Software Engineering

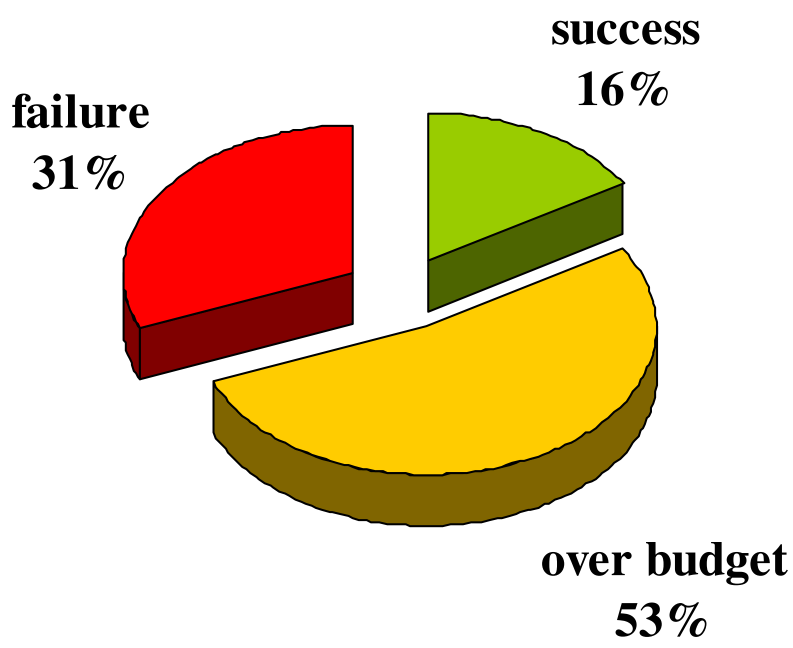
Why Software Engineering ?

- Change in nature & complexity of software
- Concept of one “guru” is over
- We all want improvement

Ready for change

The Evolving Role of Software

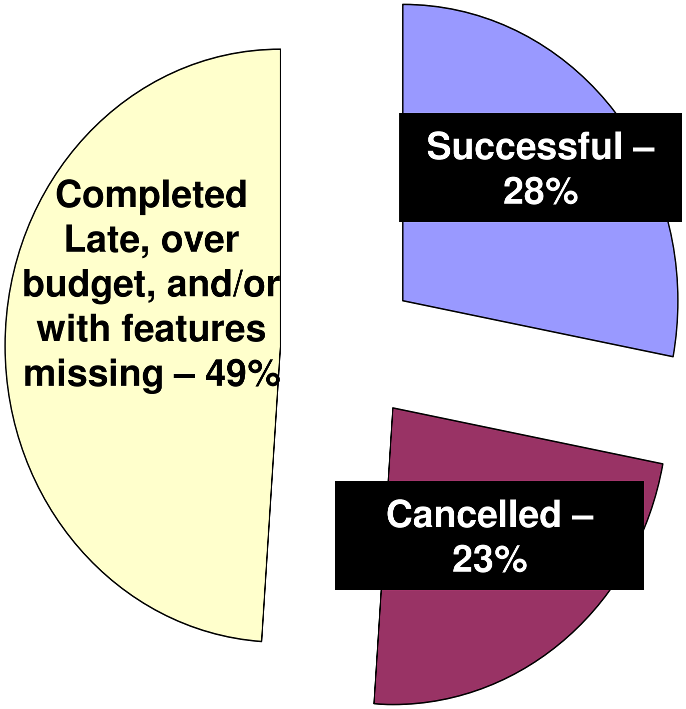
- Software industry is in Crisis!



Source: The Standish Group International, Inc. (CHAOS research)

The Evolving Role of Software

This is the SORRY state of Software Engineering Today!

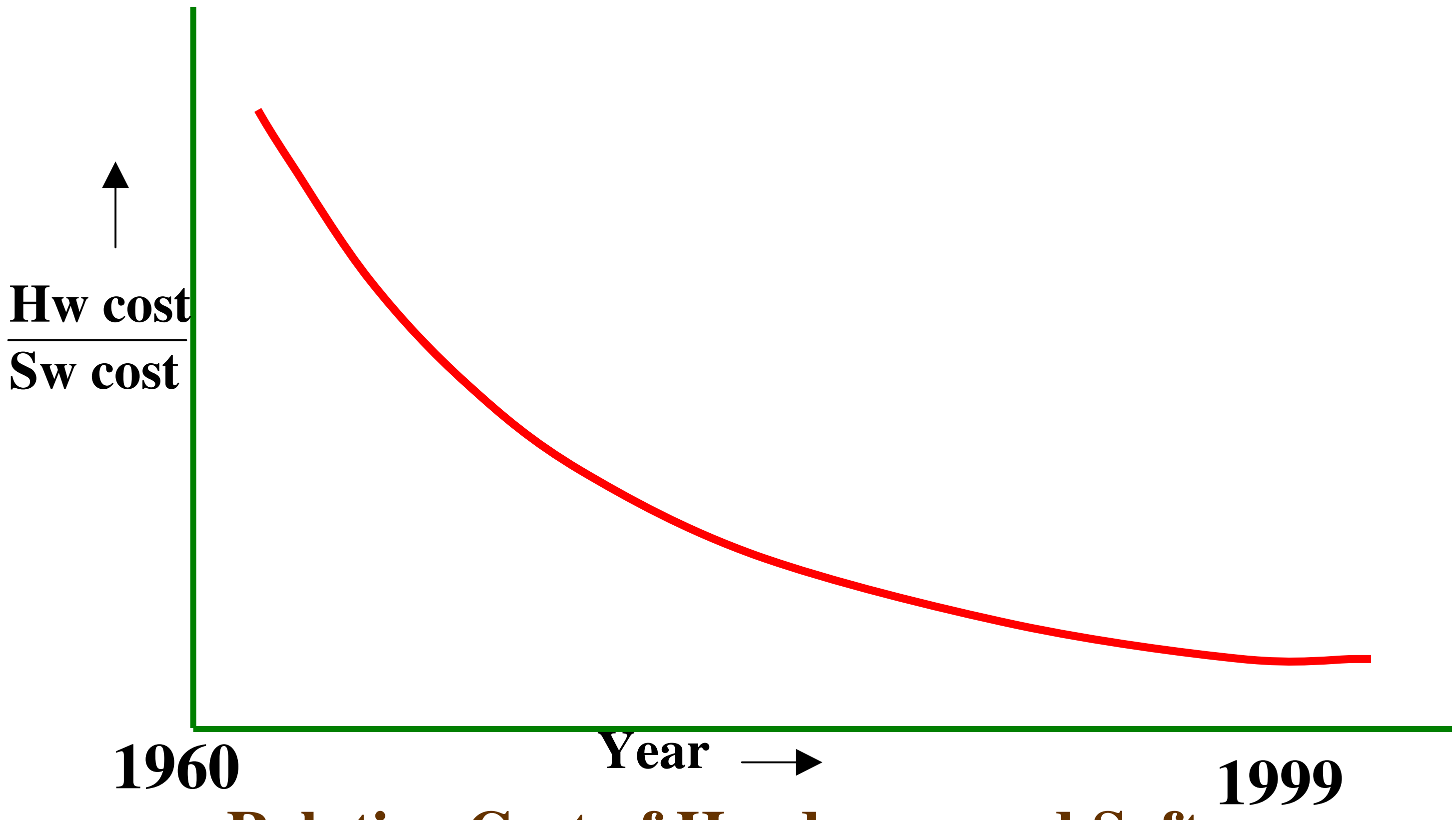


- **Data on 28,000 projects completed in 2000**

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% and for every 100 projects, there are 94 restarts".

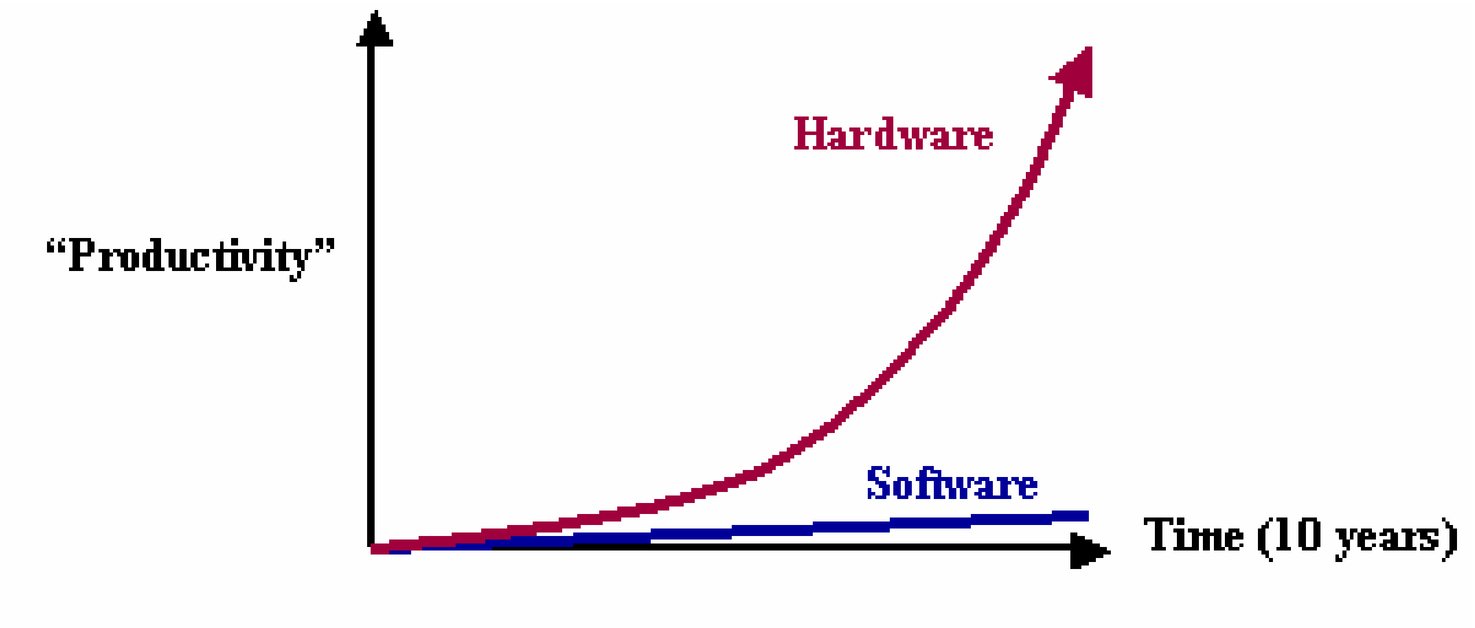
The Evolving Role of Software



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 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?
- ✓ 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.

Some Software failures

Ariane 5

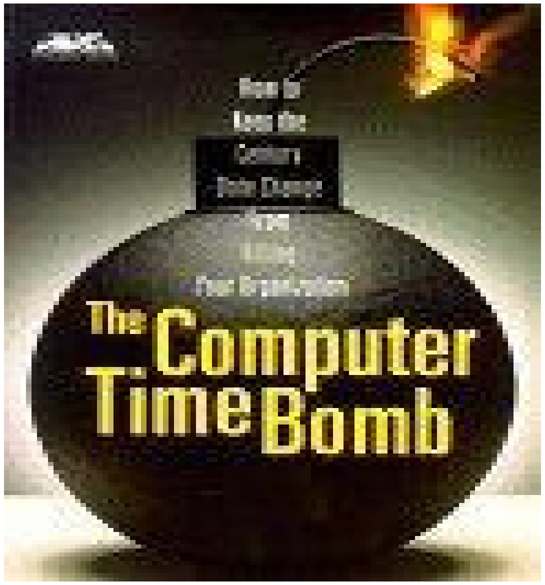
It took the European Space Agency **10 years and $7 billion** to produce Ariane 5, a giant rocket capable of hurling a pair of three-ton satellites into orbit with each launch and intended to give Europe overwhelming supremacy in the commercial space business.

The rocket was destroyed after 39 seconds of its launch, at an altitude of two and a half miles along with its payload of four expensive and uninsured scientific satellites.

Some Software failures

When the guidance system's own computer tried to convert one piece of data the sideways velocity of the rocket from a 64 bit format to a 16 bit format; the number was too big, and an overflow error resulted after 36.7 seconds. When the guidance system shutdown, it passed control to an identical, redundant unit, which was there to provide backup in case of just such a failure. Unfortunately, the second unit, which had failed in the identical manner a few milliseconds before.

Some Software failures



Y2K problem:

It was simply the ignorance about the adequacy or otherwise of using only last two digits of the year.

The 4-digit date format, like 1964, was shortened to 2-digit format, like 64.

Some Software failures

The Patriot Missile

- First time used in Gulf war
- Used as a defense from Iraqi Scud missiles
- Failed several times including one that killed 28 US soldiers in Dhahran, Saudi Arabia

Reasons:

A small timing error in the system's clock accumulated to the point that after 14 hours, the tracking system was no longer accurate. In the Dhahran attack, the system had been operating for more than 100 hours.

Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007

Some Software failures

The Space Shuttle

Part of an abort scenario for the Shuttle requires fuel dumps to lighten the spacecraft. It was during the second of these dumps that a (software) crash occurred.

...the fuel management module, which had performed one dump and successfully exited, restarted when recalled for the second fuel dump...

Some Software failures

A simple fix took care of the problem…but the programmers decided to see if they could come up with a systematic way to eliminate these generic sorts of bugs in the future. A random group of programmers applied this system to the fuel dump module and other modules.

Seventeen additional, previously unknown problems surfaced!

Some Software failures

Financial Software

Many companies have experienced failures in their accounting system due to faults in the software itself. The failures range from producing the wrong information to the whole system crashing.

Some Software failures

Windows XP

- Microsoft released Windows XP on October 25, 2001.
- On the same day company posted 18 MB of compatibility patches on the website for bug fixes, compatibility updates, and enhancements.
- Two patches fixed important security holes.

This is **Software Engineering.**

“No Silver Bullet”

The hardware cost continues to decline drastically.

However, there are desperate cries for a silver bullet something to make software costs drop as rapidly as computer hardware costs do.

But as we look to the horizon of a decade, we see no silver bullet. There is no single development, either in technology or in management technique, that by itself promises even one order of magnitude improvement in productivity, in reliability and in simplicity.

"No Silver Bullet"

The hard part of building software is the specification, design and testing of this conceptual construct, not the labour of representing it and testing the correctness of representation.

We still make syntax errors, to be sure, but they are trivial as compared to the conceptual errors (logic errors) in most systems. That is why, building software is always hard and there is inherently no silver bullet.

While there is no royal road, there is a path forward.

Is reusability (and open source) the new silver bullet?

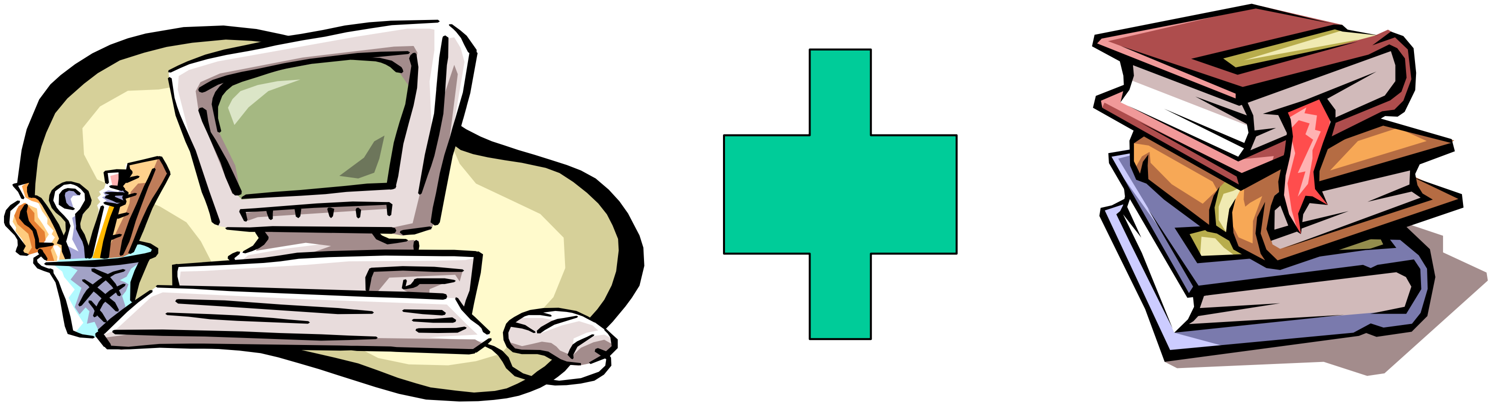
"No Silver Bullet"

The blame for software bugs belongs to:

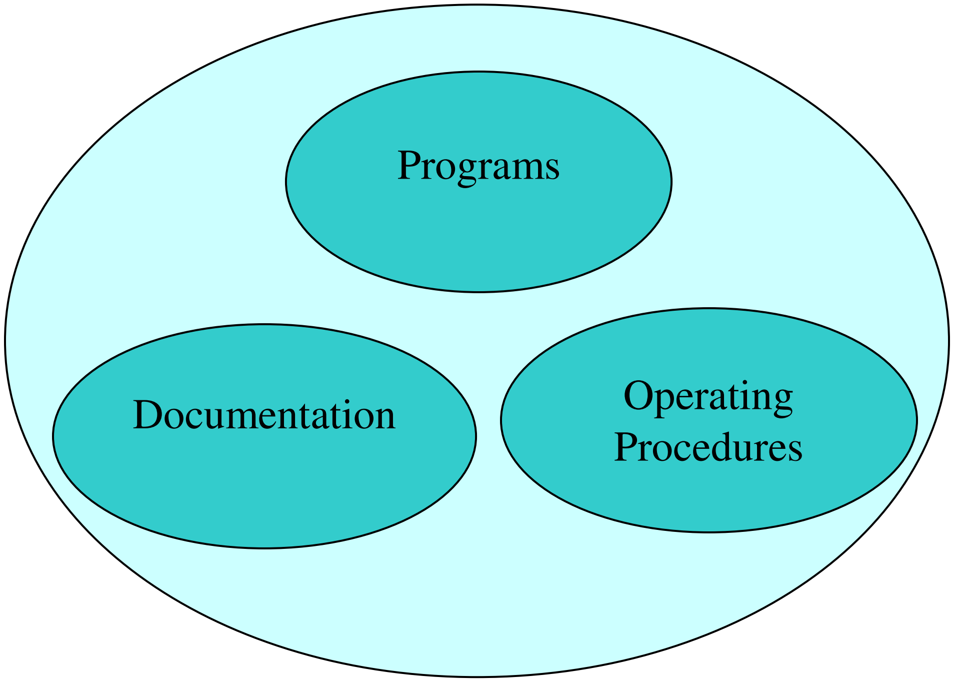
- Software companies
- Software developers
- Legal system
- Universities

What is software?

- **Computer programs** and **associated documentation**



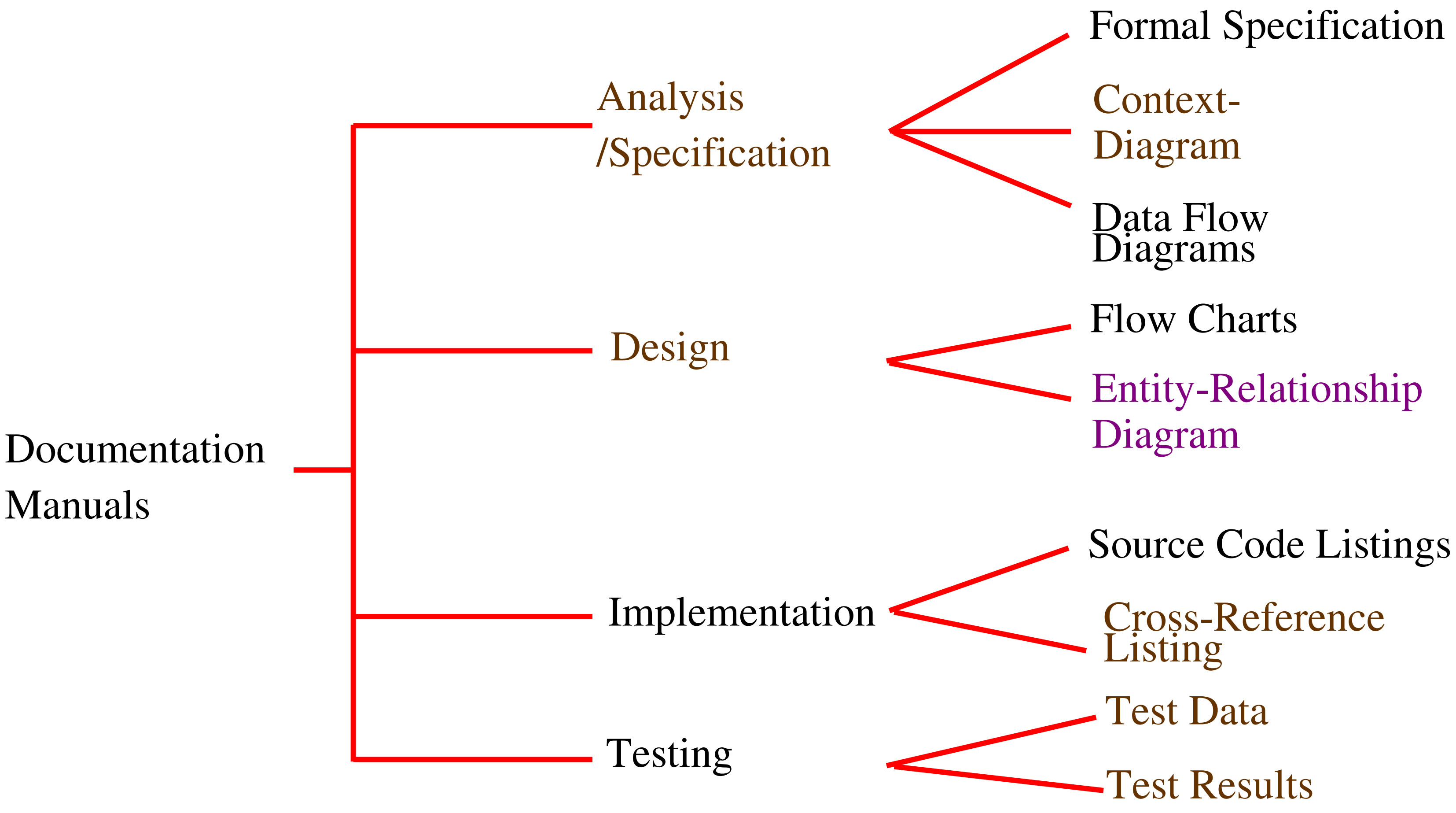
What is software?



Software=Program+Documentation+Operating Procedures

Components of software

Documentation consists of different types of manuals are



Documentation Manuals

Analysis /Specification

Formal Specification

Context-Diagram

Data Flow Diagrams

Design

Flow Charts

Entity-Relationship Diagram

Implementation

Source Code Listings

Cross-Reference Listing

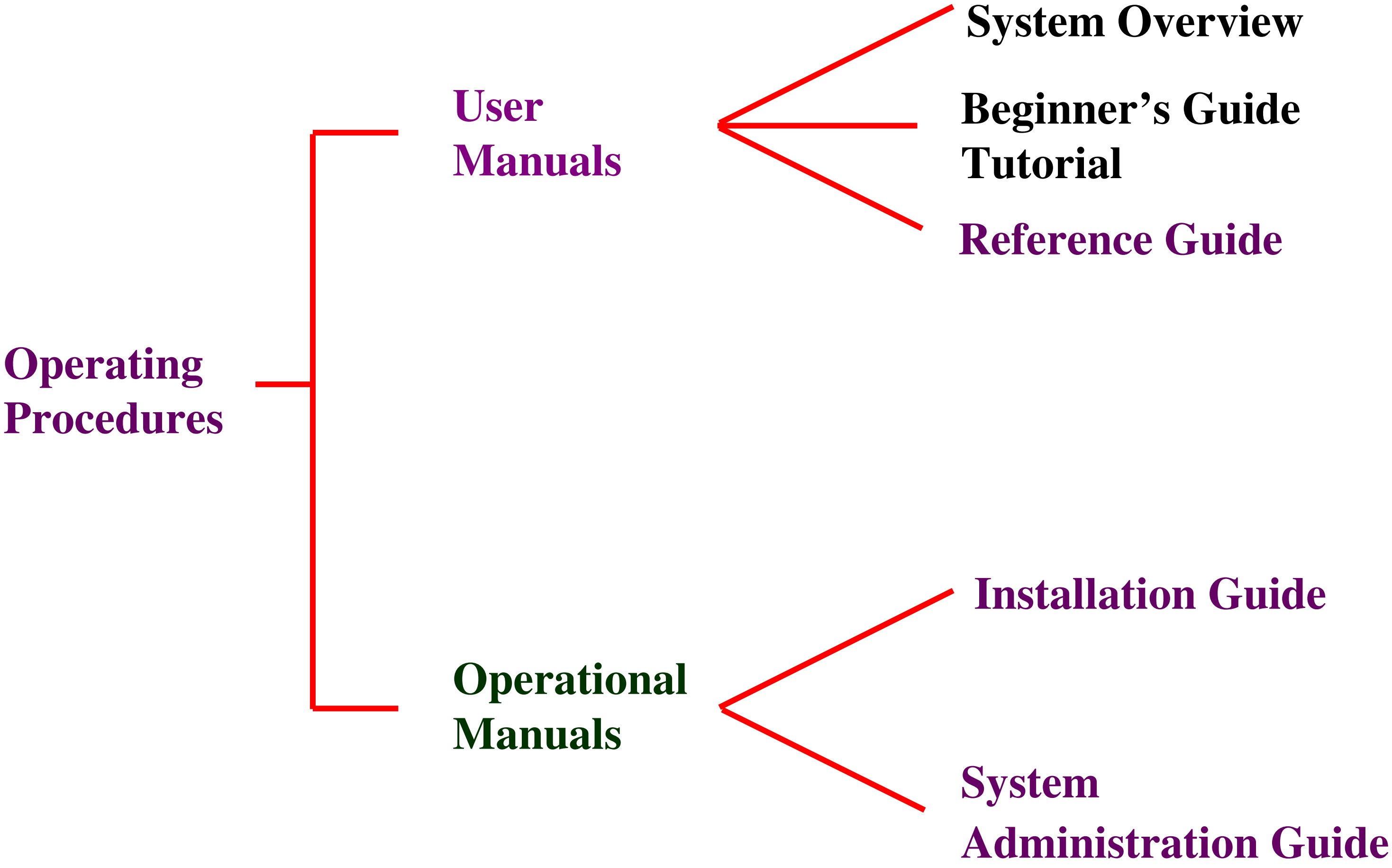
Testing

Test Data

Test Results

List of documentation manuals

Documentation consists of different types of manuals are



Operating Procedures

User Manuals

System Overview

Beginner's Guide Tutorial

Reference Guide

Operational Manuals

Installation Guide

System Administration Guide

List of operating procedure manuals.

