



Module 1 Introduction to Software Engineering Basics – Part 1

Mr. Swapnil S Sontakke (Asst. Prof.)
Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli

Content

- Software Crisis
- Need of Software Engineering Approach





Software Crisis

Software Crisis



- What is Software crisis?
- What caused the Software crisis?
- The Problem Domain
- The Software Engineering Challenges
- The Solution

What is Software crisis?

- Software Engineering (SE) is defined as

“the systematic approach to the development, operation, maintenance and retirement of software.”

- IEEE, Software Engineering Standards, Technical Report



What is Software crisis?



- Coined by NATO Software Engineering Conference, 1968 at Garmisch, Germany
- The term is used in Computing Science for the *difficulty in writing useful and efficient computer programs in the required time.*
- Reason - rapid increases in computer power and the complexity of the problems that could now be tackled

What is Software crisis?



"as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem"

- Edsger Dijkstra, The Humble Programmer (EWD340),
Communications of the ACM

What caused the Software crisis?



- Linked to overall complexity of hardware and the software development process. It was evident in several ways:
 - Projects running over-budget
 - Projects running over-time
 - Software was very inefficient
 - Software was of low quality
 - Software often did not meet requirements
 - Projects were unmanageable and code difficult to maintain
 - Software was never delivered

The Solution

- There is no single solution to the crisis.
- One possible solution of software crisis is *Software Engineering* because software engineering is a systematic, disciplined and quantifiable approach.



The Solution

- For preventing software crisis, there are some guidelines:
 - Reduction in software over-budget
 - Reduction in software over-time
 - Software must be optimized to make it efficient
 - The quality of software must be high
 - Experienced working team member on software project
 - Using standard programming style rules
 - Software must be delivered



The Problem Domain

- Industrial Strength Software
- Software is Expensive
- Late and Unreliable
- Maintenance and Rework



The Problem Domain



- Industrial Strength Software
 - Consider the following example of Building a software
 - Q: How much time will a student require to develop a 10,000 lines of code (LOC) program/software in C/C++ at college?
 - A: 2 months approx.

$$\begin{aligned}\text{Productivity} &= \text{output/input} \\ &= 10,000 / 2\end{aligned}$$

Productivity = 5,000 LOC per person-month

The Problem Domain



- Industrial Strength Software
 - For commercial software company,
Productivity = 100 - 1000 LOC per person-month
 - Why is this difference?
 - What makes these novice students more productive than expert employees in the company?

The Problem Domain



- Industrial Strength Software
 - The Problem Domain is the software that solves some problem of some users where larger systems or businesses may depend on the software, and where problems in the software can lead to significant direct or indirect loss.
 - This software is called as *industrial strength software* that is different from the one built by students which is called as *student software*.

The Problem Domain



Sr. No.	Student Software	Industrial Strength Software
1	Primarily meant for demonstration purposes; generally not used for solving any real problem of any organization.	Primarily meant for real usage; built to solve some problem of an organization
2	Developer is the client/user	Other organization is the user
3	Presence of bugs or faults in the software are tolerable	Presence of bugs or faults may lead to financial or business loss, etc. and hence are not tolerable

The Problem Domain



Sr. No.	Student Software	Industrial Strength Software
4	In general, there is no documentation	Documentation is necessary for users, organization and developing industry
5	No or very less focus on User Interface	User Interface is very important
6	No or very less amount of money is invested	High amount of money is invested; may cost about 10 times the student software

The Problem Domain



Sr. No.	Student Software	Industrial Strength Software
7	Focus on testing is less, even 5% efforts on testing may be too high	High focus on testing, about 30-50% efforts may be spent on testing
8	Standards and development processes/phases are not followed	Standards and development processes/phases are strictly followed to make it high quality
9	No requirement of backup, recovery, fault tolerance, portability, etc.	High requirement of backup, recovery, fault tolerance, portability, etc.

The Problem Domain



Sr. No.	Student Software	Industrial Strength Software
10	Productivity is higher as compared to industrial strength software	Productivity is lower as compared to student software
11	E.g. Mini Projects, Final Year Projects	E.g. Fusion 360, Solid Works, AutoCAD, MATLAB, HEC-HMS, Simulink, Microsoft Visual Studio, etc.

The Problem Domain



- Software is Expensive
 - Industrial strength software is expensive as it is labour-intensive
 - Let us have a look at the cost involved
 - Measure of software = lines of code (**LOC**) or thousands of lines of code (**KLOC**)
 - Software development cost measure = **person-month** (i.e. per person per month)
 - Productivity measure = **LOC per person-month** or **KLOC per person-month**

The Problem Domain



- Software is Expensive
 - In general, the productivity = ~300 to 1,000 LOC per person-month
 - Software companies charge the client = ~\$100,000 per person-year or \$8,000 per person-month
 - Cost per LOC = ~\$8 to \$25
 - Any small project has at least = ~50,000 LOC
 - Cost of the software = ~ \$0.5 million to \$1.25 million

The Problem Domain



- Software is Expensive
 - In general, the workstation or server that runs the software costs at most tens of thousands of dollars.
 - Earlier in 1980s the cost of hardware were high, but today the situation is reversed.
 - The cost of software is high as compared to that of hardware.

The Problem Domain



- Late and unreliable
 - Late software
 - Survey says, more than 35% software development projects are *runaways*.
 - Runaway Project is a project where budget and schedule are out of control.
 - Hence, there was an emergence of consultancy companies.

The Problem Domain



- Late and Unreliable
 - Unreliable Software
 - Meaning the software does not do what it is supposed to do or does something it is not supposed to do
 - A survey says, more than 70% equipment failures were due to the software.
 - E.g. software with electrical, hydraulic and mechanical systems, Apollo flight failure, India's test firing of missile, banking, etc.

The Problem Domain



- Late and Unreliable
 - Unreliable Software
 - Software failures are different from mechanical or electrical system failures
 - In mechanical or electrical, failure is mainly due to the change in physical or electrical properties due to aging.
 - In software, failure is due to the bugs or errors introduced during the design and development process i.e. at the start only.

The Problem Domain



- Maintenance and Rework
 - Maintenance
 - Once the software is delivered and deployed, it enters the maintenance phase.
 - Why software needs maintenance, if it does not age?
 - The answer is residual errors remained in the system.
 - It is believed that all the software has residual errors or bugs in it.

The Problem Domain



- Maintenance and Rework
 - Maintenance
 - They can be discovered only after the software has been in operation, sometimes for a long time.
 - This leads to the change in software, and is called as *corrective maintenance*.

The Problem Domain



- Maintenance and Rework
 - Maintenance
 - Software also undergoes changes (upgrades or updates) to add more features and services.
 - Change in software needs to change the environment.
 - This phenomenon is known as *law of software evolution* and the maintenance due to this phenomenon is called as *adaptive maintenance*.

The Problem Domain



- Maintenance and Rework
 - Rework
 - For large and complex software systems, all the requirement are not understood before the commencement of software development.
 - As time goes and client's understanding of software improves, they specify the additional requirement.
 - This leads to the change in the requirement, design and code to accommodate the change. This leads to the *rework*.

The Problem Domain



- Maintenance and Rework
 - Rework
 - As times change, needs also change.
 - This also leads to the rework.
 - It is estimated that the rework costs are 30-40% of the development cost.

The Software Engineering Challenges

- Basic Problem
- Primary challenges for Software Engineering
 - Scale
 - Quality and Productivity
 - Consistency and Repeatability
 - Change



The Software Engineering Challenges



- Basic Problem
- Systematic Approach: Methodologies used for development of software are repeatable.

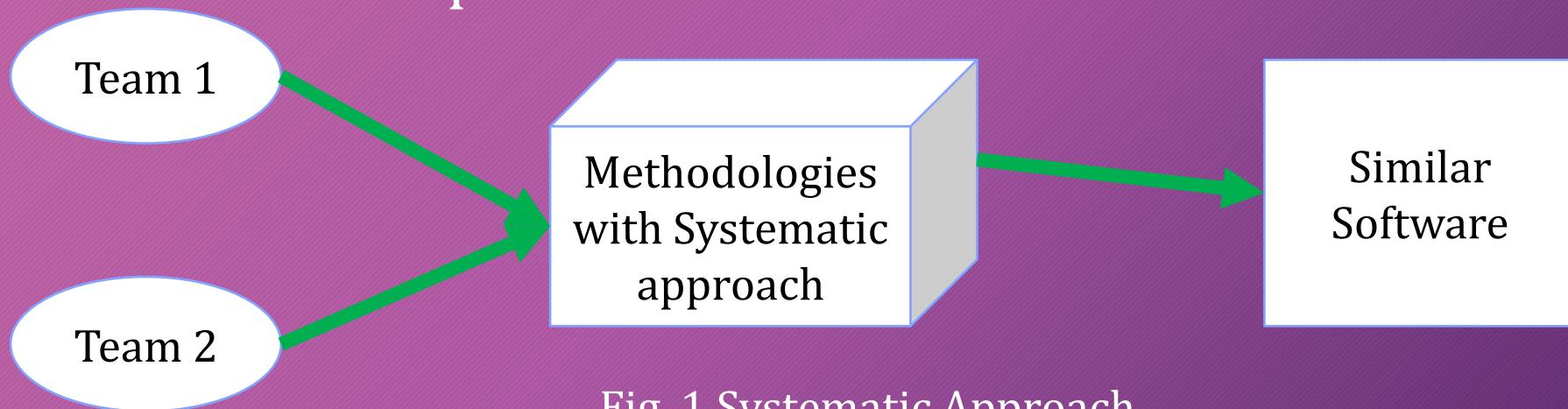


Fig. 1 Systematic Approach

The Software Engineering Challenges



- Basic Problem
- Goal of Software Engineering:
 - Take Software development closer to science and engineering where outcomes are predictable as compared to the traditional ad-hoc approaches which are being used since many years.
- The Problem:
 - To systematically develop software to satisfy the needs of some users or clients

The Software Engineering Challenges



- Basic Problem

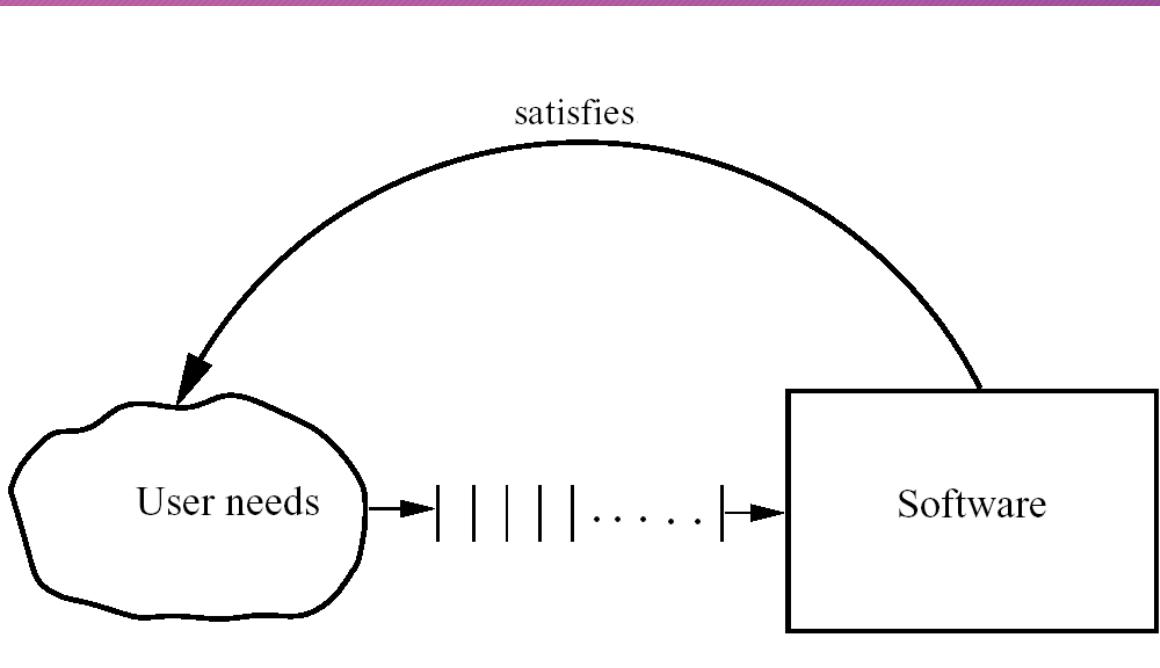


Fig. 2 Basic Problem

The Software Engineering Challenges



- Basic Problem
- There are some factors that affect the approaches selected to solve the problem.
- These factors are the primary forces that drive the *progress and development* in software engineering.
- These factors include scale, quality, productivity, consistency, repeatability and change.

The Software Engineering Challenges



- Scale
- It is measure, dimension or proportion of something.
- Methods for solving small problems do not scale up for large problems
- E.g. counting people in room vs taking census
- Methods required to develop large software must be different than that of small software

The Software Engineering Challenges



- Scale
- Two dimensions in this
 - Engineering methods – methods, procedures and tools
 - Project Management – managing the project by leading the work of a team to achieve the goals and meet success criteria at a specified time.
- Both must be more formal for developing large software.

The Software Engineering Challenges



- Scale

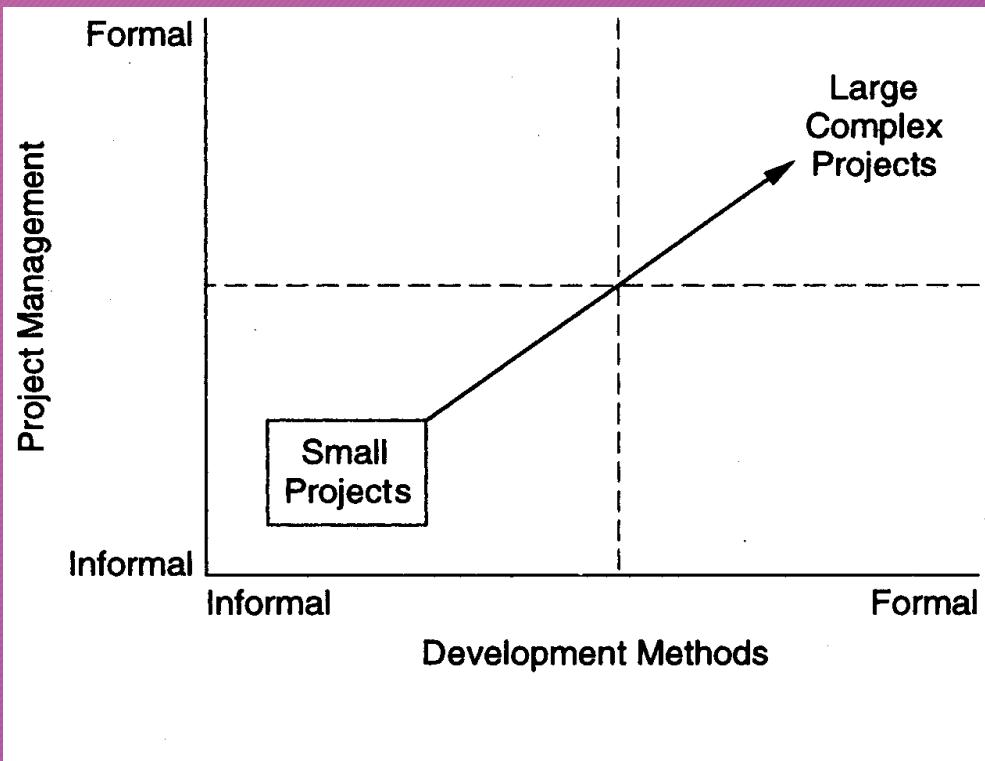


Fig. 2 The Problem of Scale

The Software Engineering Challenges



- Scale
- No universally accepted project scale
- But, informally,
 - Small Project - <10 KLOC
 - Medium Project - >10 KLOC and <100 KLOC
 - Large Project - >100 KLOC and < 1 million LOC
 - Very Large Project - >1 million LOC

The Software Engineering Challenges



- Productivity (P)
 - Software Engineering is driven by cost, schedule and quality.
 - Development cost is dominated by manpower cost.
 - Cost measure – person-month
 - Schedule – Software must be delivered within a time
 - i.e. Cycle time from concept to delivery – as small as possible
 - Productivity measure – LOC (KLOC) person-month
 - It captures cost as well as schedule

The Software Engineering Challenges



- Productivity (P)
 - If P is higher then cost will be lower
 - If P is higher then time will be less
 - This means a development team with higher productivity will complete the software in lesser time than a same-size team with lower productivity.

The Software Engineering Challenges



- Quality (Q)
 - One of the major factor driving any production.
 - Quality is fundamental goal of software engineering.
 - Six attributes or characteristics of quality.

The Software Engineering Challenges



- Quality (Q)

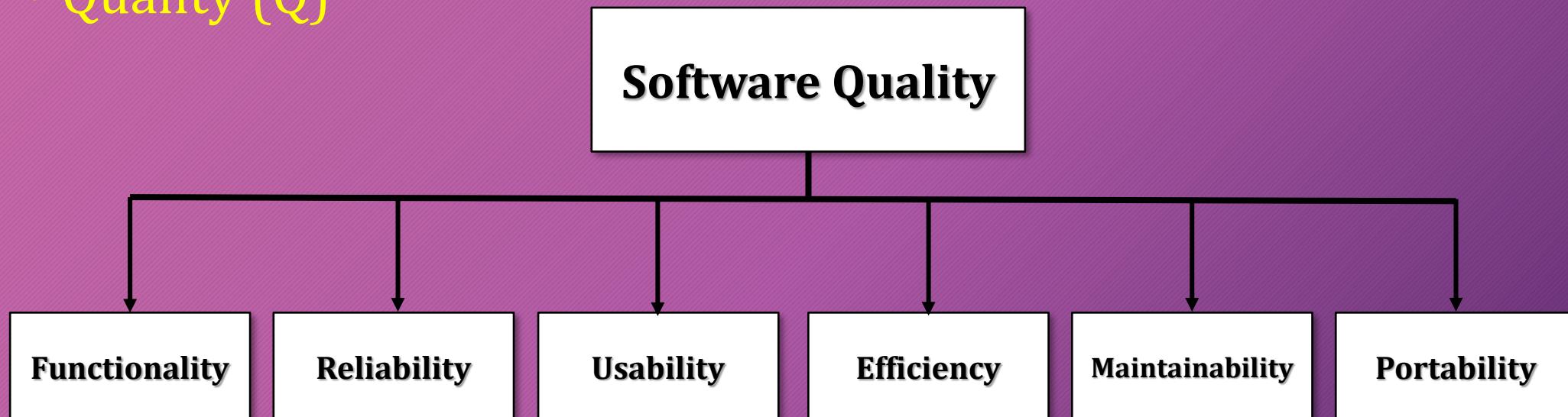


Fig. 4 Software quality attributes

The Software Engineering Challenges



- Quality (Q)
 - Two important consequences of multiple dimensions to quality
 - First - Software quality cannot be reduced to single parameter
 - Second – Concept of quality is project specific
 - In general, reliability is considered as main quality factor.

