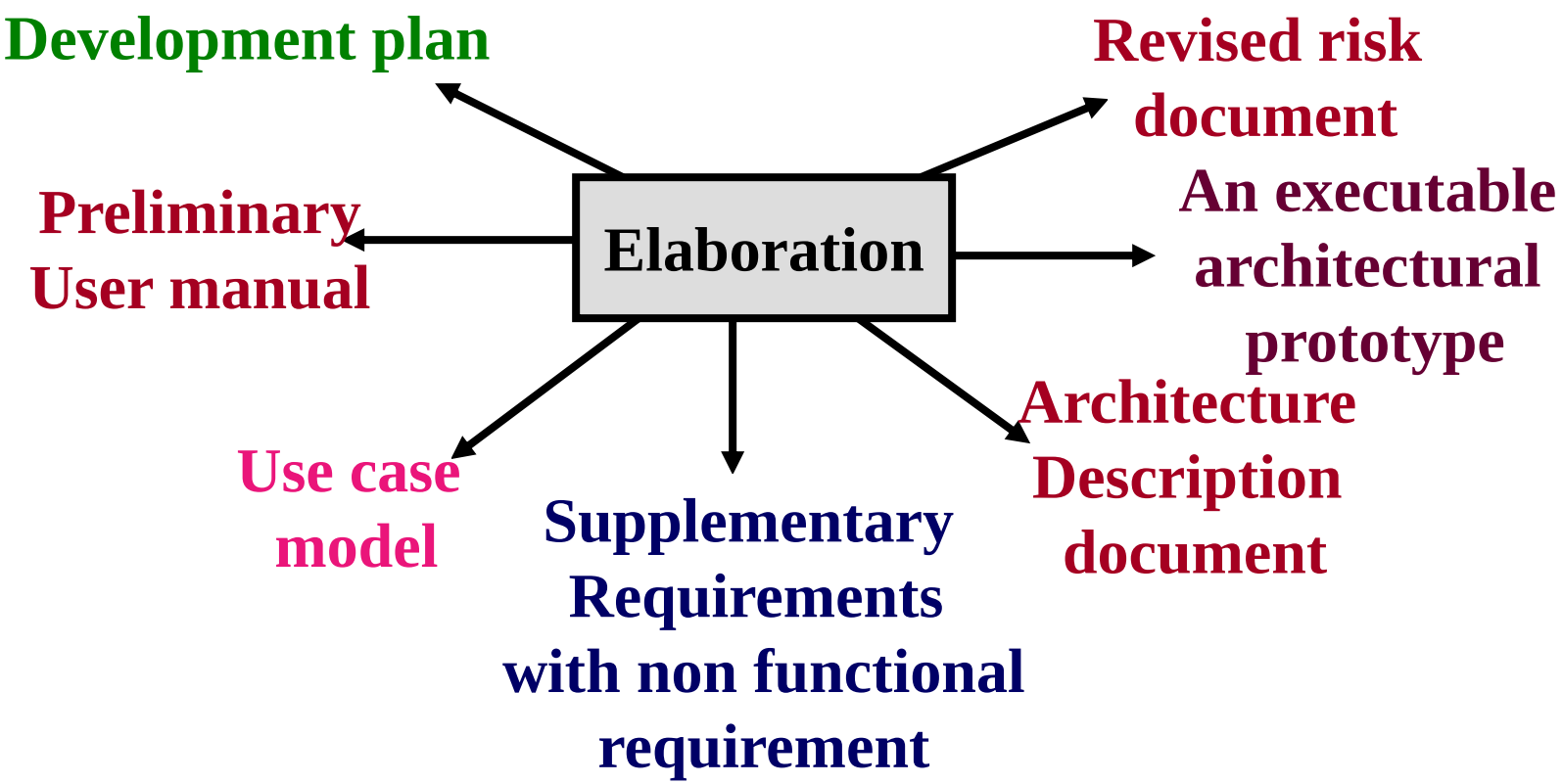


Elaboration Phase

The elaboration phase has the following objectives:

- Establishing architectural foundations.
- Design of use case model.
- Elaborating the process, infrastructure & development environment.
- Selecting component.
- Demonstrating that architecture support the vision at reasonable cost & within specified time.

Outcomes of Elaboration Phase

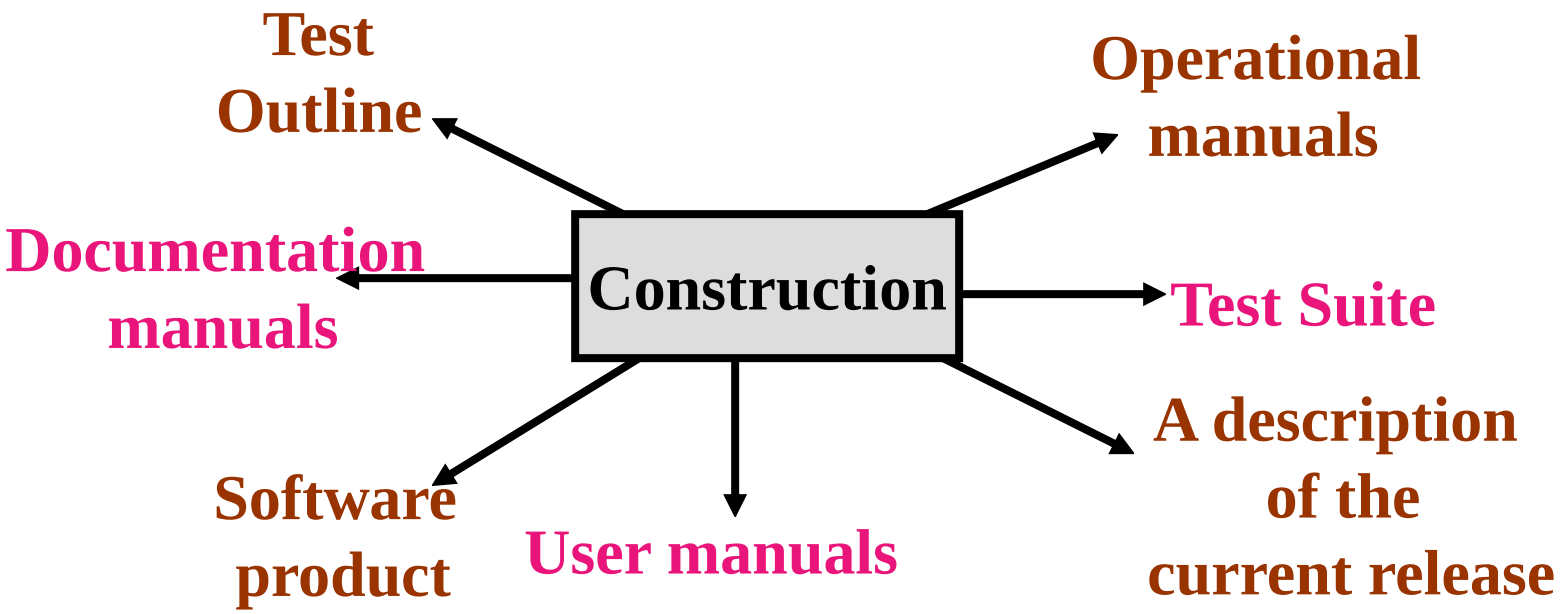


Construction Phase

The construction phase has the following objectives:

- Implementing the project.
- Minimizing development cost.
- Management and optimizing resources.
- Testing the product
- Assessing the product releases against acceptance criteria

Outcomes of Construction Phase

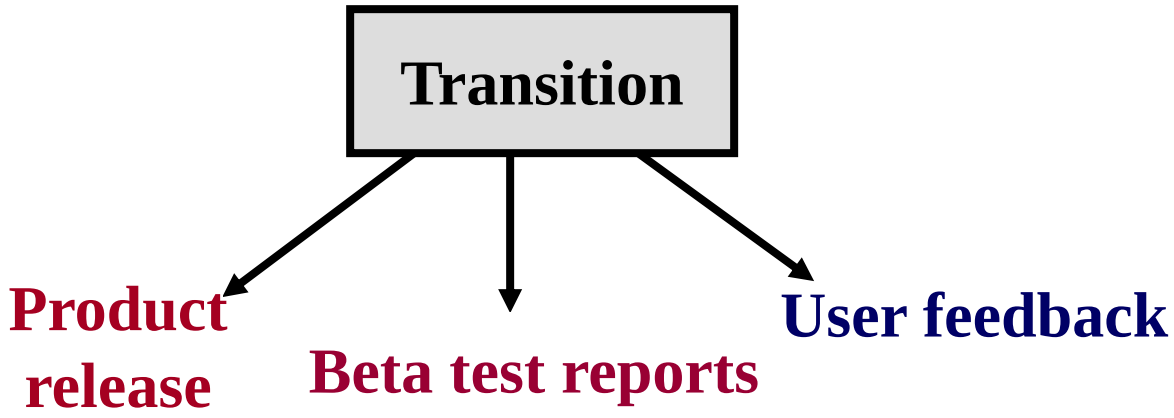


Transition Phase

The transition phase has the following objectives:

- Starting of beta testing
- Analysis of user's views.
- Training of users.
- Tuning activities including bug fixing and enhancements for performance & usability
- Assessing the customer satisfaction.