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

source codes

reports

documents

object codes

plans

manuals

data

test suites

test results

prototypes

What is software engineering?

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

Software engineers 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

What is 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 Schach 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

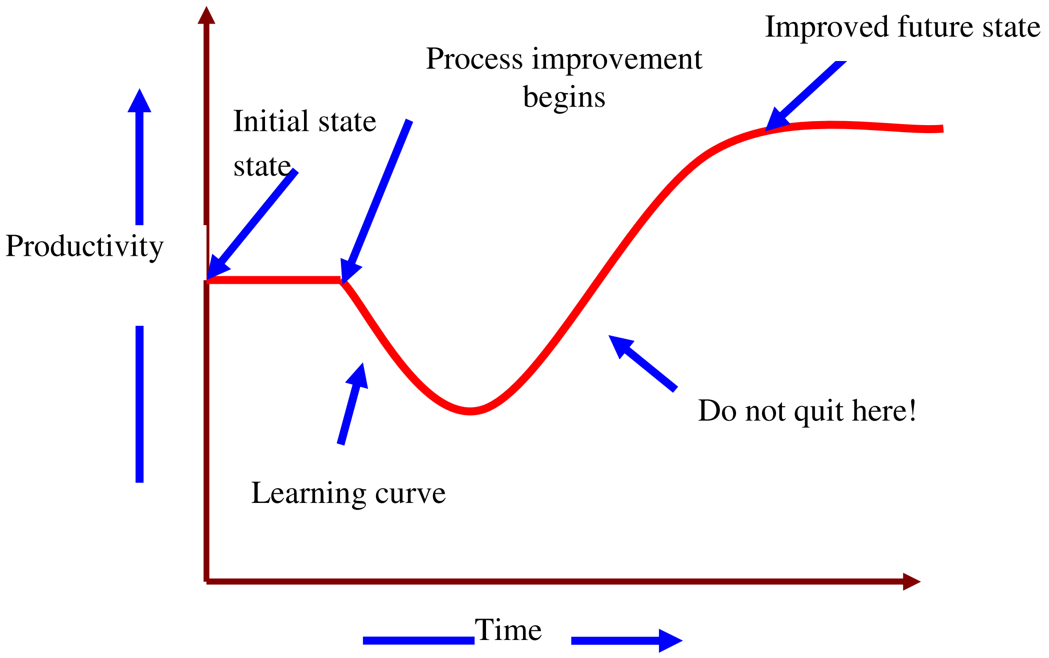
The software process is the way in which we produce software.

Why is it difficult to improve software process ?

- Not enough time
- Lack of knowledge

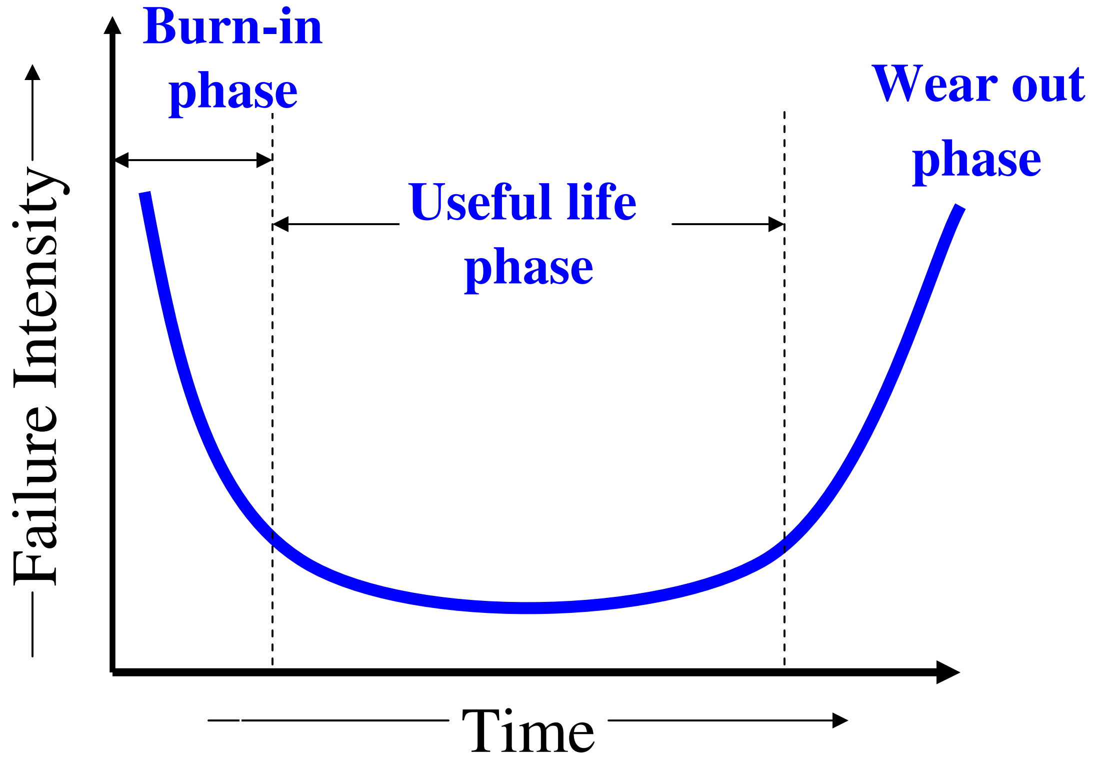
Software Process

- Wrong motivations
- Insufficient commitment



Software Characteristics:

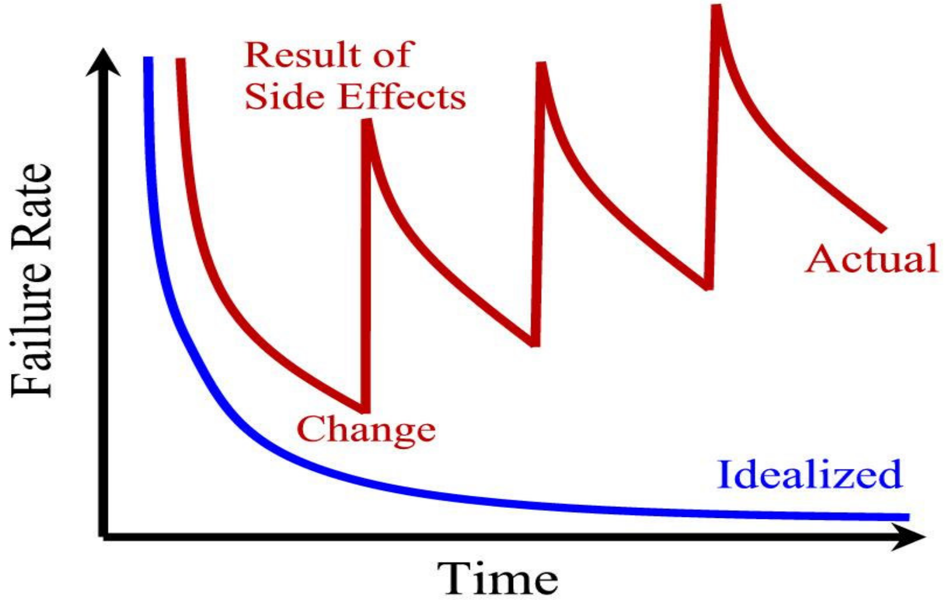
✓Software does not wear out.



Software Characteristics:

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

Failure Curve for Software



Software Characteristics:

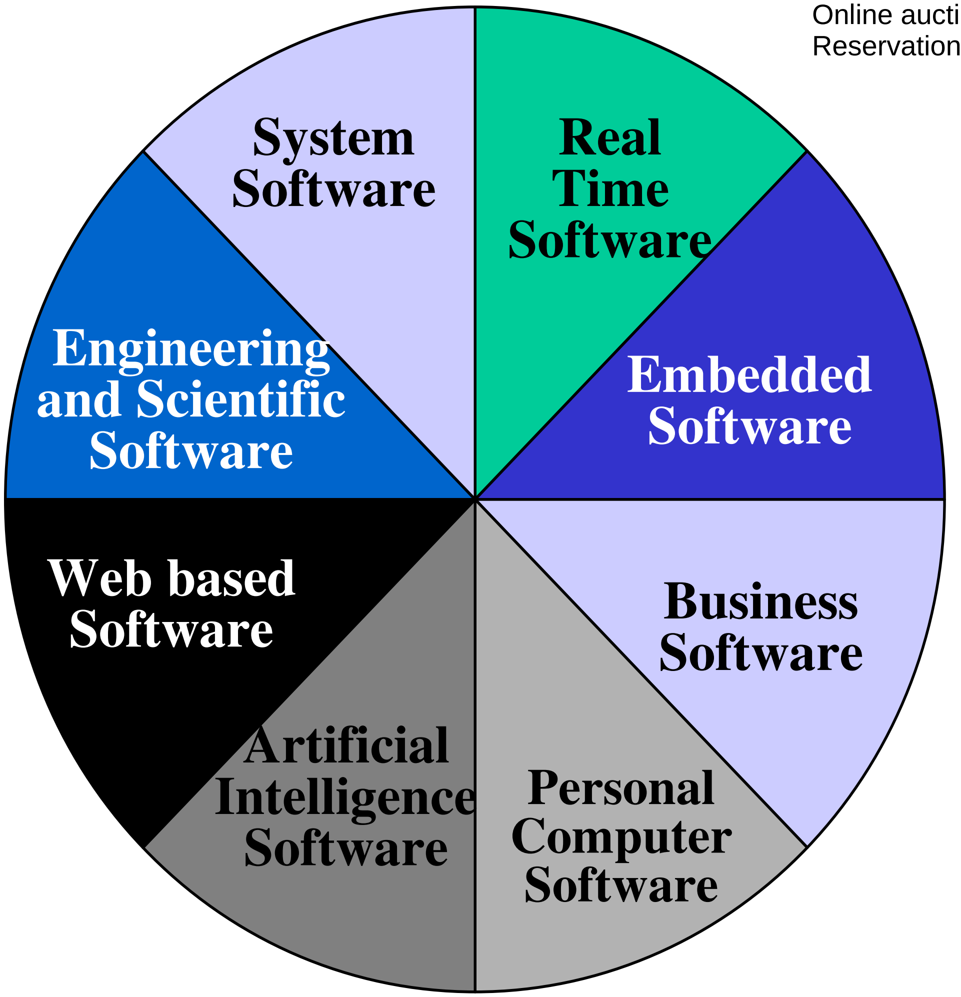
Comparison of constructing a bridge vis-à-vis writing a program.

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, the 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

Online trading systems,
Online auction systems,
Reservation systems etc

MATLAB: A software for numerical computing and visualization
AUTOCAD: A software for 2D and 3D design and drafting


System Software
Real Time Software
Engineering and Scientific Software
Embedded Software
Web based Software
Business Software
Artificial Intelligence Software
Personal Computer Software

Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007

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.

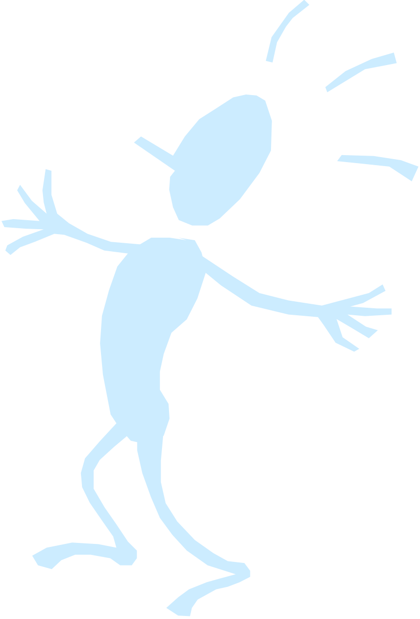
Examples

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.

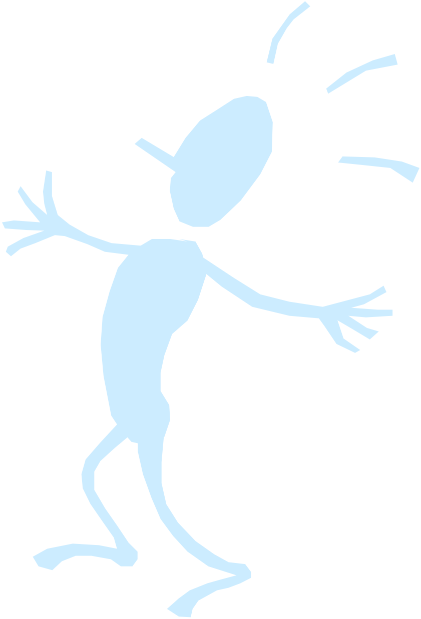
But the taste of any food item
is in the eating;
not in the Recipe !



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.

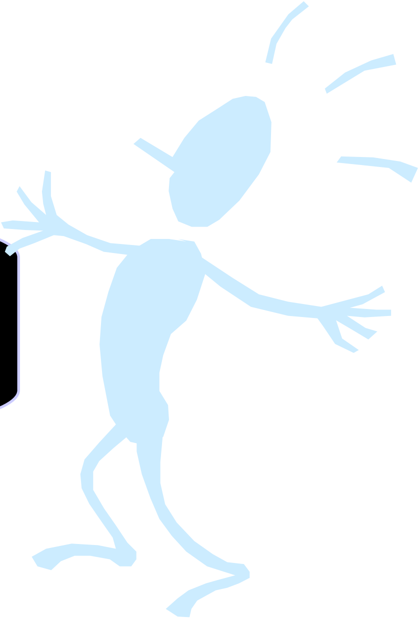
The infrastructure is only one of the several factors that determine the quality of the product!



Software Myths (Management Perspectives)

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

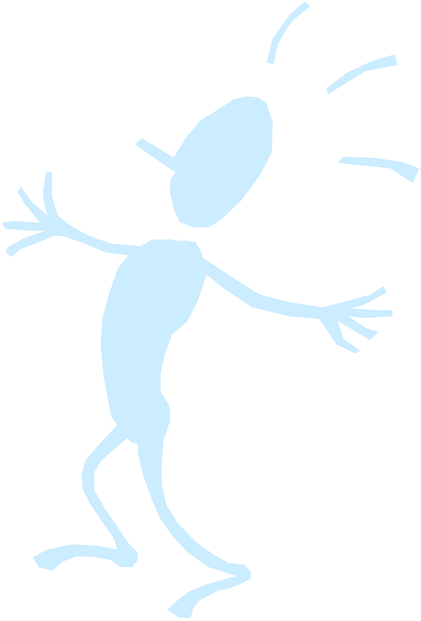
Unfortunately,
that may further delay the schedule!



Software Myths (Management Perspectives)

Software is easy to change

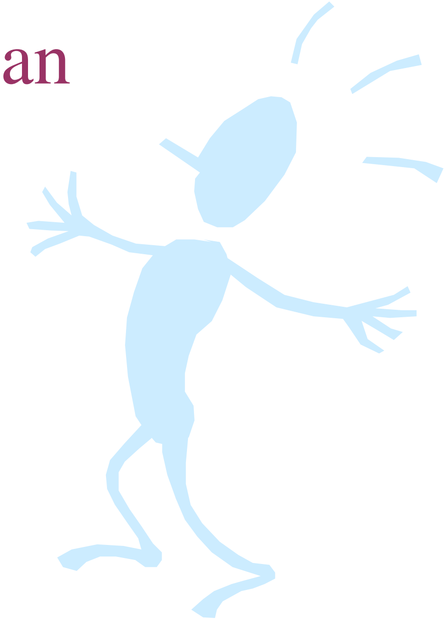
The reality is totally different.



Software Myths (Management Perspectives)

Computers provide greater reliability than the devices they replace

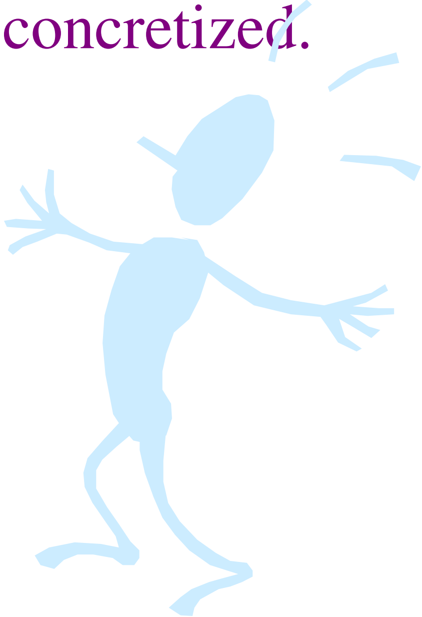
This is not always true.



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.

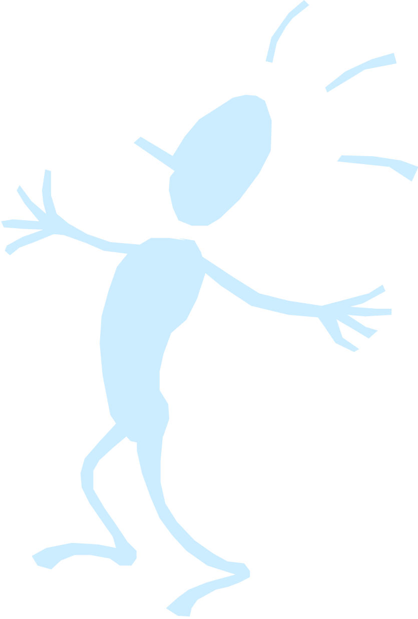


Software Myths (Customer Perspectives)

Software with more features is better software

Software can work right the first time

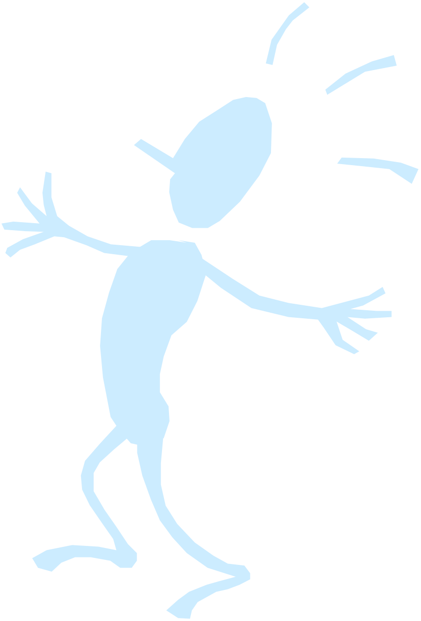
Both are only myths!



Software Myths (Developer Perspectives)

Once the software is demonstrated, the job is done.

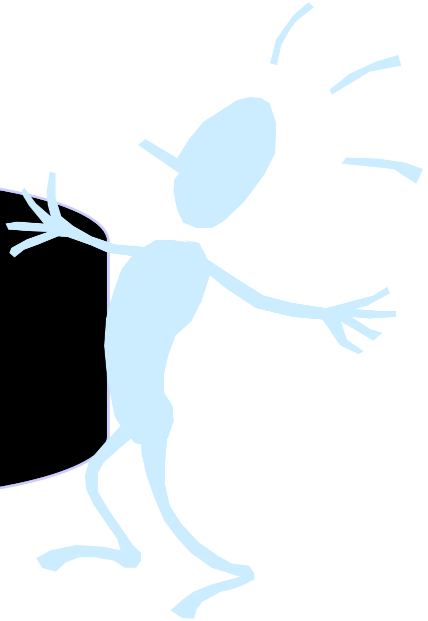
Usually, the problems just begin!



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.

Tested code is only one of the deliverable!

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

Process metrics: Productivity, Quality, Efficiency etc.

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".

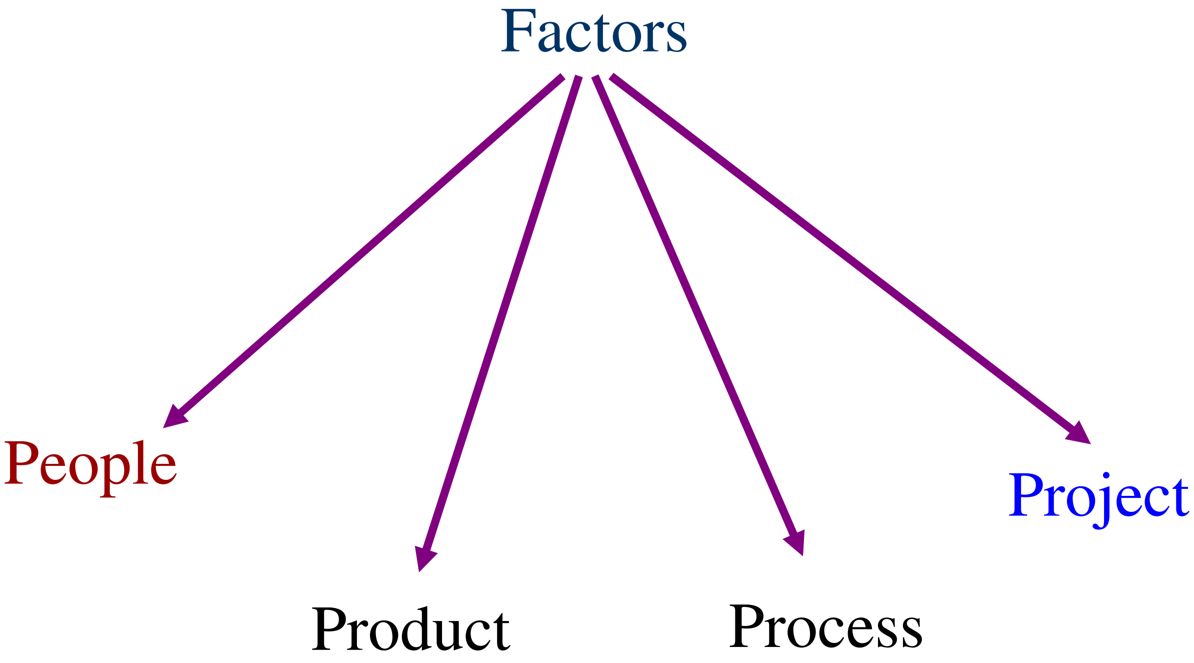
Some Terminologies

- Generic and Customized Software Products

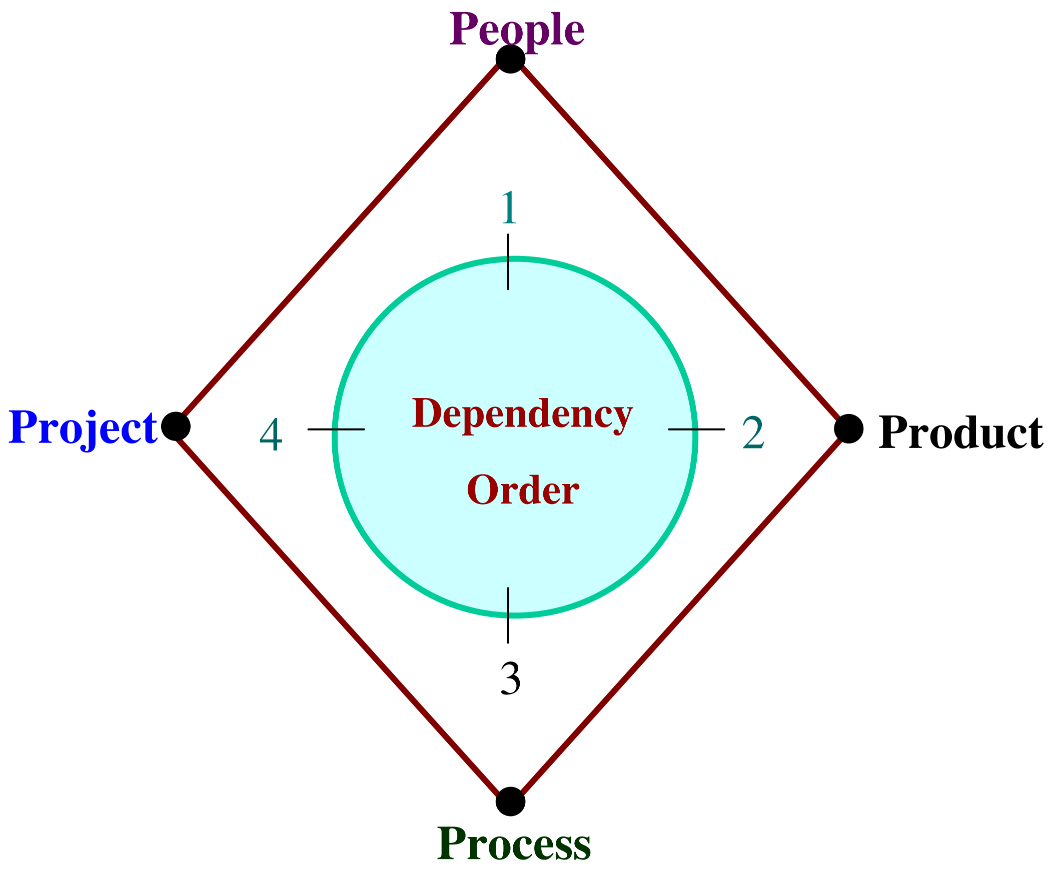
Generic products are developed for anonymous customers. The target is generally the entire world and many copies are expected to be sold. Infrastructure software like operating system, compilers, analyzers, word processors, CASE tools etc. are covered in this category.