The Software Engineering Challenges



- Quality (Q)
 - Unreliable Software = Presence of defects
 - Quality measure = number of defects in software per unit size
 - Best practice is 1 defect per 1 KLOC
 - Quality objective is to reduce the number of defects per KLOC
 - Defect definition is project specific
 - What is defect?

The Software Engineering Challenges



- Consistency and Repeatability
 - A company may deliver high quality and high productivity software once.
 - Key challenge - repeating success for all projects and consistency in quality and productivity
 - Goal of Software Engineering - system after system can be produced with high quality and high productivity
 - This means - methods that are being used should be repeatable across projects leading to consistency in quality of software

The Software Engineering Challenges



- **Consistency and Repeatability**
 - Consistency allows organization to predict the output of project with accuracy.
 - Consistency can be achieved by using methodologies in consistent manner.
 - Frameworks like ISO9001 and Capability Maturity Model (CMM) encourage organizations to standardize methodologies, use them consistently and improve them based on experience.

The Software Engineering Challenges



- Change
 - Business changes very rapidly.
 - Business changed – Supporting software has to be changed
 - Software is easy to change as it lacks physical properties.
 - Software engineering challenge is to accommodate and embrace the change.



Need of Software Engineering Approach

Need of Software Engineering Approach

- Phased Development Process
- Managing the Process



Need of Software Engineering Approach



- We now understand the problem domain and the factors that drive the software engineering.
- Objective – To develop a software with high quality and productivity consistently for large scale problems under dynamics of changes.
- Q&P is governed by three main forces: People, Processes and Technology
- Called as the '*Iron Triangle*'

Need of Software Engineering Approach

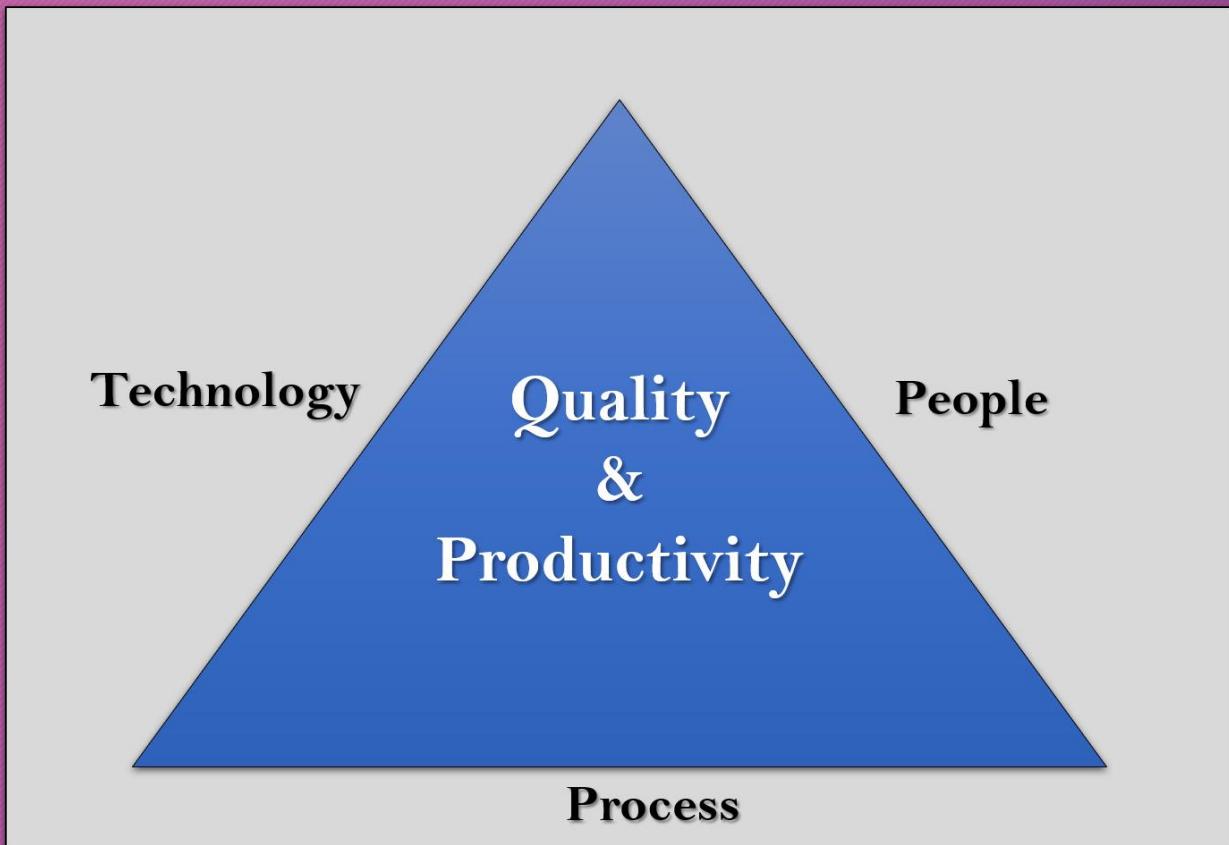


Fig. 5 The Iron Triangle

Need of Software Engineering Approach



- Technology has to be good (state-of-the-art)
- People doing the job have to be properly trained
- Good processes or methods have to be used
- Main focus: **Processes** i.e. systematic approach

Need of Software Engineering Approach



- Technology has to be good (state-of-the-art)
- People doing the job have to be properly trained
- Good processes or methods have to be used
- Main focus: **Processes** i.e. systematic approach
- Approach: To separate the process for developing software from the developed product

Need of Software Engineering Approach



- Premise: The software process determines the quality of the product and productivity.
- So, to tackle the problem domain and software engineering challenges, focus must be on software process.
- Other disciplines focus on the product while SE focuses on process for developing the product.
- Two aspects: Phased development process and Managing the process

Need of Software Engineering Approach



- Phased Development Process
 - Consists of various phases
 - Each phase ends with a defined output
 - Phase order is followed as specified by the *process model*
 - Why Phased Process?
 - It breaks the problem into phases each having its own concern.
 - Cost required will be lower as compared to that of whole project

Need of Software Engineering Approach



- Phased Development Process
 - Why Phased Process?
 - It allows proper checking for quality and progress at some defined time intervals (at the end of each phase).
 - It is central to the software engineering approach for solving the software crisis.
 - Basic software development phases: Requirement analysis, Software design, Coding and Testing

Need of Software Engineering Approach



- Phased Development Process
 - Requirement Analysis
 - Done to understand the problem the software system is to solve
 - It focuses on identifying what is needed from the system
 - Two major activities: problem understanding or analysis and requirement specification
 - Output: Software Requirement Specification

Need of Software Engineering Approach



- Phased Development Process
 - Software Design
 - Purpose is to plan a solution of the problem specified by the requirements document.
 - Problem domain -> Solution domain
 - Output: architecture design, high level design and detailed design

Need of Software Engineering Approach



- Phased Development Process
 - Coding
 - The goal is to translate the design of the system into code in a given programming language
 - It affects testing and maintenance
 - Output: Program or set of programs

Need of Software Engineering Approach



- Phased Development Process
 - Testing
 - Quality control measure
 - Main function is to detect the defects in the system/software.
 - The goal is to uncover requirement, design and coding errors in the programs.
 - Types: unit testing, integration testing, system testing and acceptance

Need of Software Engineering Approach



- Managing the Process
 - Project management handles the issues relating to managing the development process of a project.
 - It revolves around a project *plan*.
 - Plan forms a baseline that is used for monitoring and control the development process
 - Includes how to allocate resources, scheduling different activities, how to divide work within a phase, etc.

Need of Software Engineering Approach



- Managing the Process
 - *Product metrics* and *process metrics* are used for the software development
 - They quantify the characteristics of product and process resp.
 - These are important for managing a software development

References

- https://en.wikipedia.org/wiki/Software_crisis
- <https://www.ukessays.com/essays/information-technology/symptoms-and-primary-causes-of-software-crisis-information-technology-essay.php>
- <https://www.geeksforgeeks.org/software-engineering-software-crisis/>
- https://en.wikipedia.org/wiki/Spaghetti_code





Module 1 Introduction to Software Engineering Basics – Part 2

Mr. Swapnil S Sontakke (Asst. Prof.)
Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli

Content

- Software Process
- Desired Characteristics of Software Process
- Software Development Process Models
- Other Software Processes





Software Process

Software Process



- Software process
 - What is software process?
 - Process and Process Model
 - Component Software Process
 - ETVX Approach for Process Specification

Software Process



- What is software process?
 - **Process:** A particular method of doing something, generally involving a number of steps or operations
 - **Software Process:** A set of activities, together with ordering constraints, such that if the activities are performed properly and in accordance with the ordering constraints, the desired result is produced
 - It deals with the technical and management issues of software development.

Software Process



- What is software process?
 - It includes software development processes as well as non software engineering processes such as business processes, social processes and training processes.
 - Different activities are performed by different people
 - Each process has different activities and objectives
 - So, better to view software process as consisting of many component processes

Software Process



- Processes and Process Model
 - Project Success -> satisfies cost, schedule and quality expectations
 - Process Specification defines the tasks the project should perform and the order in which they should be done
 - Actual Process is distinct from process specification
 - In general, the term process refer to both

Software Process



- Processes and Process Model
 - Process model specifies the general process.
 - It includes:
 - A set of stages in which project should be divided
 - The order of execution of each stage
 - Conditions on the execution of stages
 - Any other constraints
 - Premise: Use of process model as project's process will lead to low cost, high quality or reduced cycle time

Software Process



- Processes and Process Model
 - Process is a means to reach the goals of high quality, low cost and low cycle time
 - Process model specifies the generic guidelines for developing a suitable process for a project.

Software Process



- Component Software Processes
 - Two major components in software process: a development process and a project management process
 - Development process: specifies the development and quality assurance activities that need to be performed
 - Project management process: specifies how to plan and control development process activities so that cost, schedule, quality and other objectives are met

Software Process



- Component Software Processes
 - Different software processes are executed by different people
 - Engineering Process – Developers
 - Management Process – Project Manager
 - Software Configuration Control Process – Configuration controller
 - Process Management Process – Software Engineering Process Group (SEPG)

Software Process



- ETVX approach for Process Specification
 - Each process has a set of phases (or steps).
 - Each phase has well-defined task.
 - Each task contributes in achieving the final goal (or product).
 - To reduce the cost and maintain quality, each phase needs to be verified by some means.
 - So, for each phase there should be some clearly defined output.

Software Process



- ETVX approach for Process Specification
 - Such intermediate outputs at each phase which are not a final output is called as 'work products'.
 - It includes requirements document, design document, code, prototype, etc.
 - Methodologies -> How to perform an activity at particular phase

Software Process



- ETVX approach for Process Specification
 - Que: When to start and terminate a phase?
 - Entry Criteria: specifies the conditions that input to the phase should satisfy to initiate the activities of that phase
 - Task (Activities): What is to be done in the phase
 - Verification: To find out errors and bugs at each phase
 - Exit Criteria: specifies the conditions that the work product of this phase should satisfy to terminate the phase

Software Process

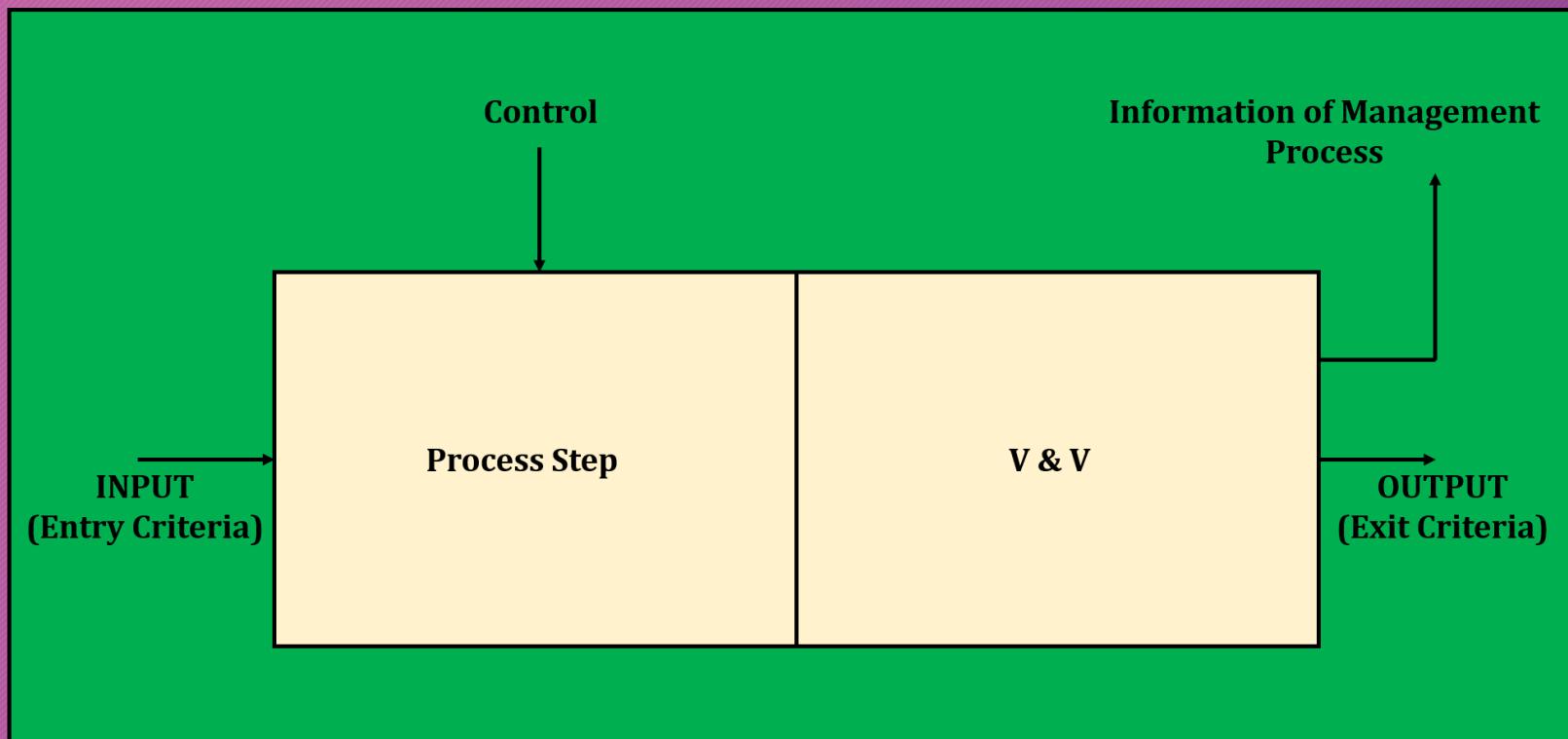


Fig. 1 A step in a development process



Desired Characteristics of Software Process

Desired Characteristics of Software Process



- Desired Characteristics of Software Process
 - Predictability
 - Support Testability and Maintainability
 - Support Change
 - Early Defect Removal
 - Process Improvement and Feedback

Desired Characteristics of Software Process



- Each software process should possess desirable characteristics apart from satisfying the project goals.
- Some of them are discussed here.