Outcomes of Transition Phase



Selection of a Life Cycle Model

Selection of a model is based on:

- Requirements
- Development team
- Users
- Project type and associated risk

Based On Characteristics Of Requirements

Requirements	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Are requirements easily understandable and defined?	**Yes**	**No**	**No**	**No**	**No**	**Yes**
Do we change requirements quite often?	**No**	**Yes**	**No**	**No**	**Yes**	**No**
Can we define requirements early in the cycle?	**Yes**	**No**	**Yes**	**Yes**	**No**	**Yes**
Requirements are indicating a complex system to be built	**No**	**Yes**	**Yes**	**Yes**	**Yes**	**No**

Based on Status of Development Team

Development team	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Less experience on similar projects?	**No**	**Yes**	**No**	**No**	**Yes**	**No**
Less domain knowledge (new to the technology)	**Yes**	**No**	**Yes**	**Yes**	**Yes**	**No**
Less experience on tools to be used	**Yes**	**No**	**No**	**No**	**Yes**	**No**
Availability of training if required	**No**	**No**	**Yes**	**Yes**	**No**	**Yes**

Based On User's Participation

Involvement of Users	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
User involvement in all phases	No	Yes	No	No	No	Yes
Limited user participation	Yes	No	Yes	Yes	Yes	No
User have no previous experience of participation in similar projects	No	Yes	Yes	Yes	Yes	No
Users are experts of problem domain	No	Yes	Yes	Yes	No	Yes

Based On Type Of Project With Associated Risk

Project type and risk	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Project is the enhancement of the existing system	No	No	Yes	Yes	No	Yes
Funding is stable for the project	Yes	Yes	No	No	No	Yes
High reliability requirements	No	No	Yes	Yes	Yes	No
Tight project schedule	No	Yes	Yes	Yes	Yes	Yes
Use of reusable components	No	Yes	No	No	Yes	Yes
Are resources (time, money, people etc.) scare?	No	Yes	No	No	Yes	No

Requirement Engineering

- Requirements describe What not How?
- Produces one large document written in natural language contains a description of what the system will do without describing how it will do it.

Crucial process steps

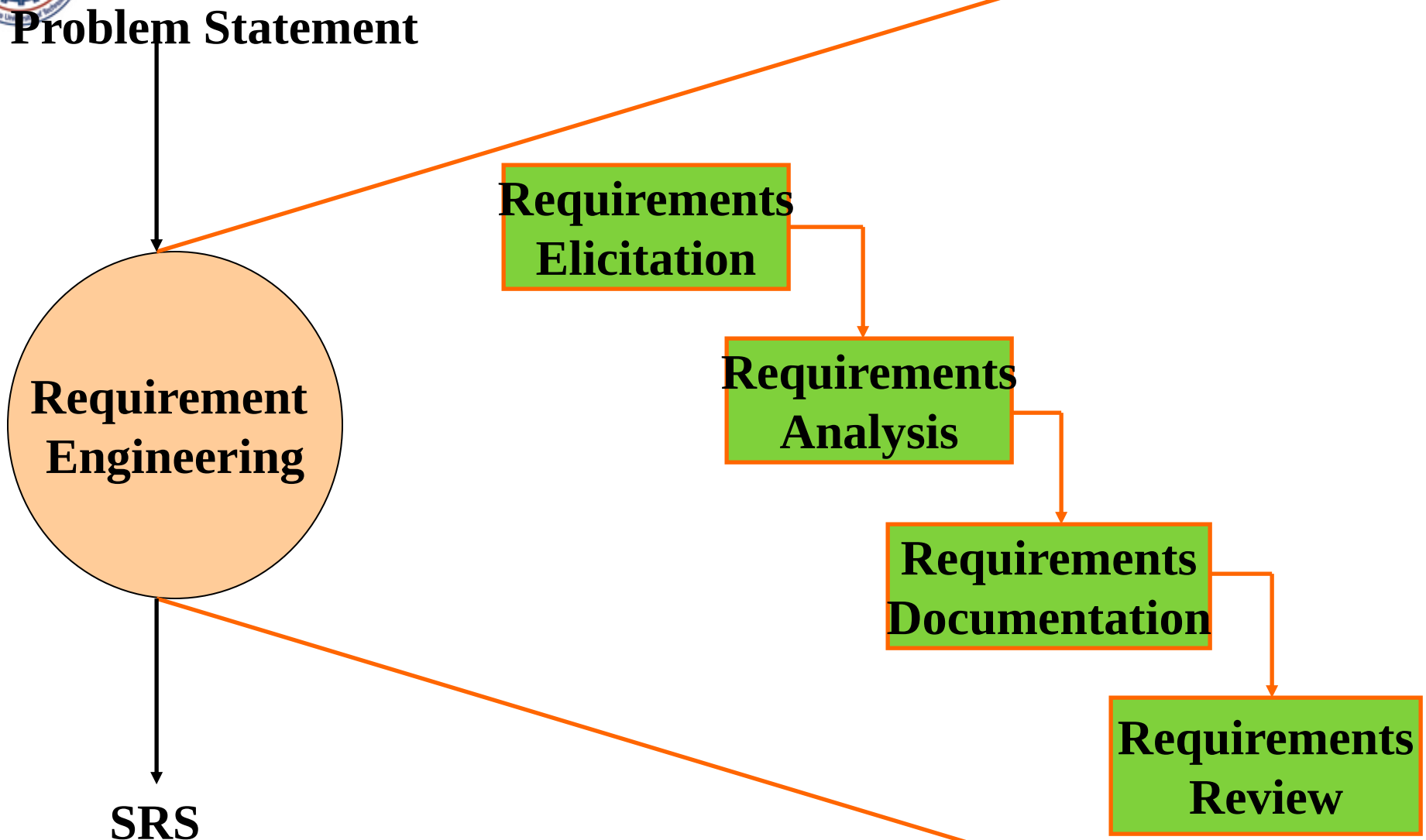
Quality of product → **Process that creates it**

Without well written document

-- Developers do not know what to build

-- Customers do not know what to expect

-- What to validate



Crucial Process Steps of requirement engineering

Requirement Engineering

- Requirement Engineering is the disciplined application of proven principles, methods, tools, and notations to describe a proposed system's intended behavior and its associated constraints.
- SRS may act as a contract between developer and customer.

Requirements are difficult to uncover because of

- Requirements change
- Over reliance on CASE Tools
- Tight project Schedule
- Communication barriers
- Market driven software development
- Lack of resources

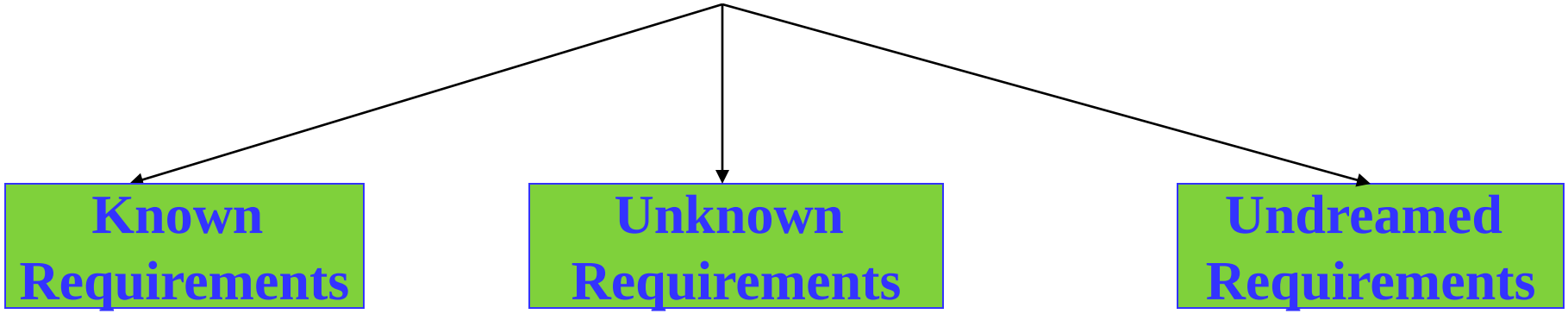
Requirement Engineering

Example

A University wish to develop a software system for the student result management of its M.Tech Programme. A problem statement is to be prepared for the software development company. The problem statement may give an overview of the existing system and broad expectations from the new software system.

Types of Requirements

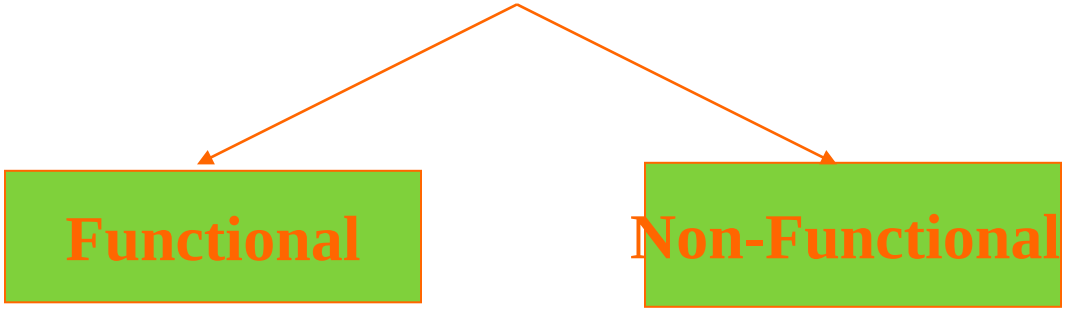
Types of Requirement



Stakeholder: Anyone who should have some direct or indirect influence on the system requirements.

--- User --- Affected persons

Requirements



Types of Requirement

- Functional requirements describe what the software has to do. They are often called product features.
- Non Functional requirements are mostly quality requirements. That stipulate how well the software does, what it has to do.

Availability
Reliability
Usability
Flexibility

} **For Users**

Maintainability
Portability
Testability

} **For Developers**

Types of Requirement

User and system requirements

- User requirements are written for the users and include functional and non functional requirement.
- System requirements are derived from user requirement.
- The user system requirements are the parts of software requirement and specification (SRS) document.