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



Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

1.1 Software is
(a) Superset of programs (b) subset of programs
(c) Set of programs (d) none of the above

1.2 Which is NOT the part of operating procedure manuals?
(a) User manuals (b) Operational manuals
(c) Documentation manuals (d) Installation manuals

1.3 Which is NOT a software characteristic?
(a) Software does not wear out (b) Software is flexible
(c) Software is not manufactured (d) Software is always correct

1.4 Product is
(a) Deliverables (b) User expectations
(c) Organization's effort in development (d) none of the above

1.5 To produce a good quality product, process should be
(a) Complex (b) Efficient
(c) Rigorous (d) none of the above

Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

1.6 Which is not a product metric?

(a) Size (b) Reliability
(c) Productivity (d) Functionality

1.7 Which is NOT a process metric?

(a) Productivity (b) Functionality
(c) Quality (d) Efficiency

1.8 Effort is measured in terms of:

(a) Person-months (b) Rupees
(c) Persons (d) Months

1.9 UML stands for

(a) Uniform modeling language (b) Unified modeling language
(c) Unit modeling language (d) Universal modeling language

1.1 An independently deliverable piece of functionality providing access to its services through interface is called

(a) Software measurement (b) Software composition
(c) Software measure (d) Software component

Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

1.11 Infrastructure software are covered under
(a) Generic products (b) Customized products
(c) Generic and Customized products (d) none of the above

1.12 Management of software development is dependent on
(a) people (b) product
(c) process (d) all of the above

1.13 During software development, which factor is most crucial?
(a) People (b) Product
(c) Process (d) Project

1.14 Program is
(a) subset of software (b) super set of software
(c) software (d) none of the above

1.15 Milestones are used to
(a) know the cost of the project (b) know the status of the project
(c) know user expectations (d) none of the above

Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

1.16 The term module used during design phase refers to

(a) Function (b) Procedure
(c) Sub program (d) All of the above

1.17 Software consists of

(a) Set of instructions + operating system
(b) Programs + documentation + operating procedures
(c) Programs + hardware manuals (d) Set of programs

1.18 Software engineering approach is used to achieve:

(a) Better performance of hardware (b) Error free software
(c) Reusable software (d) Quality software product

1.19 Concept of software engineering are applicable to

(a) Fortran language only (b) Pascal language only
(c) 'C' language only (d) All of the above

1.20 CASE Tool is

(a) Computer Aided Software Engineering (b) Component Aided Software Engineering
(c) Constructive Aided Software Engineering (d)Computer Analysis Software Engineering

Exercises

1.1 Why is primary goal of software development now shifting from producing good quality software to good quality maintainable software?

1.2 List the reasons for the "software crisis"?Why are CASE tools not normally able to control it?

1.3 "The software crisis is aggravated by the progress in hardware technology?" Explain with examples.

1.4 What is software crisis? Was Y2K a software crisis?

1.5 What is the significance of software crisis in reference to software engineering discipline.

1.6 How are software myths affecting software process? Explain with the help of examples.

1.7 State the difference between program and software. Why have documents and documentation become very important.

1.8 What is software engineering? Is it an art, craft or a science? Discuss.

Exercises

1.9 What is aim of software engineering? What does the discipline of software engineering discuss?

1.10 Define the term "Software engineering". Explain the major differences between software engineering and other traditional engineering disciplines.

1.11 What is software process? Why is it difficult to improve it?

1.12 Describe the characteristics of software contrasting it with the characteristics of hardware.

1.13 Write down the major characteristics of a software. Illustrate with a diagram that the software does not wear out.

1.14 What are the components of a software? Discuss how a software differs from a program.

1.15 Discuss major areas of the applications of the software.

1.16 Is software a product or process? Justify your answer with example

Exercises

1.17 Differentiate between the following

(i) Deliverables and milestones (ii) Product and process

(iii) Measures, metrics and measurement

1.18 What is software metric? How is it different from software measurement

1.19 Discuss software process and product metrics with the help of examples.

1.20 What is productivity? How is it related to effort. What is the unit of effort.

1.21 Differentiate between module and software component.

1.22 Distinguish between generic and customized software products. Which one has larger share of market and why?

1.23 Is software a product or process? Justify your answer with example

Exercises

1.23 Describe the role of management in software development with the help of examples.

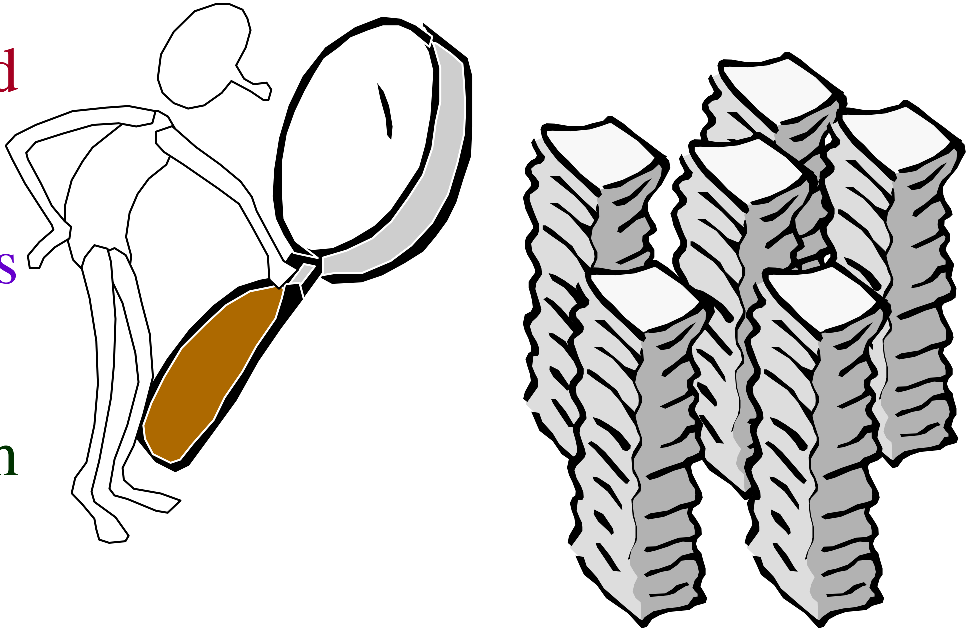
1.24 What are various factors of management dependency in software development. Discuss each factor in detail.

1.25 What is more important: Product or process? Justify your answer.

Software Certification

Software Certification

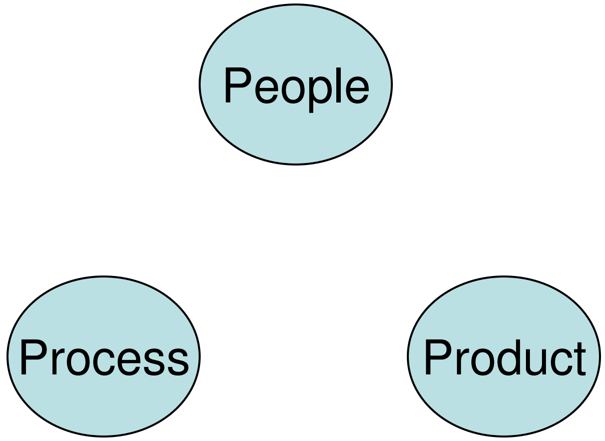
- What is certification?
- Why should we really need it?
- Who should carry out this activity?
- Where should we do such type of certification?



Software Certification

To whom should we target

- People
- Process
- Product



We have seen many certified developers (Microsoft certified, Cisco certified, JAVA certified), certified processes (like ISO or CMM) and certified products.

There is no clarity about the procedure of software certification.

Requirement of Certification

Adam Kalawa of Parasoft has given his views on certification like:

"I strongly oppose certification of software developers. I fear that it will bring more harm than good to the software industry. It may further hurt software quality by shifting the blame for bad software. The campaign for certification assumes that unqualified developers cause software problem and that we can improve software quality by ensuring that all developers have the golden stamp of approval. However, improving quality requires improving the production process and integrating in to it practices that reduce the opportunity for introducing defects into the product"

Requirement of Certification

- How often will developers require certification to keep pace with new technologies?
- How will any certification address the issues like fundamentals of computer science, analytical & logical reasoning, programming aptitude & positive attitude?
- Process certification alone cannot guarantee high quality product.
- Whether we go for certified developers or certified processes?

Can independent certification agency provide a fair playing field for each software industry??

Types of Certification

- People
  - Industry specific
- Process
  - Industry specific
- Product
  - For the customer directly and helps to select a particular product

Certification of Persons

The individual obtaining certification receives the following values:

- Recognition by peers
- Increased confidence in personal capabilities
- Recognition by software industry for professional achievement
- Improvement in processes
- Competences maintained through recertification

Certification is employees initiated improvement process which improves competence in quality assurances methods & techniques.

Certification of Persons

Professional level of competence in the principles & practices of software quality assurance in the software industry can be achieved by acquiring the designation of:

- Certified Software Quality Analyst (CSQA)
- Certified Software Tester (CSTE)
- Certified Software Project Manager (CSPM)

Some company specific certifications are also very popular like Microsoft Office Specialist (MOS) certifications in Word, Excel and PowerPoint.

MOS is far best known computer skills certification for administrator.

Certification of Processes

The most popular process certification approaches are:

- ISO 9000
- SEI-CMM

One should always be suspicious about the quality of end product, however, certification reduces the possibility of poor quality products.

Any type of process certification helps to produce good quality and stable software product.

Certification of Products

- This is what is required for the customer.
- There is no universally accepted product certification scheme.
- Aviation industry has a popular certification “RTCA DO-178B”.
- The targeted certification level is either A, B, C, D, or E.
- These levels describe the consequences of a potential failure of the software : catastrophic, hazardous severe, major, minor or no effect.

Certification of Products

DO-178B Records

Software Development Plan

Software Verification Plan

Software Configuration Management Plan

Software Quality Assurance Plan

Software Requirements Standards

Software Design Document

Software Verification Test Cases & Products

Certification of Products

DO-178B Documents

Software Verification Results

Problem Report

Software Configuration Management Records

Software Quality Assurance Records

DO-178B certification process is most demanding at higher levels.

Certification of Products

DO-178B level A will:

1. Have largest potential market
2. Require thorough labour intensive preparation of most of the items on the DO-178B support list.

DO-178B Level E would:

1. Require fewer support item and
2. Less taxing on company resources.

Certification of Products

We don’t have product certification in most of the areas. RTOS (real time operating system) is the real-time operating system certification & marked as “LinuxOS-178”.

The establishment of independent agencies is a viable option.

Third Party Certification for Component base Software Engineering

Weyukar has rightly said "For Component based Software Development (CBO) to revolutionalize software development, developers must be able to produce software significantly cheaper and faster than they otherwise could, even as the resulting software meets the same sort of high reliability standards while being easy to maintain".

Bill council has also given his views as "Currently, there is a little evidences that component based software engineering (CBSE) is revolutionizing software development, and lots of reasons to believe otherwise. I believe the primary reason is that the community is not showing how to develop trusted components".

Third Party Certification for Component base Software Engineering

Contractor:

- Gives the standard
- Directs any variations in specification
- Define patterns
- Allowable tolerances
- Fix the date of delivery

Third party certification is a method to ensure software components conform to well defined standards, based on this certification, trusted assemblies of components can be constructed

Third party certification is based on UL 1998, 2nd ed., UL standard for safety for software in programmable component.

Exercises

10.1 What is software certification? Discuss its importance in the changing scenario of software industry.

10.2 What are different types of certifications? Explain the significance of each type & which one is most important for the end user.

10.3 What is the role of third party certification in component based software engineering? Why are we not able to stabilize the component based software engineering practices.

10.4 Name few person specific certification schemes. Which one is most popular & why?

10.5 Why customer is only interested in product certification? Discuss any product certification techniques with their generic applicability.

Software Life Cycle Models

Software Life Cycle Models

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.

Software Life Cycle Models

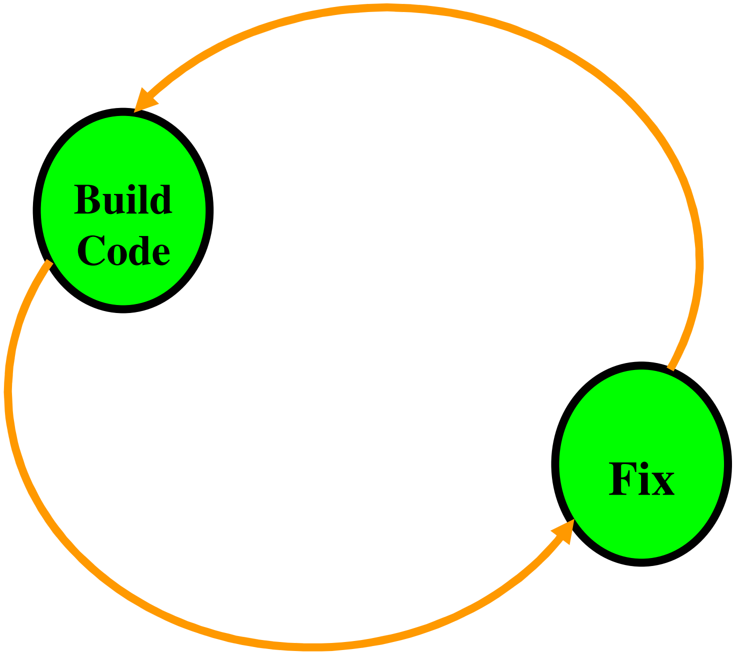
“The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a requirement phase, design phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase, and sometimes retirement phase”.

Build & Fix Model

- Product is constructed without specifications or any attempt at design

- Adhoc approach and not well defined

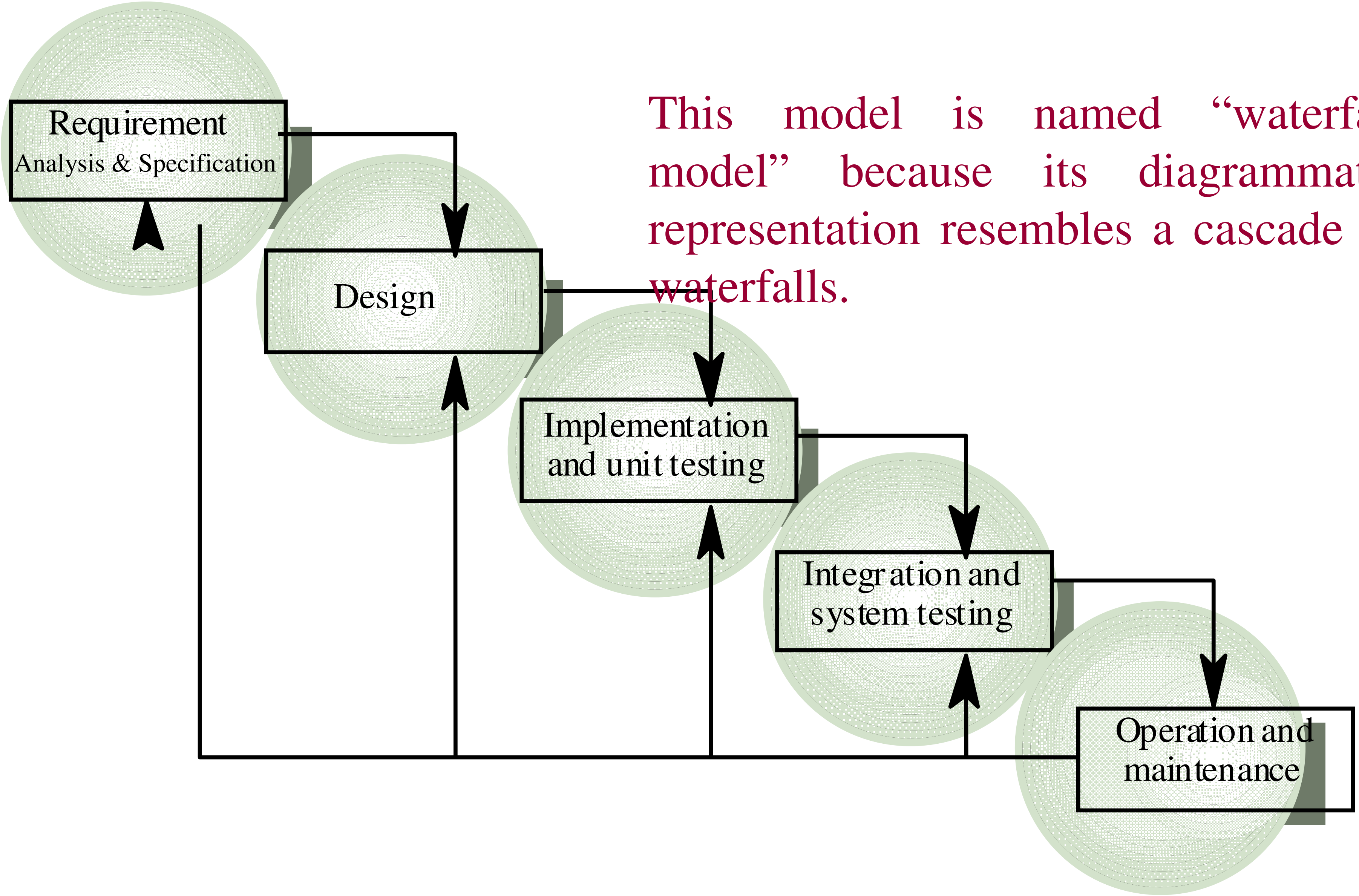
- Simple two phase model



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


Requirement
Analysis & Specification
Design
Implementation and unit testing
Integration and system testing
Operation and maintenance

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

i. It is difficult to define all requirements at the beginning of a project

ii. This model is not suitable for accommodating any change

iii. A working version of the system is not seen until late in the project's life

iv. It does not scale up well to large projects.

v. Real projects are rarely sequential.

Incremental Process Models

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.

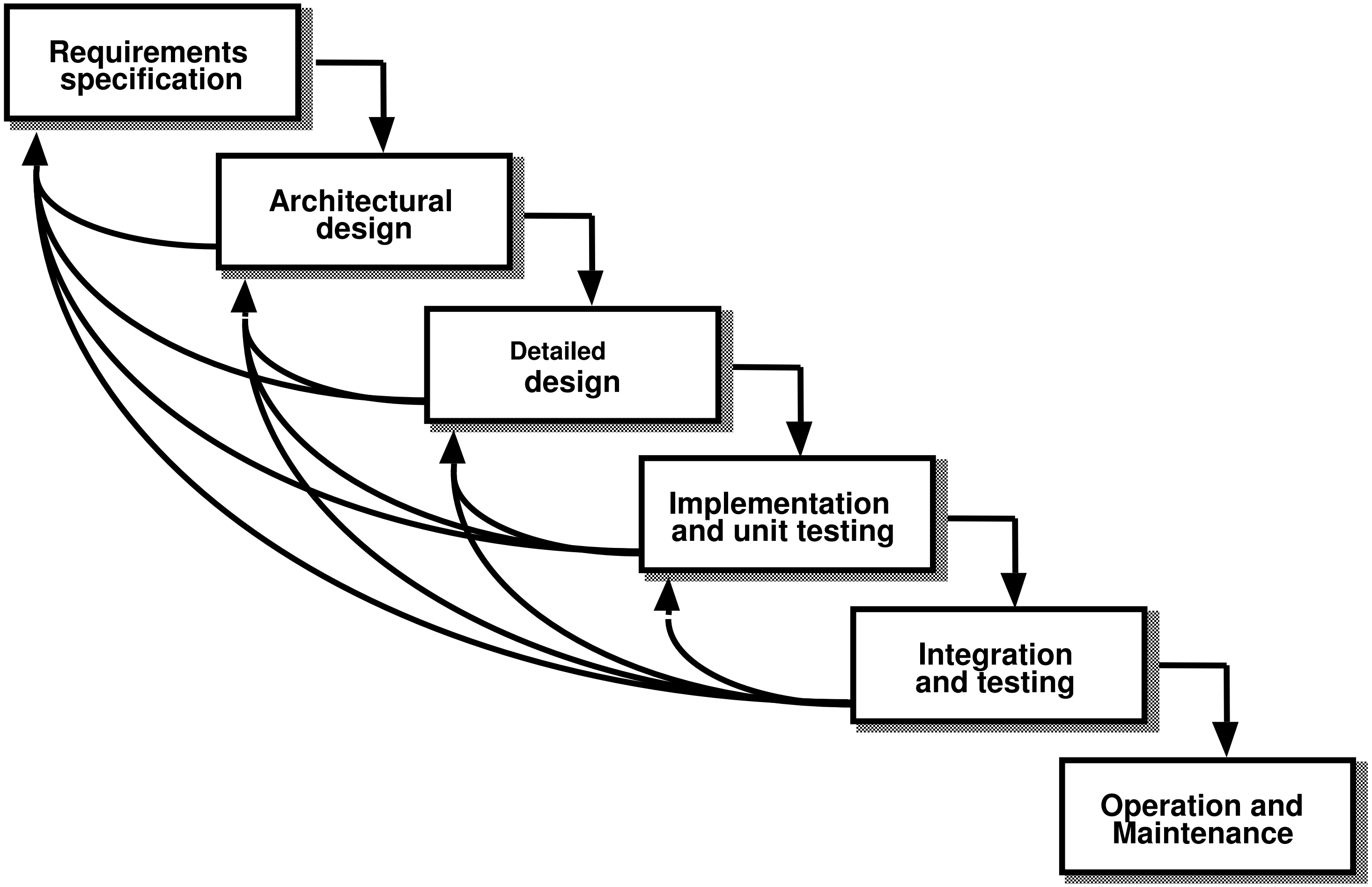
Popular particularly when we have to quickly deliver a limited functionality system.

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



Requirements specification

Architectural design

Detailed design

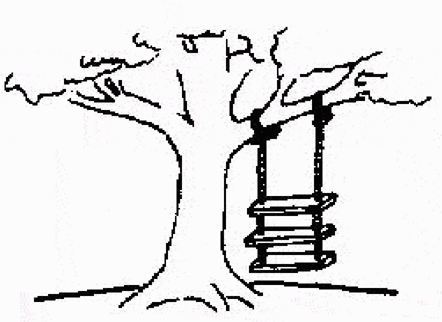
Implementation and unit testing

Integration and testing

Operation and Maintenance

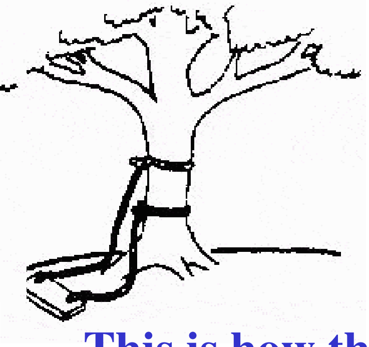
The Rapid Application Development (RAD) Model

- Developed by IBM in 1980
- User participation is essential

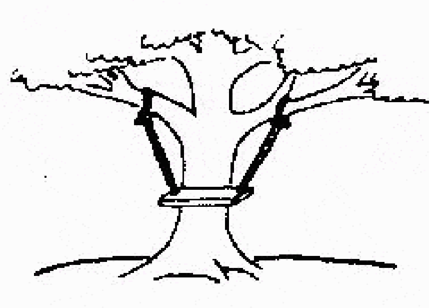


The requirements specification was defined like this

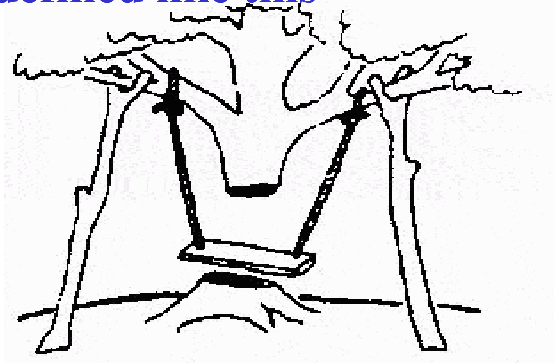
The developers understood it in that way



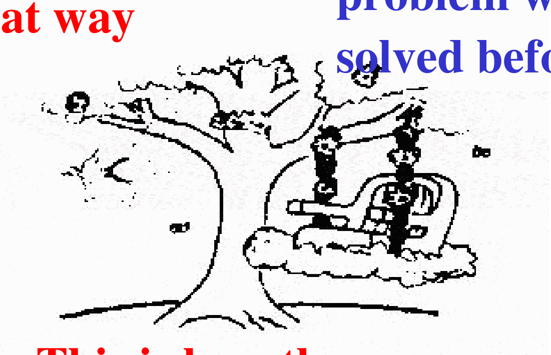
This is how the problem was solved before.



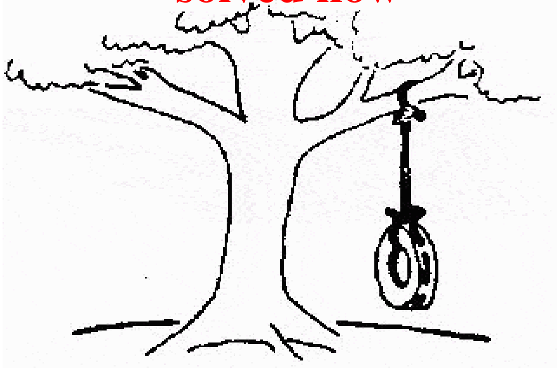
This is how the problem is solved now



That is the program after debugging



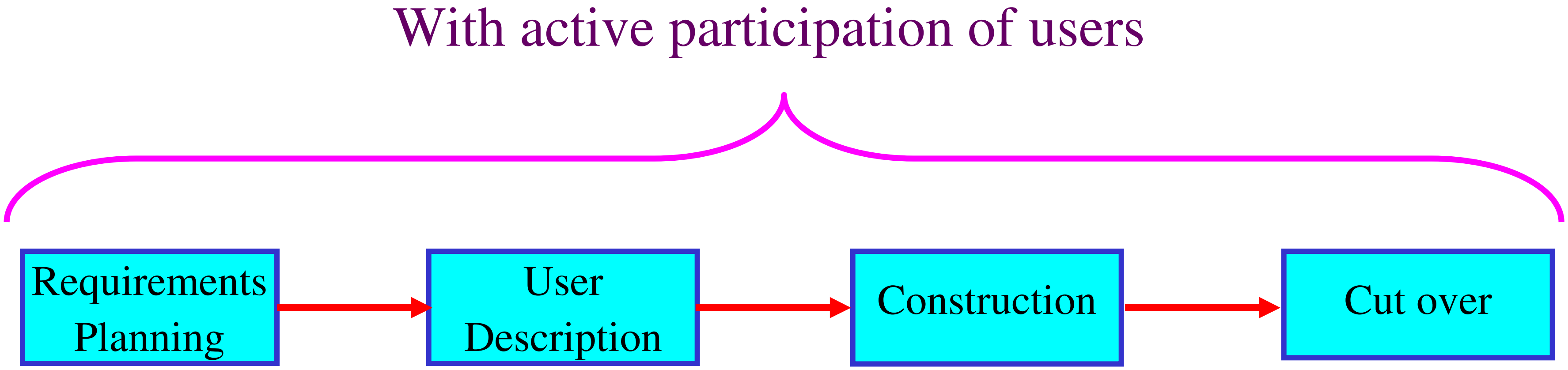
This is how the program is described by marketing department



This, in fact, is what the customer wanted …

The 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

Requirements Planning → User Description → Construction → Cut over

The Rapid Application Development (RAD) Model

Not an appropriate model in the absence of user participation.