Desired Characteristics of Software Process



- Predictability
 - Fundamental property of any process
 - Predicts the outcome of selected process in a project before its completion
 - E.g. Two projects A and B following same process, cost, schedule and results may be predicted in advance if the process is predictable

Desired Characteristics of Software Process



- Predictability
 - A predictable process is also said to be *under statistical control*
 - This means the same process produces similar results.
 - Variation may be there, it is mostly due to random causes

Desired Characteristics of Software Process

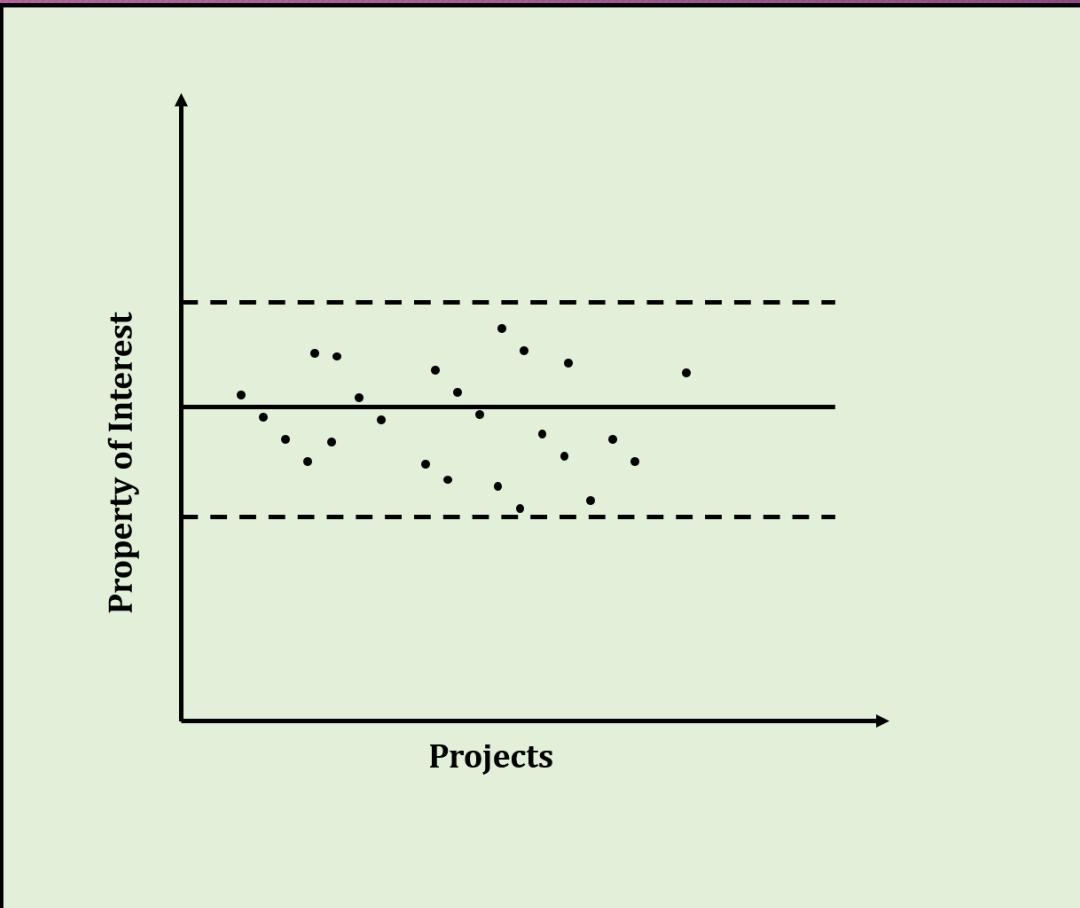


Fig. 2
Process
under
statistical
control

Desired Characteristics of Software Process



- Support Testability and Maintainability
 - In general, maintenance cost > development cost
 - Also, testing takes 50% of total cost
 - It is also found that programmers spend only 13% of time in writing programs.
 - Objective: To produce software that is easy to maintain and reduce the cost of testing and maintenance.

Desired Characteristics of Software Process



- Support Change
 - Software changes for a variety of reasons
 - Main focus: Changes due to requirements changes
 - Other reasons:
 - Business change, so software needs to be changed
 - Needs of customer may change
 - Alternatives and other possibilities may come to customer's mind

Desired Characteristics of Software Process



- Early Defect Removal
 - Errors can occur at any stage during the development
 - For example,
 - requirements 20%
 - Design 30%
 - Coding 50%
 - Cost of error depends on when the error is detected and corrected.

Desired Characteristics of Software Process

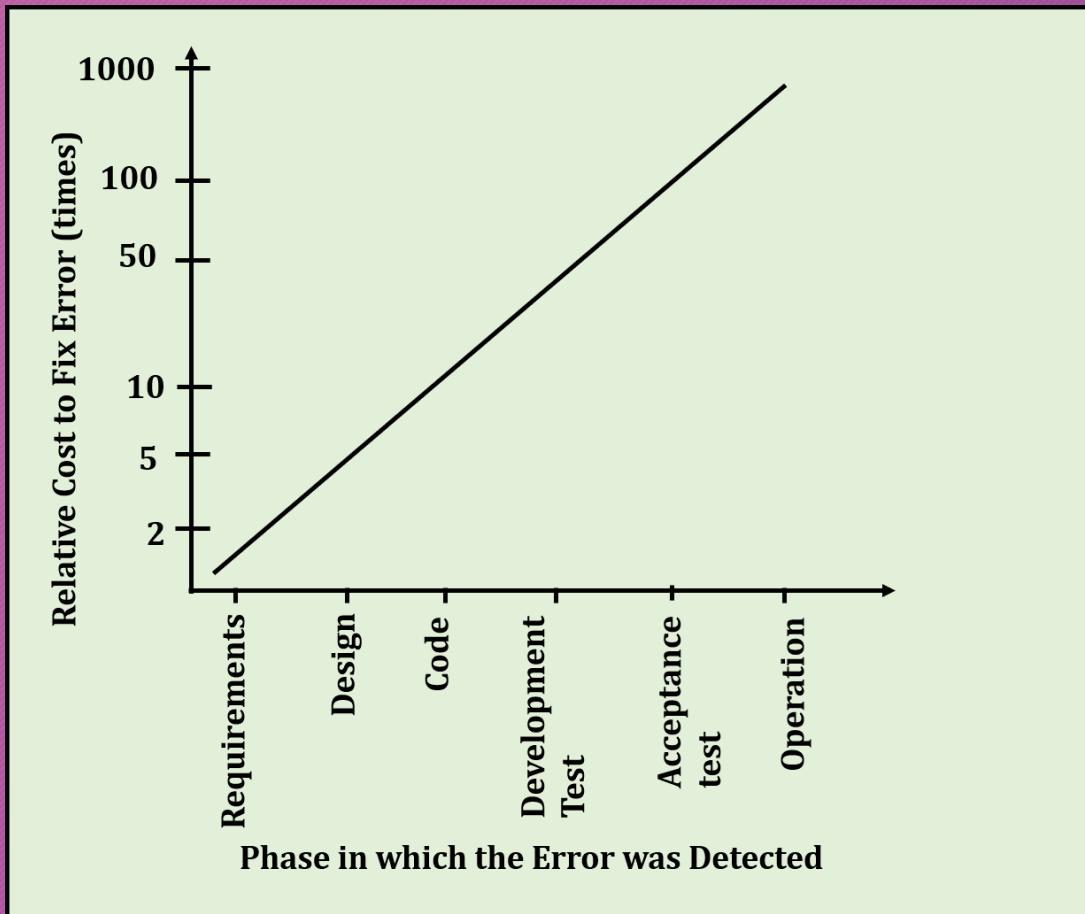


Fig. 3 Cost of Correcting Error

Desired Characteristics of Software Process



- Early Defect Removal
 - Correcting an error in acceptance testing can cost 100 times more than that of during the requirements phase.
 - Therefore, there is V in the ETVX process

Desired Characteristics of Software Process



- **Process Improvement and Feedback**
 - Quality and Productivity is largely determined by the process
 - So, to improve the quality and reduce the cost, process must be improved.
 - For this, process must be closed-loop.
 - i.e. improvement must be based on the past experiences.
 - This requires the feedback from previous project as well as from the early parts of the project.



Software Development Process Models

Software Development Process Models



- Software Development Process Models
 - Waterfall Model
 - Prototyping
 - Iterative Development
 - Timeboxing Model
 - Comparison of Models

Software Development Process Models



- Software Development Process Model
 - We focus majorly on the activities directly related to development of software, e.g. design, coding and testing.
 - Management process is based on the development process.
 - So, development process is important.
 - So, development process model have been proposed to ensure the success in the software development.

Software Development Process Models



- The Waterfall Model
 - Simplest process model
 - Phases are organized in linear manner
 - Proposed by Royce

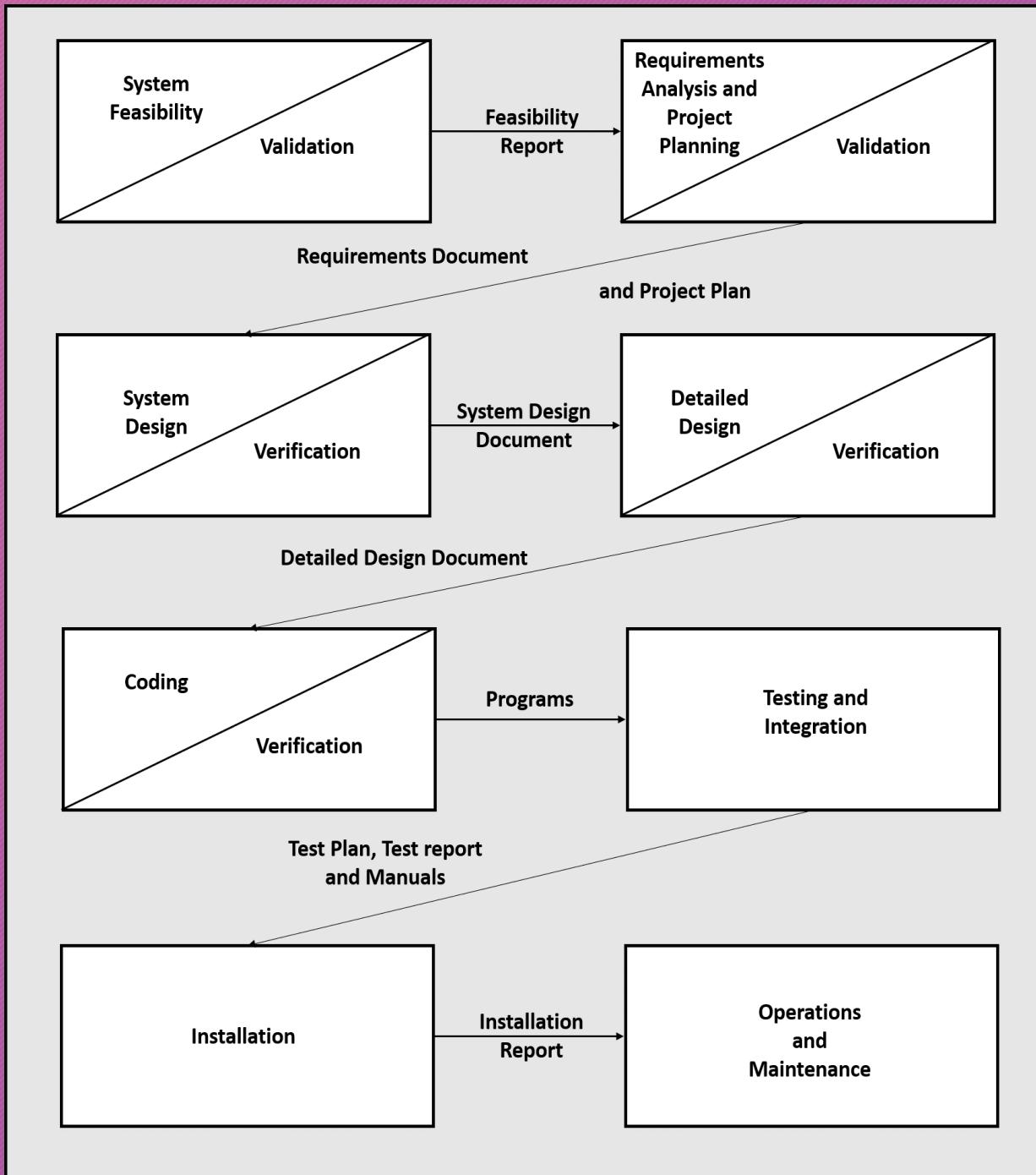


Fig. 4 The Waterfall Model

Software Development Process Models



- The Waterfall Model
 - Verification and validation is applied at the end of each phase
 - It ensures each phase has some defined output.
 - These intermediate outputs are “work products”.
 - Common documents produced (work products) for each of the projects include:
 - requirements document
 - Project Plan

Software Development Process Models



- The Waterfall Model
 - Common documents produced (work products) for each of the projects include:
 - Design Documents (architecture, system, detailed)
 - Test Plan and Test Report
 - Final Code
 - Software Manuals (user, installation, etc.)

Software Development Process Models



- The Waterfall Model - Advantages
 - Simple – divides large task into cleanly divided phases and each phase has separate logical concern
 - Easy to administer in a contractual setup

Software Development Process Models



- The Waterfall Model - Disadvantages
 - Assumes requirements can be frozen before the design begins
 - Requires hardware fixing at early stage. For large and longer projects, hardware may become obsolete.
 - Follows “Big Bang” approach. Software will be delivered fully at the end. Money problem may lead no software production. i.e. all or nothing
 - Document-driven process – requires formal documents at the end of each phase

Software Development Process Models



- The Waterfall Model - Usage
 - Widely used
 - Used where requirements are well understood and fixed
 - Used where organization is familiar with problem domain

Software Development Process Models



- **Prototyping**
 - Goal: overcome the limitations of Waterfall model
 - Instead of freezing the requirements, a prototype is built to understand the requirements.
 - Prototype is built based on the currently known requirements
 - Client interacts with the prototype which helps the client to better understand the requirements of the desired system.

Software Development Process Models



- Prototyping
 - Development of prototype starts with initial requirements from SRS
 - Only reasonable understanding of the system and its needs are known at this stage
 - Needs may change
 - Prototype is then given to clients/end users for use
 - Based on the experience, they provide feedback to developers

Software Development Process Models



- Prototyping
 - Feedback includes what is correct, what is missing, what is not needed, what needs to be modified, etc.
 - Based on the feedback, prototype is modified to incorporate suggested changes that can be done easily.
 - Modified prototype is again given to clients/end users.
 - This cycle is repeated until benefits of change and feedback outweighed by cost and time of making changes and taking feedback

Software Development Process Models



- Prototyping
 - To make requirements analysis feasible, its cost must be kept low.
 - So, only important features that give valuable return from the users are added in the prototype.
 - Features like exception handling, recovery, conformance to standards and formats are omitted.

Software Development Process Models



- **Prototyping**
 - Prototype is to be discarded after its use.
 - “Quick and dirty” development approach is used i.e. focus on quick development and not on the quality.
 - requirements that are not understood are only implemented.
 - Minimal testing is done.
 - Only minimal documentation is produced.
 - All of this minimizes the cost of prototype development.

Software Development Process Models

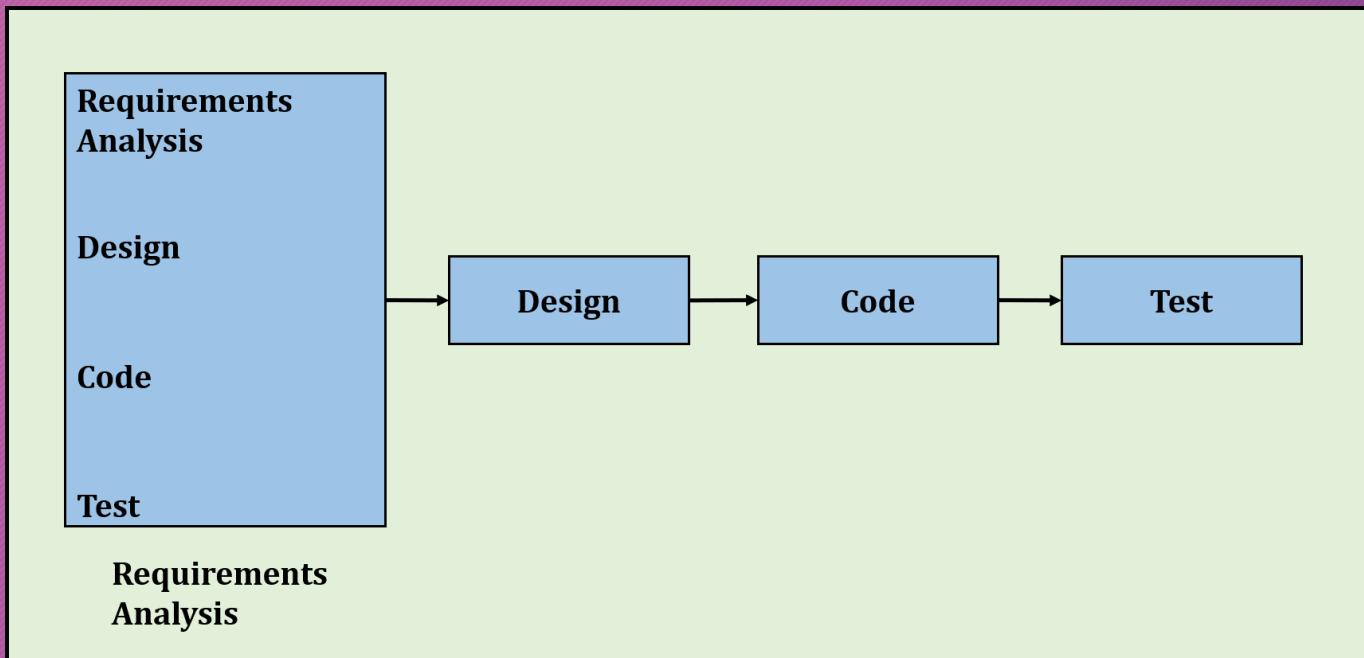


Fig. 5 The Prototyping Model

Software Development Process Models



- Prototyping - Advantages
 - requirements will be more stable
 - Prototyping experience helps developers in actual development process.
 - Hardware changes can be accommodated.

Software Development Process Models



- Prototyping - Disadvantages
 - Potential hit on cost and schedule

Software Development Process Models



- Prototyping - Usage
 - Useful for complicated and large systems for which there is no manual process or existing system to determine the requirements.
 - Useful where requirements are hard to determine, confidence in stated requirements is low and requirements are changing.
 - Used where cost of a project without prototyping may be more than with the prototyping.

Software Development Process Models



- Iterative Development
 - This model solves the “big bang” approach limitation of Waterfall model.
 - Tries to combine the benefits of Waterfall and Prototyping model

Software Development Process Models



- Iterative Development
 - Software is developed and delivered in increments.
 - Each increment adds some functionality to the system.
 - This way full system is implemented.
 - At each increment, extensions and design modifications can be made.

Software Development Process Models



- Iterative Development – Iterative Enhancement Model
 - Can be viewed as a sequence of waterfall model
 - In first step, simple implementation is done on the key aspects of the problem.
 - *Project Control List* is created.
 - It contains ordered list of all the tasks that needs to be performed to get the final implementation.

Software Development Process Models



- Iterative Development – Iterative Enhancement Model
 - At each increment next task is removed, list is updated, design, coding, testing and analysis is done.
 - The process is repeated until the list gets empty and final implementation becomes available.