Feasibility Study

- **Is cancellation of a project a bad news?**

  As per IBM report, "31% projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% & for every 100 projects, there are 94 restarts.

- **How do we cancel a project with the least work?**

➡ **CONDUCT A FEASIBILTY STUDY**

Types of Feasibility Study

- Economic Feasibility
- Technical Feasibility
- Operational Feasibility
- Social Feasibility
- Behavioural Feasibility
- Legal Feasibility

Feasibility Study

Feasibility depends upon non technical issues like:

- Are the project's cost and schedule assumption realistic?
- Does the business model realistic?
- Is there any market for the product?

Feasibility Study

Purpose of feasibility study

- Evaluation or analysis of the potential impact of a proposed project or program.

Focus of feasibility studies

- Is the product concept viable?
- Will it be possible to develop a product that matches the project's vision statement?
- What are the current estimated cost and schedule for the project?

Feasibility Study

Focus of feasibility studies

- How big is the gap between the original cost & schedule targets & current estimates?
- Is the business model for software justified when the current cost & schedule estimate are considered?
- Have the major risks to the project been identified & can they be surmounted?
- Is the specifications complete & stable enough to support remaining development work?

Feasibility Study

Focus of feasibility studies

- Have users & developers been able to agree on a detailed user interface prototype? If not, are the requirements really stable?

- Is the software development plan complete & adequate to support further development work?

Requirements Elicitation

Perhaps

- Most critical
- Most error prone
- Most difficult
- Most communication intensive

Succeed

→ **effective customer developer partnership**

Selection of any method

- It is the only method that we know
- It is our favorite method for all situations
- Understand that this method is effective in the present circumstances.

Normally we rely on first two reasons.

Methods of Requirements Elicitation

- Interviews
- Brainstorming Sessions
- Facilitated Application specification Techniques
- Quality Function Deployment
- The Use Case Approach
- Questionnaires
- On Site Observations

Interviews

Both parties have a common goal

- Open ended
- Structured

} Interview → Success of the project

Selection of stakeholder

- Entry level personnel
- Middle level stakeholder
- Managers
- Users of the software (Most important)

Interviews

Types of questions askes:

- Any problems with existing system
- Any Calculation errors
- Possible reasons for malfunctioning
- No. of Student Enrolled
- Possible benefits
- Satisfied with current policies

Interviews

- How are you maintaining the records of previous students?
- Any requirement of data from other system
- Any specific problems
- Any additional functionality
- Most important goal of the proposed development

At the end, we may have wide variety of expectations from the proposed software.

Brainstorming Sessions

It is a group technique

↓

Group Discussions

New ideas Quickly | **Creative Thinking**

Prepare long list of requirements

Categorized
Prioritized
Pruned

Idea is to generate views, not to vet them.

Brainstorming Sessions

Groups

1. Users 2. Middle Level managers 3. Total Stakeholders

A Facilitator may handle group bias, conflicts carefully.

- Facilitator may follow a published agenda
- Every idea will be documented in a way that everyone can see it.
- A detailed report is prepared.

Facilitated Application specification Techniques

FAST

- Similar to brainstorming sessions.
- Team oriented approach
- Creation of joint team of customers and developers

Facilitated Application specification Techniques

Guidelines

- Arrange a meeting at a neutral site.
- Establish rules for participation.
- Informal agenda to encourage free flow of ideas.
- Appoint a facilitator.
- Prepare definition mechanism board, worksheets, wall stickier.
- Participants should not criticize or debate.

Each attendee is asked to make a list of objects that are:

Facilitated Application specification Techniques

FAST session Preparations

- Part of environment that surrounds the system.
- Produced by the system.
- Used by the system.
  - List of constraints
  - Functions
  - Performance criteria

Activities of FAST session

- Every participant presents his/her list
- Combine list for each topic
- Discussion
- Consensus list
- Sub teams for mini specifications
- Presentations of mini-specifications
- Validation criteria
- A sub team to draft specifications

Quality Function Deployment

Incorporate voice of the customer

Technical requirements

Documented

Prime concern is customer satisfaction

What is important for customer?

- Normal requirements
- Expected requirements
- Exciting requirements

Quality Function Deployment

Steps

- Identify stakeholders
- List out requirements
- Degree of importance to each requirement.

Quality Function Deployment

5 Points : V. Important

4 Points : Important

3 Points : Not Important but nice to have

2 Points : Not important

1 Points : Unrealistic, required further exploration

Requirement Engineer may categorize like:

- It is possible to achieve
- It should be deferred & Why
- It is impossible and should be dropped from consideration

First Category requirements will be implemented as per priority assigned with every requirement.

The Use Case Approach

Ivar Jacobson & others introduced Use Case approach for elicitation & modeling.

Use Case – give functional view

The terms

Use Case

Use Case Scenario

Use Case Diagram

} **Often Interchanged**

But they are different

Use Cases are structured outline or template for the description of user requirements modeled in a structured language like English.

The Use Case Approach

- Use case Scenarios are unstructured descriptions of user requirements.

- Use case diagrams are graphical representations that may be decomposed into further levels of abstraction.

Components of Use Case approach

Actor:

An actor or external agent, lies outside the system model, but interacts with it in some way.

Actor → Person, machine, information System

The Use Case Diagram

- Cockburn distinguishes between Primary and secondary actors.
- A Primary actor is one having a goal requiring the assistance of the system.
- A Secondary actor is one from which System needs assistance.

Use Cases

A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied.

Use Case

- It describes the sequence of interactions between actors and the system necessary to deliver the services that satisfies the goal.
- Alternate sequence
- System is treated as black box.

Thus, Use Case captures who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals.

Use Case

- Defines all behavior required of the system, bounding the scope of the system.
- Jacobson & others proposed a template for writing Use cases as shown below:

1. Introduction
   Describe a quick background of the use case.
2. Actors
   List the actors that interact and participate in the use cases.
3. Pre Conditions
   Pre conditions that need to be satisfied for the use case to perform.
4. Post Conditions
   Define the different states in which we expect the system to be in, after the use case executes.

Use Case

5. Flow of Events

5.1 Basic Flow

List the primary events that will occur when this use case is executed

5.2 Alternative Flows

Any Subsidiary events that can occur in the use case should be separately listed. List each such event as an alternative flow.A use case can have many alternative flows as required.

6.Special Requirements

Business rules should be listed for basic & information flows as special requirements in the use case narration .These rules will also be used for writing test cases. Both success and failures scenarios should be described.

7.Use Case relationships

For Complex systems it is recommended to document the relationships between use cases. Listing the relationships between use cases also provides a mechanism for traceability

Use Case Template.

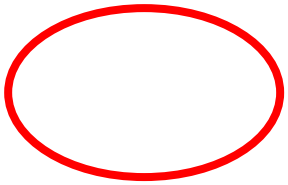
Use Case Guidelines

- Identify all users
- Create a user profile for each category of users including all roles of the users play that are relevant to the system.
- Create a use case for each goal, following the use case template maintain the same level of abstraction throughout the use case. Steps in higher level use cases may be treated as goals for lower level (i.e. more detailed), sub-use cases.
- Structure the use case
- Review and validate with users.

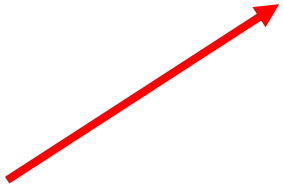
Use Case Diagrams

- Represents what happens when actor interacts with a system.
- Captures functional aspect of the system.

Actor



Use Case



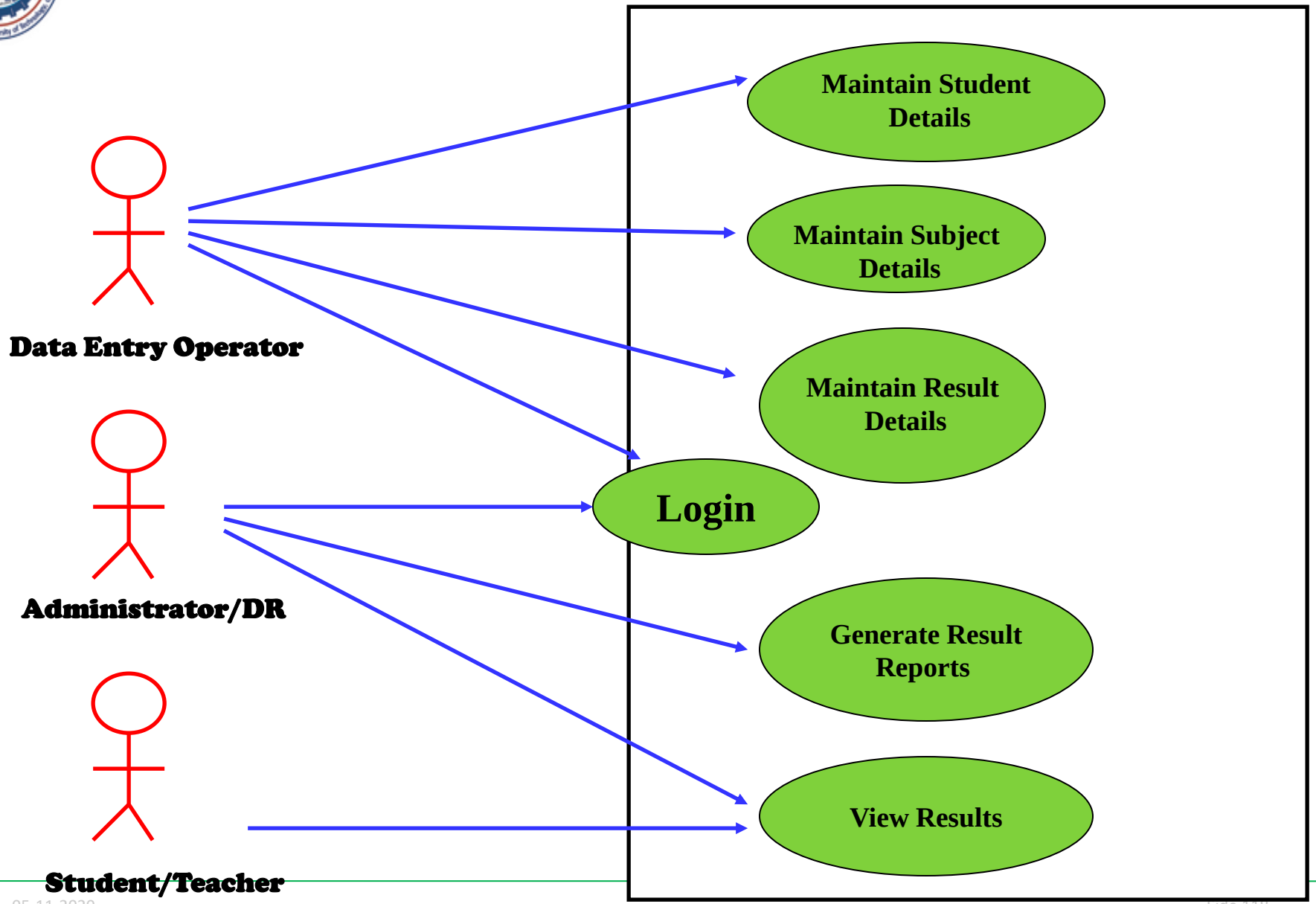
Relationship between actors and use case and/or between the use cases.