Reusable components are required to reduce development time.

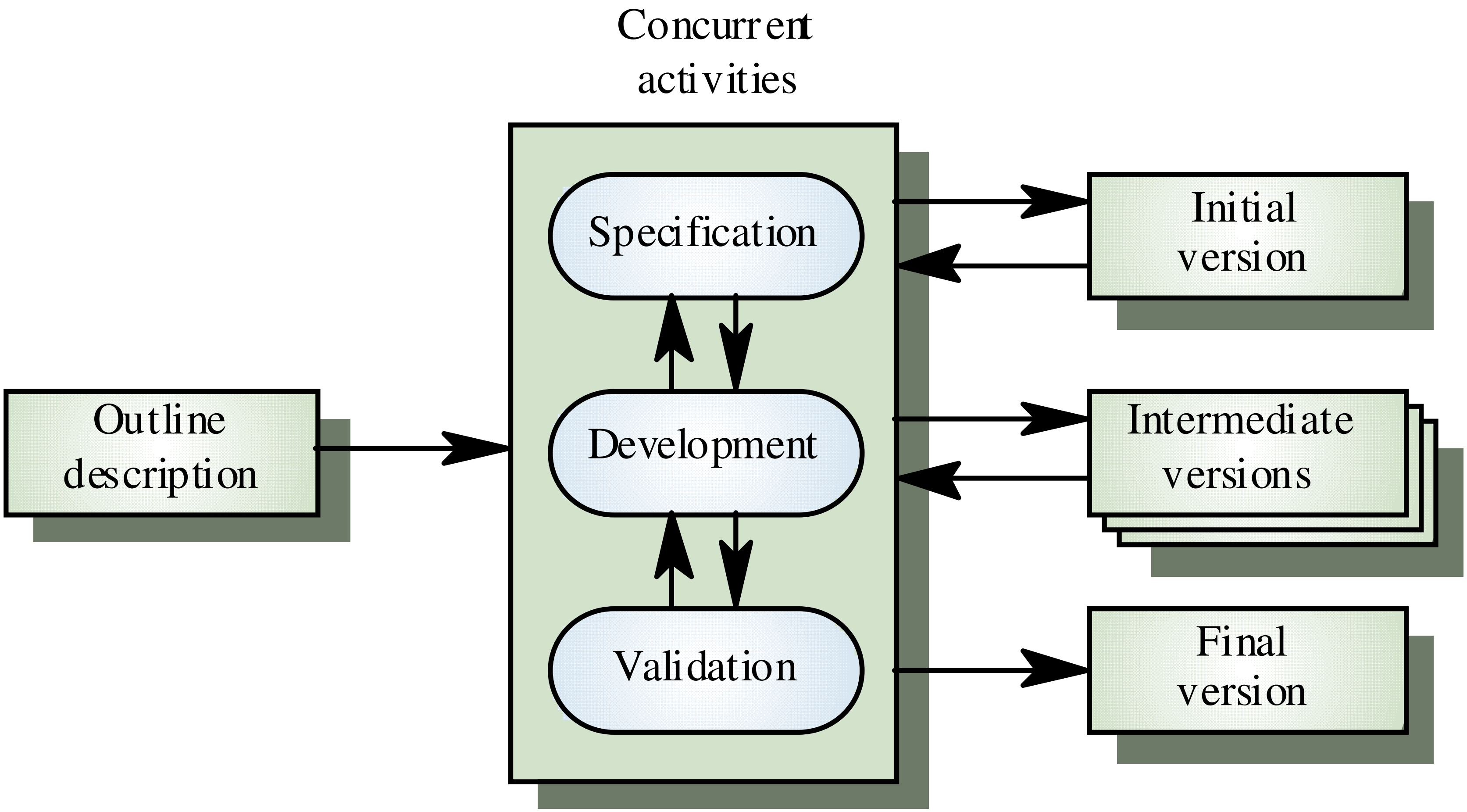
Highly specialized & skilled developers are required and such developers are not easily available.

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

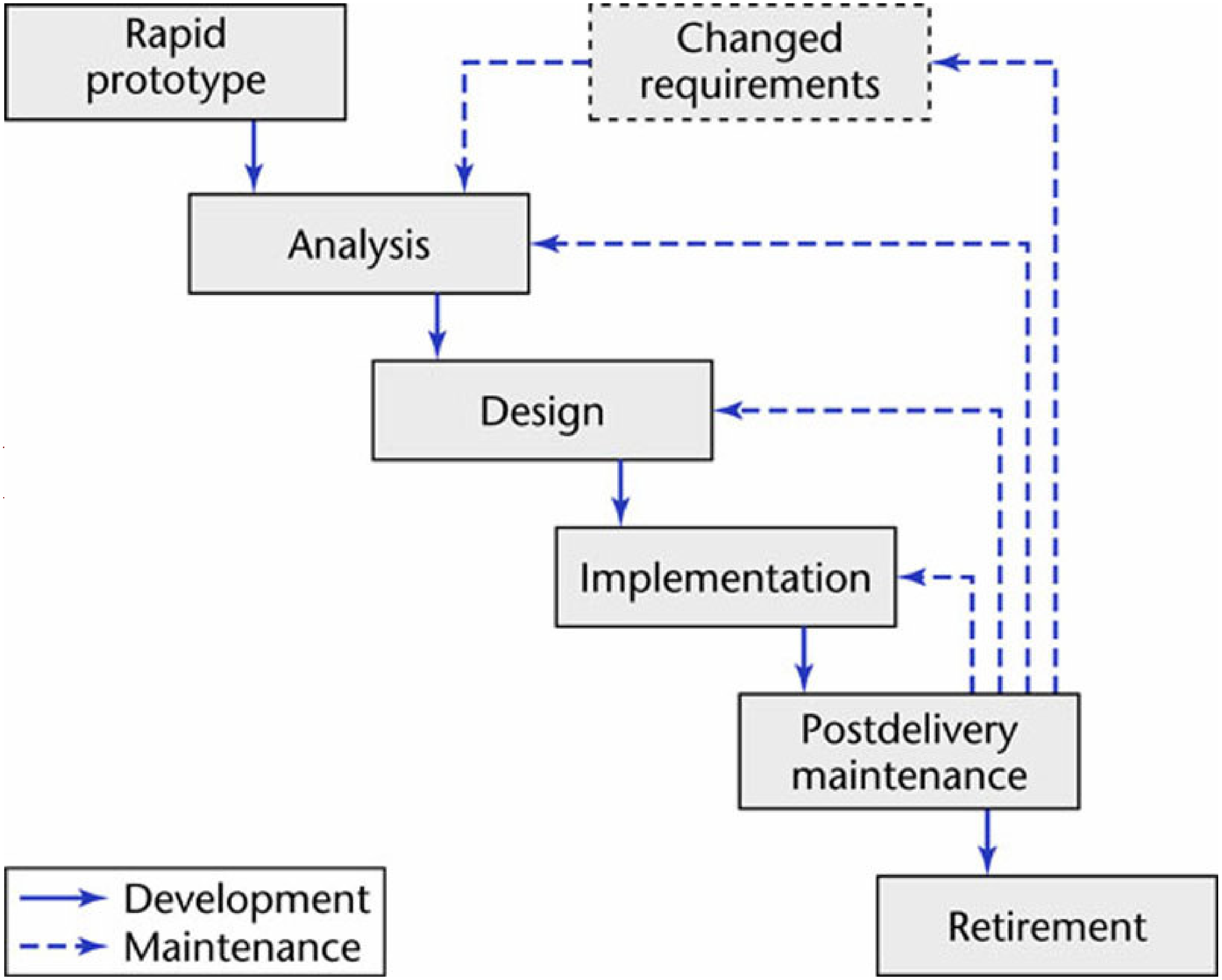
- The prototype may be a usable program but is not suitable as the final software product.

- 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”



Spiral Model

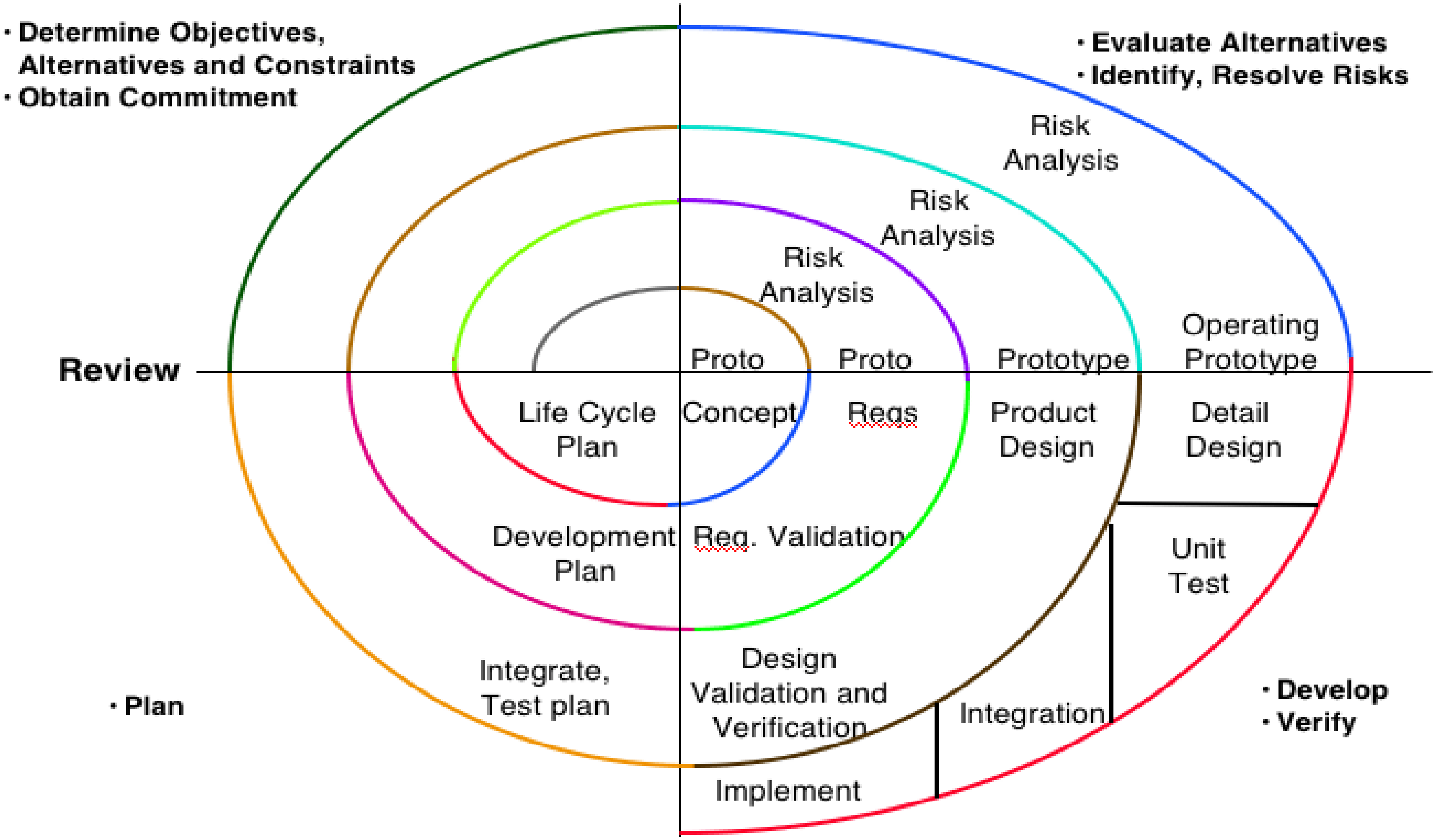
Models do not deal with uncertainly which is inherent to software projects.

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

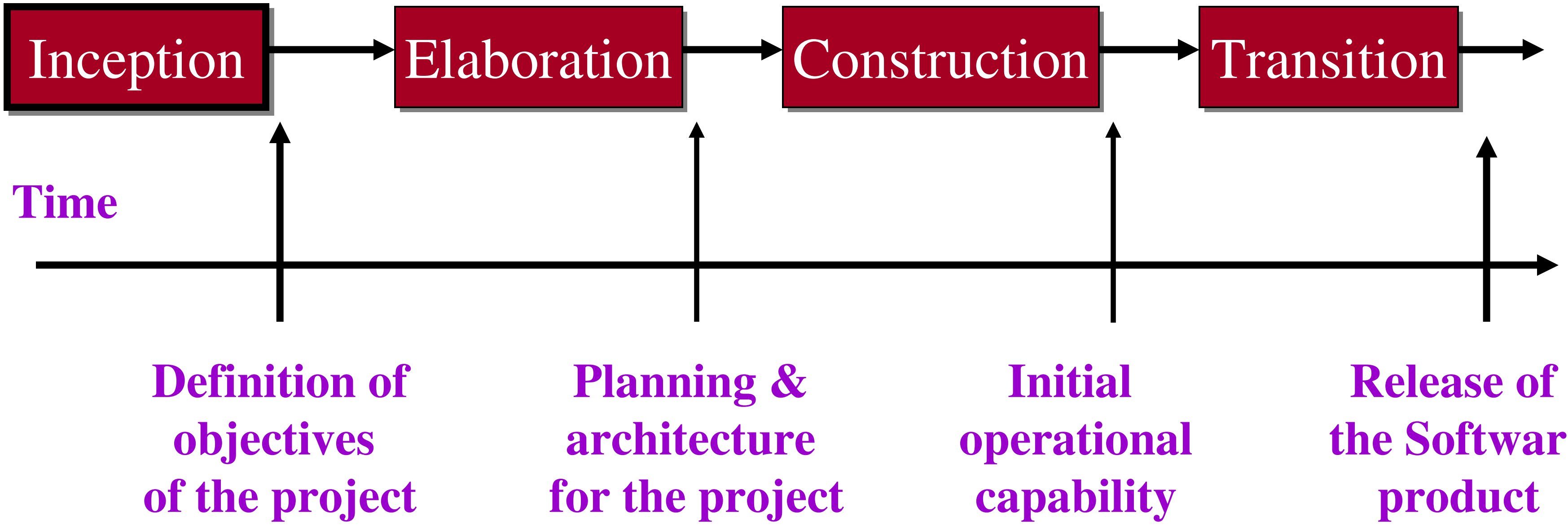
- Developed by I.Jacobson, G.Booch and J.Rumbaugh.

- 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


Inception
Elaboration
Construction
Transition
Time
Definition of objectives of the project
Planning & architecture for the project
Initial operational capability
Release of the Software product

Phases of the Unified Process

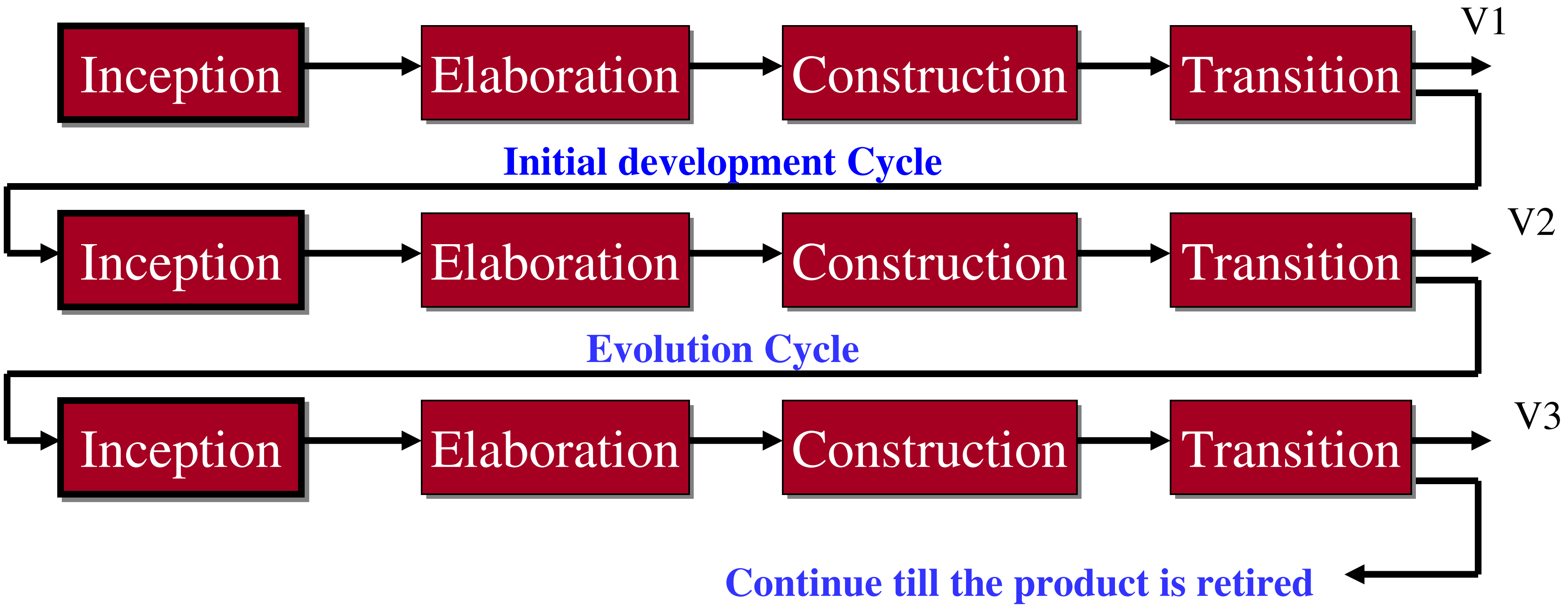
- Inception: defines scope of the project.
- Elaboration

  - How do we plan & design the project?

  - What resources are required?

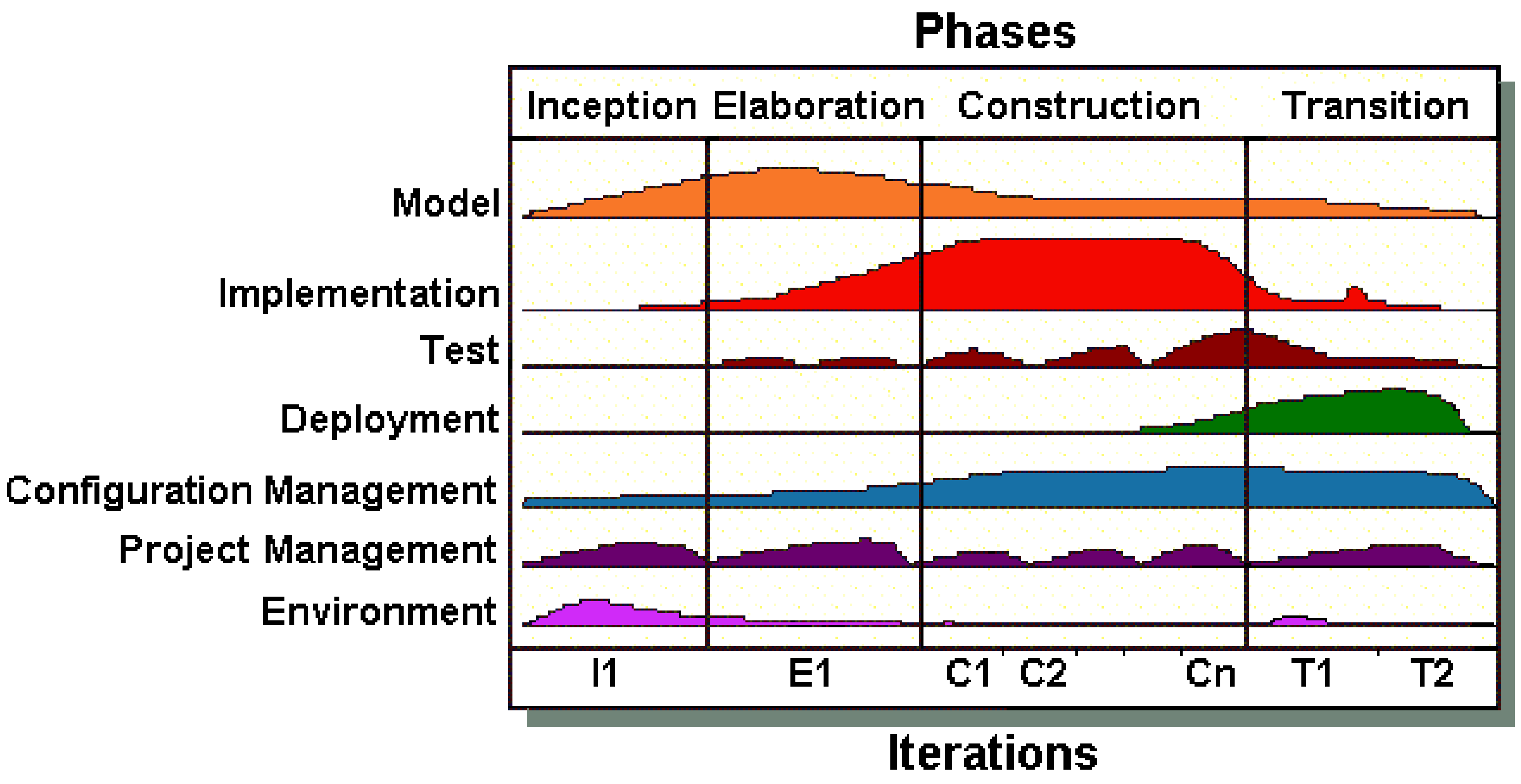
  - What type of architecture may be suitable?
- Construction: the objectives are translated in design & architecture documents.
- Transition : involves many activities like delivering, training, supporting, and maintaining the product.