Software Development Process Models

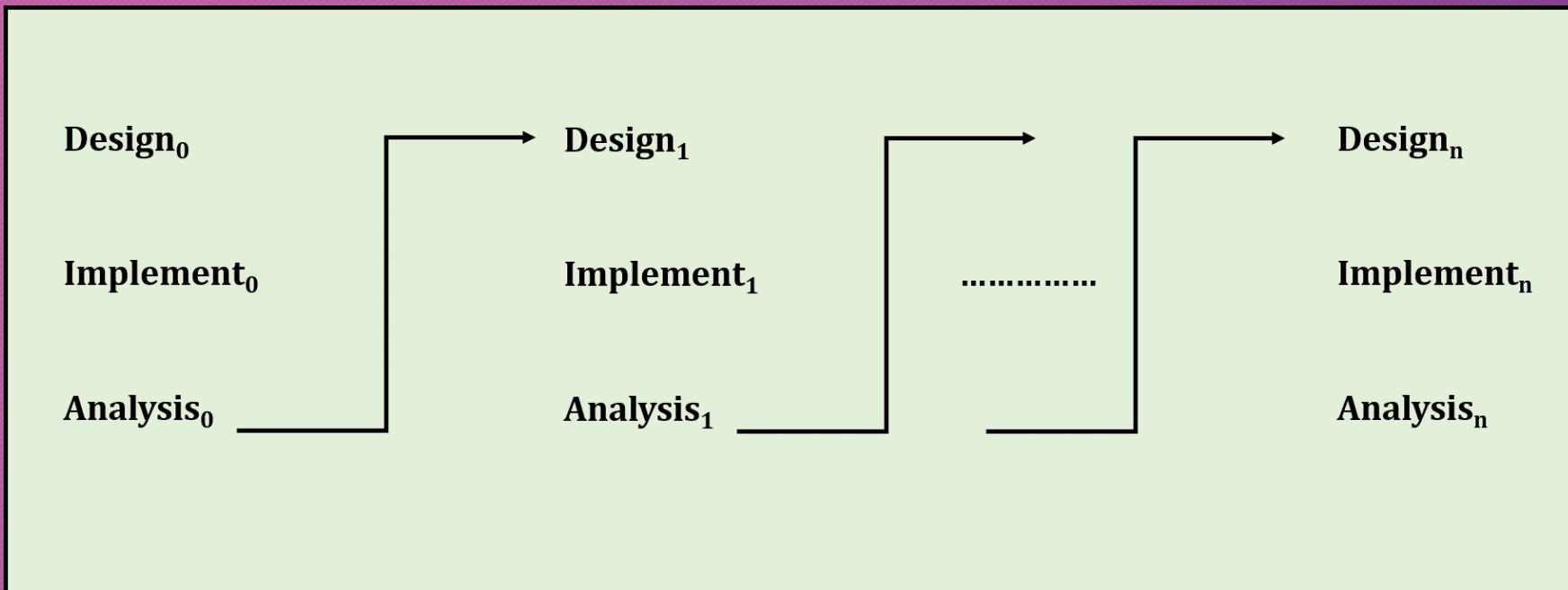


Fig. 5 The Iterative Enhancement Model

Software Development Process Models



- Iterative Development - Advantages
 - Software is delivered in increments.
 - Reduces risk of “all or nothing”
 - Better testing can be done
 - Feedback from client on each increment is useful for determining final requirements of the system.
 - Less chance of error and redesign work due to the Project Control List

Software Development Process Models



- Iterative Development - Disadvantages
 - Sometimes rework may be more
 - Design may not be optimal
 - Total cost may be more

Software Development Process Models



- Iterative Development - Usage
 - Useful in product development, in which developers themselves provide the specifications
 - Useful in customized software development, where client has to provide and approve the specifications
 - Useful where businesses are changing rapidly and requirements are changing rapidly
 - Useful where new features are constantly being added

Software Development Process Models



- Timeboxing Model
 - To speed up development, parallelism between different iterations can be employed.
 - New iteration can be started before completion of previous iteration.
 - This needs that each iteration has to be structured properly and teams have to be organized suitably.
 - Timeboxing model follows this approach.

Software Development Process Models



- Timeboxing Model
 - It differs from iterative approach.
 - In iterative approach, functionality is selected and accordingly time to deliver is determined.
 - Basic development unit in timeboxing model – **time box of fixed duration** and not the functionality
 - In timeboxing, duration is fixed and accordingly requirements or features that can be fitted in the time box.

Software Development Process Models



- Timeboxing Model
 - Each timebox is divided into a sequence of stages, like in the Waterfall model.
 - Each stage performs a clearly defined task for the iteration and produces a clearly defined output.
 - Duration for each stage is generally same.
 - Dedicated team for each stage is required.
 - It can be seen following a hardware pipelining approach.

Software Development Process Models



- Timeboxing Model
 - Consider a time box TB with duration T and consisting of n stages S₁, S₂, ..., S_n.
 - Time to finish each stage = T/n
 - When a team of a stage i completes the task of that stage for a time box k , it passes the output to the team of stage $i+1$ and starts its execution for the next time box $k+1$
 - When first time box is in completion other $n-1$ time boxes are in different stages of execution.

Software Development Process Models



- Timeboxing Model
 - Consider a time box with 3 stages: requirements, build and deployment.
 - In the timebox-1, in requirements stage, analysts prepare a list of requirements and high level design and hand over it to build team
 - Build team then implements the requirements, performs testing and hands over the tested code to deployment team.

Software Development Process Models



- Timeboxing Model
 - Deployment team then performs pre-deployment tests and installs the system for production use.
 - After giving requirements to build team in timebox-1, the requirements team starts preparing requirements for timebox-2.
 - When build team hands over the tested code to deployment team for timebox-1, it moves to building code for requirements of timebox-2 provided by requirements team.

Software Development Process Models

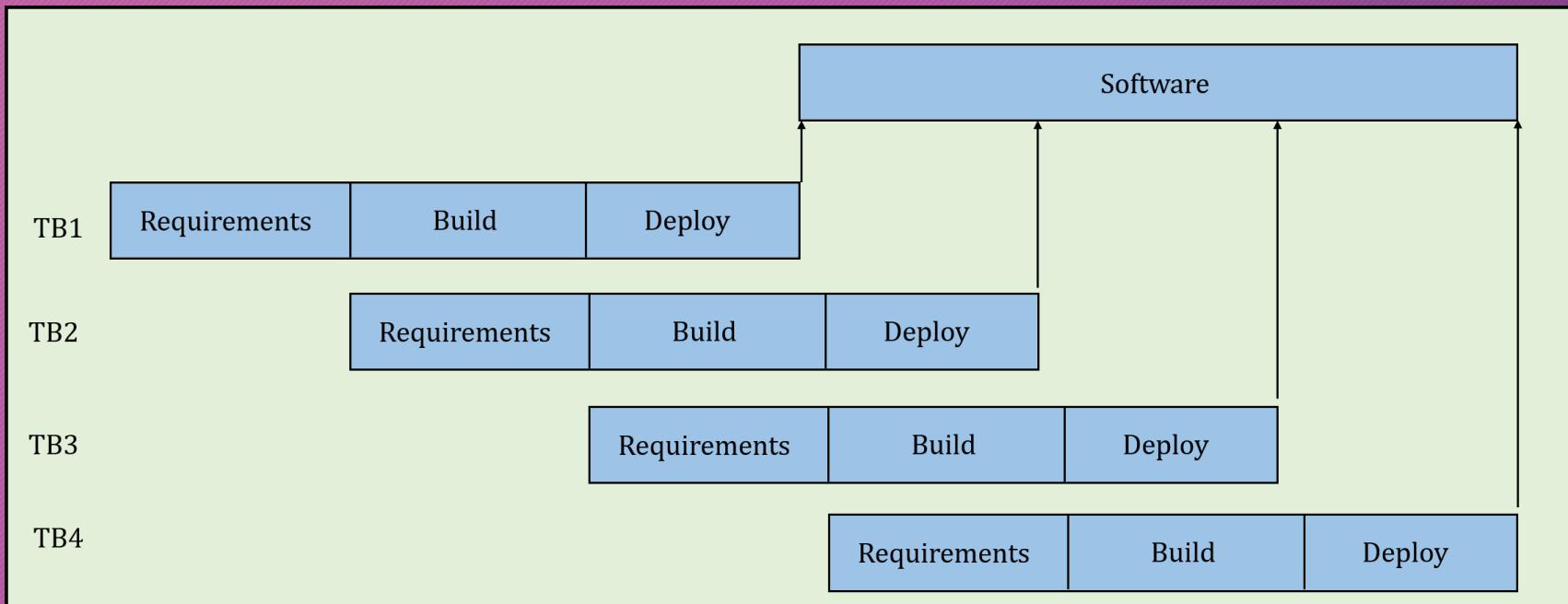


Fig. 5 The Timeboxing Model

Software Development Process Models



- Timeboxing Model – Reduced time
 - Duration of each iteration is same
 - Total work done in time box is also same
 - Total effort spent in time box is also same
 - How cost of reducing time is achieved?
 - Answer: Separate teams for each stage

Software Development Process Models



- Timeboxing Model – Reduced time
 - E.g. In the Waterfall model, if requirements phase requires 2 persons for 2 weeks, building phase 4 persons for 4 weeks and 3 persons for 2 weeks for test and deploy,
 - Team size = 4
 - In case of timeboxing model, it will be $(2+4+3) = 9$ persons
 - This increases throughput and average delivery time.

Software Development Process Models



- Timeboxing Model - Advantages
 - It has all the features of iterative model
 - Parallel execution with additional manpower reduces the overall delivery time
 - Structured way

Software Development Process Models



- Timeboxing Model - Disadvantages
 - It requires larger team size
 - Project management is complex
 - Possibly increase in cost
 - Not suitable where equal duration stages is not possible
 - Not suitable for projects where integrating the stages is not flexible

Software Development Process Models



- Timeboxing Model - Usage
 - Suitable for projects where large number of features need to be developed within short period of time
 - Suitable for projects which have flexibility in grouping the features

Software Development Process Models



- Comparison of Models
 - Based on the Strengths, Weaknesses and Applications
 - Every model has its own application domain
 - Based on the project needs, the appropriate model can be used

References

- Pankaj Jalote, "An Integrated Approach to Software Engineering", Narosa Publishing House, 2nd Edition.





Module 1 Introduction to Software Engineering Basics – Part 3

Mr. Swapnil S Sontakke (Asst. Prof.)
Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli

Content

- Other Software Processes
 - Project Management Process
 - Configuration Management Process
 - Process Management Process





Other Software Processes

Other Software Process



- Other Processes
 - Development process is the central process in software processes.
 - But, other processes are needed to properly execute development process and achieve desired characteristics of software processes.
 - These processes are for each of the activities in the development process, e.g. requirements, design, testing, etc.
 - We discuss here some of the important processes

Other Software Process



- Project Management Process
 - To meet cost, quality and schedule objectives of software development:
 - Resources have to be properly allocated to each activity
 - Progress of different activities has to be monitored
 - Corrective measures, if needed, need to be taken
 - Scheduling of each activity has to be done

Other Software Process



- Project Management Process
 - To meet cost, quality and schedule objectives of software development:
 - Project risk plan has to be created
 - Change requests have to be managed
 - These and many other activities that ensure cost, quality and schedule objectives are part of *Project Management Process*.

Other Software Process



- Project Management Process
 - Basic Task: To plan the detailed implementation of the development process chosen for a project and then ensure the plan is followed.
 - The focus is on issues like
 - Planning a project
 - Estimating resource and schedule
 - Monitoring and controlling the project

Other Software Process



- Project Management Process
 - Project Management process activities are broadly grouped in three phases:
 - Planning
 - Monitoring and Control
 - Termination Analysis

Other Software Process



- Project Management Process
 - Planning
 - First phase of project management process
 - Important phase and forms basis for *monitoring and control*
 - Goal: To develop a *plan* for software development to meet project objectives successfully and efficiently
 - Software plan is produced before the development begins
 - It is updated as development proceeds and project progress data becomes available.

Other Software Process



- Project Management Process
 - Planning
 - Major activities include
 - Cost estimation
 - Schedule and milestone determination
 - Project staffing
 - Quality control plans
 - Controlling and monitoring plans

Other Software Process



- Project Management Process
 - Monitoring and Control
 - Longest phase in terms of duration
 - Encompasses most of development process
 - It includes all activities that project management has to perform while development is going on
 - It also ensure that project objectives are met and development proceeds according to plan

Other Software Process



- Project Management Process
 - Monitoring and Control
 - Monitoring potential risk is important activity which focuses on potential risks, if any during the development of project.
 - Monitoring a development process requires proper information about the project.

Other Software Process



- Project Management Process
 - Monitoring and Control
 - Major activities include
 - Measuring planned performance vs actual performance.
 - Ongoing assessment of the project's performance to identify any preventive or corrective actions needed.

Other Software Process



- Project Management Process
 - Monitoring and Control
 - Major activities include
 - Keeping accurate, timely information based on the project's output and associated documentation.
 - Monitoring the implementation of any approved changes or schedule amendments.

Other Software Process



- Project Management Process
 - Termination Analysis
 - Last phase of management process
 - Performed when the development is over
 - Goal: To provide the information about the development process and learn from the project in order to improve the process.
 - Also called as *postmortem analysis*.

Other Software Process

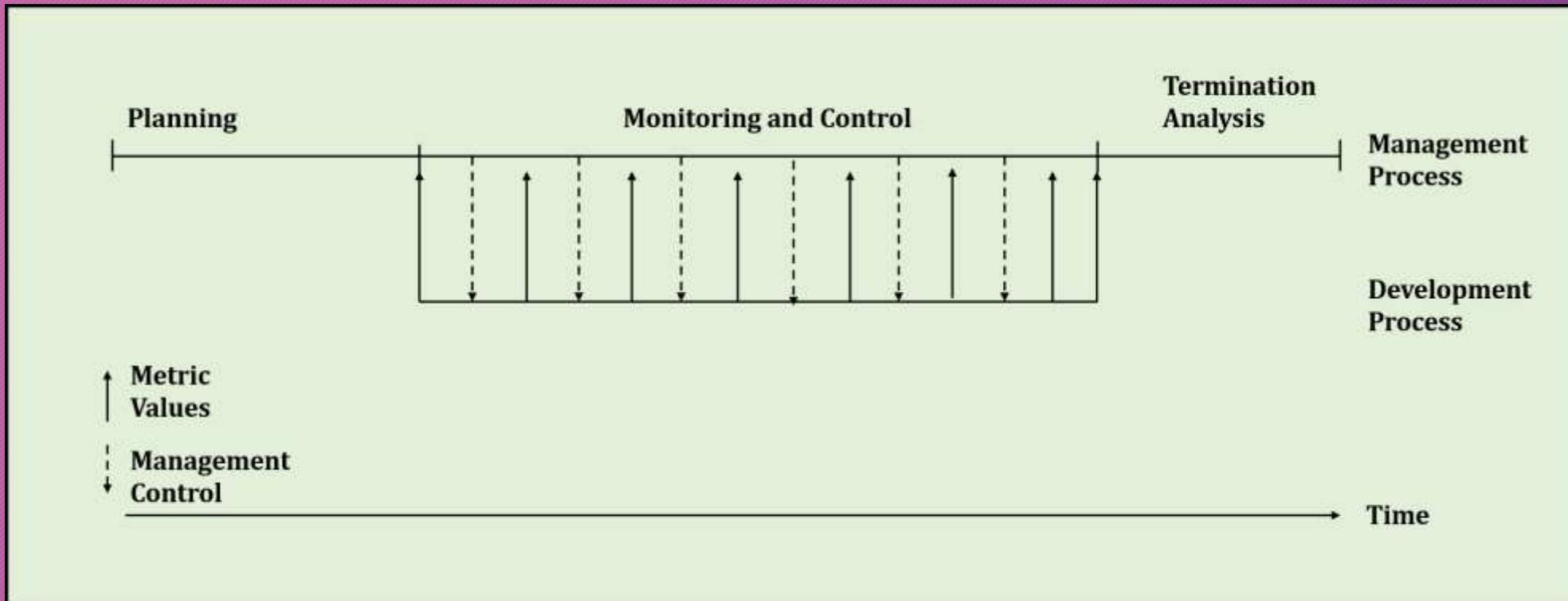


Fig. 1 Temporal Relationship Between Development and Management Process

Other Software Process



- Software Configuration Management Process
 - Changes continuously take place in a software development project
 - Changes due to evolution of work products
 - Changes due to bugs
 - Changes due to requirements changes
 - These changes are reflected as changes in the files containing source, data, documentation, manuals, other work products.

Other Software Process



- Software Configuration Management Process
 - To systematically control these changes *Configuration Management (CM)* or *Software Configuration Management (SCM)* is required.
 - IEEE defines SCM as

"the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items."