- Actors appear outside the rectangle.
- Use cases within rectangle providing functionality.
- Relationship association is a solid line between actor & use cases.

Use Case Diagram

- Use cases should not be used to capture all the details of the system.
- Only significant aspects of the required functionality
- No design issues
- Use Cases are for “what” the system is , not “how” the system will be designed
- Free of design characteristics

Use case diagram for Result Management System



Requirements Elicitation

1.Maintain student Details

Add/Modify/update students details like name, address.

2. Maintain subject Details

Add/Modify/Update Subject information semester wise

3.Maintain Result Details

Include entry of marks and assignment of credit points for each paper.

4. Login

Use to Provide way to enter through user id & password.

5. Generate Result Report

Use to print various reports

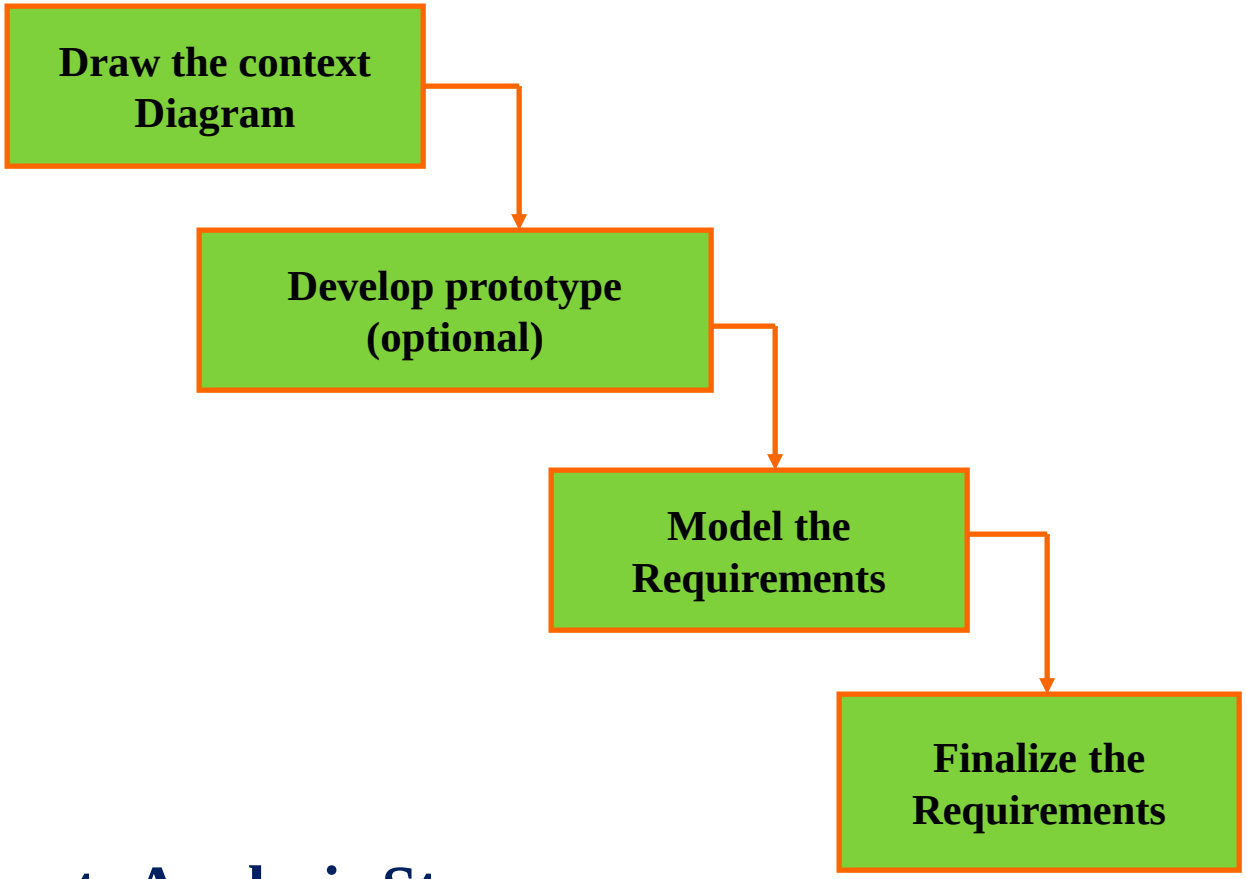
6. View Result

- According to course code
- According to Enrollment number/roll number

Requirements Analysis

We analyze, refine and scrutinize requirements to make consistent & unambiguous requirements.

Steps

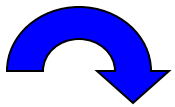
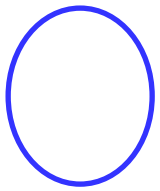


Requirements Analysis Steps

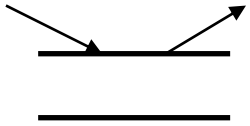
Data Flow Diagrams

DFD show the flow of data through the system.

- All names should be unique
- It is not a flow chart
- Suppress logical decisions
- Defer error conditions & handling until the end of the analysis

Symbol	Name	Function
	**Data Flow**	**Connect process**
	**Process**	**Perform some transformation of its input data to yield output data.**

Data Flow Diagram

Symbol	Name	Function
	Source or sink	A source of system inputs or sink of system outputs
	Data Store	A repository of data, the arrowhead indicate net input and net outputs to store

Leveling

- DFD represent a system or software at any level of abstraction.
- A level 0 DFD is called fundamental system model or context model represents entire software element as a single bubble with input and output data indicating by incoming & outgoing arrows.

Data Dictionaries

DFD ⟶ **DD**

Data Dictionaries are simply repositories to store information about all data items defined in DFD.

Includes :

- Name of data item
- Aliases (other names for items)
- Description/Purpose
- Related data items
- Range of values
- Data flows
- Data structure definition

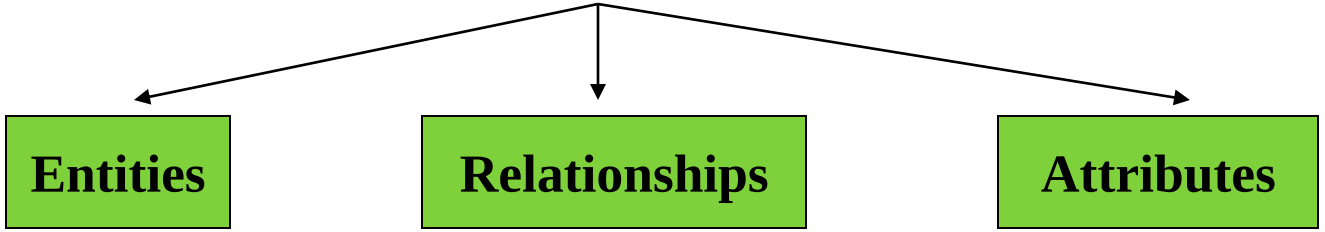
Data Dictionaries

Notation	Meaning
x= a+b	x consists of data element a & b
x={a/b}	x consists of either a or b
x=(a)	x consists of an optional data element a
x= y{a}	x consists of y or more occurrences
x={a}z	x consists of z or fewer occurrences of a
x=y{a}z	x consists of between y & z occurrences of a{