Initial development & Evolution Cycles



V1=version1, V2 =version2, V3=version3

Iterations & Workflow of Unified Process

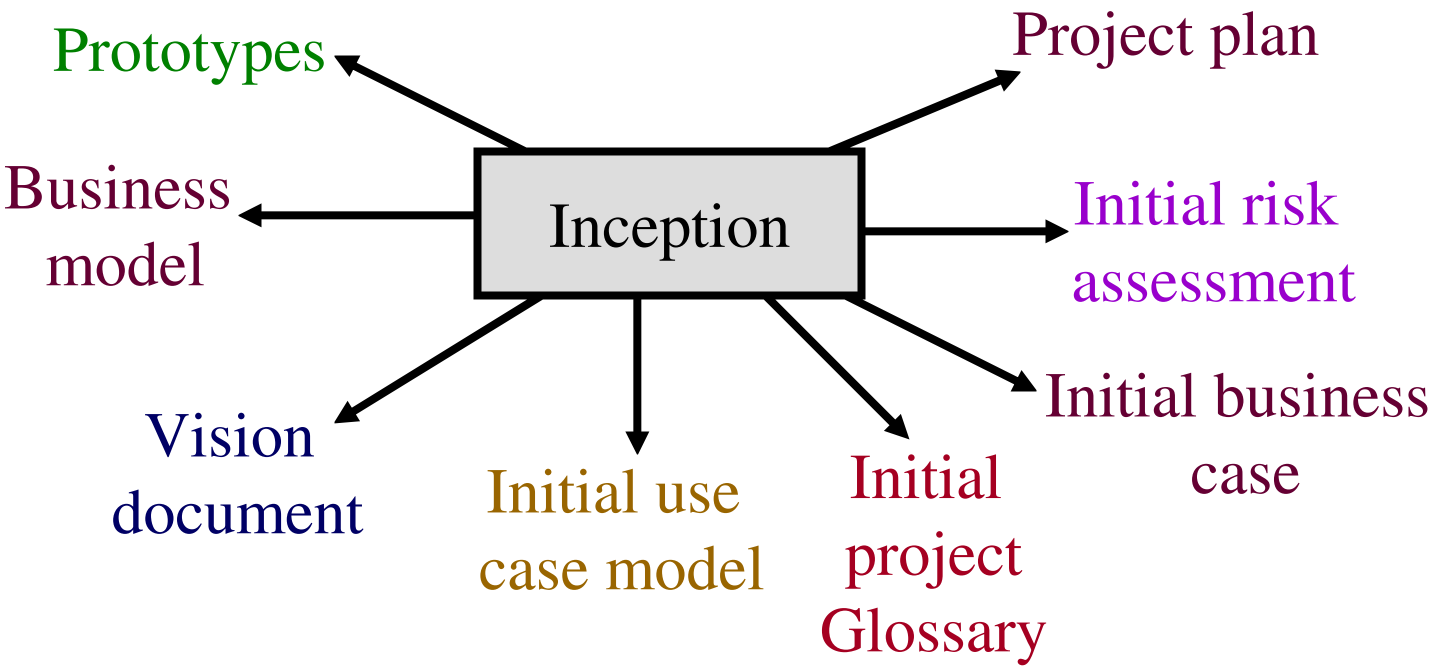


Inception Phase

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



Inception

Prototypes

Project plan

Business model

Initial risk assessment

Vision document

Initial use case model

Initial project Glossary

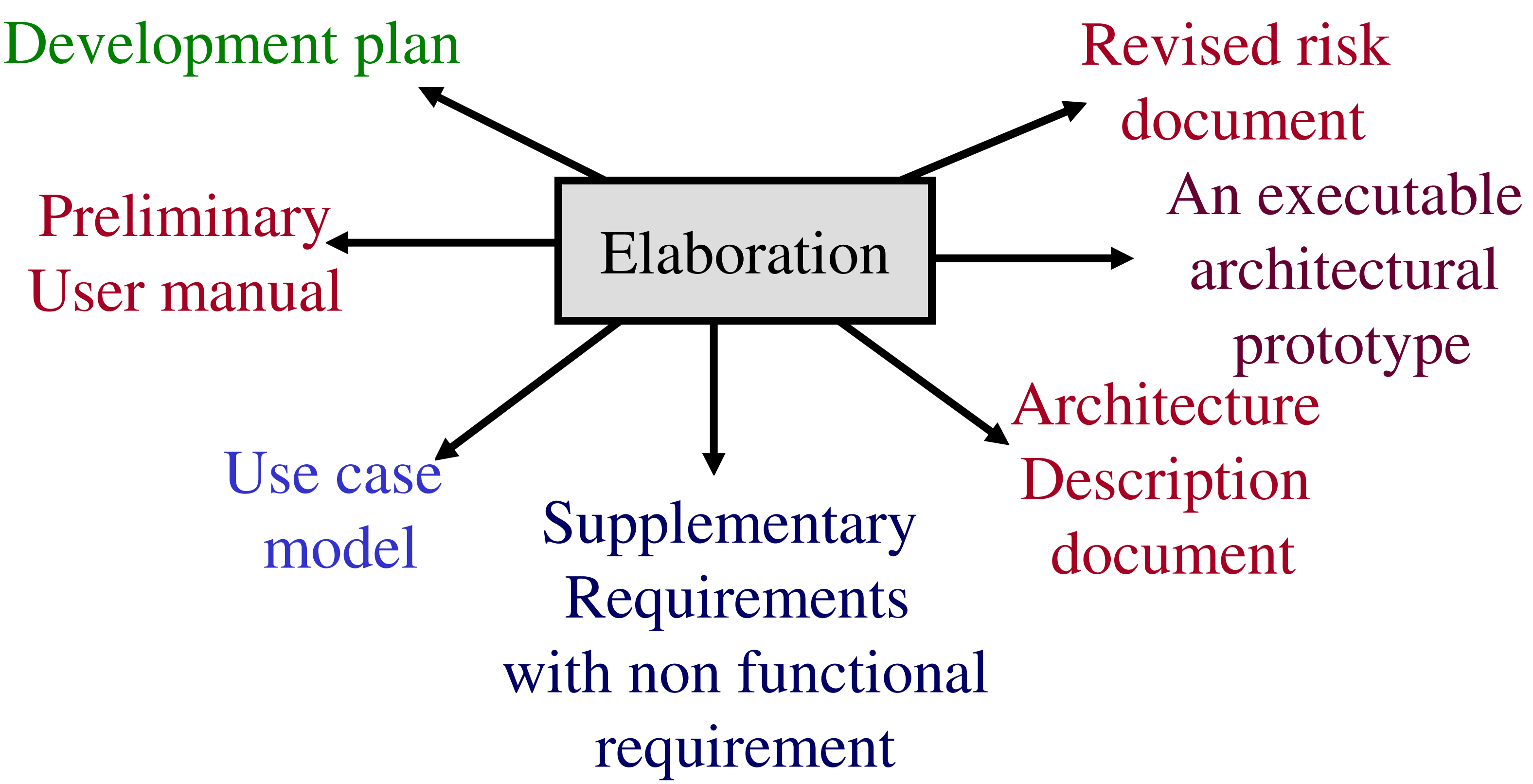
Initial business case

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

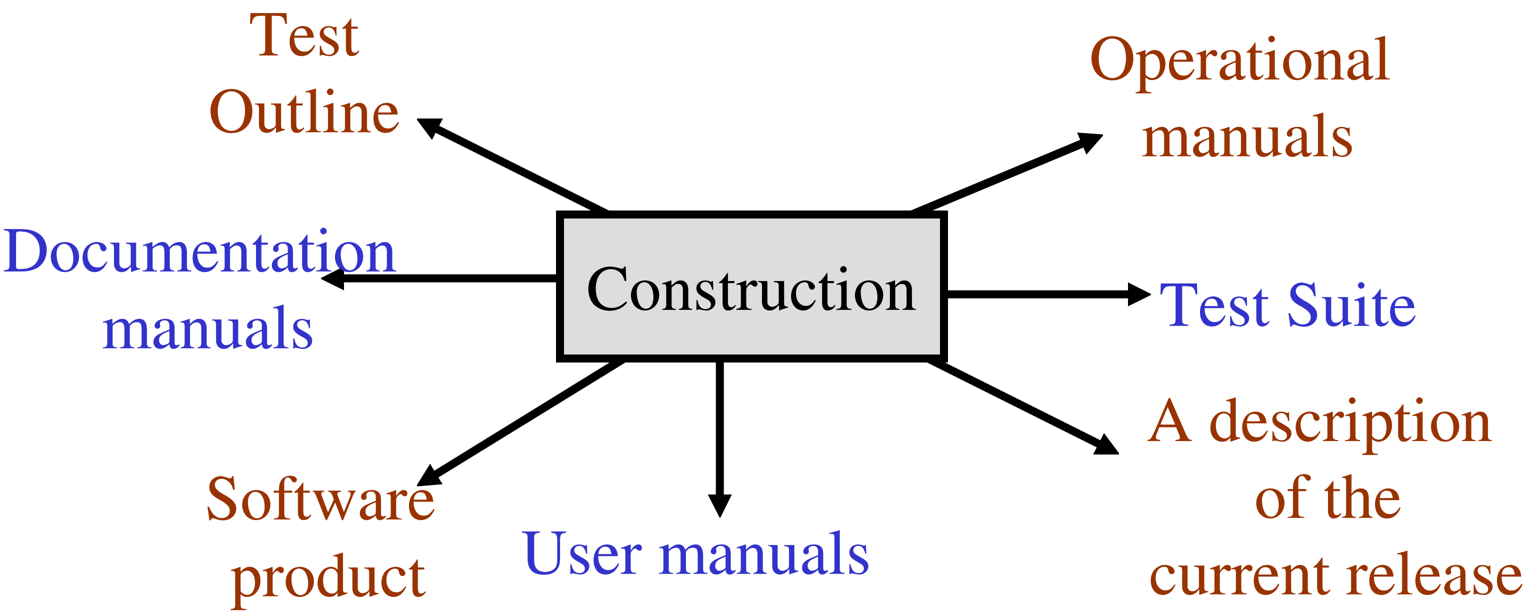

Development plan
Revised risk document
Preliminary User manual
Elaboration
An executable architectural prototype
Architecture Description document
Use case model
Supplementary Requirements with non functional requirement

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

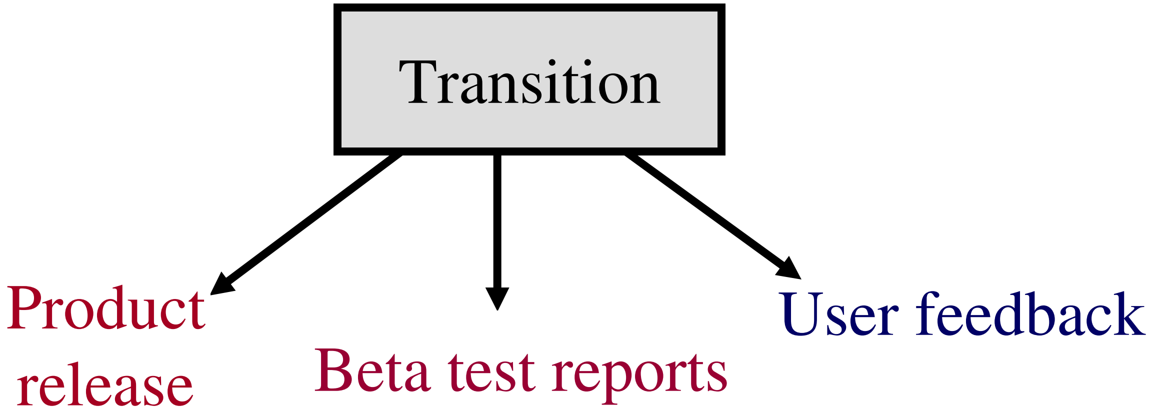


Transition Phase

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



Transition

Product release

Beta test reports

User feedback

Selection of a Life Cycle Model

Selection of a model is based on:

a) Requirements

b) Development team

c) Users

d) 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

Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

2.1 Spiral Model was developed by

(a) Bev Littlewood (b) Berry Boehm
(c) Roger Pressman (d) Victor Basili

2.2 Which model is most popular for student's small projects?

(a) Waterfall model (b) Spiral model
(c) Quick and fix model (d) Prototyping model

2.3 Which is not a software life cycle model?

(a) Waterfall model (b) Spiral model
(c) Prototyping model (d) Capability maturity model

2.4 Project risk factor is considered in

(a) Waterfall model (b) Prototyping model
(c) Spiral model (d) Iterative enhancement model

2.5 SDLC stands for

(a) Software design life cycle (b) Software development life cycle
(c) System development life cycle (d) System design life cycle

Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

2.6 Build and fix model has

(a) 3 phases (b) 1 phase

(c) 2 phases (d) 4 phases

2.7 SRS stands for

(a) Software requirements specification (b) Software requirements solution

(c) System requirements specification (d) none of the above

2.8 Waterfall model is not suitable for

(a) small projects (b) accommodating change

(c) complex projects (d) none of the above

2.9 RAD stands for

(a) Rapid application development (b) Relative application development

(c) Ready application development (d) Repeated application development

2.10 RAD model was proposed by

(a) Lucent Technologies (b) Motorola

(c) IBM (d) Microsoft

Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

2.11 If requirements are easily understandable and defined,which model is best suited?

(a) Waterfall model (b) Prototyping model

(c) Spiral model (d) None of the above

2.12 If requirements are frequently changing, which model is to be selected?

(a) Waterfall model (b) Prototyping model

(c) RAD model (d) Iterative enhancement model

2.13 If user participation is available, which model is to be chosen?

(a) Waterfall model (b) Iterative enhancement model

(c) Spiral model (d) RAD model

2.14 If limited user participation is available, which model is to be selected?

(a) Waterfall model (b) Spiral model

(c) Iterative enhancement model (d) any of the above

2.15 If project is the enhancement of existing system, which model is best suited?

(a) Waterfall model (b) Prototyping model

(c) Iterative enhancement model (d) Spiral model

Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

2.16 Which one is the most important feature of spiral model?

(a) Quality management (b) Risk management

(c) Performance management (d) Efficiency management

2.17 Most suitable model for new technology that is not well understood is:

(a) Waterfall model (b) RAD model

(c) Iterative enhancement model (d) Evolutionary development model

2.18 Statistically, the maximum percentage of errors belong to the following phase of SDLC

(a) Coding (b) Design

(c) Specifications (d) Installation and maintenance

2.19 Which phase is not available in software life cycle?

(a) Coding (b) Testing

(c) Maintenance (d) Abstraction

2.20 The development is supposed to proceed linearly through the phase in

(a) Spiral model (b) Waterfall model

(c) Prototyping model (d) None of the above

Multiple Choice Questions

Note: Select most appropriate answer of the following questions:

2.21 Unified process is maintained by

(a) Infosys	(b) Rational software corporation
(c) SUN Microsystems	(d) None of the above

2.22 Unified process is

(a) Iterative	(b) Incremental
(c) Evolutionary	(d) All of the above

2.23 Who is not in the team of Unified process development?

(a) I.Jacobson	(b) G.Booch
(c) B.Boehm	(d) J.Rumbaugh

2.24 How many phases are in the unified process?

(a) 4	(b) 5
(c) 2	(d) None of the above

2.25 The outcome of construction phased can be treated as:

(a) Product release	(b) Beta release
(c) Alpha release	(d) All of the above

Exercises

2.1 What do you understand by the term Software Development Life Cycle (SDLC)? Why is it important to adhere to a life cycle model while developing a large software product?

2.2 What is software life cycle? Discuss the generic waterfall model.

2.3 List the advantages of using waterfall model instead of adhoc build and fix model.

2.4 Discuss the prototyping model. What is the effect of designing a prototype on the overall cost of the project?

2.5 What are the advantages of developing the prototype of a system?

2.6 Describe the type of situations where iterative enhancement model might lead to difficulties.

2.7 Compare iterative enhancement model and evolutionary process model.

Exercises

2.8 Sketch a neat diagram of spiral model of software life cycle.

2.9 Compare the waterfall model and the spiral model of software development.

2.10 As we move outward along with process flow path of the spiral model, what can we say about software that is being developed or maintained.

2.11 How does "project risk" factor effect the spiral model of software development.

2.12 List the advantages and disadvantages of involving a software engineer throughout the software development planning process.

2.13 Explain the spiral model of software development. What are the limitations of such a model?

2.14 Describe the rapid application development (RAD) model.Discuss each phase in detail.

2.15 What are the characteristics to be considered for the selection of the life cycle model?

Exercises

2.16 What is the role of user participation in the selection of a life cycle model?.

2.17 Why do we feel that characteristics of requirements play a very significant role in the selection of a life cycle model?

2.18 Write short note on “status of development team” for the selection of a life cycle model?.

2.19 Discuss the selection process parameters for a life cycle model.

2.20 What is unified process? Explain various phases along with the outcome of each phase.

2.21 Describe the unified process work products after each phase of unified process.

2.22 What are the advantages of iterative approach over sequential approach? Why is unified process called as iterative or incremental?

Software Requirements Analysis and specification

Requirement Engineering

Requirements describe

<u>What</u> not <u>How</u>

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

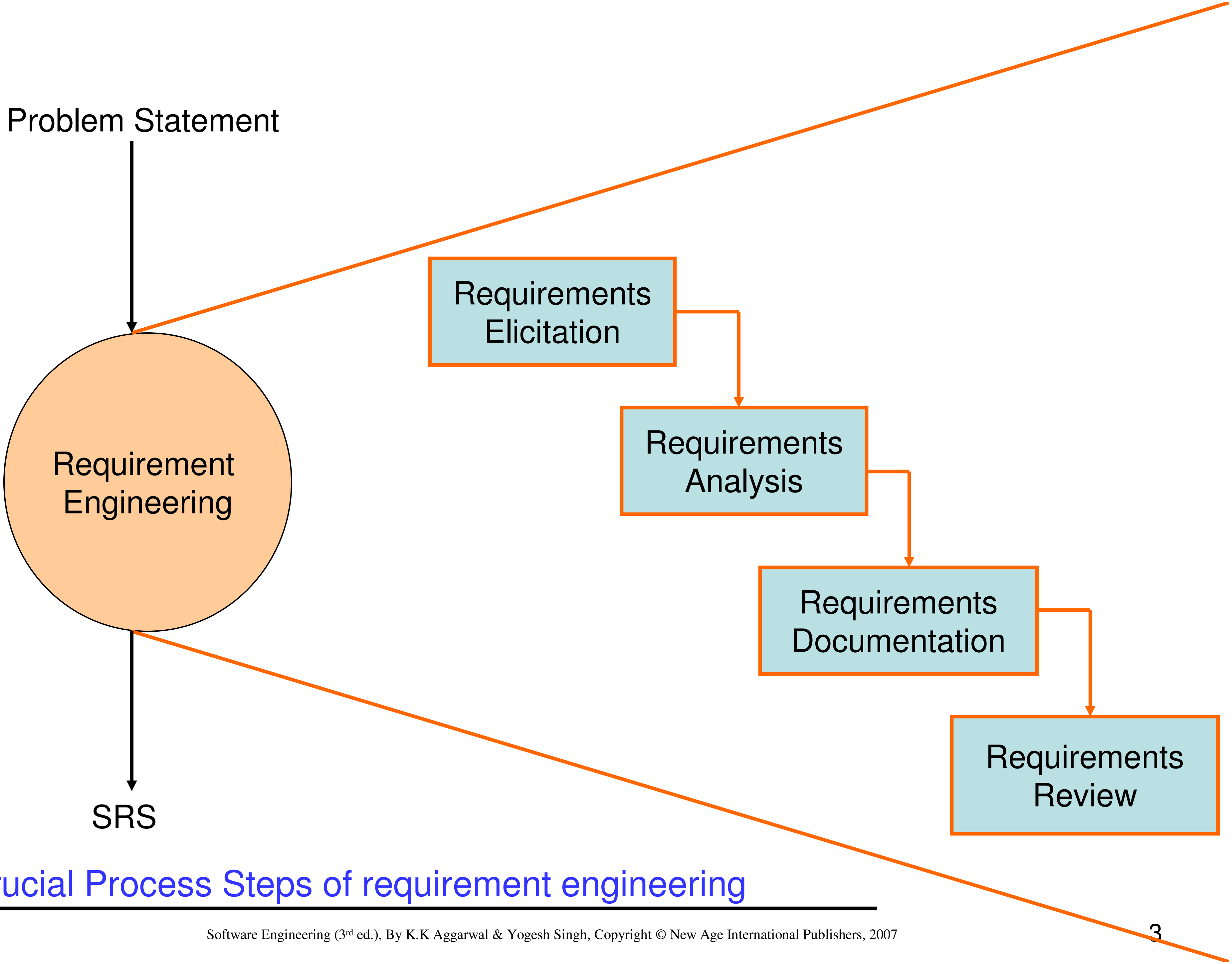
Quality of product → Process that creates it

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.

State of practice

Requirements are difficult to uncover

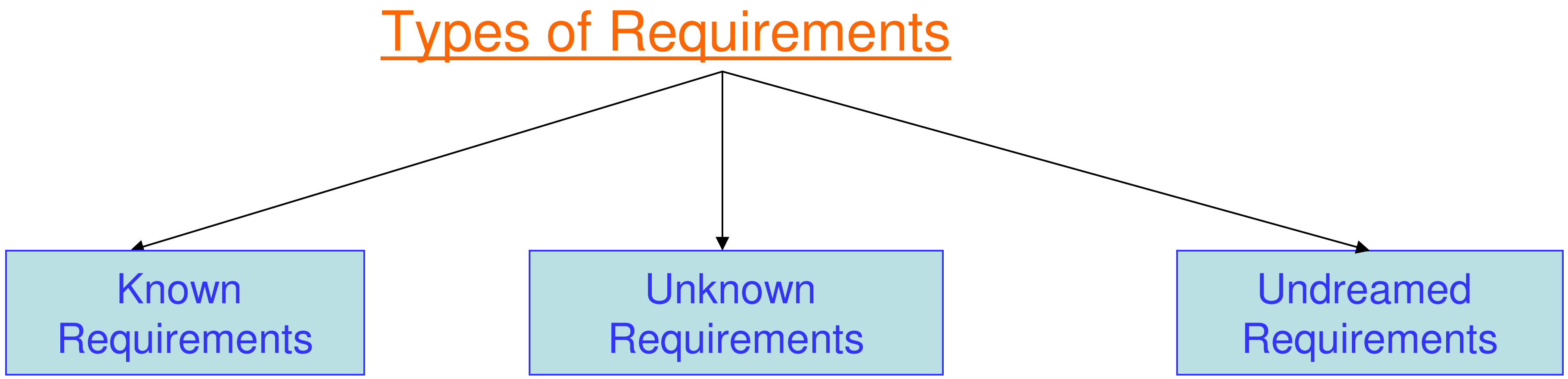
- Requirements change
- Over reliance on CASE Tools
- Tight project Schedule
- Communication barriers
- Market driven software development
- Lack of resources

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 Requirements

Known Requirements

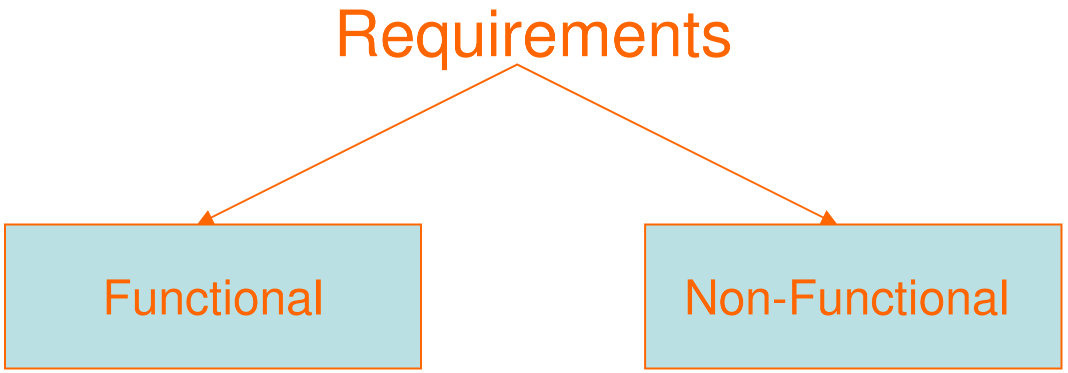
Unknown Requirements

Undreamed Requirements

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

--- User

--- Affected persons



Requirements

Functional

Non-Functional

Types of Requirements

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 Requirements

User and system requirements

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

Types of Requirements

Interface Specification

- Important for the customers.

TYPES OF INTERFACES

- Procedural interfaces (also called Application Programming Interfaces (APIs)).
- Data structures
- Representation of data.

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

Feasibility Study

Technical feasibility

- Is it technically feasible to provide direct communication connectivity through space from one location of globe to another location?

- Is it technically feasible to design a programming language using “Sanskrit”?

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

Succeed

→ effective customer developer partnership

Selection of any method

1. It is the only method that we know
2. It is our favorite method for all situations
3. We understand intuitively that the method is effective in the present circumstances.

Normally we rely on first two reasons.

Requirements Elicitation

1. Interviews

Both parties have a common goal → Success of the project

--- open ended
--- structured } Interview

Selection of stakeholder

1. Entry level personnel
2. Middle level stakeholder
3. Managers
4. Users of the software (Most important)

Requirements Elicitation

Types of questions.

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

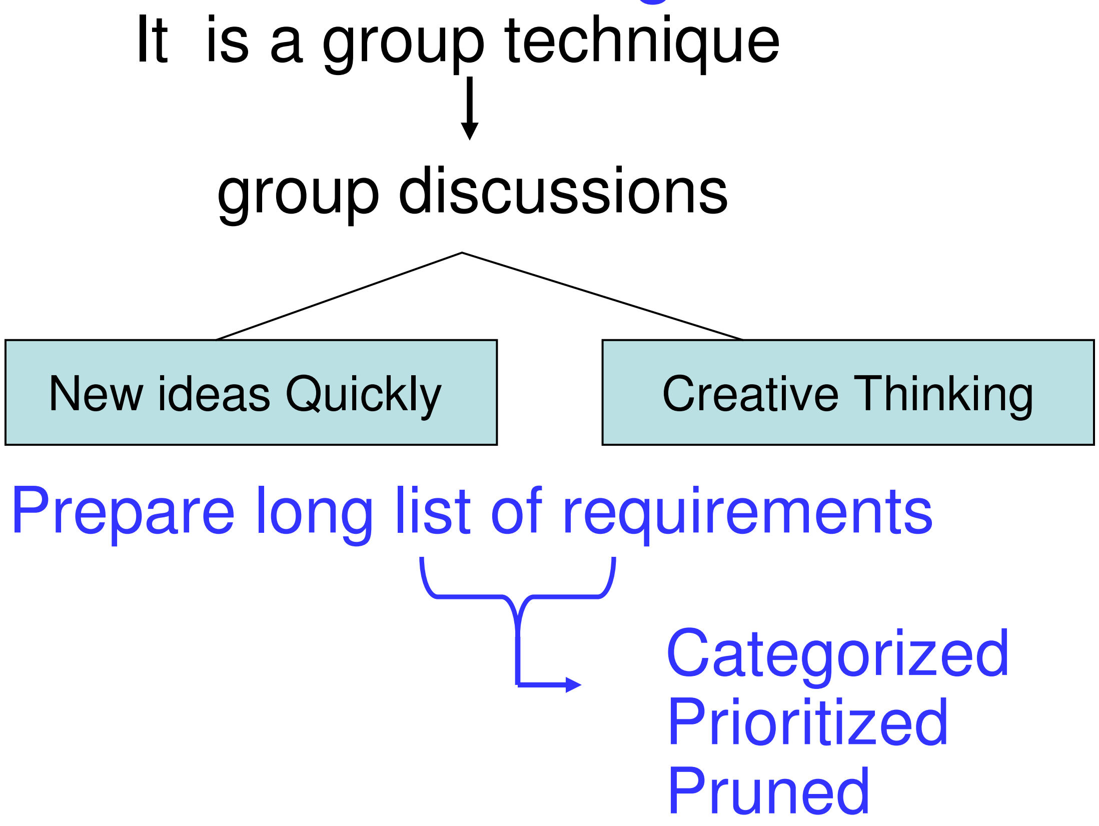
Requirements Elicitation

5. Possible benefits
6. Satisfied with current policies
7.How are you maintaining the records of previous students?
8. Any requirement of data from other system
9. Any specific problems
10. Any additional functionality
11. Most important goal of the proposed development

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

Requirements Elicitation

2. 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.

Groups

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

Requirements Elicitation

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.

3. Facilitated Application specification Techniques (FAST)

-- Similar to brainstorming sessions.

-- Team oriented approach

-- Creation of joint team of customers and developers.