Other Software Process



- Software Configuration Management Process
 - It is independent of development process.
 - Development models look at the macro picture and not on changes to individual files
 - Development process is brought under SCM, so changes are allowed in controlled manner.
 - SCM directly controls the products of a process
 - It indirectly influences the activities producing the products

Other Software Process

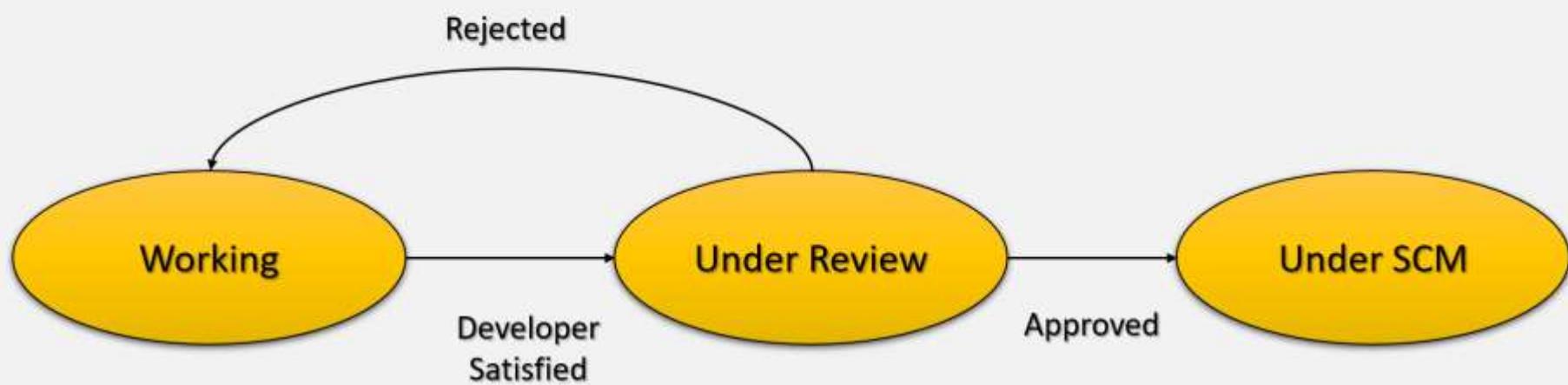


Fig. 2 Configuration Management and development process

Other Software Process



- Software Configuration Management Process Functionalities
 - Give latest version of a program
 - Changes should be done in the latest version of a file.
 - CM process will ensure that the latest version of a file can be obtained easily
 - Undo a change or revert back to a specified version
 - Change made in a program or many programs to implement some change request may sometimes need to be undone
 - The CM process must allow this to happen smoothly

Other Software Process



- Software Configuration Management Process Functionalities
 - Prevent unauthorised changes or deletions
 - A change in program may be done that has some adverse side effects
 - The CM process must ensure that the unapproved changes are not permitted

Other Software Process



- Software Configuration Management Process Functionalities
 - Gather all sources, documents and other information for the current system
 - All sources and related files are needed for releasing the product.
 - These files may be needed for reinstallation or recovery.
 - The CM process must provide this functionality.

Other Software Process



- Software Configuration Management Process
 - CM is essential to satisfy the basic objective of a project:
Delivery of high quality software product to the client

Other Software Process



- Requirements Change Management Process
 - Project requirements can change any time during the life of a project.
 - The farther down in the life cycle the requirements change, the more severe the impact on the project.
 - Requirements changes can take as much as 40% of the total cost of the project.
 - Therefore project manager must be prepared to handle any change requests as and when they come.

Other Software Process



- Requirements Change Management Process
 - The change management process defines the set of activities that are performed when there are some new requirements or changes to existing requirements
 - This process also focuses on the major changes like design changes and bug fixes.

Other Software Process



- Requirements Change Management Process
 - This process has following steps:
 - Log the changes
 - Perform impact analysis on the work products
 - Estimate the impact on the effort and schedule
 - Review impact with concerned stakeholder
 - Rework work products

Other Software Process



- Requirements Change Management Process
 - A change is initiated by a *change request*.
 - A *change request log* is maintained to keep track of the change requests.
 - Each entry in the log contains
 - A change request number
 - A brief description of the change
 - The effect of the change
 - The Status of the change requests and key dates

Other Software Process



- Requirements Change Management Process
 - Then effect of a change request is assessed by performing *impact analysis*.
 - It involves
 - Identifying work products and configuration items that need to be changed
 - Evaluating quantum of change to each work product and items
 - Reassessing the project risks by revisiting the risk management plan

Other Software Process



- Requirements Change Management Process
 - It involves
 - Evaluating the overall implications of the changes for the effort and *schedule estimates*.
 - Once the change is reviewed and approved, then it is implemented.
 - i.e. changes to all items are made
 - The actual tracking of implementation may handled by SCM process.

Other Software Process



- Requirements Change Management Process
 - One danger: Even if some changes are not large, over the life of the project cumulative impact of changes are large.
 - So, cumulative impact of changes must also be monitored.
 - For this, the change log is used frequently as a spreadsheet.

Other Software Process



- **Process Management Process**
 - A software process is not a static entity.
 - To meet cost, quality and schedule objectives software process must continuously be improved.
 - Improving quality and productivity of the process is main objective of process management process.
 - Project management and process management process are different.

Other Software Process



- **Process Management Process**
 - Process management focuses on *improving the process which in turn improves the general quality and productivity for the products produced using the process.*
 - Project management focuses on *executing the current project and ensuring that the objectives of the project are met.*
 - Time duration for project management process is typically the duration of the project while process management has larger duration as each project provides a data point for the process.

Other Software Process



- Process Management Process – Capability Maturity Model
 - To improve the process
 - First need to understand current status of a process
 - Then, build a plan to improve the process
 - Changes to a process are best introduced in the small increments
 - Revolutionizing the whole process takes time as new methods require time to internalize and truly follow.

Other Software Process



- **Process Management Process – Capability Maturity Model**
 - The next question is what order should be followed or what small change should be introduced first?
 - This depends on the current status of a process.
 - This concept of introducing changes in small increments based on the current state of a process has been captured in *Capability Maturity Model (CMM) framework*.

Other Software Process



- Process Management Process – Capability Maturity Model
 - *Software process capability* describes the range of expected results that can be achieved by following the process.
 - It determines what can be expected from the organization in terms of quality and productivity.
 - The goal of process improvement is to improve the process capability.

Other Software Process



- Process Management Process – Capability Maturity Model
 - A *maturity level* is a well-defined evolutionary plateau towards achieving a mature software process.
 - CMM suggests that there are five levels of well-defined maturity levels for a software process.
 - These are initial (level 1), repeatable, defined, managed, and optimizing (level 5).
 - When a process improves it moves from one level to next until it reaches level 5.

Other Software Process



- Process Management Process – Capability Maturity Model
 - In each level it specifies the areas in which improved can be absorbed and will bring the maximum benefits.

Other Software Process

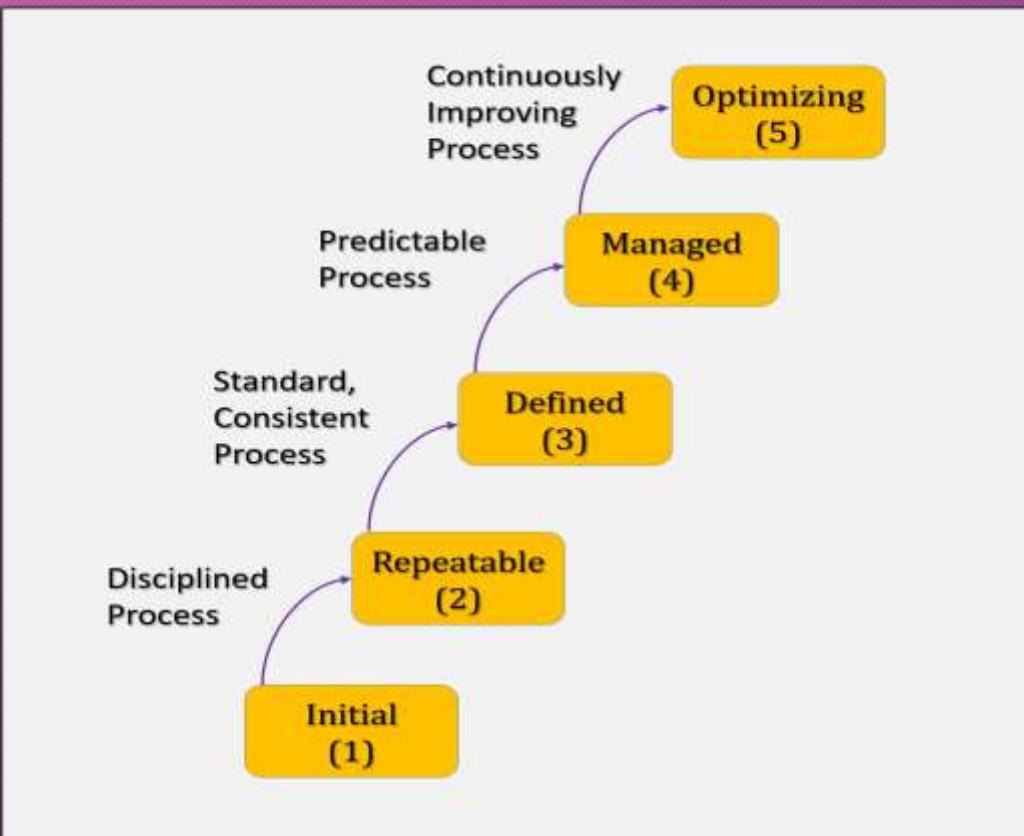


Fig. 3 Capability Maturity Model

Other Software Process



- Process Management Process – Capability Maturity Model
 - The initial process (level 1)
 - Ad hoc process
 - Has no formalized method for any activity
 - Basic project controls for ensuring the activity are done properly but there is no project plan
 - Unstable development environment
 - No basis for predicting the product quality, cost and schedule activity

Other Software Process



- Process Management Process – Capability Maturity Model
 - The initial process (level 1)
 - Success depends on the quality and capability of individuals
 - Need to improve project management, quality assurance and change control

Other Software Process



- Process Management Process – Capability Maturity Model
 - The repeatable process (level 2)
 - Project management policies for a software project are well defined.
 - Key characteristics are
 - Project commitments are realistic and based on the past experience with similar projects
 - Cost and schedule are tracked and problems are resolved when they arise

Other Software Process



- Process Management Process – Capability Maturity Model
 - The repeatable process (level 2)
 - Key characteristics are
 - Formal configuration control mechanisms are in place
 - Software project standards are defined and followed
 - The process is repeatable i.e. results obtained by this process can be repeated as project planning and tracking is formal.

Other Software Process



- Process Management Process – Capability Maturity Model
 - The defined level (level 3)
 - Organization has standardized a software process and is well documented.
 - Organization has software process group that owns and manages the process
 - Key characteristics are
 - Each step/stage is carefully defined with verifiable entry and exit criteria, methodologies for performing the step,

Other Software Process



- Process Management Process – Capability Maturity Model
 - The defined level (level 3)
 - and verification mechanism for output
 - Both development and management processes are formal

Other Software Process



- Process Management Process – Capability Maturity Model
 - The managed level (level 4)
 - In this level quantitative goals exists for organization for process and products
 - Key characteristics are
 - Data is collected from software processes which is used to build models to characterize the process.
 - Hence organization has a good insights of the process capability and its deficiencies.

Other Software Process



- Process Management Process – Capability Maturity Model
 - The managed level (level 4)
 - Key characteristics are
 - The results can be predicted in quantitative terms.

Other Software Process



- Process Management Process – Capability Maturity Model
 - The optimizing level (level 5)
 - Highest level of process maturity in CMM
 - Focus is on continuous process improvement
 - Key characteristics are
 - Data is collected and routinely analysed to identify the areas that can be strengthened to improve the quality or productivity

Other Software Process



- Process Management Process – Capability Maturity Model
 - The optimizing level (level 5)
 - Key characteristics are
 - New technologies and tools are introduced and their effect is measured to improve the performance of the process
 - Best software engineering and management practices are used throughout the organization

References

- Pankaj Jalote, "An Integrated Approach to Software Engineering", Narosa Publishing House, 2nd Edition
- <https://blog.proofhub.com/what-are-key-project-activities-in-project-management-bf5c939e8c67>
- <https://www.projectmanagementqualification.com/blog/2019/10/21/project-monitoring-control/>
- <https://www.javatpoint.com/software-engineering-tutorial>
- <https://www.geeksforgeeks.org/software-engineering-capability-maturity-model-cmm/>





Module 2 Software Quality and Project Planning – Part 1

Mr. Swapnil S Sontakke (Asst. Prof.)
Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli

Content

- What is quality?
- Software Quality Factors/Attributes
- Cost of Quality
- Quality Standards



What is quality?



- The International Organization for Standardization (ISO) defines quality as:

“The totality of features and characteristics of a product or service that bears on its ability to satisfy stated or implied needs.”

- ISO 8402 (1986)

What is quality?



- A decade ago the emphasis was on the products and quality was considered as the ability of product to conform to specifications i.e. client needs.
- Quality now includes measures of work life, workplace diversity, environmental conditions and competitiveness.
- It encompasses the features or characteristics related to organization's many stakeholders such as community, suppliers, shareholders, employees and management.

Software Quality (Q) Factors/Attributes



- Quality is one of the major factor that drives any production.
- Quality is fundamental goal of software engineering.
- Six factors or attributes or characteristics of quality.

Software Quality (Q) Factors/Attributes

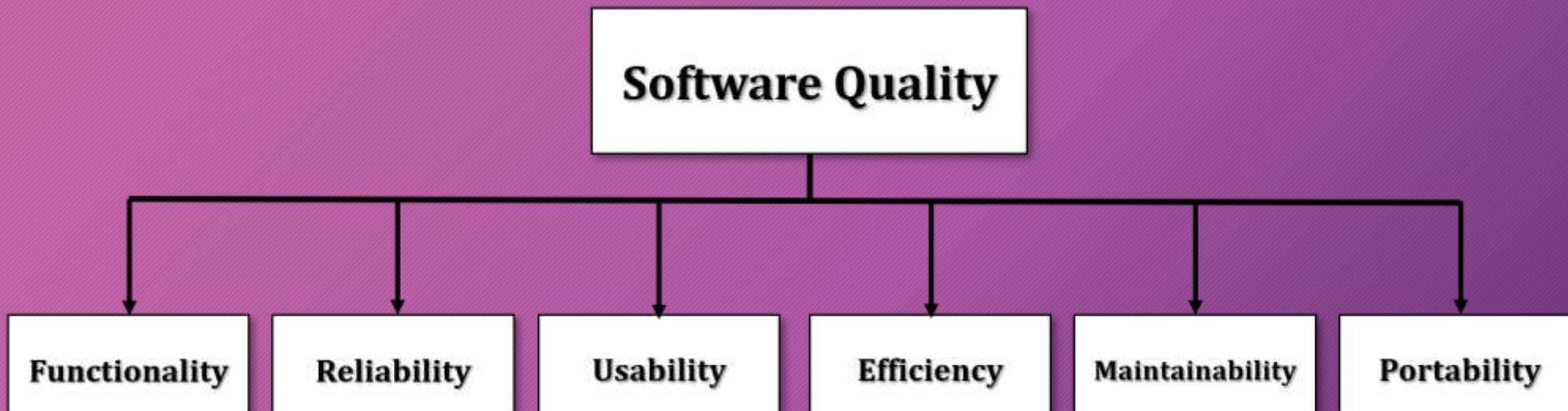


Fig. 1 Software Quality Factors/Attributes

Software Quality (Q) Factors/Attributes



- **Functionality:**
 - *It is the capability to provide functions which meet stated and implied needs when the software is used.*
 - It further includes
 - Suitability (whether appropriate set of functions provided)
 - Accuracy (the results are accurate)
 - Security (The capability to protect information and data so that the unauthorised persons or systems cannot read or modify them and authorised persons or systems cannot deny access to them.)

Software Quality (Q) Factors/Attributes



- Reliability:
 - *It is the capability to maintain a specified level of performance.*
 - This means the extent to which the program can be expected to perform its intended function with the required precision.

Software Quality (Q) Factors/Attributes



- **Usability:**
 - *It is the capability to be understood, learned and used.*
 - It further includes understandability, learnability and operability.
- **Efficiency:**
 - *It is the capability to provide appropriate performance relative to the amount of resources used.*

Software Quality (Q) Factors/Attributes



- **Maintainability:**
 - *It is the capability to be modified for purposes of making corrections, improvements or adaptation.*
 - It further includes changeability, testability and stability.

Software Quality (Q) Factors/Attributes



- Portability:
 - *It is the capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product.*
 - It further has adaptability, installability, etc.

Software Quality (Q) Factors/Attributes



- Two important consequences of multiple dimensions to quality
- First – Software quality cannot be reduced to single parameter
- Second – Concept of quality is project specific
- In general, reliability is considered as main quality factor.

Software Quality (Q) Factors/Attributes



- Unreliable Software = Presence of defects
- Quality measure = number of defects in software per unit size
- Best practice is 1 defect per 1 KLOC
- Quality objective is to reduce the number of defects per KLOC

Cost of Quality



- Cost of quality comprises of three components:
 - Cost of Prevention
 - Cost of Appraisal
 - Cost of Failure
- From these three costs, we can get the total cost of quality.

Cost of Quality



- Cost of Prevention
 - *Cost of prevention is the cost incurred in preventing an error from occurring.*
 - It could be doing a good requirements analysis, doing a good design and following a good standard.
 - *Prevention costs increase with increased degree of quality.*

Cost of Quality



- Cost of Appraisal
 - *Cost of appraisal is the cost incurred for all testing we do including code reviews and other forms of testing.*
 - As the cost of appraisal increases degree of quality also increases to an extend, but after a certain limit the degree of quality does not increase.
 - *Appraisal costs increases with increased degree of quality but reduces after it reaches a certain point.*

Cost of Quality



- Cost of Failure
 - *Cost of failure is the cost incurred in removing all the errors that have got into the software.*
 - Cost of failure also includes the good will lost which may not be measurable.
 - *Failure costs reduce with increase in degree of quality.*
- Fig. 2 shows Graph of Cost of Quality which shows there is usually an optimum level of quality at any given point of time.