Entity-Relationship Diagrams

Entity-Relationship Diagrams

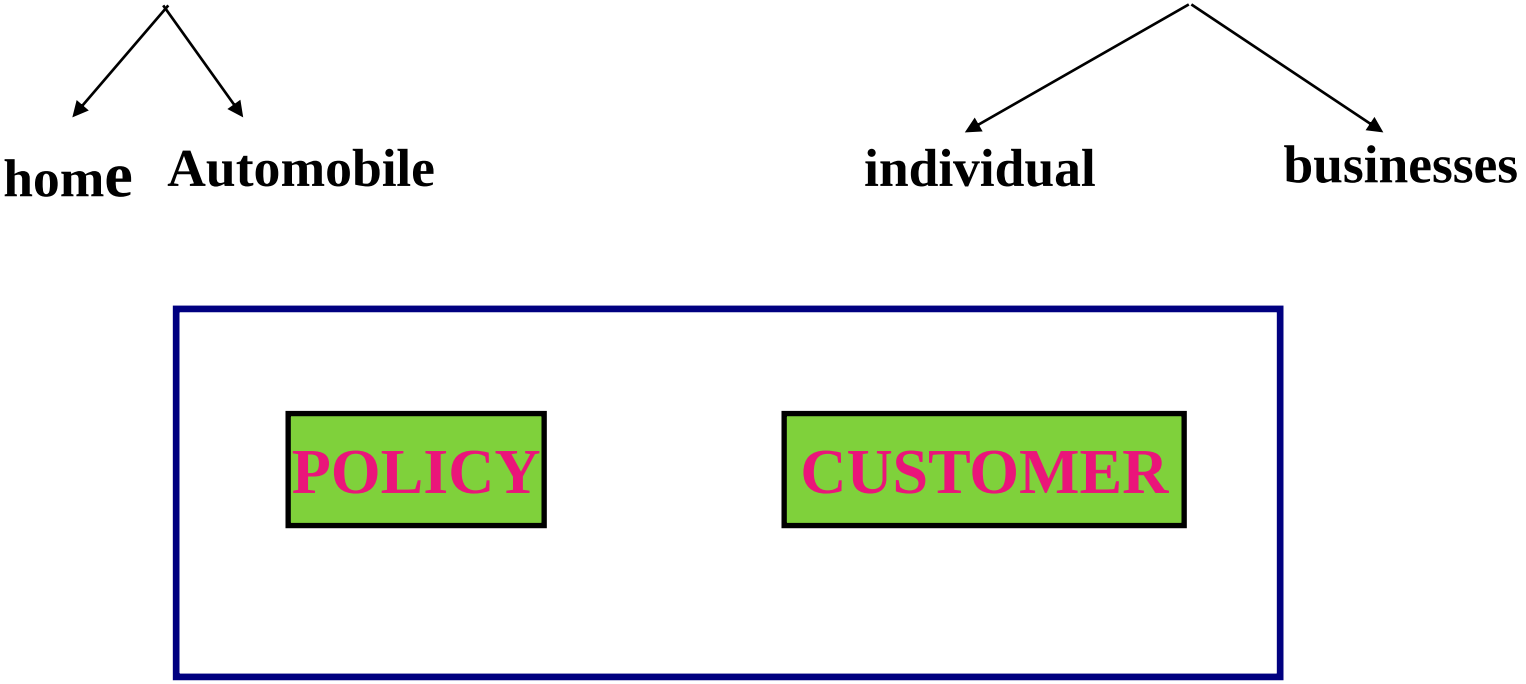
It is a detailed logical representation of data for an organization and uses three main constructs.



Entities

- Fundamental thing about which data may be maintained. Each entity has its own identity.
- Entity Type is the description of all entities to which a common definition and common relationships and attributes apply.

Entity-Relationship Diagrams



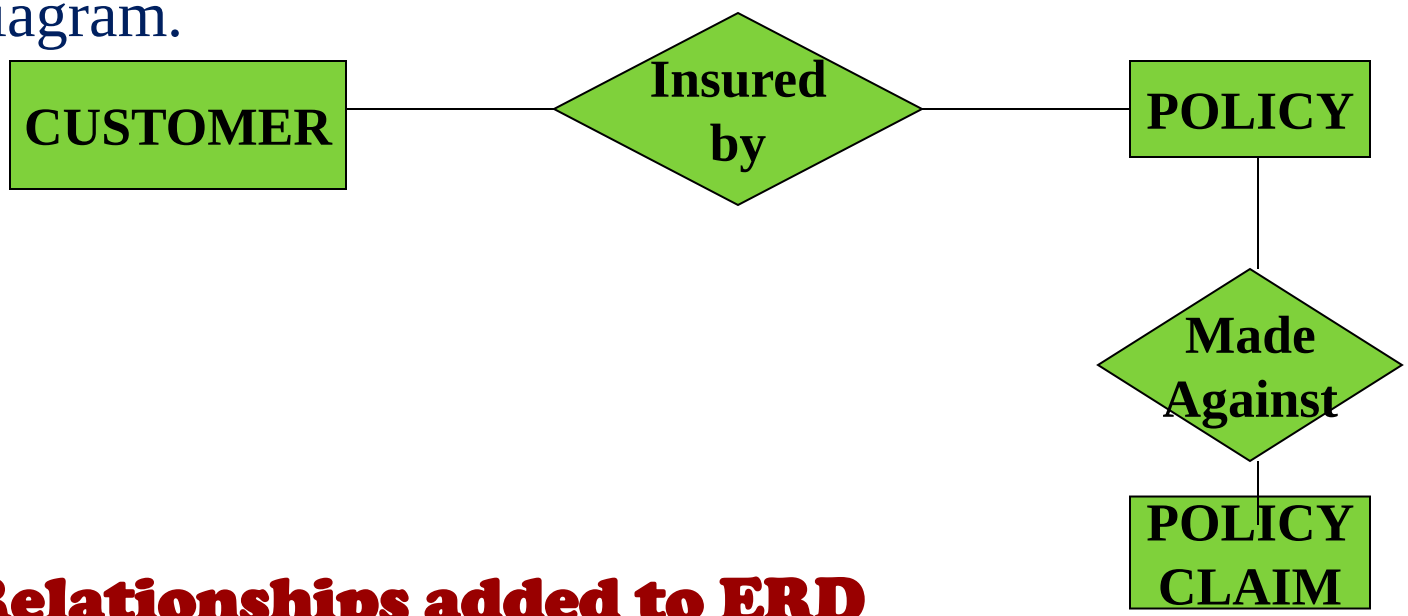
Entity-Relationship Diagrams

Relationships

A relationship is a reason for associating two entity types.

Binary relationships ⟶ involve two entity types

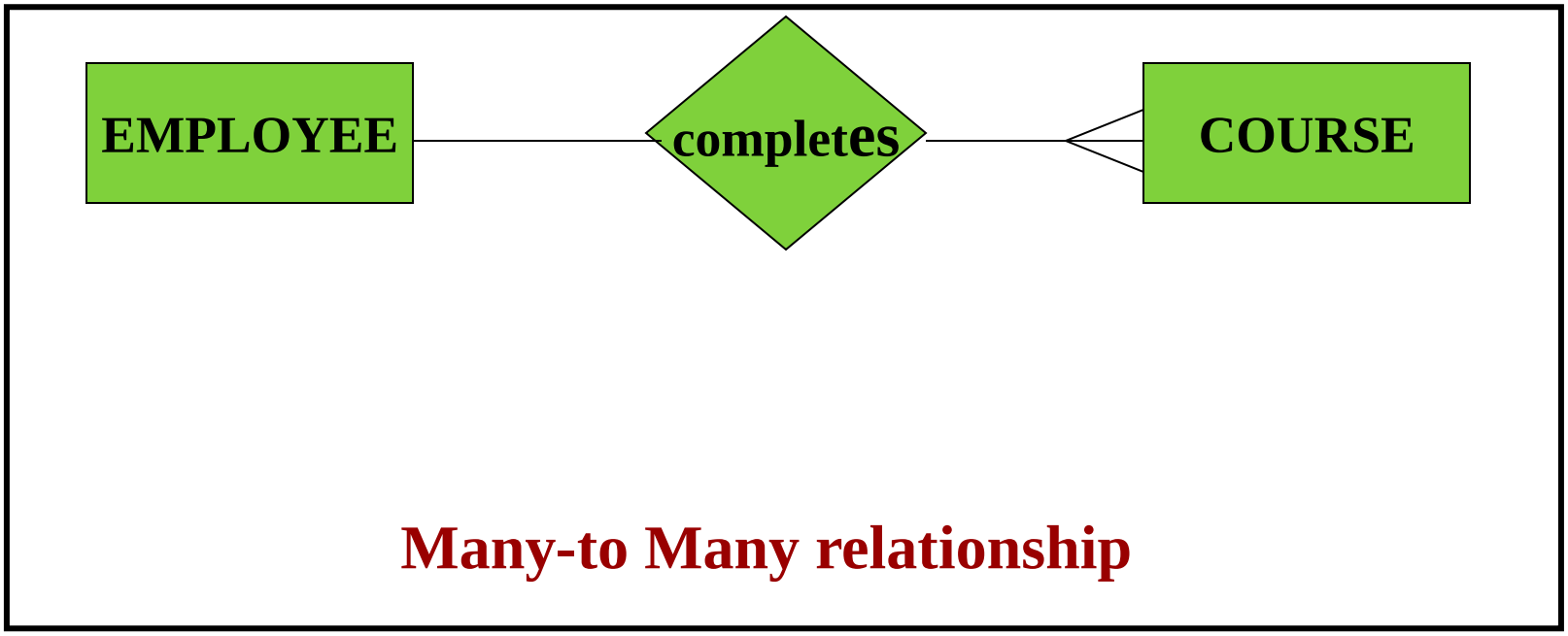
- A CUSTOMER is insured by a POLICY. A POLICY CLAIM is made against a POLICY.
- Relationships are represented by diamond notation in a ER diagram.



Relationships added to ERD

Entity-Relationship Diagrams

- A training department is interested in tracking which training courses each of its employee has completed.