Requirements Elicitation

Guidelines

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

FAST session Preparations

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

Requirements Elicitation

1. Part of environment that surrounds the system.
2. Produced by the system.
3. Used by the system.

A. List of constraints
B. Functions
C. Performance criteria

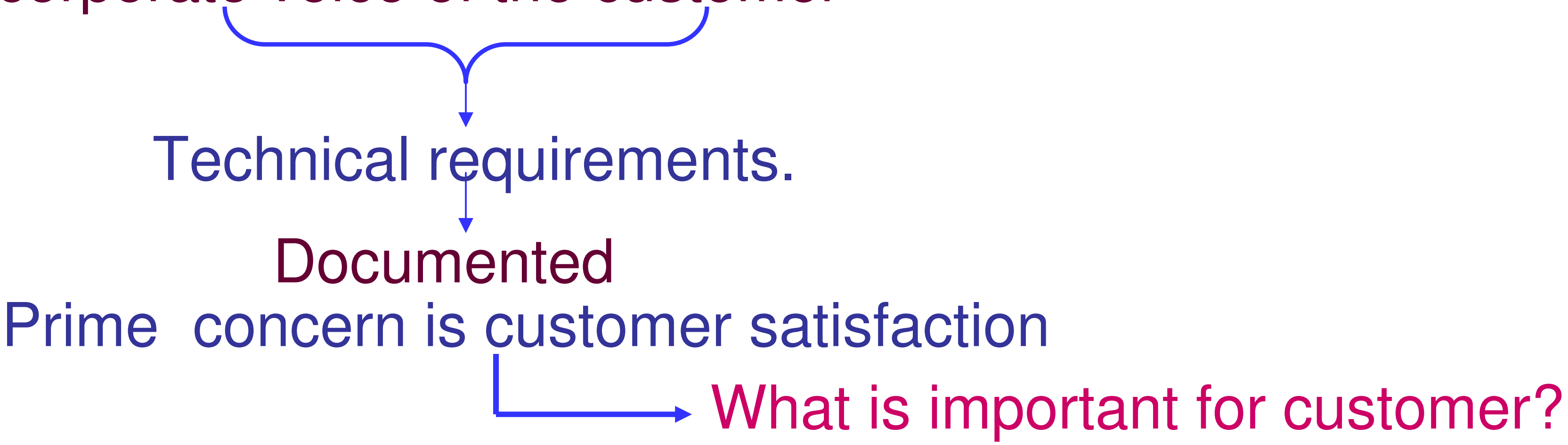
Activities of FAST session

1. Every participant presents his/her list
2. Combine list for each topic
3. Discussion
4. Consensus list
5. Sub teams for mini specifications
6. Presentations of mini-specifications
7. Validation criteria
8. A sub team to draft specifications

Requirements Elicitation

4. 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

Requirements Elicitation

Steps

1. Identify stakeholders
2. List out requirements
3. Degree of importance to each requirement.

Requirements Elicitation

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:

(i) It is possible to achieve

(ii) It should be deferred & Why

(iii) It is impossible and should be dropped from consideration

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

Requirements Elicitation

5. 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.

Requirements Elicitation

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

Requirements Elicitation

- 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.

Requirements Elicitation

* 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.

Requirements Elicitation

*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.

Requirements Elicitation

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

1. Identify all users

2. Create a user profile for each category of users including all roles of the users play that are relevant to the system.

3. 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.

4. Structure the use case

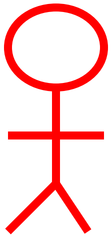
5. Review and validate with users.

Requirements Elicitation

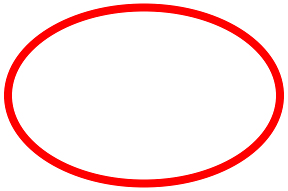
Use case Diagrams

-- represents what happens when actor interacts with a system.

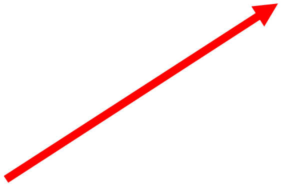
-- captures functional aspect of the system.



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.

Requirements Elicitation

*Use cases should not be used to capture all the details of the system.

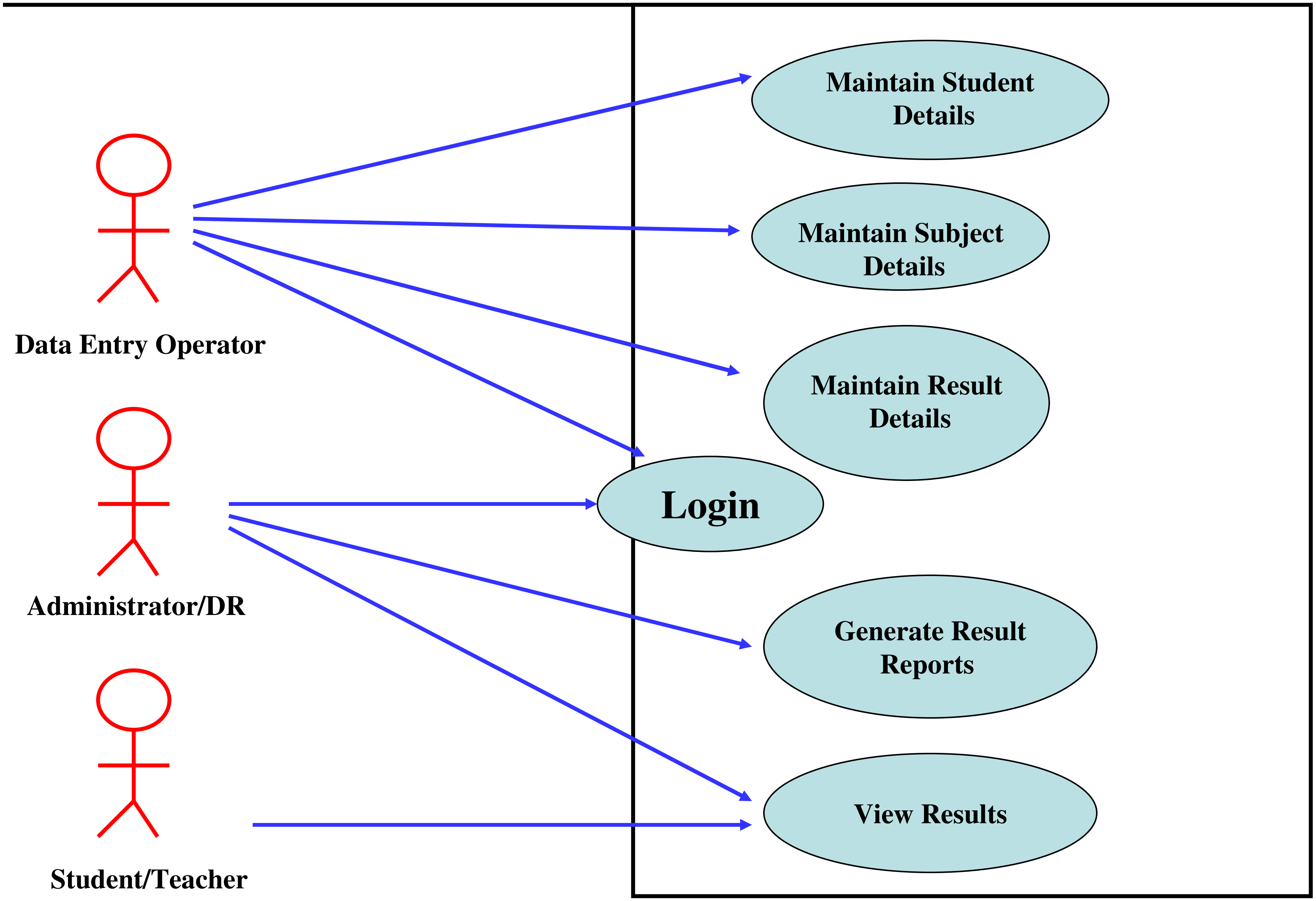
*Only significant aspects of the required functionality

*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
   (i) According to course code
   (ii) According to Enrollment number/roll number

Requirements Elicitation (Use Case)

Login

1.1 Introduction : This use case describes how a user logs into the Result Management System.

1.2 Actors : (i) Data Entry Operator
(ii) Administrator/Deputy Registrar

1.3 Pre Conditions : None

1.4 Post Conditions : If the use case is successful, the actor is logged into the system. If not, the system state is unchanged.

Requirements Elicitation (Use Case)

1.5 Basic Flow : This use case starts when the actor wishes to login to the Result Management system.

(i) System requests that the actor enter his/her name and password.

(ii) The actor enters his/her name & password.

(iii) System validates name & password, and if finds correct allow the actor to logs into the system.

Use Cases

1.6 Alternate Flows

1.6.1 Invalid name & password

If in the basic flow, the actor enters an invalid name and/or password, the system displays an error message. The actor can choose to either return to the beginning of the basic flow or cancel the login, at that point, the use case ends.

1.7 Special Requirements:

None

1.8 Use case Relationships:

None

Use Cases

2.Maintain student details

2.1 Introduction : Allow DEO to maintain student details. This includes adding, changing and deleting student information

2.2 Actors : DEO

2.3 Pre-Conditions: DEO must be logged onto the system before this use case begins.

Use Cases

2.4 Post-conditions : If use case is successful, student information is added/updated/deleted from the system. Otherwise, the system state is unchanged.

2.5 Basic Flow : Starts when DEO wishes to add/modify/update/delete Student information.

(i) The system requests the DEO to specify the function, he/she would like to perform (Add/update/delete)

(ii) One of the sub flow will execute after getting the information.

Use Cases

- ❑ If DEO selects "Add a student", "Add a student" sub flow will be executed.

- ❑ If DEO selects "update a student", "update a student" sub flow will be executed.

- ❑ If DEO selects "delete a student", "delete a student" sub flow will be executed.

2.5.1 Add a student

(i) The system requests the DEO to enter:

Name
Address
Roll No
Phone No
Date of admission

(ii) System generates unique id

Use Cases

2.5.2 Update a student

(i) System requires the DEO to enter student-id.

(ii) DEO enters the student_id. The system retrieves and display the student information.

(iii) DEO makes the desired changes to the student information.

(iv) After changes, the system updates the student record with changed information.

Use Cases

2.5.3 Delete a student

(i) The system requests the DEO to specify the student-id.

(ii) DEO enters the student-id. The system retrieves and displays the student information.

(iii) The system prompts the DEO to confirm the deletion of the student.

(iv) The DEO confirms the deletion.

(v) The system marks the student record for deletion.

2.6 Alternative flows

2.6.1 Student not found

If in the update a student or delete a student sub flows, a student with specified_id does not exist, the system displays an error message .The DEO may enter a different id or cancel the operation. At this point ,Use case ends.

Use Cases

2.6.2 Update Cancelled

If in the update a student sub-flow, the data entry operator decides not to update the student information, the update is cancelled and the basic flow is restarted at the begin.

Use Cases

2.6.3 Delete cancelled

If in the delete a student sub flows, DEO decides not to delete student record ,the delete is cancelled and the basic flow is re-started at the beginning.

2.7 Special requirements

None

2.8 Use case relationships

None

3. Maintain Subject Details

3.1 Introduction

The DEO to maintain subject information. This includes adding, changing, deleting subject information from the system

3.2 Actors : DEO

3.3 Preconditions : DEO must be logged onto the system before the use case begins.

Use Cases

3.4 Post conditions :

If the use case is successful, the subject information is added, updated, or deleted from the system, otherwise the system state is unchanged.

3.5 Basic flows :

The use case starts when DEO wishes to add, change, and/or delete subject information from the system.

(i) The system requests DEO to specify the function he/she would like to perform i.e.

- Add a subject
- Update a subject
- Delete a subject.

Use Cases

(ii) Once the DEO provides the required information, one of the sub flows is executed.

- ❑ If DEO selected “Add a subject” the “Add-a subject sub flow is executed.
- ❑ If DEO selected “Update-a subject” the “update-a- subject” sub flow is executed
- ❑ If DEO selected “Delete- a- subject”, the “Delete-a-subject” sub flow is executed.

3.5.1 Add a Subject

(i) The System requests the DEO to enter the subject information. This includes :

* Name of the subject

Use Cases

* Subject Code
* Semester
* Credit points

(ii) Once DEO provides the requested information, the system generates and assigns a unique subject-id to the subject. The subject is added to the system.

(iii) The system provides the DEO with new subject-id.

Use Cases

3.5.2 Update a Subject

(i) The system requests the DEO to enter subject_id.

(ii) DEO enters the subject_id. The system retrieves and displays the subject information.

(iii) DEO makes the changes.

(iv) Record is updated.

Use Cases

3.5.3 Delete a Subject

(i) Entry of subject_id.

(ii) After this, system retrieves & displays subject information.

* System prompts the DEO to confirm the deletion.
* DEO verifies the deletion.
* The system marks the subject record for deletion.

3.6 Alternative Flow

3.6.1 Subject not found

If in any sub flows, subject-id not found, error message is displayed. The DEO may enter a different id or cancel the case ends here.

3.6.2 Update Cancelled

If in the update a subject sub-flow, the data entry operator decides not to update the subject information, the update is cancelled and the basic flow is restarted at the begin.

Use Cases

3.6.3 Delete Cancellation

If in delete-a-subject sub flow, the DEO decides not to delete subject, the delete is cancelled, and the basic flow is restarted from the beginning.

3.7 Special Requirements:

None

3.8 Use Case-relationships

None

Use Cases

4. Maintain Result Details

4.1 Introduction

This use case allows the DEO to maintain subject & marks information of each student. This includes adding and/or deleting subject and marks information from the system.

4.2 Actor

DEO

Use Cases

4.3 Pre Conditions

DEO must be logged onto the system.

4.4 Post Conditions

If use case is successful ,marks information is added or deleted from the system. Otherwise, the system state is unchanged.

Use Cases

4.5 Basic Flow

This use case starts, when the DEO wishes to add, update and/or delete marks from the system.

(i) DEO to specify the function

(ii) Once DEO provides the information one of the subflow is executed.

- * If DEO selected "Add Marks ", the Add marks subflow is executed.
- * If DEO selected "Update Marks", the update marks subflow is executed.
- * If DEO selected "Delete Marks", the delete marks subflow is executed.

Use Cases

4.5.1 Add Marks Records

Add marks information .This includes:

a. Selecting a subject code.

b. Selecting the student enrollment number.

c. Entering internal/external marks for that subject code & enrollment number.

Use Cases

(ii) If DEO tries to enter marks for the same combination of subject and enrollment number,the system gives a message that the marks have already been entered.

(iii) Each record is assigned a unique result_id.

4.5.2 Delete Marks records

1. DEO makes the following entries:
   a. Selecting subject for which marks have to be deleted.
   b. Selecting student enrollment number.
   c. Displays the record with id number.
   d. Verify the deletion.
   e. Delete the record.

Use Cases

4.5.2 Update Marks records

1. The System requests DEO to enter the record_id.

2. DEO enters record_id. The system retrieves & displays the information.

3. DEO makes changes.

4. Record is updated.

Use Cases

4.5.3 Compute Result

(i) Once the marks are entered, result is computed for each student.

(ii) If a student has scored more than 50% in a subject, the associated credit points are allotted to that student.

(iii) The result is displayed with subject-code, marks & credit points.

4.6 Alternative Flow

4.6.1 Record not found

If in update or delete marks sub flows, marks with specified id number do not exist, the system displays an error message. DEO can enter another id or cancel the operation.

4.6.2 Delete Cancelled

If in Delete Marks, DEO decides not to delete marks, the delete is cancelled and basic flow is re-started at the beginning.

4.7 Special Requirements

None

4.8 Use case relationships

None

Use Cases

5 View/Display result

5.1 Introduction

This use case allows the student/Teacher or anyone to view the result. The result can be viewed on the basis of course code and/or enrollment number.

5.2 Actors

Administrator/DR, Teacher/Student

5.3 Pre Conditions

None

5.4 Post Conditions

If use case is successful, the marks information is displayed by the system. Otherwise, state is unchanged.

Use Cases

5.5 Basic Flow

Use case begins when student, teacher or any other person wish to view the result.

Two ways

-- Enrollment no.

-- Course code

Use Cases

(ii) After selection, one of the sub flow is executed.

Course code → Sub flow is executed

Enrollment no. → Sub flow is executed

5.5.1 View result enrollment number wise

(i) User to enter enrollment number

(ii) System retrieves the marks of all subjects with credit points

(iii) Result is displayed.

Use Cases

5.6 Alternative Flow

5.6.1 Record not found

Error message should be displayed.

5.7 Special Requirements

None

5.8 Use Case relationships

None

Use Cases

6. Generate Report

6.1 Introduction

This use case allows the DR to generate result reports. Options are

a. Course code wise
b. Semester wise
c. Enrollment Number wise

6.2 Actors

DR

6.3 Pre-Conditions

DR must logged on to the system

Use Cases

6.4 Post conditions

If use case is successful, desired report is generated. Otherwise, the system state is unchanged.

6.5 Basic Flow

The use case starts, when DR wish to generate reports.

(i) DR selects option.

(ii) System retrieves the information displays.

(iii) DR takes printed reports.

Use Cases

6.6 Alternative Flows

6.6.1 Record not found

If not found, system generates appropriate message. The DR can select another option or cancel the operation. At this point, the use case ends.

6.7 Special Requirements

None

6.8 Use case relationships

None

7. Maintain User Accounts

7.1 Introduction

This use case allows the administrator to maintain user account. This includes adding, changing and deleting user account information from the system.

7.2 Actors

Administrator

7.3 Pre-Conditions

The administrator must be logged on to the system before the use case begins.

Use Cases

7.4 Post-Conditions

If the use case was successful, the user account information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

7.5 Basic Flow

This use case starts when the Administrator wishes to add, change, and/or delete use account information from the system.

(i) The system requests that the Administrator specify the function he/she would like to perform (either Add a User Account, Update a User Account, or Delete a User Account).

(ii) Once the Administrator provides the requested information, one of the sub-flows is executed

Use Cases

* If the Administrator selected “Add a User Account”, the **Add a User Account** sub flow is executed.

* If the Administrator selected “Update a User Account”, the **Update a User Account** sub-flow is executed.

* If the Administrator selected “Delete a User Account”, the **Delete a User Account** sub-flow is executed.22

7.5.1 Add a User Account

1. The system requests that the Administrator enters the user information. This includes:

(*a*) User Name

(*b*) User ID-should be unique for each user account

(*c*) Password

(*d*) Role

Use Cases

2. Once the Administrator provides the requested information, the user account information is added.

7.5.2 Update a User Account

1. The system requests that the Administrator enters the User ID.
2. The Administrator enters the User ID. The system retrieves and displays the user account information.
3. The Administrator makes the desired changes to the user account information. This includes any of the information specified in the **Add a User Account** sub-flow.
4. Once the Administrator updates the necessary information, the system updates the user account record with the updated information.

Use Cases

7.5.3 Delete a User Account

1. The system requests that the Administrator enters the User ID.
2. The Administrator enters the User ID. The system retrieves and displays the user account information.
3. The system prompts the Administrator to confirm the deletion of the user account.
4. The Administrator confirms the deletion.
5. The system deletes the user account record.

Use Cases

7.6 Alternative Flows

7.6.1 User Not Found

If in the **Update a User Account** or **Delete a User Account** sub-flows, a user account with the specified User ID does not exist, the system displays an error message. The Administrator can then enter a different User ID or cancel the operation, at which point the use case ends.

7.6.2 Update Cancelled

If in the **Update a User Account** sub-flow, the Administrator decides not to update the user account information, the update is cancelled and the **Basic Flow** is re-started at the beginning.

7.6.3 Delete Cancelled

If in the **Delete a User Account** sub-flow, the Administrator decides not to delete the user account information, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

7.7 Special Requirements

None

7.8 Use case relationships

None

8. Reset System

8.1 Introduction

This use case allows the allows the administrator to reset the system by deleting all existing information from the system .

8.2 Actors

Administrator

8.3 Pre-Conditions

The administrator must be logged on to the system before the use case begins.

Use Cases

8.4 Post-Conditions

If the use case was successful, all the existing information is deleted from the backend database of the system. Otherwise, the system state is unchanged.

8.5 Basic Flow

This use case starts when the Administrator wishes to reset the system.

i. The system requests the Administrator to confirm if he/she wants to delete all the existing information from the system.

ii. Once the Administrator provides confirmation, the system deletes all the existing information from the backend database and displays an appropriate message.

Use Cases

8.6 Alternative Flows

8.6.1 Reset Cancelled

If in the Basic Flow, the Administrator decides not to delete the entire existing information, the reset is cancelled and the use case ends.

8.7 Special Requirements

None

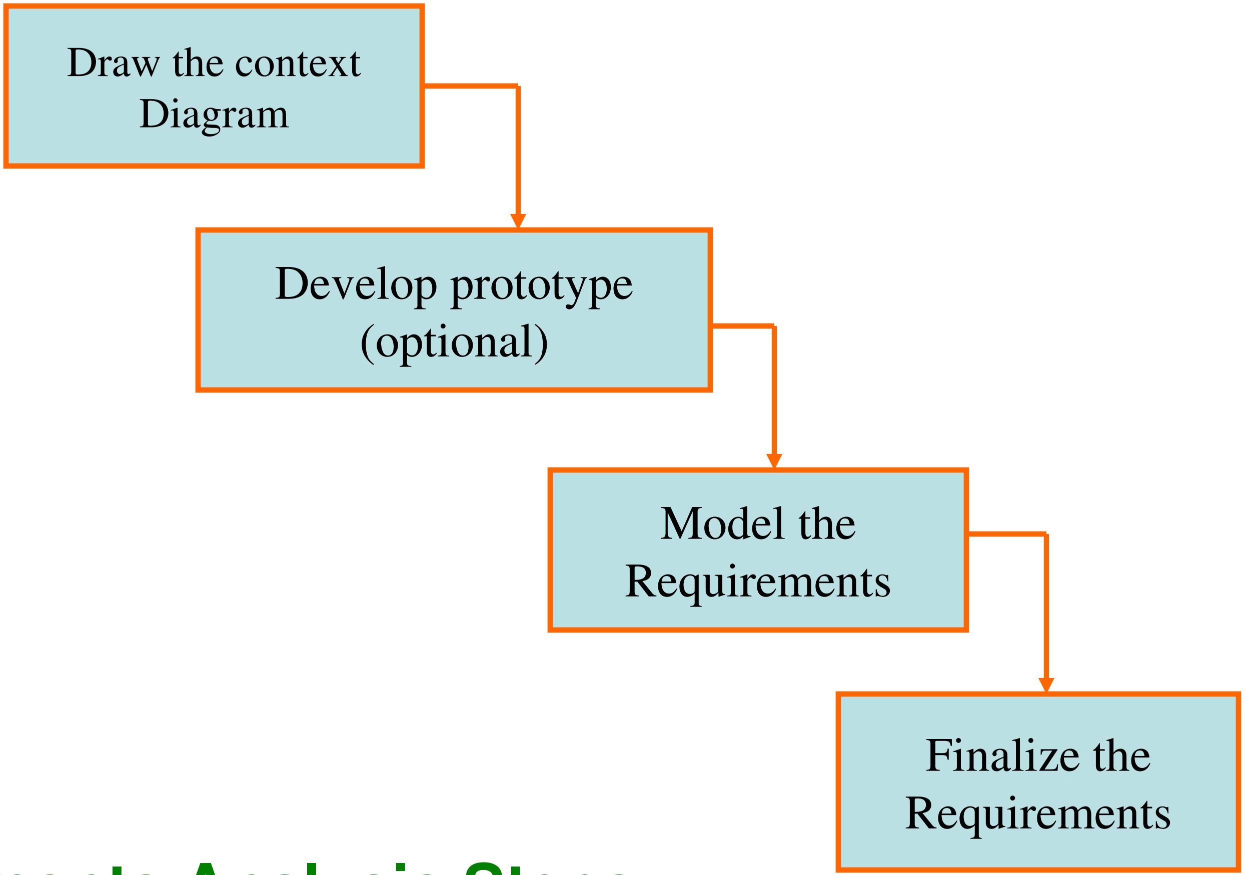
8.8 Use case relationships

None

Requirements Analysis

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

Steps



Draw the context Diagram

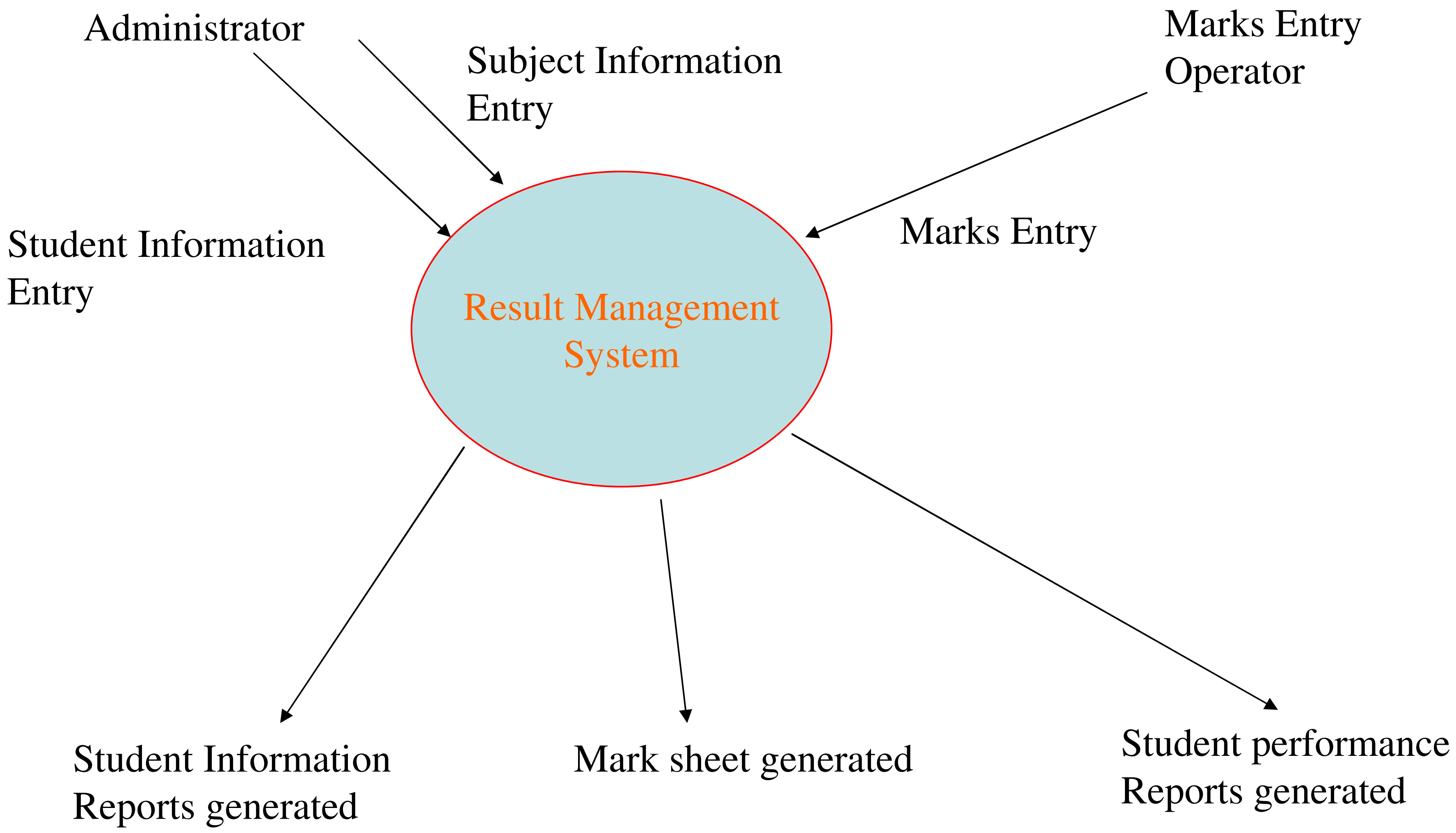
Develop prototype (optional)

Model the Requirements

Finalize the Requirements

Requirements Analysis Steps

Requirements Analysis


Administrator
Subject Information Entry
Marks Entry Operator
Student Information Entry
Result Management System
Marks Entry
Student Information Reports generated
Mark sheet generated
Student performance Reports generated

Requirements Analysis

Data Flow Diagrams

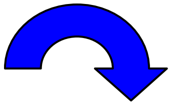
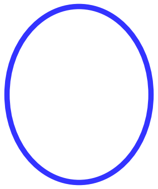
DFD show the flow of data through the system.