Cost of Quality

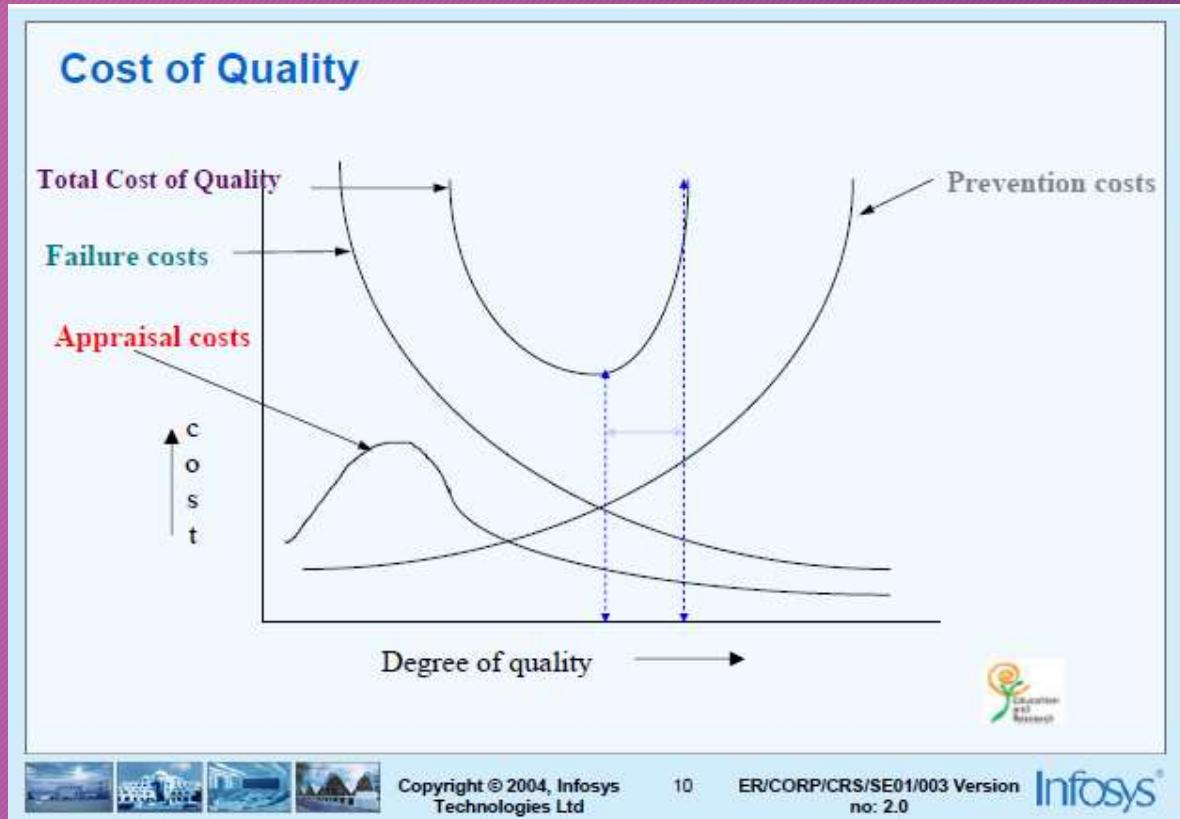


Fig. 2 Graph of Cost of Quality

Quality Standards



- ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization.
- ISO declared its 9000 series of standards in 1987.
- The ISO 9000 standard determines the guidelines for maintaining a quality system.
- The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc.

Quality Standards



- ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.
- It is series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically.
- For software, ISO 9001 and ISO 9004:2018 are suitable

Quality Standards

- ISO 9001:
 - ISO 9001 sets out the criteria for a quality management system and is the only standard in the family that can be certified to.
 - It specifies requirements for a quality management system (QMS).
 - It can be used by any organization, large or small, regardless of its field of activity.



Quality Standards

- ISO 9001:
 - Using ISO 9001 helps ensure that customers get consistent, good-quality products and services, which in turn brings many business benefits.



Quality Standards



- ISO 9004:2018
 - It gives guidelines for enhancing an organization's ability to achieve sustained success.
 - This guidance is consistent with the quality management principles given in ISO 9000:2015.
 - provides a self-assessment tool to review the extent to which the organization has adopted the concepts in this document.
 - It is applicable to any organization, regardless of its size, type and activity.

Quality Standards



- **Capability Maturity Model (CMM)**
 - The CMM is a procedure used to develop and refine an organization's software development process.
 - The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.
 - These are initial (level 1), repeatable, defined, managed, and optimizing (level 5).
 - When a process improves it moves from one level to next until it reaches level 5.

Quality Standards

- Capability Maturity Model (CMM)
 - For detailed information, refer **Module 1 part 3 PPT.**
 - Following table 1 shows comparison of ISO and CMM.



Quality Standards



ISO Vs CMM

ISO	CMM
ISO is a generic standard. It is applicable for all industries.	CMM is very specific to software industry.
ISO is a standard. It tells what needs to be done, in an organization.	CMM is a model. It doesn't mandate the practices.
ISO focuses on the entire organization's processes	CMM is specific to software processes
ISO Certification is followed by surveillance audit once in 6 months	After CMM assessment, there are no such checks. It is up to the organization to use it for internal process improvements.
ISO is continuous.	CMM is staged (It has different levels of process maturity)
Internal Audits are mandatory	Not mandatory



Table 1. Comparison of ISO and CMM

References

- Pankaj Jalote, "An Integrated Approach to Software Engineering", Narosa Publishing House, 2nd Edition
- <https://asq.org/quality-resources/iso-9000>
- <https://www.iso.org/iso-9001-quality-management.html>
- <https://asq.org/quality-resources/iso-9001>
- <https://www.iso.org/standard/70397.html>





Module 3 Software Development Phases – Part 1

Mr. Swapnil S Sontakke (Asst. Prof.)
Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli

Content

- Software Requirement Process
- Design Principles
- Coding Standards
- Levels of Testing





Software Requirement Process

Software Requirement Process



- Introduction
- Need of SRS
- Requirement Process

Software Requirement Process



- ## Introduction

- As the complexity and size of the software systems increases, new problems occur that were not present in smaller systems.
- For complex and large systems – goal could not be easily comprehended.
- For such systems, requirement analysis is most difficult, intractable and error-prone task.
- One of the difficulty is due to the scope of the activity.

Software Requirement Process



- **Introduction**

- The software project is initiated by the client's needs.
- These needs are in minds of various people and requirements analyst has to identify and understand requirements by talking to them.
- Information in minds is in general informal and not organized.
- So, input to this phase is informal, imprecise and likely to be incomplete.

Software Requirement Process



- **Introduction**
 - Software requirements phase translates the informal ideas in minds into a set of precisely specified document which hopefully are complete and consistent.
 - The process of identifying the requirements cannot be totally formal as input is in general imprecise and ambiguous.
 - Hence it cannot be fully automated and methods used to identify requirements can be at best a set of guidelines.

Software Requirement Process



- Introduction
 - IEEE defines a requirement as

“1. A condition of capability needed by a user to solve a problem or achieve an objective;

2. A condition or a capability that must be met or processed by a system...to satisfy a contract, standard, specification, or other formally imposed document.”

Software Requirement Process



- **Introduction**
 - Requirements = Capability of a system
 - The goal: To produce Software Requirements Specification (SRS) that describes *what* the proposed software should do without describing *how* the software will do it.
 - It requires a lot of efforts to produce high quality and relatively stable SRS.
 - Some of the reasons are discussed here.

Software Requirement Process



- Need of SRS
 - Three major parties interested in a new software system:
 - The client
 - The developer
 - The users
 - The requirements that satisfies the needs of the clients and the concerns of the users have to be communicated to the developer.

Software Requirement Process



- Need of SRS
 - Problem: *The client usually does not understand the software or the software development process, and the developer does not understand the client's problem and application area.*
 - This causes a communication gap between all the parties.
 - Solution: SRS bridges this gap.
 - SRS is a medium through which the client and user needs are accurately specified to the developer.

Software Requirement Process



- Need of SRS
 - **Advantage:** *An SRS establishes the basis for agreement between the client and the supplier on what software product will do.*
 - This basis for agreement is frequently formalised into a legal contract between the client and the developer (the supplier).
 - So, through SRS client clearly describes the what is expects from the supplier, and developer understands what capabilities to build in the software.

Software Requirement Process



- Need of SRS
 - Advantage: *An SRS provides a reference for validation of the final product.*
 - The SRS helps the client determine if the software meets the requirements.

Software Requirement Process



- Need of SRS
 - A study shows in some projects about 45% of errors originated during requirement and early design stages.
 - Also a report on errors shows many of the errors are made during the requirements phase.
 - An error in SRS will manifest as an error in final system.

Software Requirement Process



- Need of SRS
 - That is if SRS is specified wrongly, even a correct implementation of SRS will not satisfy the client.
 - Conclusion: *A high-quality SRS is a prerequisite to high-quality software*

Software Requirement Process



- Need of SRS
 - Quality of SRS has impact on cost of a project.
 - We know the cost of fixing an error increases almost exponentially as time progresses.
 - Following table shows approximate average cost of fixing requirement errors in person-hours depending on the phase in which it is detected.

Software Requirement Process



Phase	Cost (person-hours)
Requirements	2
Design	5
Coding	15
Acceptance Test	50
Operation and Maintenance	150

Software Requirement Process



- Need of SRS
 - In a project, following distribution is found: of the requirements errors that remain after the requirements phase
 - Errors detected during Design = 65%
 - Errors detected during Coding = 2%
 - Errors detected during Testing = 30%
 - Errors detected during Operation and Maintenance = 3%

Software Requirement Process



- Need of SRS
 - With the given distribution, the cost to fix 100 requirement errors that were detected in later phases would be:
$$(65 * 5) + (2 * 15) + (30 * 50) + (3 * 150)$$
$$= 2305 \text{ person-month}$$
 - In this example, by investing additional 100 person-hours in the requirements phase can reduce the cost of 2205 person-hours.

Software Requirement Process



- Need of SRS
 - Conclusion: *A high quality SRS reduces the development cost.*
 - Hence, quality of SRS impacts customer (and developer) satisfaction, system validation, quality of the final software and the software development cost.

Software Requirement Process



- **Requirement Process**

- It is the sequence of activities that need to be performed in the requirements phase and that culminate in producing a high-quality document containing the SRS.
- Typically, it has three basic tasks:
 - Problem or requirements analysis
 - Requirements specification
 - Requirements validation

Software Requirement Process

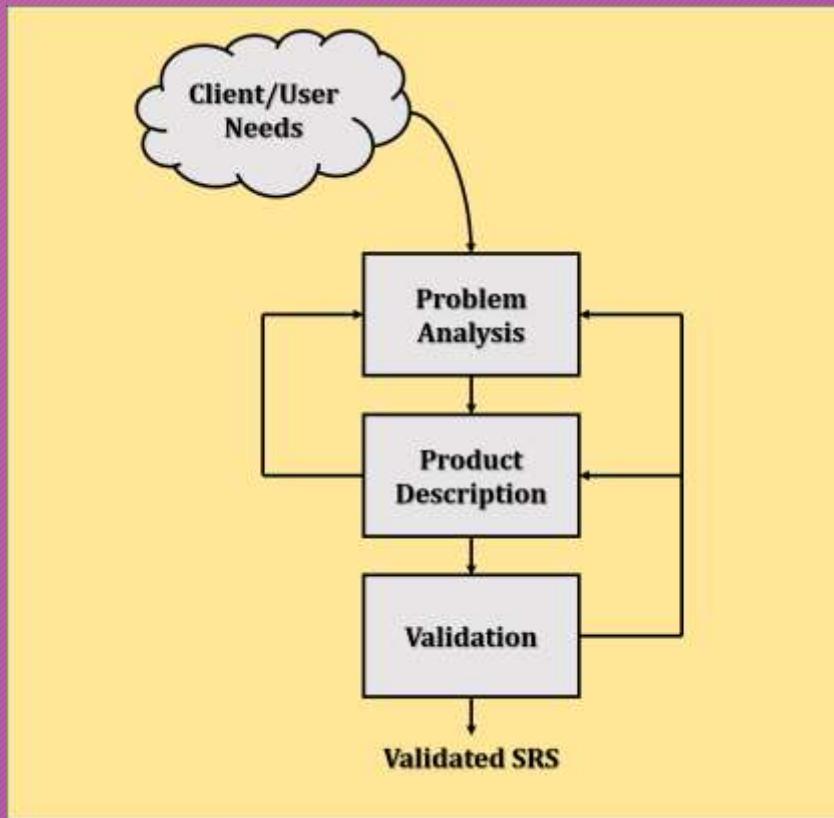


Fig. 1
The Requirement
Process

Software Requirement Process



- Requirement Process – Problem Analysis
 - It starts with a high-level problem statement.
 - Problem domain and the environment are modelled to understand the system behaviour, constraints on the system, its inputs and outputs, etc.
 - Goal: To obtain thorough understanding of what the software needs to provide.
 - It forms a basis of requirements specification.

Software Requirement Process



- Requirement Process – Requirements Specification
 - It focuses on clearly specifying the requirements in a document.
 - Issues such as representation, specification languages and tools, etc. are addressed during this activity.
 - Goal: To properly organize and describe the requirements out of large information and knowledge produced by analysis activity.

Software Requirement Process



- Requirement Process – Requirements Validation
 - It focuses on validation of requirements specified in the SRS.
 - Ensures that what has been specified in the SRS are indeed all the requirements of the software.
 - Also ensures that the SRS is of good quality.
 - Last activity in the requirements process.

Software Requirement Process



- **Requirement Process**

- Though requirements process seems to be linear sequence of activities, there is considerable overlap and feedback between these activities.
- This means analysis of many parts of the system can be going on at a time.
- If validation activity reveals any problem in SRS, then analysis and specification activities may need to be performed again as shown in the figure 1.

Software Requirement Process



- Requirement Process
 - Similarly, if specification activity shows any shortcomings in the knowledge of problem then further analysis needs to be done.
 - The level of details that the requirement process shows depends on the objective of the requirement phase.

Software Requirement Process



- **Problem Analysis**
 - Goal: To obtain thorough understanding of needs of the clients and users, what exactly is desired from the software and what the constraints on the solutions are.
 - In general, clients and users do not understand or know their needs fully.
 - The analysts have to ensure that these real needs are uncovered.

Software Requirement Process



- Problem Analysis
 - Basic principal used in analysis is *divide and conquer*.
 - To understand the complex problem, *partition* it into subproblems, understand each subproblem and its relationship with other subproblems.
 - Two concepts are effectively used in the partitioning:
 - State
 - Projection

Software Requirement Process



- Problem Analysis
 - State
 - Represents some condition about the system.
 - System is viewed as operating in one of the possible states and then detailed analysis is performed.
 - Useful in real-time system or process control software

Software Requirement Process



- Problem Analysis
 - Projection
 - System is defined from multiple points of view and then is analysed from these different perspectives.
 - These projections are combined to form the analysis of the whole system.

Software Requirement Process



- Problem Analysis – Informal Approach
 - No defined methodology is used for analysis
 - Information about the system is obtained by interaction with client, end users, questionnaire, study of existing systems, brainstorming, etc.
 - The problem and system model is built in the minds of analysts and are directly translated into the SRS
 - Analysts will have series of meeting with client and end users.

Software Requirement Process



- Problem Analysis – Informal Approach
 - In early meetings, analyst is basically listener and obtains information such as about work, their environment, and basic needs as per their perception.
 - In the final meetings, initial draft is prepared.
 - Widely used approach.

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Often referred to as *structured analysis technique*
 - It focuses on the functions performed in the problem domain and the data consumed and produced by these functions.
 - It is a top-down refinement approach, originally called "*structured analysis and specification*"
 - It uses Data Flow Diagrams and Data Dictionary

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Data Flow Diagram (DFD)
 - Also called as Data Flow Graphs
 - Useful in understanding the system and can be used effectively during problem analysis
 - It shows flow of data through a system
 - It views a system as a function that transforms the inputs into desired outputs.

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Data Flow Diagram (DFD)
 - In case of complex system, transformation is not a single step rather it is series of transformations.
 - The aim of DFD is to capture the transformation to the input data that eventually produces output data.
 - It depicts the system requirements graphically.

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Data Flow Diagram (DFD)
 - It shows how data enters and leaves the system, what changes the information, and where data is stored.
 - It acts as a communication tool between analyst and client or end users.
 - There are various symbols to represent the DFD elements.

Software Requirement Process



Table 1: Symbols used in DFD

	Process or Bubble	Agent that performs the transformation of data from state to another. (Transformation of input data to output data)
	Source or Sink	Originator or consumer of data (input and output of system)
	Data Flows	Connects the processes, source or sink
	Data Store or Data Warehouse	Repository of data. The arrow heads indicate net inputs and net outputs to data store. E.g. Database, external files

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Data Flow Diagram (DFD)
 - Multiple data flows between two processes is represented by * (A AND B relationship)
 - One of the data flows between processes is represented by + (A OR B relationship)

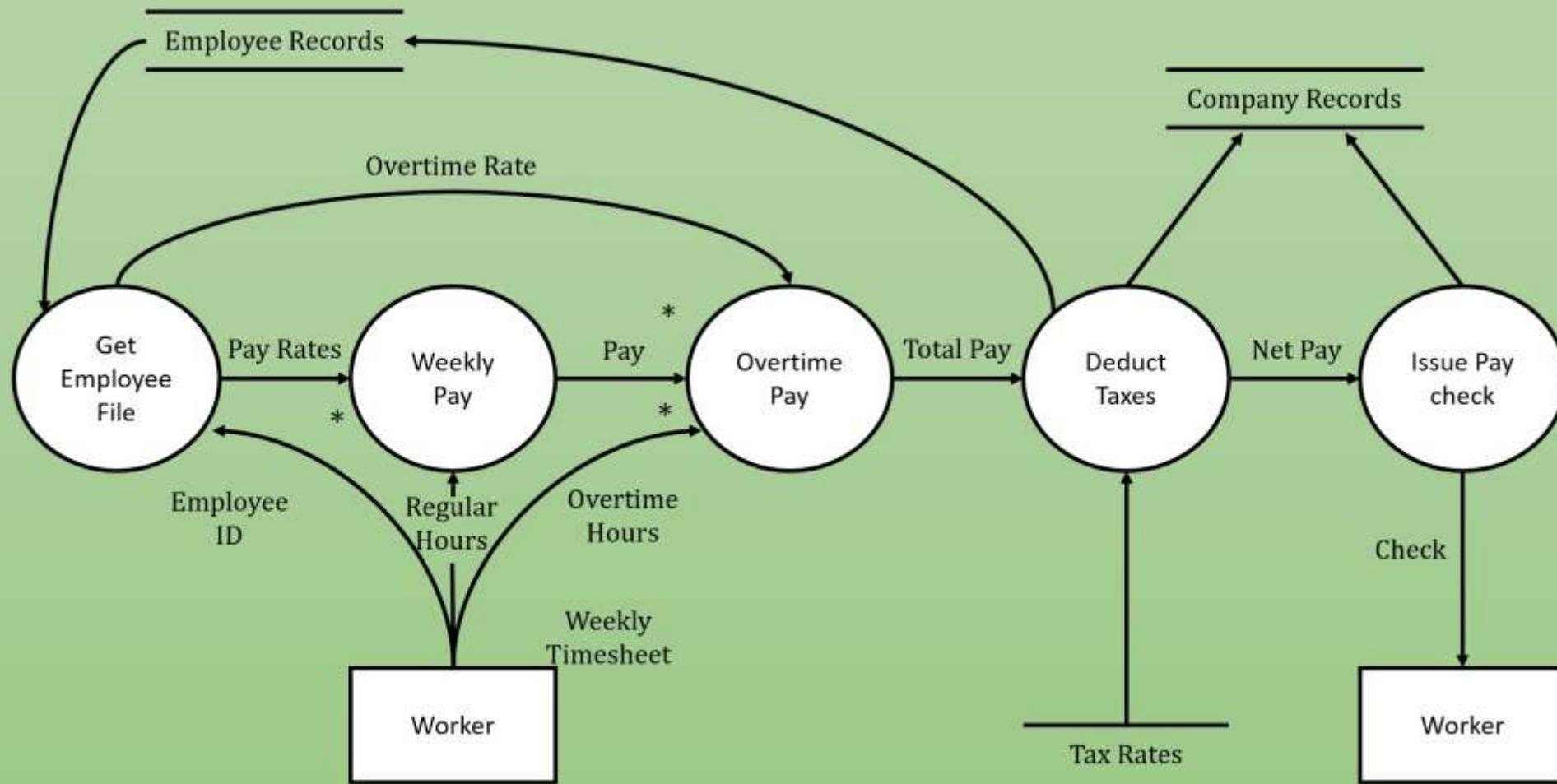


Fig. 2 DFD of a system that pays workers

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Data Flow Diagram (DFD)
 - DFDs can be hierarchically organized to help progressively partition and analyse the larger systems.
 - Such a DFDs together are called *levelled DFD set*.
 - There are different levels level 0, level 1, etc.
 - A starting DFD (level 0) gives abstract representation of system, covering major inputs, outputs and processes.