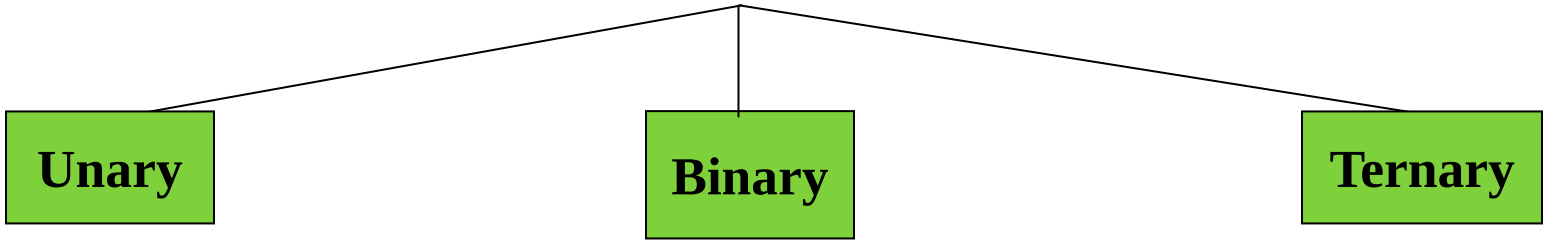


- Each employee may complete more than one course, and each course may be completed by more than one employee.

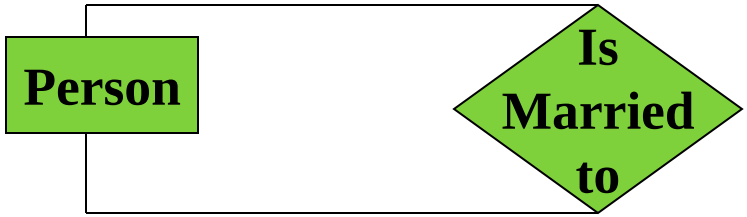
Entity-Relationship Diagrams

Degree of relationship

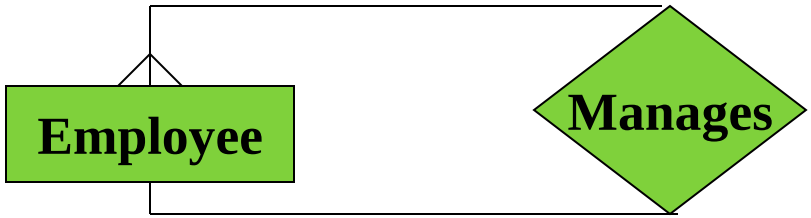
It is the number of entity types that participates in that relationship.



Unary relationship



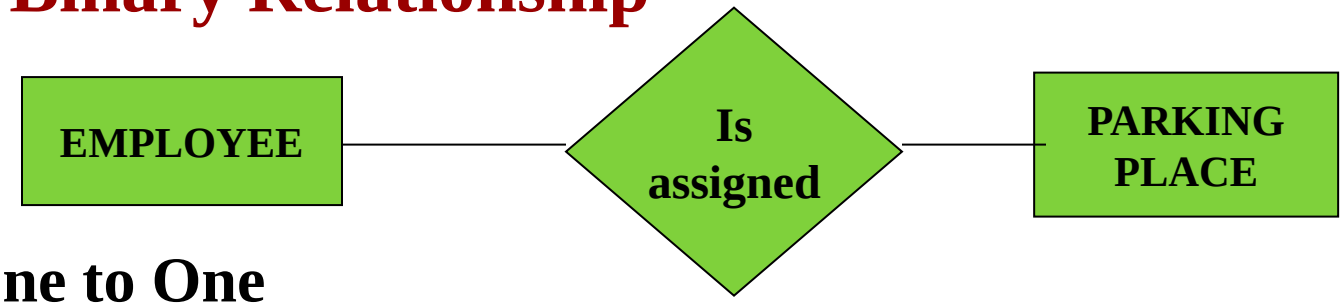
One to One



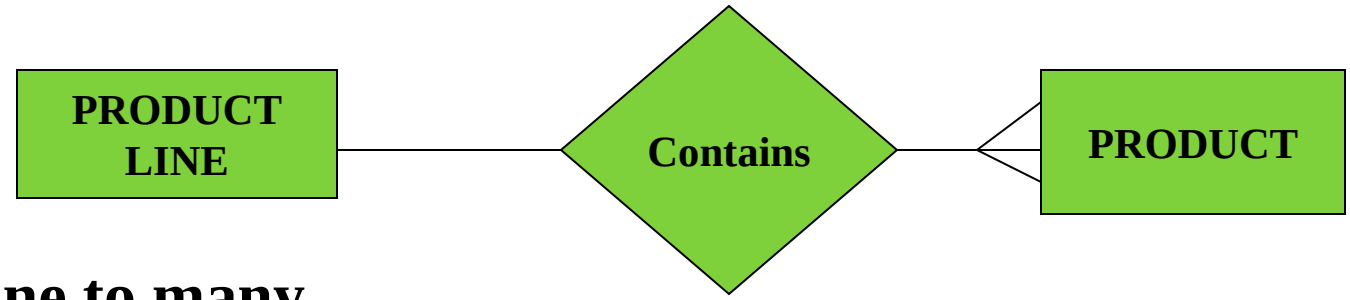
One to many

Entity-Relationship Diagrams

Binary Relationship



One to One

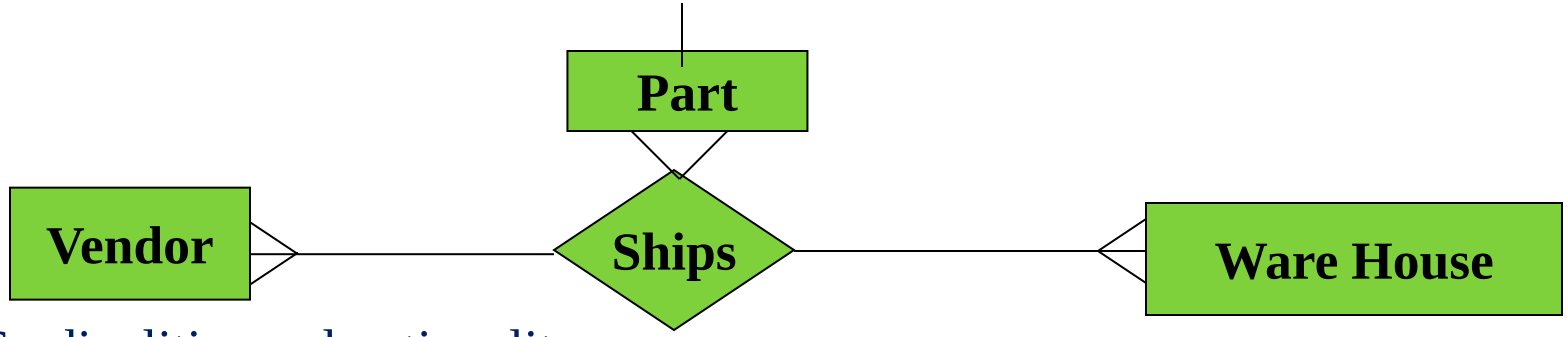


One to many

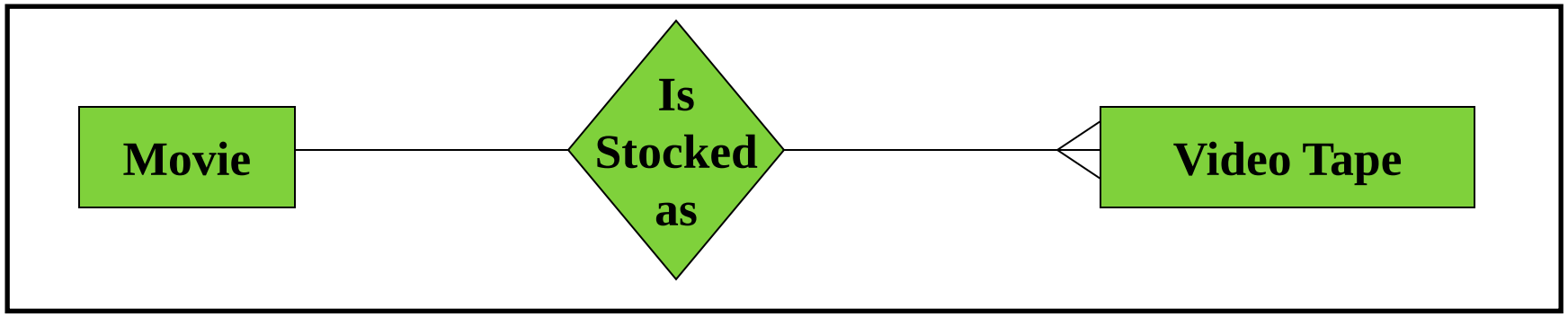
Many to many

Entity-Relationship Diagrams

Ternary relationship



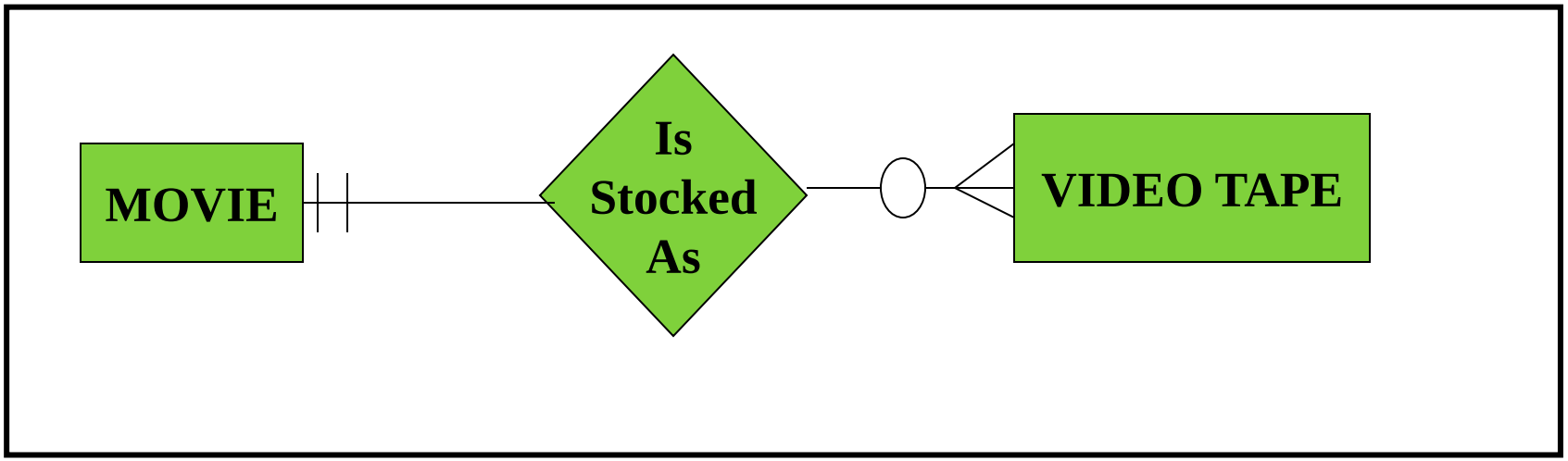
- Cardinalities and optionality
- Two entity types A, B, connected by a relationship.
- The cardinality of a relationship is the number of instances of entity B that can be associated with each instance of entity A