--All names should be unique

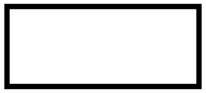
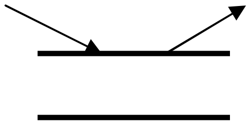
-- It is not a flow chart

-- 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.

Requirements Analysis

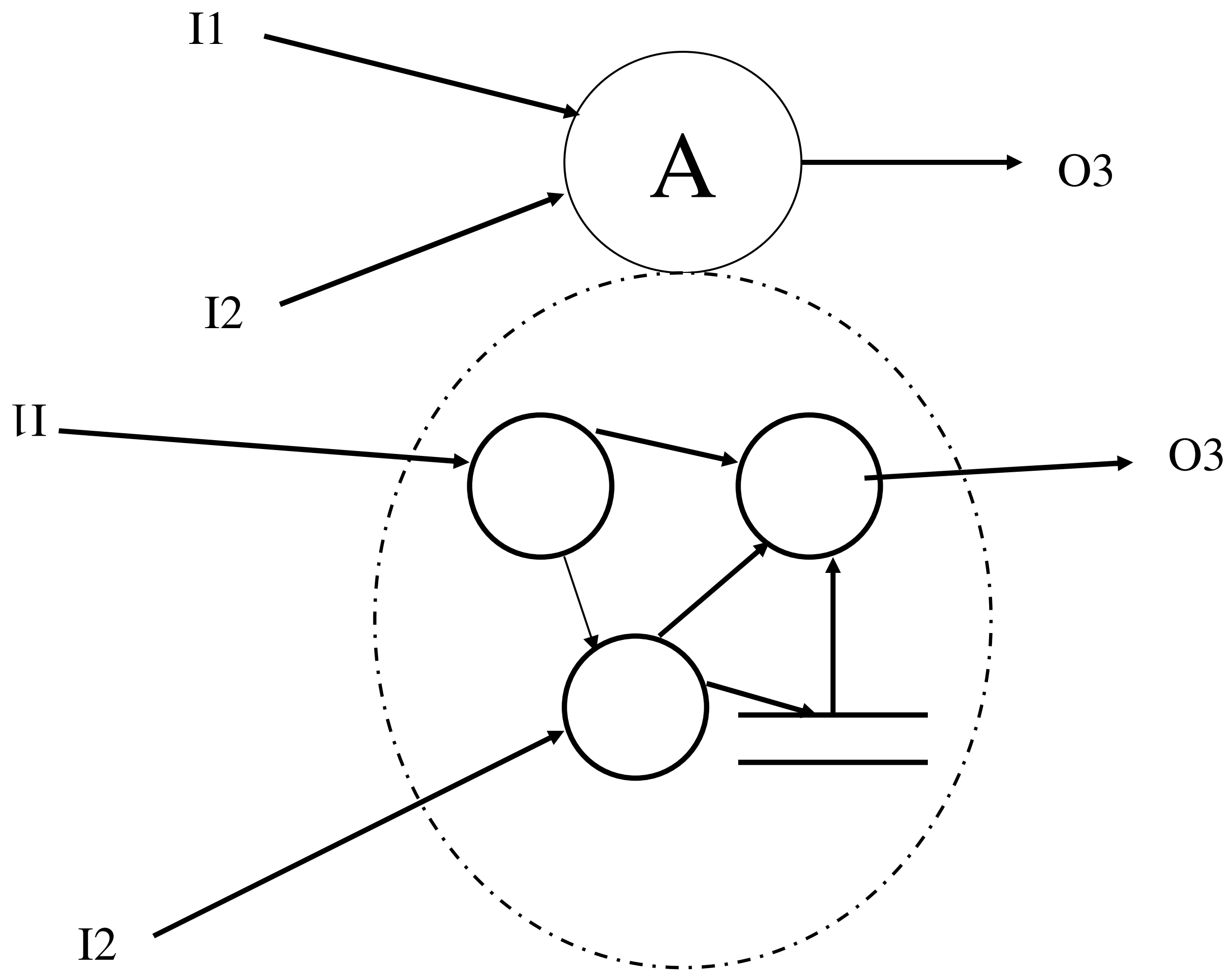
Symbol	Name	Function
	Source or sink	A source of system inputs or sink of system outputs
	Data Store	A repository of data, the arrowhead indicate net input and net outputs to store

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.

Requirements Analysis



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

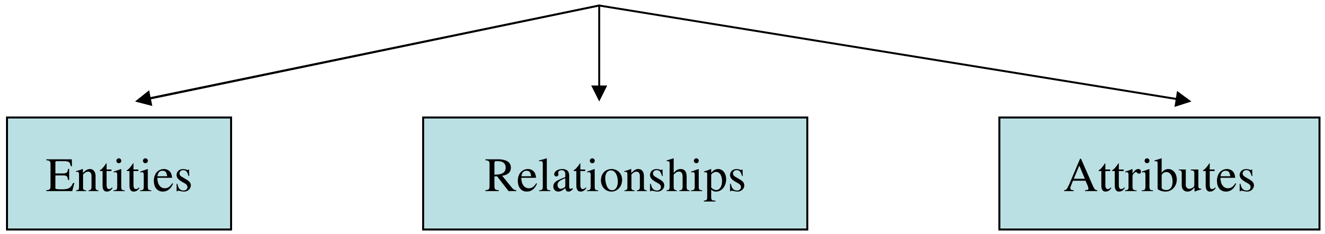
Data Dictionaries

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.



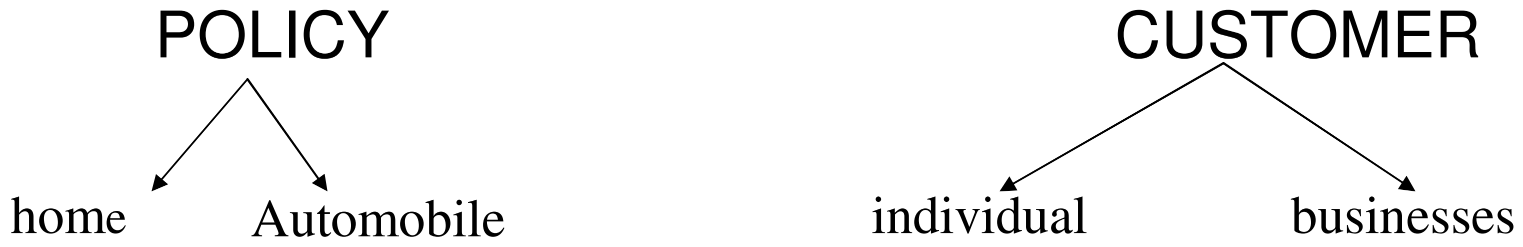
Entities

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

Consider an insurance company that offers both home and automobile insurance policies .These policies are offered to individuals and businesses.



POLICY

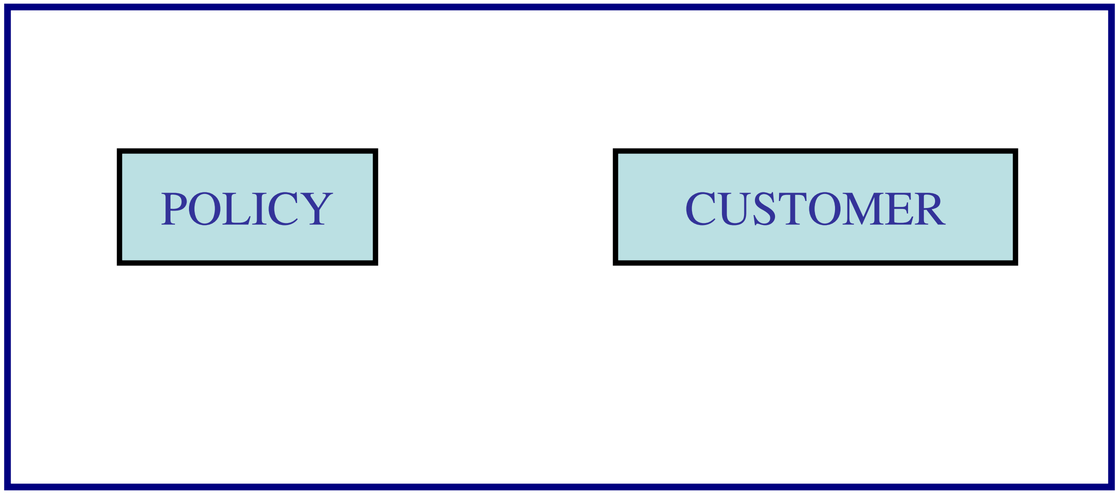
home

Automobile

CUSTOMER

individual

businesses



POLICY

CUSTOMER

Entity-Relationship Diagrams

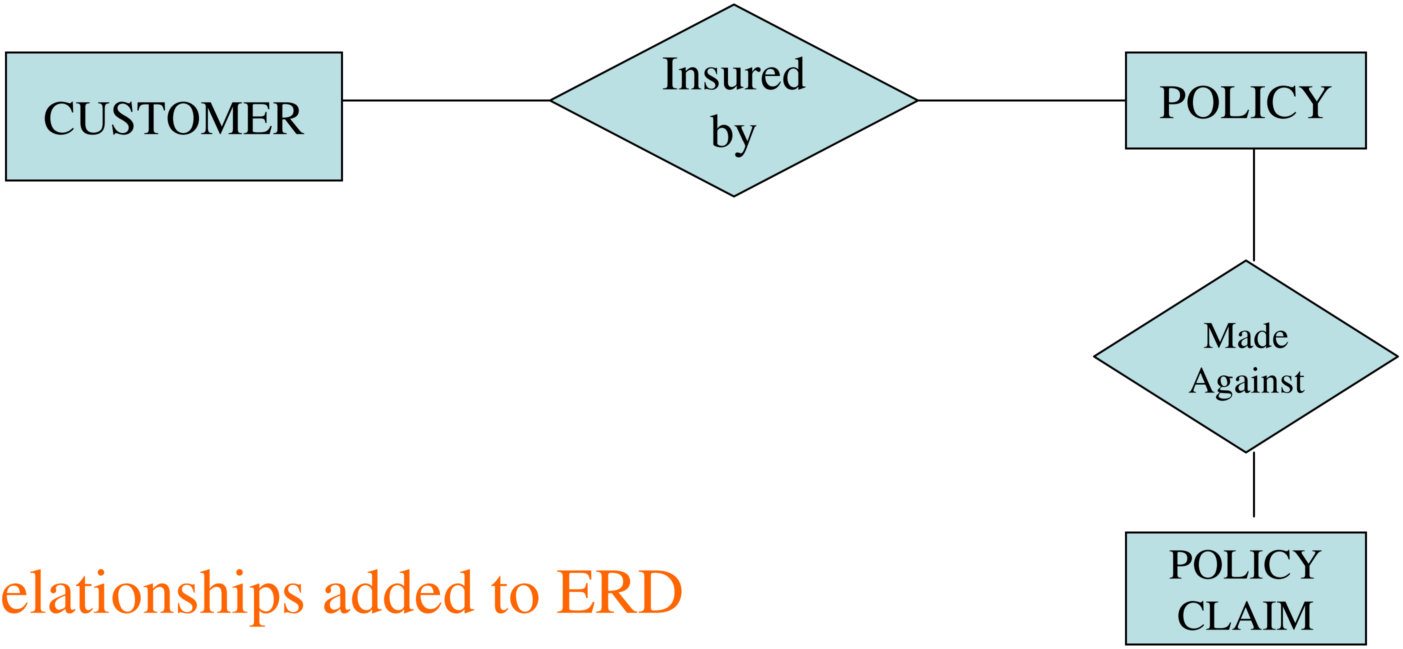
Relationships

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.

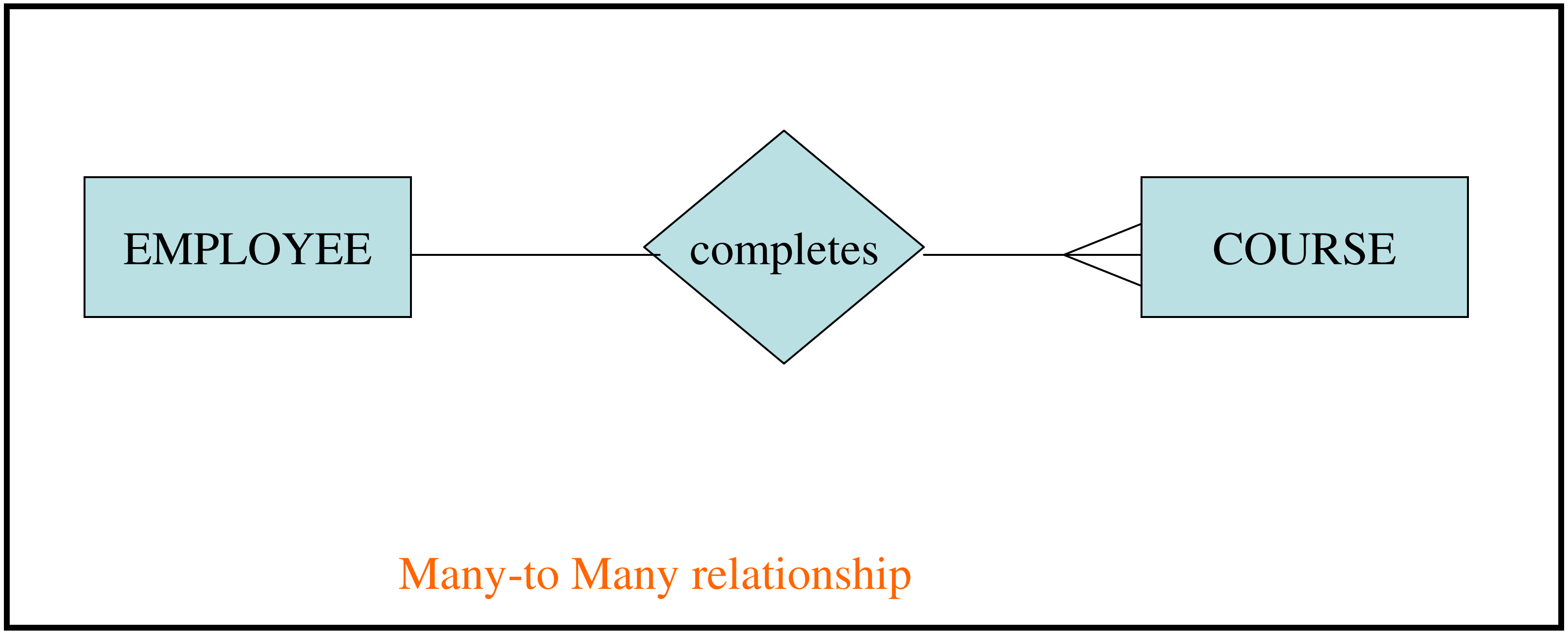
Relationships are represented by diamond notation in a ER diagram.



Relationships added to ERD

Entity-Relationship Diagrams

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



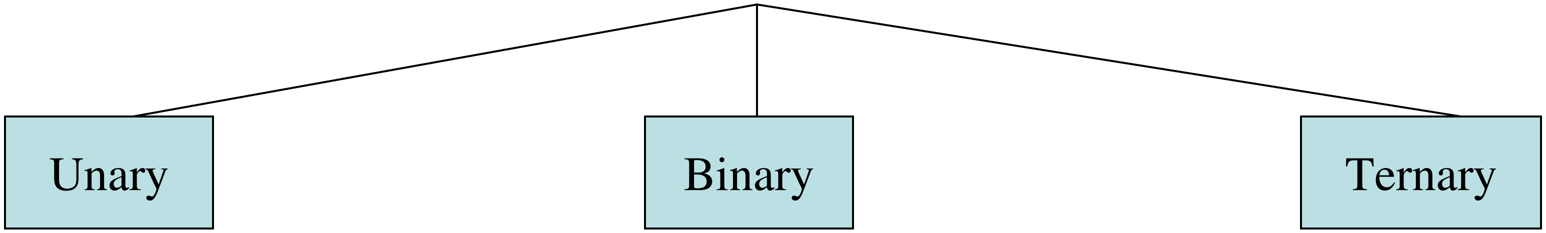
Many-to Many relationship

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.

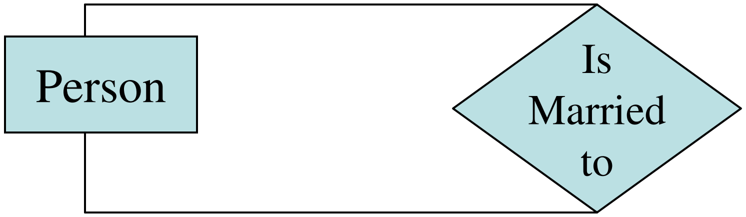


Unary

Binary

Ternary

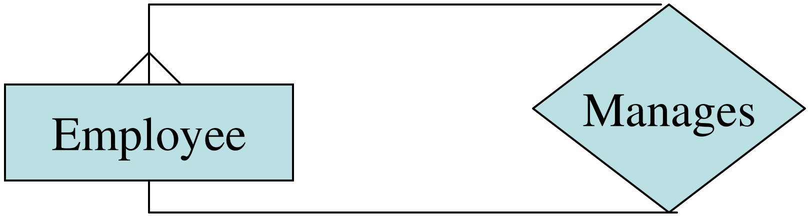
Unary relationship



Person

Is Married to

One to One



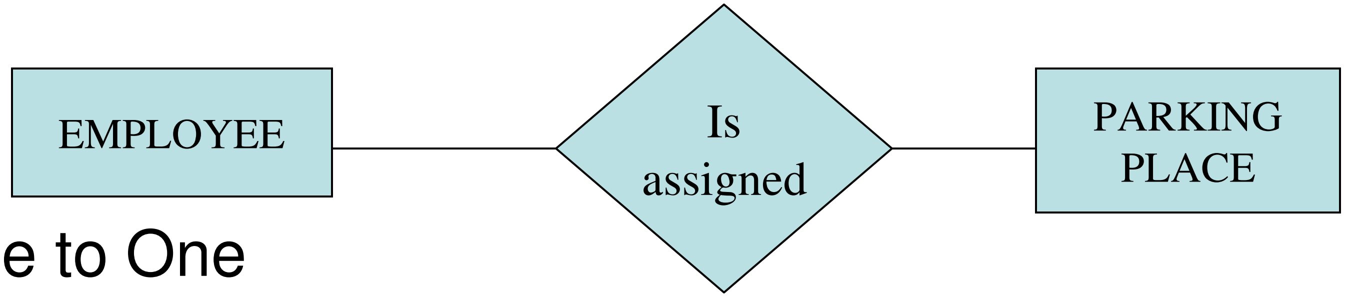
Employee

Manages

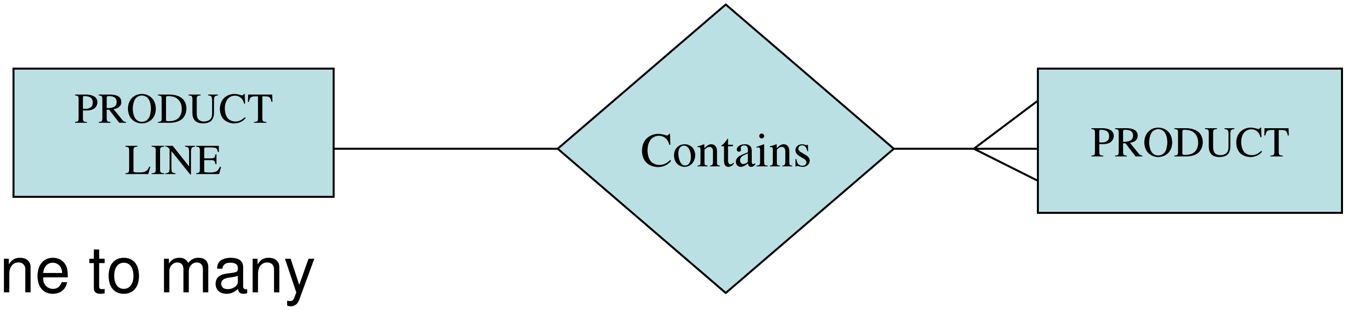
One to many

Entity-Relationship Diagrams

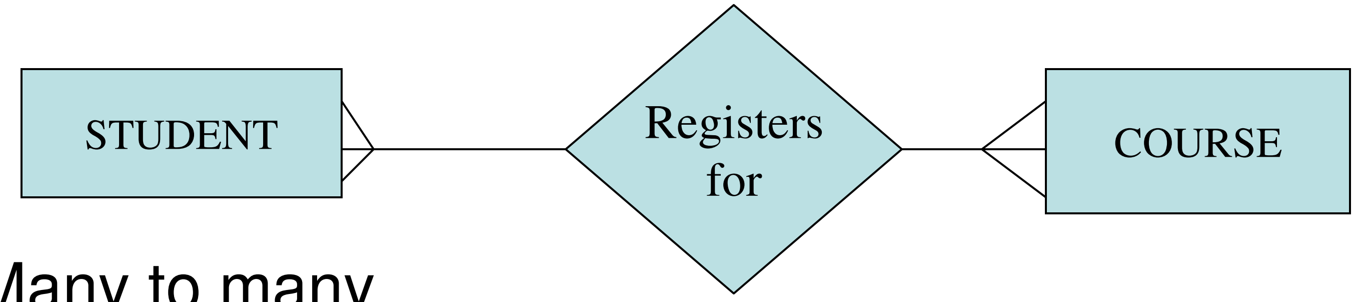
Binary Relationship



One to One



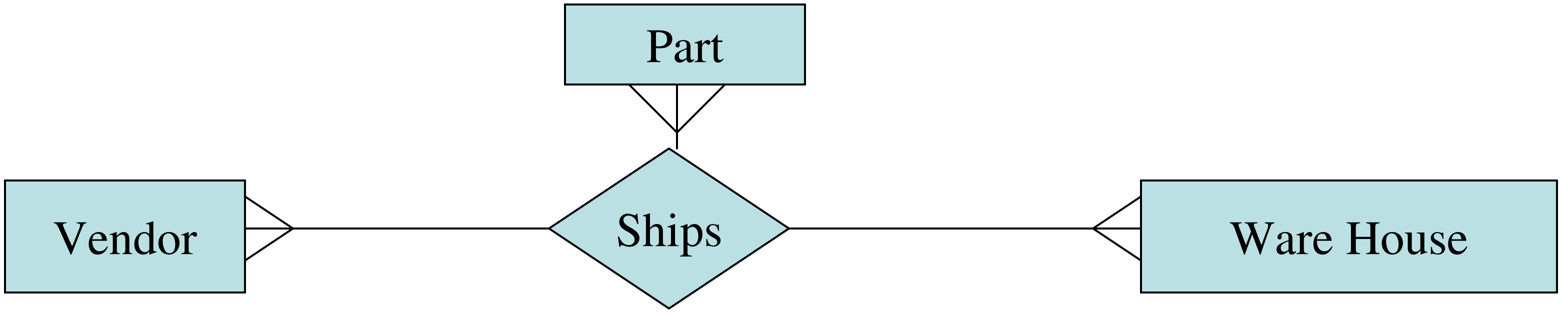
One to many



Many to many

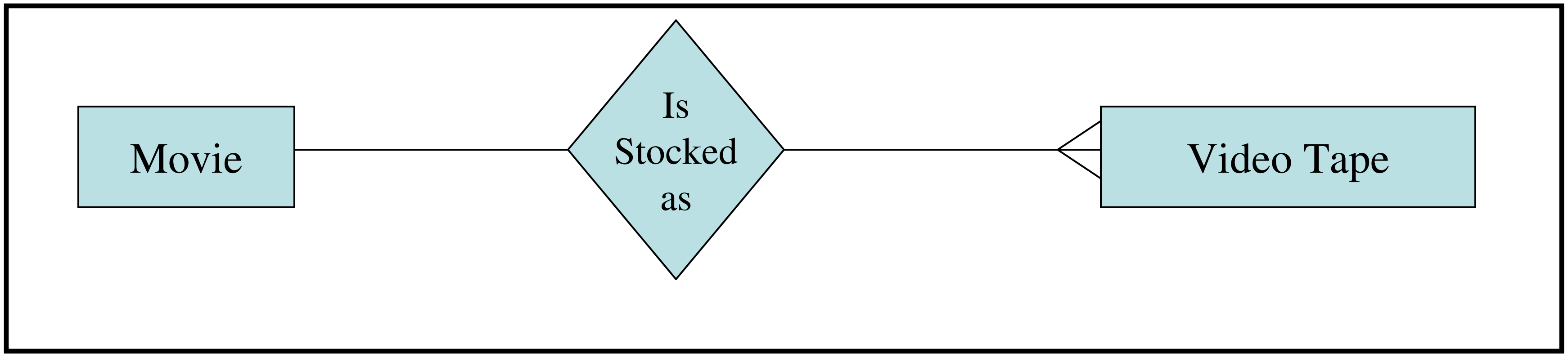
Entity-Relationship Diagrams

Ternary relationship



Cardinalities and optionality

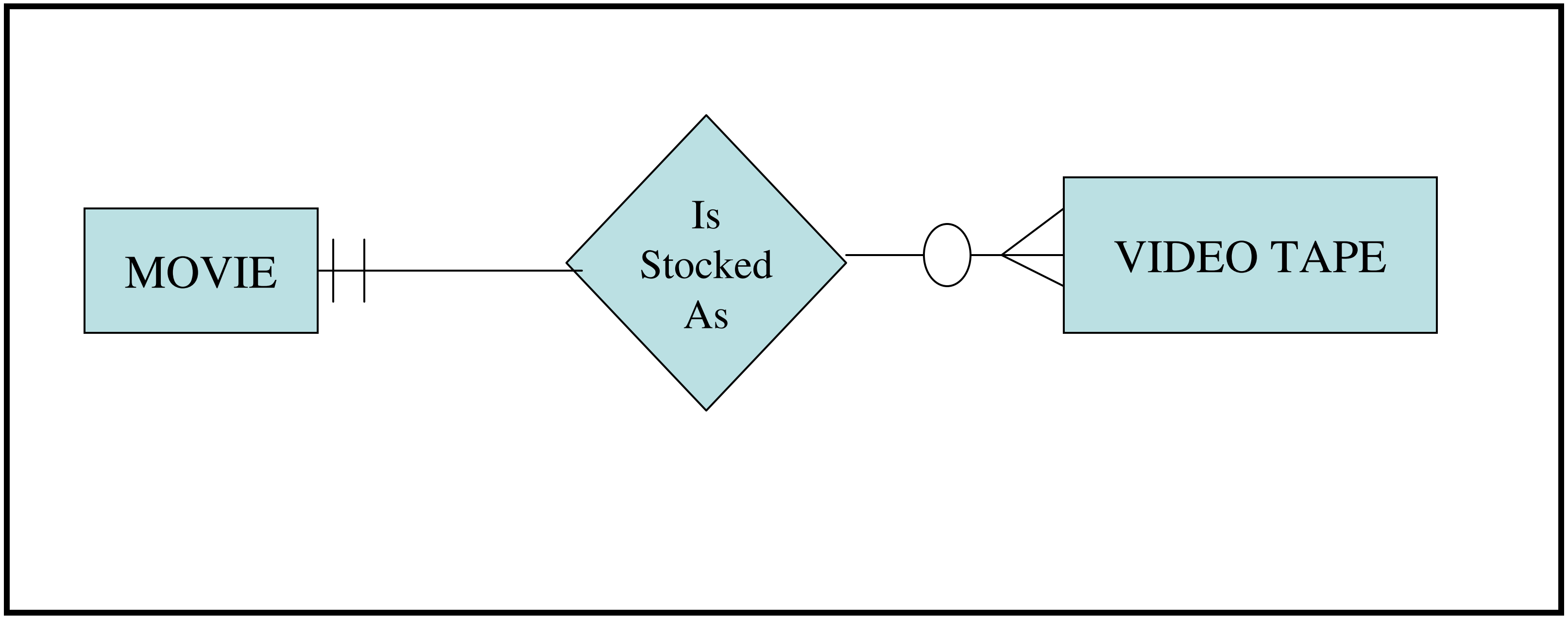
Two entity types A,B, connected by a relationship.
The cardinality of a relationship is the number of instances of entity B that can be associated with each instance of entity A



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.