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Data Flow Diagram (DFD)
 - Every level then in detail refine each of the process.
 - This refinement stops when the process is considered to be *atomic*.

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Data Dictionary
 - DFD specifies data flows which are identified by unique names.
 - But they don't specify the precise structure of data flows.
 - *Data dictionary is a repository of various data flows defined in DFD that precisely specifies their structure.*

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Data Dictionary
 - Important Notations are:
 - + represents sequence or composition (select one AND other)
 - | represents selection (means select one OR other)
 - * represents iteration (means one or more occurrences)
 - Example:

Software Requirement Process



```
weekly timesheet =  
    Employee Name +  
    Employee ID +  
    [Regular Hours + Overtime Hours]*  
  
pay_rate =  
    [Hourly | Daily | Weekly] +  
    Dollar_amount  
  
Employee_name =  
    Last + First + Middle_initial  
  
Employee_ID =  
    digit + digit + digit + digit
```

Fig. 3 Data Dictionary

Software Requirement Process



- Problem Analysis – Data Flow Modelling
 - Other Problem Analysis Methods
 - Object-Oriented Modelling
 - Prototyping

Software Requirement Process



- Characteristics of an SRS
 - To properly satisfy the basic goals, an SRS should have certain properties and contain different requirements. A good SRS is:

1	Correct	5	Consistent
2	Complete	6	Ranked for importance and/or stability
3	Unambiguous	7	Modifiable
4	Verifiable	8	Traceable

Software Requirement Process



- Characteristics of an SRS
 - **Correct:**
 - An SRS is *correct* if every requirement included in the SRS represents something required in the final system.
 - It ensures that what is specified is done correctly.
 - It is easier to establish as it requires examining each requirement to make sure that it represents the user requirement.

Software Requirement Process



- Characteristics of an SRS
 - **Complete:**
 - An SRS is *complete* if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS.
 - It ensures that everything is indeed specified.
 - It is difficult property to establish as it requires to detect the absence of specifications.

Software Requirement Process



- Characteristics of an SRS
 - **Unambiguous:**
 - An SRS is *unambiguous* if and only if every requirement that has stated has one and only one interpretation.
 - Generally, SRS is written in natural language which is inherently ambiguous.
 - So care must be taken to ensure that there are no ambiguities.

Software Requirement Process



- Characteristics of an SRS
 - **Unambiguous:**
 - Some formal requirement specification language should be used to avoid ambiguity.
 - But, it requires a lot of efforts and cost.
 - Also, there is increased difficulty in reading and understanding formally stated requirements for users and clients.

Software Requirement Process



- Characteristics of an SRS
 - **Verifiable:**
 - An SRS is *verifiable* if and only if every stated requirement is verifiable.
 - A requirement is *verifiable* if there exists some cost effective process that can check whether the final software meets that requirement.
 - Unambiguity is essential for verifiability.

Software Requirement Process



- Characteristics of an SRS
 - **Verifiable:**
 - As verification of requirements is often done through reviews, it implies that the SRS should be understandable, at least by the developer, the clients and the users.

Software Requirement Process



- Characteristics of an SRS
 - **Consistent:**
 - An SRS is *consistent* if there is no requirement that conflicts with another.
 - Terminologies can cause inconsistencies. E.g. different requirements can use different terms to refer to the same object.
 - There may be logical or temporal conflict between requirements that causes inconsistencies.

Software Requirement Process



- Characteristics of an SRS
 - **Consistent:**
 - This occurs if the SRS contains two or more requirements whose logical or temporal characteristics can not be satisfied together by any software system.
 - E.g. one requirement states event e should occur before event f another requirement states, directly or indirectly that event f should occur before event e .

Software Requirement Process



- Characteristics of an SRS
 - **Ranked for importance and/or stability:**
 - Generally, all requirements are not of equal importance.
 - Some are critical, some are important but not critical and some are desirable but not important.
 - Similarly, some requirements are *core* which are not likely to change in the future, and some are time dependent.

Software Requirement Process



- Characteristics of an SRS
 - **Ranked for importance and/or stability:**
 - An SRS is *ranked for importance and/or stability* if for each requirement the importance and the stability of the requirement are indicated.
 - Stability reflects the chances of requirement change in the future.

Software Requirement Process



- Characteristics of an SRS
 - **Modifiable**
 - An SRS is *modifiable* if its structure and style are such that any necessary change can be made easily while preserving completeness and consistency
 - This is required as SRS is an iterative process.
 - The SRS needs to be modified when there is change in needs of customer. Hence, it should be easily modifiable.

Software Requirement Process



- Characteristics of an SRS
 - **Modifiable**
 - A redundancy is a major hindrance to modifiability, as it can easily lead to errors.
 - E.g. If one requirement is stated at two places and that same requirement needs to be modified after some time.
 - If its only one occurrence is modified then resulting SRS will be inconsistent which is not acceptable.

Software Requirement Process



- Characteristics of an SRS
 - **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.
 - *Forward traceability* means that each requirement should be traceable to some design and code elements.

Software Requirement Process



- Characteristics of an SRS
 - **Traceable**
 - *Backward traceability* means that it is possible to trace design and code elements to the requirements they support.
 - Traceability aid verification and validation.

Software Requirement Process



- Characteristics of an SRS
 - Of all these characteristics, completeness is most important and hardest to ensure.
 - Also, incompleteness is a major source of disagreement between the client and the supplier.

Software Requirement Process



- Components of an SRS
 - An SRS should be complete, but it is difficult to achieve and verify.
 - Some guidelines about what an SRS should specify will help in completely specifying the requirements.
 - The basic issues that an SRS must address are: *Functionality, Performance, Design constraints imposed on an implementation and external interfaces.*

Software Requirement Process



- Components of an SRS
 - **Functional Requirements**
 - Specify which outputs should be produced from the given inputs.
 - Describe the relationship between inputs and outputs.
 - For each functional requirement, a detailed description of all the data inputs and their source, the units of measure and the range of valid inputs must be specified.

Software Requirement Process



- Components of an SRS
 - **Functional Requirements**
 - All operations to be performed on the input data to obtain the output should be specified. This includes
 - Validity checks on inputs and outputs,
 - Parameters affected by operation, and
 - Equations or other logical operations used to transform input into corresponding output

Software Requirement Process



- Components of an SRS
 - **Functional Requirements**
 - It must clearly state system behaviour in abnormal situations like invalid inputs, errors during computation, valid input but normal operation cannot be performed, etc.
 - E.g. Airline Reservation System, a valid passenger cannot book a ticket because the airplane is fully booked.

Software Requirement Process



- Components of an SRS
 - **Performance Requirements**
 - Specifies the performance constraints on the system
 - Two types of requirements: *static* and *dynamic*

Software Requirement Process



- Components of an SRS
 - **Performance Requirements**
 - Static Requirements
 - Do not impose constraint on execution characteristics of the system
 - It includes no of terminals to be supported, no of simultaneous users to be supported, no of files that the system has to process and their sizes
 - Also called as *capacity* requirements of the system

Software Requirement Process



- Components of an SRS
 - **Performance Requirements**
 - Dynamic Requirements
 - Impose constraint on execution behavior of the system
 - It includes response time and throughput constraints
 - *Response time is the expected time for the completion of operation under specified circumstances.*
 - *Throughput is the expected number of operations that can be performed in a unit time.*

Software Requirement Process



- Components of an SRS
 - **Performance Requirements**
 - Dynamic Requirements
 - E.g. an SRS may specify response time and throughput of the system
 - Acceptable ranges for performance parameters and acceptable performance for normal and peak workload conditions must be specified.

Software Requirement Process



- Components of an SRS
 - **Performance Requirements**
 - All these requirements should be stated in measurable terms.
 - E.g. “response time of command x should be less than 1 second 90% of the time” instead of “response time should be good”

Software Requirement Process



- Components of an SRS
 - **Design Constraints**
 - Number of factors in client's environment may restrict the choices of a designer.
 - Such factors include standards that must be followed, resource limits, operating environment, reliability and security requirements, and policies that may have impact on the system
 - An SRS must identify and specify such constraints.

Software Requirement Process



- Components of an SRS
 - **Design Constraints – Standard Compliance**
 - Specifies the standards that must be followed
 - It may include report format and accounting procedures.
 - There may be audit tracing requirements, which requires changes or operations must be recorded in audit file.

Software Requirement Process



- Components of an SRS
 - **Design Constraints – Hardware Limitations**
 - Software may require predetermined hardware to operate.
 - It can include type of machine to be used, type of OS on the machine, languages supported, limits on primary and secondary storage.

Software Requirement Process



- Components of an SRS
 - **Design Constraints – Reliability and Fault Tolerance**
 - Fault tolerant requirements make system more complex and expensive.
 - It includes the requirements about system behaviour in the face of certain kinds of faults.
 - Recovery requirements specify what system should do if some failure occurs to ensure certain properties.

Software Requirement Process



- Components of an SRS
 - **Design Constraints – Security**
 - Security requirements are significant in defence systems and many database systems.
 - They place restrictions on the use of certain commands, control access to data, provide different kinds of access for different people, require the use of passwords and cryptography techniques and maintain the log of activities in the system.

Software Requirement Process



- Components of an SRS
 - **Design Constraints – Security**
 - They may also require proper assessment of security threats, proper programming techniques, and use of tools to detect flaws.

Software Requirement Process



- **Specification Language**

- Natural language like English is preferred to write the specification as it is understandable by both client and the developer.
- As natural language is ambiguous and imprecise, for some part of the specification, formal language can be used.
- To reduce the drawbacks of natural language, it is generally used in structured fashion where requirements are broken into sections and paragraphs.

Software Requirement Process



- Specification Language
 - Paragraphs are broken into subparagraphs.
 - A general rule is followed while using natural language:
 - Be precise, factual and brief
 - Organize requirements hierarchically whenever possible
 - Give unique numbers to each separate requirement
 - Formal language include use of regular expressions, decision tables, etc.

Software Requirement Process



- Structure of a SRS
 - To specify all the requirements clearly and concisely, an SRS should be organized in sections and subsections.
 - Many methods and standards are proposed for organizing the SRS.
 - Goal: To fit SRS in certain pattern so that it is easy to understand and help ensure that the analyst does not forget some major property.

Software Requirement Process



- Structure of a SRS
 - According to IEEE standards there is no single method to organize SRS.
 - But in all cases, first two sections are same.
 - Figure 4 shows a general structure of an SRS.

Software Requirement Process



- 1. Introduction**
 - 1.1 Purpose**
 - 1.2 Scope**
 - 1.3 Definitions, Acronyms, and Abbreviations**
 - 1.4 References**
 - 1.5 Overview**
- 2. Overall Description**
 - 2.1 Product Perspective**
 - 2.2 Product Functions**
 - 2.3 User Characteristics**
 - 2.4 General Constraints**
 - 2.5 Assumptions and Dependencies**
- 3. Specific Requirements**

Fig. 4 General Structure of an SRS

Software Requirement Process



- Structure of a SRS
 - The first section contains purpose, scope, references, etc.
 - The second section describes the general factors that affect the product and its requirements.
 - Specific requirements are not specified, only general overview is presented to make the understanding of specific requirements easier.
 - Schematic diagrams can be used in second section.

Software Requirement Process



- Structure of a SRS
 - The third section describes all the details that a developer needs to know for designing and developing a system.
 - Largest and important part of the document
 - One method to organize this is shown in the figure 5.



3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interface

3.1.2 Hardware Interface

3.1.3 Software Interface

3.1.4 Communication Interface

3.2 Functional Requirements

3.2.1 Mode 1

3.2.1.1 Functional Requirement 1.1

.

.

3.2.1.n Functional Requirement 1.n

.

.

3.2.m Mode m

3.2.m.1 Functional Requirement 1.1

.

.

3.2.m.n Functional Requirement 1.n

3.3 Performance Requirements

3.4 Design Constraints

3.5 Attributes

3.6 Other Requirements

Fig. 5 One organization for specific requirements

Software Requirement Process



- Structure of a SRS
 - External interface requirements specifies all interfaces of software to people, hardware, other software and other systems.
 - UI: How human interact with system e.g. menus, screen formats, etc.
 - HI: Hardware requirement to run the software

Software Requirement Process



- Structure of a SRS
 - SI: Other software and interfaces needed to run this software
 - CI: How software communicates with other systems with interfaces required
 - In functional requirements section, functional capabilities for all the modes of the systems are specified.
 - They include required inputs, desired outputs and processing requirements

Software Requirement Process



- Structure of a SRS
 - Performance section specifies all static and dynamic performance requirements.
 - In design constrain section, design constraints are specified such as fault tolerance, security, etc.
 - In attributes section, some of the overall attributes of the system.
 - Requirements that are not covered are listed in other requirements.

Software Requirement Process



- Validation of SRS
 - An SRS determines whether or not the delivered software is acceptable.
 - Also, due to nature of the requirements specification phase, there is a possibility of misunderstanding and committing errors.
 - It is quite possible that the document may not accurately represent the client's needs.
 - Also, longer an error remains undetected, greater the cost to correct it.

Software Requirement Process



- Validation of SRS
 - Hence, it is necessary to detect the errors in the requirements before the design and development of the software begins.
 - The main objective of requirements validation phase is to ensure that SRS reflects the actual requirements accurately and clearly.
 - Another objective is to check that the SRS document is itself of “good quality”.
 - Before validation we discuss types of errors that occur in SRS.

Software Requirement Process



- Validation of SRS
 - Out of many possible errors, most common errors can be classified into four types:
 - Omission
 - Inconsistency
 - Incorrect fact
 - Ambiguity

Software Requirement Process



- Validation of SRS
 - **Omission:**
 - In this, some user requirement is not included in SRS.
 - Such a requirement can be related to the behaviour, performance, constraints of the system or can be any other
 - Omission directly affects the completeness of the SRS.

Software Requirement Process



- Validation of SRS
 - **Inconsistency:**
 - It can be due to
 - The contradiction within the requirements themselves
 - Incompatibility of the stated requirements with the actual requirements of the clients or with environment in which the software will operate.

Software Requirement Process



- Validation of SRS
 - **Incorrect Fact:**
 - This error occurs when some fact recorded in the SRS is not correct.
 - **Ambiguity:**
 - This error occurs when there are some requirements that have multiple meaning, i.e. their interpretation is not unique.

Software Requirement Process



- Validation of SRS
 - Data collected from some projects shows:
 - On an average, more than 250 errors were detected and their percentage is as follows:

Omission	Incorrect Fact	Inconsistency	Ambiguity
26%	10%	38%	26%

Software Requirement Process



- Validation of SRS
 - Data collected from A-7 project shows:
 - Total 80 errors were detected of which 23% were clerical in nature and percentage of remaining is as follows:

Omission	Incorrect Fact	Inconsistency	Ambiguity
32%	49%	13%	5%

Software Requirement Process



- Validation of SRS
 - Even if the distribution differs in two table, if we take average of both tables, it shows all four types of errors are significant.
 - This implies, quality of SRS should be improved i.e. no clerical errors and validation should focus on revealing these errors.
 - As SRS is a textual document, inspections are used for validation.
 - This inspections of SRS is called as *requirements review*.

Software Requirement Process



- Validation of SRS
 - *It is a review by group of people to find errors and point out other matters of other concerns in the SRS of a system.*
 - The group include author of SRS document, a person who understands the needs of the client (or client), a person from design team, and the person responsible for maintaining the SRS.

Software Requirement Process



- Validation of SRS
 - In general review process focuses on
 - Revealing any errors in the requirements
 - Considering factors affecting quality such as testability and readability
 - One of the job of reviewer is to uncover the requirements that are too subjective and too difficult to define criteria for testing that requirement.