Entity-Relationship Diagrams

- Minimum cardinality is the minimum number of instances of entity B that may be associated with each instance of entity A.

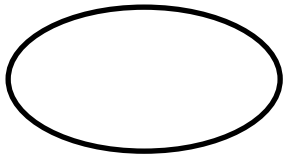
- Minimum no. of tapes available for a movie is zero.We say VIDEO TAPE is an optional participant in the is-stocked-as relationship.



Entity-Relationship Diagrams

Attributes

- Each entity type has a set of attributes associated with it.
- An attribute is a property or characteristic of an entity that is of interest to organization.



Attribute

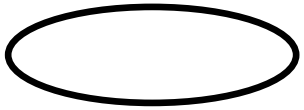
Entity-Relationship Diagrams

- A candidate key is an attribute or combination of attributes that uniquely identifies each instance of an entity type.

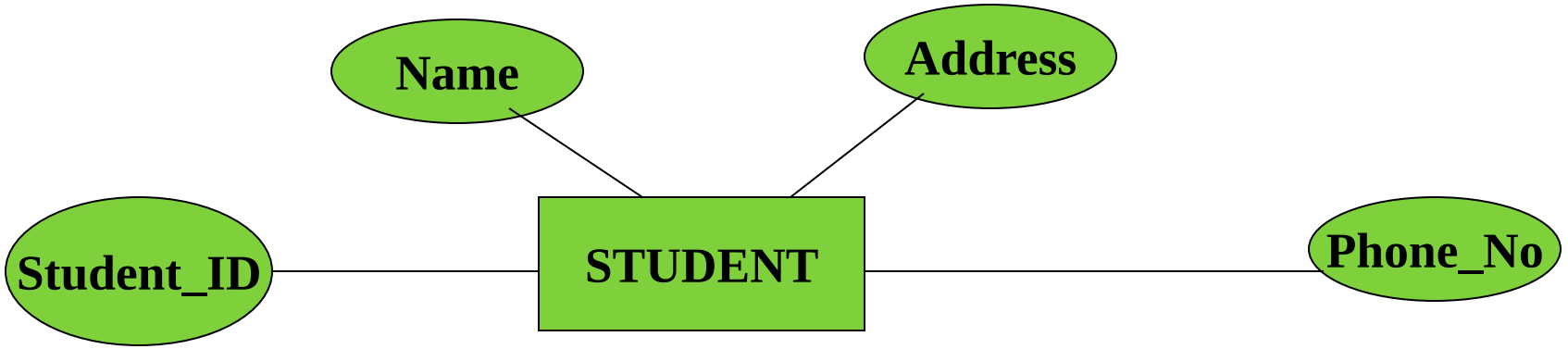
Student_ID ⟶ **Candidate Key**

- If there are more candidate keys, one of the key may be chosen as the Identifier.

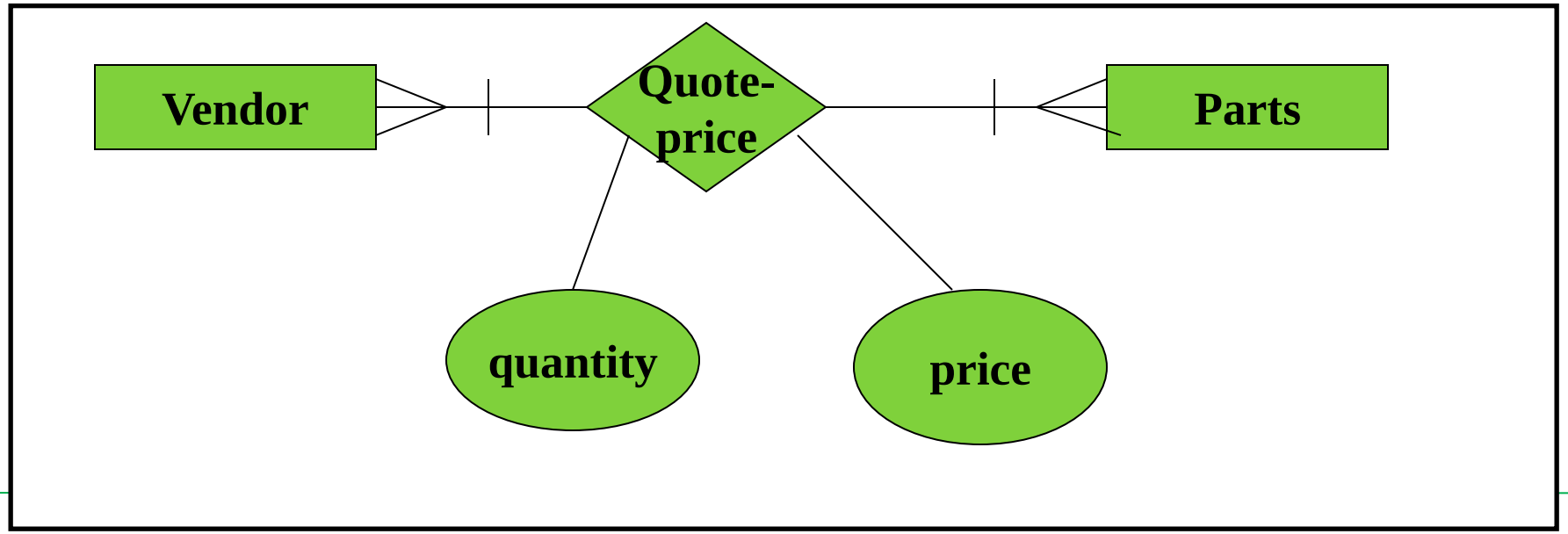
- It is used as unique characteristic for an entity type.

Identifier 

Entity-Relationship Diagrams



- Vendors quote prices for several parts along with quantity of parts.
- Draw an E-R diagram.



Approaches to problem analysis

- List all inputs, outputs and functions.
- List all functions and then list all inputs and outputs associated with each function.

Structured requirements definition (SRD)

Step1

Define a user level DFD. Record the inputs and outputs for each individual in a DFD.

Step2

Define a combined user level DFD.

Step3

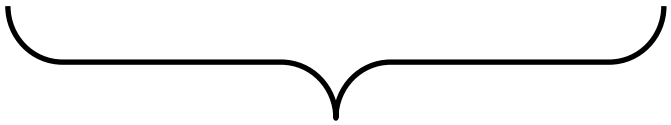
Define application level DFD.

Step4

Define application level functions.

Requirements Documentation

- This is the way of representing requirements in a consistent format
- SRS serves many purpose depending upon who is writing it.
- **written by customer**
- **written by developer**



- Serves as contract between customer & developer.

Requirements Documentation

Nature of SRS

Basic Issues

- Functionality
- External Interfaces
- Performance
- Attributes
- Design constraints imposed on an Implementation

Requirements Documentation

SRS Should

- Correctly define all requirements
- not describe any design details
- not impose any additional constraints

Characteristics of a good SRS

An SRS Should be

- Correct
- Unambiguous
- Complete
- Consistent

Requirements Documentation

- Ranked for important and/or stability
- Verifiable
- Modifiable
- Traceable

Requirements Documentation

Correct

An SRS is correct if and only if every requirement stated therein is one that the software shall meet.

Unambiguous

An SRS is unambiguous if and only if, every requirement stated therein has only one interpretation.

Complete

An SRS is complete if and only if, it includes the following elements

- All significant requirements, whether related to functionality, performance, design constraints, attributes or external interfaces.

Requirement Documentation

- Responses to both valid & invalid inputs.
- Full Label and references to all figures, tables and diagrams in the SRS and definition of all terms and units of measure.

Consistent

An SRS is consistent if and only if, no subset of individual requirements described in it conflict.

Ranked for importance and/or Stability

If an identifier is attached to every requirement to indicate either the importance or stability of that particular requirement.

Requirements Documentation

Verifiable

An SRS is verifiable, if and only if, every requirement stated therein is verifiable.

Modifiable

An SRS is modifiable, if and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining structure and style.

Traceable

An SRS is traceable, if the origin of each of the requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.

Software Requirement Specification (SRS)

Organization of the SRS

IEEE has published guidelines and standards to organize an SRS.