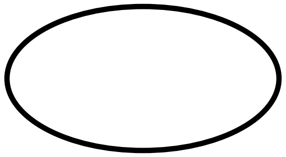


Entity-Relationship Diagrams

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

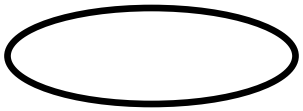
Entity-Relationship Diagrams

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

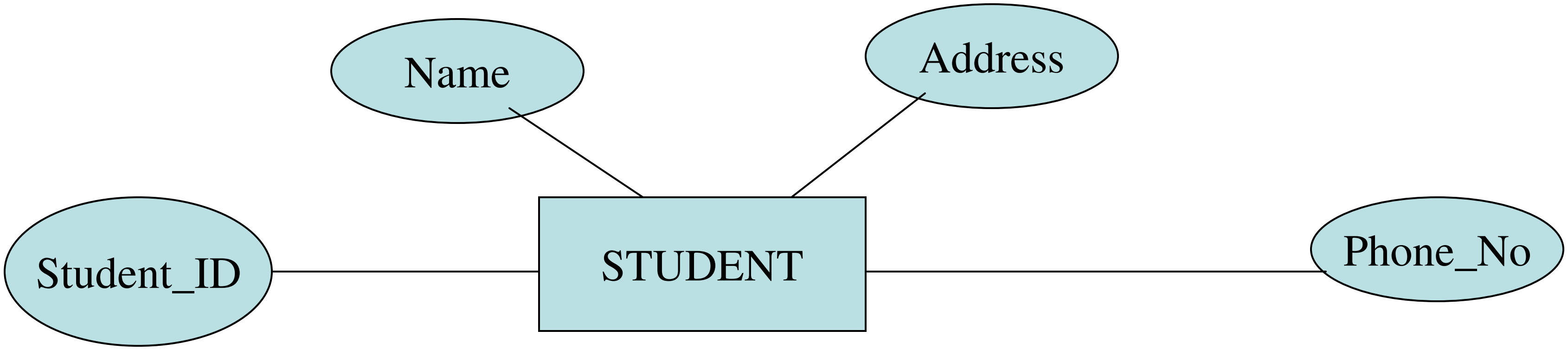
Student_ID → Candidate Key

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.

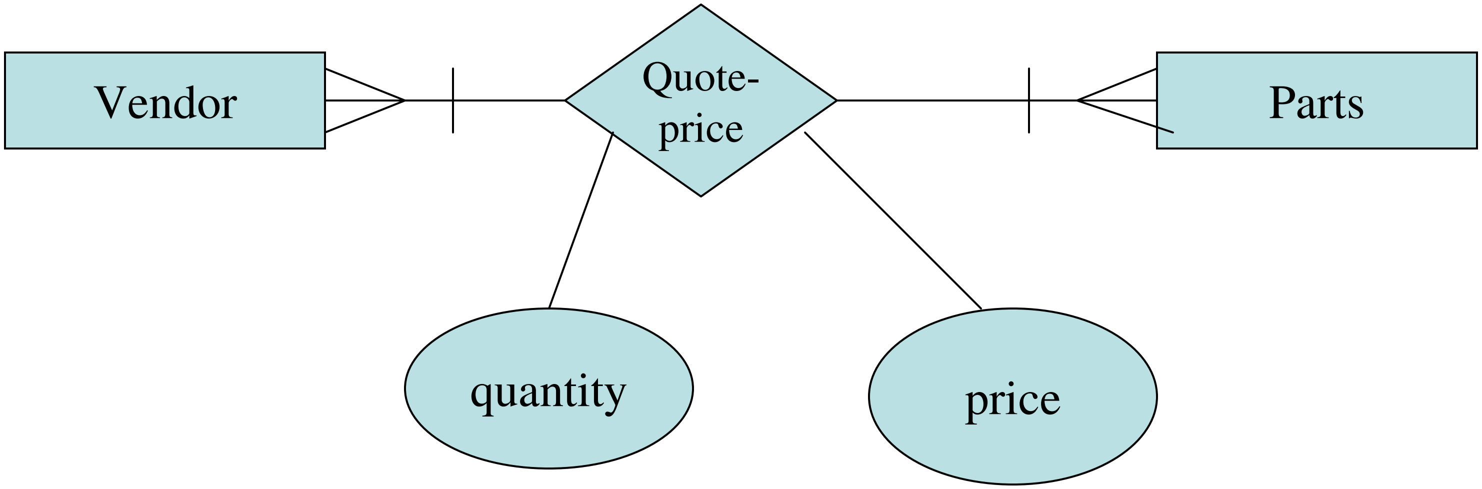
Identifier



Entity-Relationship Diagrams



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



Approaches to problem analysis

1. List all inputs, outputs and functions.
2. 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

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

Requirements Documentation

(ii) Responses to both valid & invalid inputs.

(iii) 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.

Organization of the SRS

IEEE has published guidelines and standards to organize an SRS.
First two sections are same. The specific tailoring occurs in section-3.

1. Introduction

1.1 Purpose
1.2 Scope
1.3 Definition, Acronyms and abbreviations
1.4 References
1.5 Overview

Requirements Documentation

2. The Overall Description

2.1 Product Perspective

2.1.1 System Interfaces

2.1.2 Interfaces

2.1.3 Hardware Interfaces

2.1.4 Software Interfaces

2.1.5 Communication Interfaces

2.1.6 Memory Constraints

2.1.7 Operations

2.1.8 Site Adaptation Requirements

Requirements Documentation

2.2 Product Functions
2.3 User Characteristics
2.4 Constraints
2.5 Assumptions for dependencies
2.6 Apportioning of requirements

3. Specific Requirements

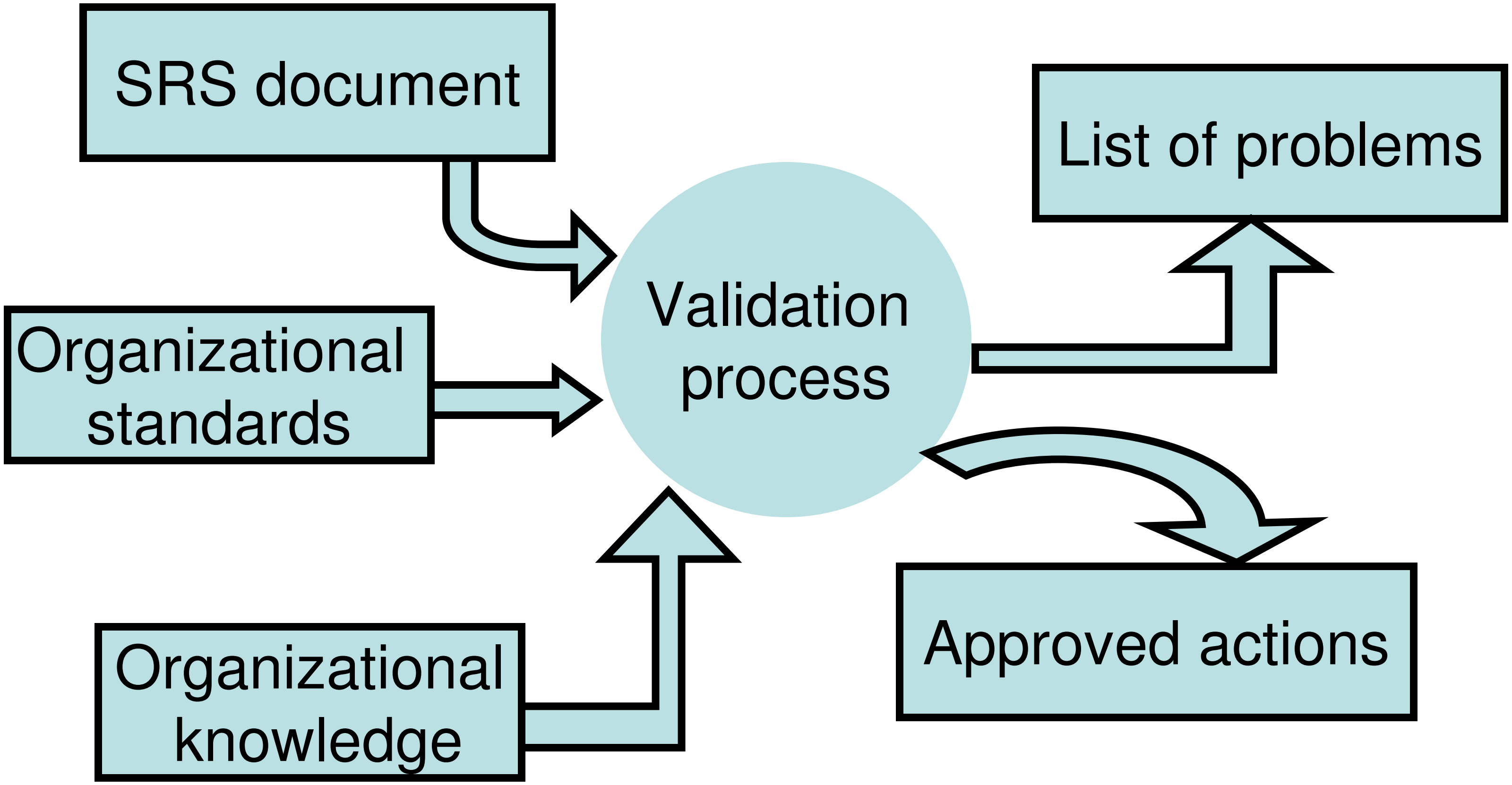
3.1 External Interfaces
3.2 Functions
3.3 Performance requirements
3.4 Logical database requirements
3.5 Design Constraints
3.6 Software System attributes
3.7 Organization of specific requirements
3.8 Additional Comments.

Requirements Validation

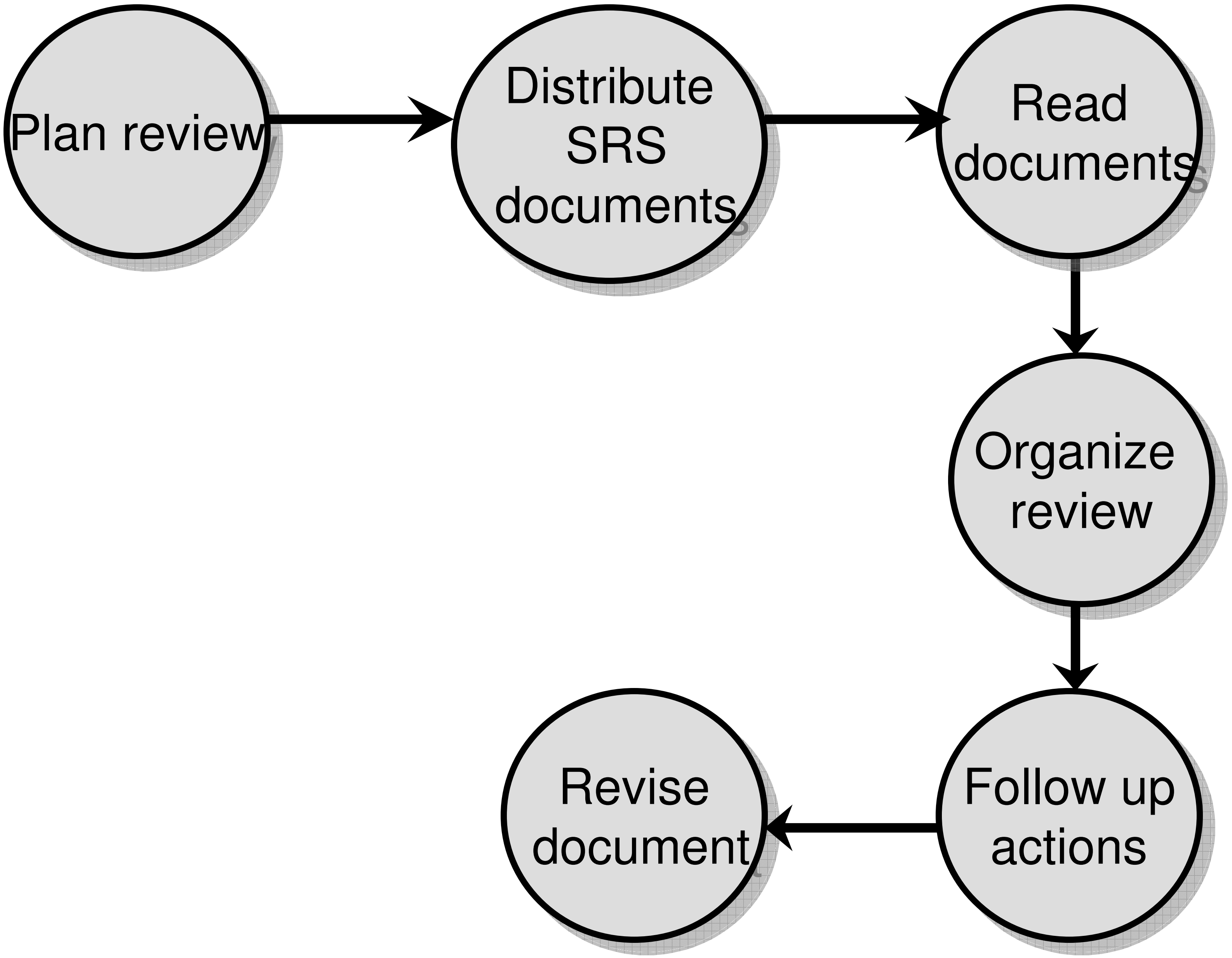
Check the document for:

- ✓ 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

Review Checklists

- ✓ Redundancy
- ✓ Completeness
- ✓ Ambiguity
- ✓ Consistency
- ✓ Organization
- ✓ Conformance
- ✓ Traceability

Prototyping

Validation prototype should be reasonably complete & efficient & should be used as the required system.

Requirements Management

- Process of understanding and controlling changes to system requirements.

ENDURING & VOLATILE REQUIREMENTS

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

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

- o 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

Download from

Q:\IRM\PRIVATE\INITIATIATI\QA\QAPLAN\SRSPLAN.DOC

Multiple Choice Questions

Note. Choose the most appropriate answer of the following questions.

3.1 Which one is not a step of requirement engineering?

(a) Requirements elicitation
(b) Requirements analysis
(c) Requirements design
(d) Requirements documentation

3.2 Requirements elicitation means

(a) Gathering of requirements
(b) Capturing of requirements
(c) Understanding of requirements
(d) All of the above

Multiple Choice Questions

3.3 **SRS stands for**

(a) Software requirements specification
(b) System requirements specification
(c) Systematic requirements specifications
(d) None of the above

3.4 **SRS document is for**

(a) “What” of a system?
(b) How to design the system?
(c) Costing and scheduling of a system
(d) System’s requirement.

Multiple Choice Questions

3.5 Requirements review process is carried out to

(a) Spend time in requirements gathering ✗
(b) Improve the quality of SRS ✗
[✓ **(c) Document the requirements**]
(d) None of the above

3.6 Which one of the statements is not correct during requirements engineering?

(a) Requirements are difficult to uncover
(b) Requirements are subject to change ✓
(c) Requirements should be consistent
(d) Requirements are always precisely known.

Multiple Choice Questions

3.7 Which one is not a type of requirements?

(a) Known requirements
(b) Unknown requirements
(c) Undreamt requirements
[✓ (d) Complex requirements]

A Use-case actor is always a person having a role that different people may play.
a) True
[✓ b) False]

3.8 Which one is not a requirements elicitation technique?

(a) Interviews
(b) The use case approach
(c) FAST
[✓ (d) Data flow diagram.]

Multiple Choice Questions

3.9 FAST stands for

(a) Functional Application Specification Technique
(b) Fast Application Specification Technique
[✓ (c) Facilitated Application Specification Technique]
(d) None of the above

3.10 QFD in requirement engineering stands for

(a) Quality function design
(b) Quality factor design
(c) Quality function development
[✓ (d) Quality function deployment]

Multiple Choice Questions

3.11 Which is not a type of requirements under quality function deployment

(a) Normal requirements
[✓ (b) Abnormal requirements]
(c) Expected requirements
(d) Exciting requirements

3.12 Use case approach was developed by

[✓ (a) I. Jacobson and others]
(b) J.D. Musa and others
(c) B. Littlewood
(d) None of the above

Multiple Choice Questions

3.13 Context diagram explains

[✓ (a) The overview of the system]
(b) The internal view of the system ✗
(c) The entities of the system ✗
(d) None of the above

3.14 DFD stands for

(a) Data Flow design
(b) Descriptive functional design
[✓ (c) Data flow diagram]
(d) None of the above

Multiple Choice Questions

3.15 Level-O DFD is similar to

(a) Use case diagram
[✓ (b) Context diagram]
(c) System diagram
(d) None of the above

3.16 ERD stands for

[✓ (a) Entity relationship diagram]
(b) Exit related diagram
(c) Entity relationship design
(d) Exit related design

Multiple Choice Questions

3.17 Which is not a characteristic of a good SRS?

(a) Correct
(b) Complete
(c) Consistent
(d) [✓ Brief]

3.18 Outcome of requirements specification phase is

(a) [✓ Design Document]
(b) Software requirements specification
(c) Test Document
(d) None of the above

Multiple Choice Questions

3.19 The basic concepts of ER model are:

[✓ **(a) Entity and relationship**]
(b) Relationships and keys
(c) Entity, effects and relationship
(d) Entity, relationship and attribute

3.20 The DFD depicts

[✓ **(a) Flow of data**]
(b) Flow of control
(c) Both (a) and (b)
(d) None of the above

Multiple Choice Questions

3.21 Product features are related to:

[✓ (a) Functional requirements]
(b) Non functional requirements
(c) Interface requirement
(d) None of the above

3.22 Which one is a quality attribute?

(a) Reliability
(b) Availability
(c) Security
[✓ (d) All of the above]

Multiple Choice Questions

3.23 IEEE standard for SRS is:

(a) IEEE Standard 837-1998
(b) IEEE Standard 830-1998
(c) IEEE Standard 832-1998
(d) IEEE Standard 839-1998

3.24 Which one is not a functional requirement?

(a) Efficiency
(b) Reliability
(c) Product features
(d) Stability

Multiple Choice Questions

3.23 APIs stand for:

(a) Application performance interfaces

[✓ (b) Application programming interfaces]

(c) Application programming integration

(d) Application performance integration

Exercises

3.1 Discuss the significance and use of requirement engineering. What are the problems in the formulation of requirements?

3.2 Requirements analysis is unquestionably the most communication intensive step in the software engineering process. Why does the communication path frequently break down ?

3.3 What are crucial process steps of requirement engineering ? Discuss with the help of a diagram.

3.4 Discuss the present state of practices in requirement engineering. Suggest few steps to improve the present state of practice.

3.5 Explain the importance of requirements. How many types of requirements are possible and why ?

3.6 Describe the various steps of requirements engineering. Is it essential to follow these steps ?

3.7 What do you understand with the term "requirements elicitation" ? Discuss any two techniques in detail.

3.8 List out requirements elicitation techniques. Which one is most popular and why ?

Exercises

3.9 Describe facilitated application specification technique (FAST) and compare this with brainstorming sessions.

3.10 Discuss quality function deployment technique of requirements elicitation. Why an importance or value factor is associated with every requirement ?

3.11. Explain the use case approach of requirements elicitation. What are use-case guidelines ?

3.12. What are components of a use case diagram. Explain their usage with the help of an example.

3.13. Consider the problem of library management system and design the following:

(*i*) Problem statement

(*ii*) Use case diagram

(*iii*) Use cases.

Exercises

3.14. Consider the problem of railway reservation system and design the following:

(*i*) Problem statement

(*ii*) Use case diagram

(*iii*) Use cases.

3.15. Explain why a many to many relationship is to be modeled as an associative entity ?

3.16. What are the linkages between data flow and E–R diagrams ?

3.17. What is the degree of a relationship ? Give an example of each of the relationship degree.

3.18. Explain the relationship between minimum cardinality and optional and mandatory participation.

3.19. An airline reservation is an association between a passenger, a flight, and a seat. Select a few

pertinent attributes for each of these entity types and represent a reservation in an E–R diagram.

Exercises

3.20. A department of computer science has usual resources and usual users for these resources. A software is to be developed so that resources are assigned without conflict. Draw a DFD specifying the above system.

3.21. Draw a DFD for result preparation automation system of B. Tech. courses (or MCA program) of any university. Clearly describe the working of the system. Also mention all assumptions made by you.

3.22. Write short notes on

(*i*) Data flow diagram

(*ii*) Data dictionary.

3.23. Draw a DFD for borrowing a book in a library which is explained below: "A borrower can borrow a book if it is available else he/she can reserve for the book if he/she so wishes. He/she can borrow a maximum of three books".

3.24. Draw the E–R diagram for a hotel reception desk management.

Explain why, for large software systems development, is it recommended that prototypes should be "throw-away" prototype ?

Exercises

3.26. Discuss the significance of using prototyping for reusable components and explain the problems,which may arise in this situation.

3.27. Suppose a user is satisfied with the performance of a prototype. If he/she is interested to buy this

for actual work, what should be the response of a developer ?

3.28. Comment on the statement: “The term throw-away prototype is inappropriate in that these prototypes expand and enhance the knowledge base that is retained and incorporated in the final prototype; therefore they are not disposed of or thrown away at all.”

3.29. Which of the following statements are ambiguous ? Explain why.

(*a*) The system shall exhibit good response time. –

(*b*) The system shall be menu driven. PARTIALY

(*c*) There shall exist twenty-five buttons on the control panel, numbered PF1 to PF25. ×

(*d*) The software size shall not exceed 128K of RAM. ×

Exercises

3.30. Are there other characteristics of an SRS (besides listed in section 3.4.2) that are desirable ? List a few and describe why ?

3.31. What is software requirements specification (SRS) ? List out the advantages of SRS standards.

Why is SRS known as the black box specification of a system ?

3.32. State the model of a data dictionary and its contents. What are its advantages ?

3.33. List five desirable characteristics of a good SRS document. Discuss the relative advantages of formal requirement specifications. List the important issues, which an SRS must address.

3.34. Construct an example of an inconsistent (incomplete) SRS.

3.35. Discuss the organization of a SRS. List out some important issues of this organization.

Exercises

3.36. Discuss the difference between the following:

(a) Functional & nonfunctional requirements

(b) User & system requirements

Software Project Planning