1. Introduction

- Purpose
- Scope
- Definition, Acronyms and abbreviations
- References
- Overview

Software Requirement Specification (SRS)

2. The Overall Description

2.1 Product Perspective

- System Interfaces
- Interfaces
- Hardware Interfaces
- Software Interfaces
- Communication Interfaces
- Memory Constraints
- Operations
- Site Adaptation Requirements

Software Requirement Specification (SRS)

- Product Functions
- User Characteristics
- Constraints
- Assumptions for dependencies
- Apportioning of requirements

3. Specific Requirements

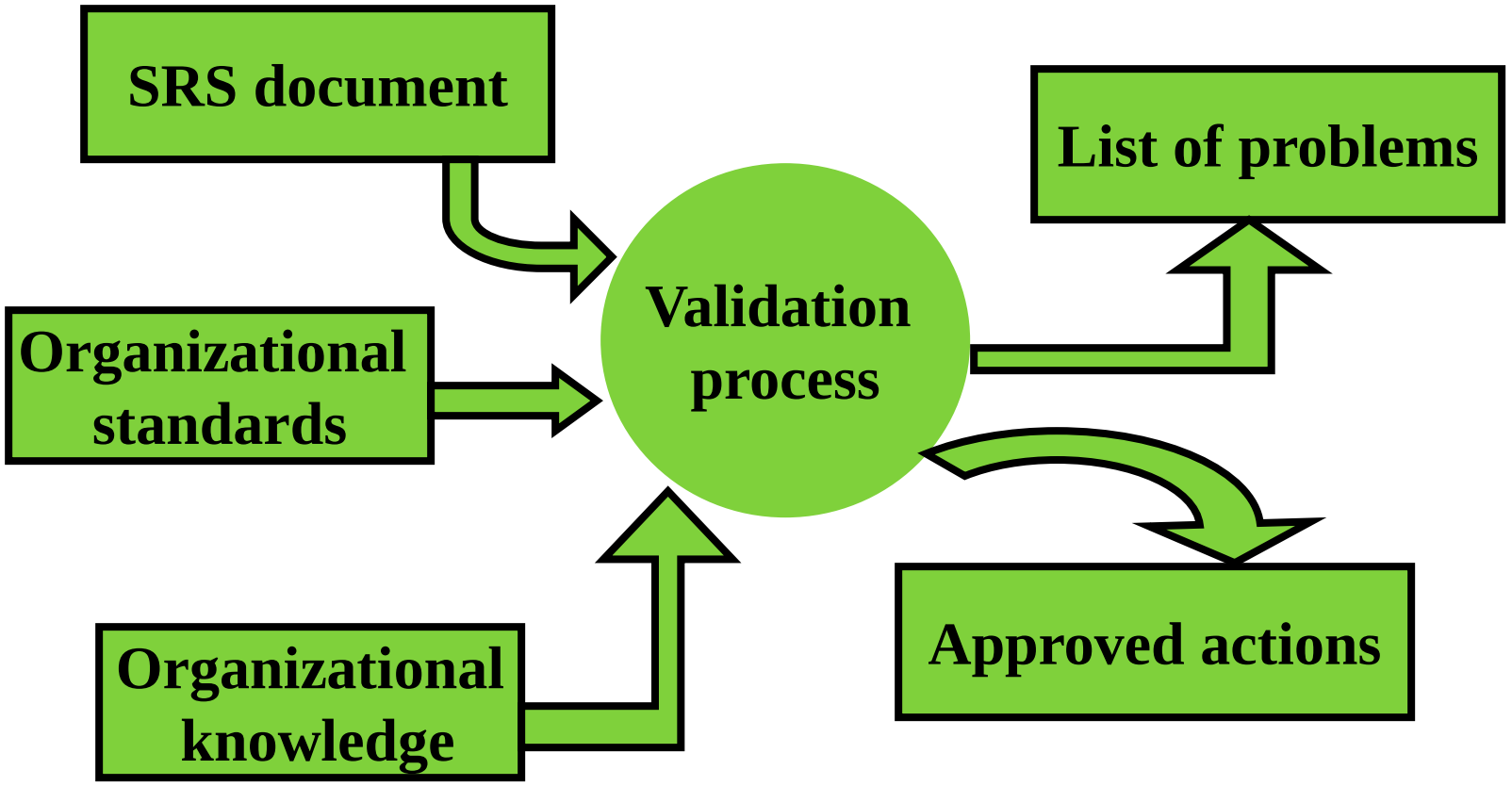
- External Interfaces
- Functions
- Performance requirements
- Logical database requirements
- Design Constraints
- Software System attributes
- Organization of specific requirements
- Additional Comments.

Software Requirement Specification (SRS)

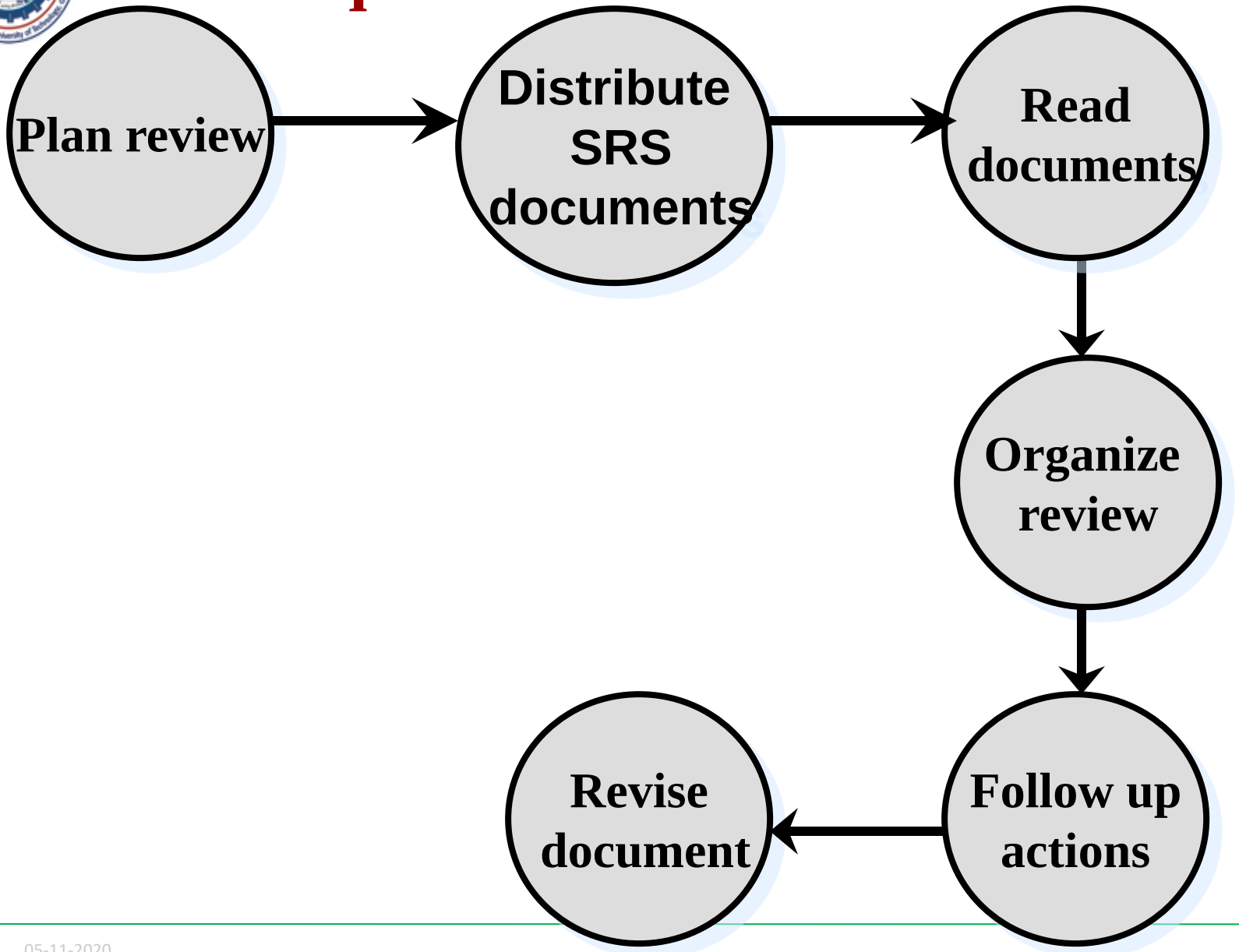
Check the document for:

- Completeness & consistency
- Conformance to standards
- Requirements conflicts
- Technical errors
- Ambiguous requirements

Requirements Validation



Requirements Review Process



Requirements Validation

Problem actions

- Requirements clarification
- Missing information
  - *find this information from stakeholders*
- Requirements conflicts
  - *Stakeholders must negotiate to resolve this conflict*
- Unrealistic requirements
  - *Stakeholders must be consulted*
- Security issues
  - *Review the system in accordance to security standards*

Requirements Management

Process of understanding and controlling changes to system requirements.

ENDURING & VOLATILE REQUIREMENTS

- Enduring requirements: They are core requirements & are related to main activity of the organization.

  Example: issue/return of a book, cataloging etc.

- Volatile requirements: likely to change during software development lifer cycle or after delivery of the product

Requirements Management Planning

- Very critical.
- Important for the success of any project.

Requirements Change Management

- Allocating adequate resources
- Analysis of requirements
- Documenting requirements
- Requirements traceability
- Establishing team communication
- Establishment of baseline

References

1. K.K Agarwal, Yogesh Singh, "Software Engineering" 3rd edition, New Age International Publishers, 2008
2. Rajib Mall, "Fundamentals of Software Engineering", Prentice-Hall of India.
3. Roger S. Pressman, "Software Engineering: A Practitioner's Approach", Mc Graw Hill Education
4. http://www.tutorialspoint.com