Software Requirement Process



- Validation of SRS
 - Checklists are frequently used in reviews to focus the reviews effort and ensure that no major source of errors is overlooked by the reviewers.
 - Figure 6 shows a sample checklist:

Software Requirement Process



- Are all hardware resources defined?
- Have the response times of the functions been specified?
- Have all the hardware, external software and data interfaces been defined?
- Have all the functions required by the client been specified?
- Is each requirement testable?
- Is the initial state of the system defined?
- Are the responses to exceptional conditions specified?
- Does the requirement contain restrictions that can be controlled by the designer?
- Are possible modifications specified?

Fig. 6 Sample Checklist used by reviewers

References

- Pankaj Jalote, "An Integrated Approach to Software Engineering", Narosa Publishing House, 2nd Edition
- <https://www.javatpoint.com/software-engineering-data-flow-diagrams>





Module 3 Software Development Phases – Part 2

Mr. Swapnil S Sontakke (Asst. Prof.)
Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli

Content

- Design Principles
- Coding Standards
- Testing Fundamentals





Design Principles

Design Principles



- Introduction
 - Design activity begins when the SRS of the software becomes available and architecture has been designed.
 - *The design of a system is essentially a blueprint or a plan for the system.*
 - The design process often has two levels:
 - System design or top-level design
 - Logic design or detailed design

Design Principles



- Introduction
 - System design or top-level design
 - Focus is on deciding which modules are needed for the system, their specifications and how these modules should be interconnected.
 - Logic design or detailed design
 - The internal design of the modules or how the specifications of the module can be satisfied, is decided.

Design Principles



- **Introduction**

- The design of a system should be correct, verifiable, complete and traceable.
- The goal of design process is not simply to produce a design but is to find the best possible design within the limitations imposed by requirements and physical and social environment in which the system will operate.
- Two most important properties are *efficiency* and *simplicity*.

Design Principles



• Introduction

- *Efficiency* of any system is concerned with the proper use of scarce resources by the system.
- If some resources are scarce and expensive, then they must be used efficiently.
- *Simplicity* of any system is concerned with how simple and understandable the system is and it depends on the design.
- Simplicity affects the maintainability of the system.

Design Principles



- **Introduction**
 - Only after having the thorough knowledge of the system, the maintainer can do necessary modification in the system and maintain the system.
 - Following design principles help achieve good design of a system.

Design Principles



- Problem Partitioning and Hierarchy
 - Small problems can be solved entirely, but it is quite difficult to solve large and complex problems all at once.
 - The basic principle is the *time-based principle of divide and conquer*.
 - The goal of software design is to divide the problem into manageable smaller pieces so that each piece can be conquered (solved) separately.

Design Principles



- **Problem Partitioning and Hierarchy**
 - It is believed that the cost of solving the entire problem is more than the sum of the cost of solving all the pieces.
 - However, all the pieces cannot work independently as they together form the system.
 - They have to cooperate and communicate to solve the larger problem.
 - This communication due to partitioning adds complexity which may not have existed in the original problem.

Design Principles



- Problem Partitioning and Hierarchy
 - Also, cost of partitioning and combining along with the complexity become more than saving achieved by partitioning.
 - It can be argued that maintenance is minimized if each part can be easily related to the application and each piece can be modified easily.

Design Principles



- Problem Partitioning and Hierarchy
 - Problem partitioning leads to the hierarchies in the design.
 - So, design can be represented as a hierarchy of components.
 - In general, hierarchical structure makes complex system easier to understand.

Design Principles



- Abstraction
 - Abstraction is used in all engineering disciplines.
 - *It is a tool that permits a designer to consider a component at an abstract level without worrying about the details of the implementation of the component.*
 - Every component provides a service to its environment.
 - An abstraction of a component describes the external behaviour of that component without bothering with the internal details that produce the behaviour.

Design Principles



- Abstraction
 - Abstraction is essential part of the design process and problem partitioning.
 - Various components interact with other components and to understand this the designer has to know the external behaviour of other components.
 - It should not determine the behaviour of other components.
 - To allow the designer to concentrate on one component at a time, abstraction of other components is used.

Design Principles



- **Abstraction**
 - Abstraction is used for existing components as well as components that are being designed.
 - Also, it helps during the maintenance phase.
 - Using the abstractions, behaviour of entire system can be understood which help determine how modifying a component affects the system.

Design Principles

- Abstraction
 - Two common abstraction mechanisms:
 - Functional Abstraction
 - Data Abstraction



Design Principles



- **Abstraction – Functional Abstraction**
 - In this a module is specified by the function it performs.
 - E.g. a module to sort an array can be shown by the specification of sorting, a module to compute the log of a value can be represented by the function log, etc.
 - It is a basis of partitioning in function-oriented design i.e. when the problem is being partitioned, the overall transformation function for the system is partitioned into smaller functions that comprise the whole system.

Design Principles



- Abstraction – Data Abstraction
 - In this data is treated as objects with some predefined operations on them.
 - The operations defined on a data object are the only operations that can be performed on those objects.
 - From the outside of the object, the internal details of the object are hidden; only operations are visible.
 - It forms a basis for *object-oriented design*.

Design Principles



- **Modularity**
 - A system is considered *modular* if it consists of discrete components so that each component can be implemented separately, and a change to one component has minimal impact on other components.
 - Modularity helps in system debugging, system repair and system building.

Design Principles



- **Modularity**
 - For modularity to work well in software system each module should
 - support a well-defined abstraction
 - have clear interface through which it can interact with other modules.
 - For easily understandable and maintainable system, modularity is important and partitioning and abstraction help achieve the modularity.

Design Principles



- **Top-Down and Bottom-Up Strategies**
 - In general, a system is a hierarchy of components and each component can have its sub-components.
 - Highest-level component corresponds to the whole system
 - To approaches to design a hierarchy:
 - Top-down approach
 - Bottom-up approach

Design Principles



- Top-Down and Bottom-Up Strategies – Top-down Approach
 - Top-down approach starts from highest-level component and proceeds through to lowest levels.
 - It starts by identifying the major components of the system, decomposing them into their lower-level components and iterating until the desired level is achieved.
 - It results in *stepwise refinement*.

Design Principles



- **Top-Down and Bottom-Up Strategies – Top-down Approach**
 - Design starts with abstract design and at each step the design is refined to more concrete level until no refinement is needed and design can be implemented directly.
 - This is suitable when the system specifications are well-known and the development is from scratch.

Design Principles



- Top-Down and Bottom-Up Strategies – Bottom-up Approach
 - Design starts with most basic or primitive components and proceeds to higher-level components that use these lower-level components.
 - It works with *layers of abstraction*.
 - Starts from very bottom, operations that provide the layer of abstraction are implemented.
 - The operations at this layer are then used to implement more powerful operations.

Design Principles



- Top-Down and Bottom-Up Strategies – Bottom-up Approach
 - This is continued until operations supported by the layer are those desired by the system.
 - This is suitable if a system is to be built from an existing system.



Coding Standards

Coding Standards



• Introduction

- Programmers spend a little time on writing the code and more time on reading the code during debugging and enhancement.
- People other than author spend a considerable time in reading as the code is often maintained by someone other than the author.
- So, it is important that code should be easy to read, understand and maintain.

Coding Standards



- **Introduction**
 - Coding standards provide rules and guidelines for programmers for some aspects of programming in order to make the code easier to read, understand and maintain.
 - These guidelines may be regarding naming, file organization, statements and declarations and layouts and comments.
 - We discuss here some of the generalised guidelines with respect to Java programming language.

Coding Standards



- Coding Standards – Naming Conventions
 - Variable names should be nouns starting with *lower case* (*e.g. name, amount, sum*)
 - Constant names should be *upper case* (*e.g. PI, MAX, MIN*)
 - Method names should be verbs starting with *lower case* (*e.g. getValue(), computeArea()*)
 - Package names should be *lower case* (*e.g. mypackage, game.ui*)
 - Loop iterators should be named *I, j, k, l, etc.*

Coding Standards



- Coding Standards – Naming Conventions
 - Boolean variable and method names should start with *is* to avoid the confusion (*e.g. isValid()*, *isNumber*)
 - Also negative Boolean names should be avoided (*e.g. isNotCorrect*, instead use *isFalse*)

Coding Standards



- Coding Standards – Files
 - Java source file should have the extension *.java*
 - Class name should be same as the file name
 - Line length should be less than 80 columns and special characters such as *, :, ;, etc should be avoided

Coding Standards



- Coding Standards – Statements
 - Variable should be initialised where declared
 - Related variables should be declared in a common statement (*e.g. num1, num2 and result should be declared as*
int num1, num2, result=0;
And not
int num1;
int num2;
int result;

Coding Standards



- Coding Standards – Indentation
 - Proper indentation should be used as it makes the code readable. Some of the spacing conventions are:
 - Each nested block should be properly indented and spaced.
 - Proper Indentation should be there at the beginning and at the end of each block in the program.
 - All braces should start from a new line and the code following the end of braces also start from a new line.
 - There must be a space after giving a comma between two function arguments.

Coding Standards



- Coding Standards – Comments
 - *Comments are the textual statements that are meant for the program reader to aid the understanding of the code.*
 - In general comments should explain what the code is doing and why the code is written. Some guidelines are:
 - Single line comments for a block should aligned with the code they are meant for
 - Comments for major variables should be there
 - Trailing comments should be short

Coding Standards



```
import java.util.Scanner;
public class HelloWorld

public static void main(String[] args)
{
    // Creates a reader instance which takes
    // input from standard input - keyboard
    Scanner reader = new Scanner(System.in);
    System.out.print("Enter a number: ");
    // nextInt() reads the next integer from the keyboard
    int number = reader.nextInt();
    // println() prints the following line to the output screen
    System.out.println("You entered: " + number);
}
```

Fig. 1a Code without Indentation

```
import java.util.Scanner;
public class HelloWorld

public static void main(String[] args)
{
    // Creates a reader instance which takes
    // input from standard input - keyboard
    Scanner reader = new Scanner(System.in);
    System.out.print("Enter a number: ");
    // nextInt() reads the next integer from the keyboard
    int number = reader.nextInt();
    // println() prints the following line to the output screen
    System.out.println("You entered: " + number);
}
```

Fig. 1b Code with Indentation



Testing Fundamentals

Testing Fundamentals



- Introduction
- Fundamentals Terms - Error, Fault and Failure
- Test Cases
- Testing Process

Testing Fundamentals



- **Introduction**
 - *Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce the quality product.*
 - Testing performs a critical role for ensuring the software quality.
 - We discuss testing fundamentals here.

Testing Fundamentals



- Fundamentals Terms - Error
 - Error has two different meanings.
 - *An error refers to the discrepancy between a computed, observed or measured value and the true, specified, or theoretically correct value.*
 - That is error refers to the difference between the actual output of a software and the correct output.
 - *Error also refers to the human action that results in software containing a defect or fault.*

Testing Fundamentals



- Fundamentals Terms - Fault
 - *Fault is a condition that causes a system to fail in performing the required function.*
 - It is the basic reason for software malfunction.
 - It is commonly called as a *bug*.
 - Faults in software system are only because of design faults and not because of wear and tear in software.

Testing Fundamentals



- Fundamentals Terms - Failure
 - *Failure is the inability of a system or component to perform a required function according to its specifications.*
 - It occurs if the behavior of the software is different from the specified behavior.
 - It may be caused due to functional or performance reasons.

Testing Fundamentals



- Fundamentals Terms - Failure
 - It is produced only when there is a fault in the system.
 - Presence of fault is necessary but not sufficient condition for failure to occur.

Testing Fundamentals



- Test Cases
 - A *TEST CASE* is a documented set of preconditions (prerequisites), procedures (inputs / actions) and postconditions (expected results) which a tester uses to determine whether a system under test satisfies requirements or works correctly.
 - A test case can have one or multiple *test scripts* and a collection of test cases is called a *test suite*.

Testing Fundamentals



- Test Cases - Characteristics
 - **Accurate:** Exacts the purpose.
 - **Economical:** No unnecessary steps or words.
 - **Traceable:** Capable of being traced to requirements.
 - **Repeatable:** Can be used to perform the test over and over.
 - **Reusable:** Can be reused if necessary.

Testing Fundamentals



- Test Cases – Sample examples
 - **Test Case 1:** Check results on entering valid User Id & Password
 - **Test Case 2:** Check results on entering Invalid User ID & Password
 - **Test Case 3:** Check response when a User ID is Empty & Login Button is pressed

Testing Fundamentals



- Test Cases – Sample examples
 - Each test case costs money as efforts are needed to generate it, machine time to execute it and efforts to evaluate the results.
 - So, fundamental goal of testing is to *maximize the number of errors detected and minimize the number of test cases*.
 - To achieve this, *test selection criterion (or test criterion)* is used.

Testing Fundamentals



- Testing Process
 - *It is a process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of a component or system and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.*

Testing Fundamentals



- Testing Process – Levels of Testing
 - Testing is usually relied upon to detect the faults from earlier stages, in addition to the faults introduced during coding itself.
 - So, to test the different aspects of the system, different levels of testing are used.
 - These are *unit, integration, system* and *acceptance testing*.
 - Figure 2 shows the relation of the faults introduced in different phases and different levels of testing.

Testing Fundamentals

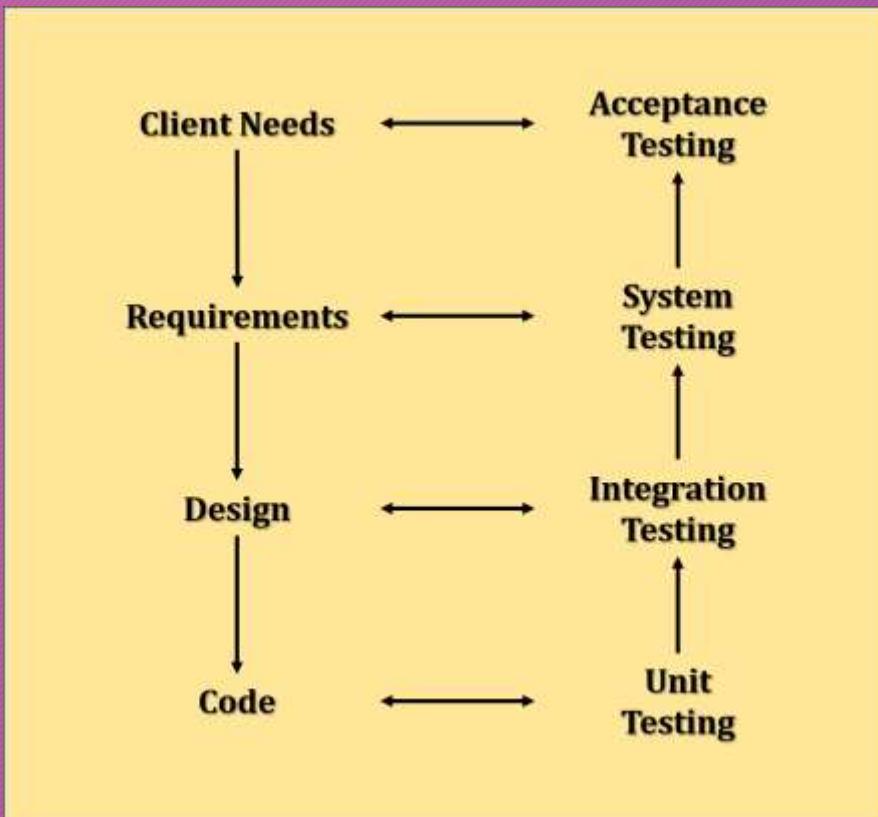


Fig. 2 Levels of Testing

Testing Fundamentals



- Testing Process – Levels of Testing
 - Unit Testing
 - In this different modules are tested against the specifications produced during design for the modules.
 - It is generally used for verification of code, hence the goal is to test the internal logic of the modules.
 - This testing is typically done by programmer of the module.

Testing Fundamentals



- Testing Process – Levels of Testing
 - Integration Testing
 - In this different unit tested modules are combined into subsystems, which are then tested.
 - The goal is to see if the modules can be integrated properly.
 - The emphasis is on testing interfaces between modules.
 - This can be considered testing the design.

Testing Fundamentals



- Testing Process – Levels of Testing
 - System and Acceptance Testing
 - In this entire software is tested against the requirements specified in the SRS.
 - The goal is to see if the software meets its requirements.
 - This is generally validation activity.
 - In acceptance testing, sometimes the realistic data from client is used to demonstrate that the software is working satisfactorily.

Testing Fundamentals



- Testing Process – Levels of Testing
 - System and Acceptance Testing
 - Here testing focuses on the behavior of the system and not the internal logic.
 - Mostly, functional testing is performed at these levels.

Testing Fundamentals



- Testing Process – Levels of Testing
 - Regression Testing
 - This testing is performed when some changes are made to an existing system.
 - It ensures the desired behavior of new services as well as the desired behavior of old services.

Testing Fundamentals



- Testing Process – Test Plan
 - *It is a general document for entire project that defines the scope, approach to be taken and the schedule of the testing and identifies the test items for the entire testing process and the personnel responsible for the different testing activities.*
 - Testing commences with test plan and terminates with acceptance testing.
 - Test plan can be defined before testing commences and can be done in parallel with design and coding activities.

Testing Fundamentals



- Testing Process – Test Plan
 - Inputs for test plan are: *project plan, requirements document* and *system design document*.
 - A test plan should contain:
 - Test Unit Specification: Module or few modules or entire system
 - Features to be tested: All software features or combination of features
 - Approach for testing: Generally, testing criterion for evaluating the test cases.

Testing Fundamentals



- Testing Process – Test Plan
 - A test plan should contain:
 - Test Deliverables: It can be list of test cases, detailed results including list of defects found, test summary reports, etc.
 - Schedule and Task Allocation: If test plan is separate document from project management plan, then it specifies the detailed schedule and task allocation to each testing activities.

References

- Pankaj Jalote, "An Integrated Approach to Software Engineering", Narosa Publishing House, 2nd Edition
- <https://www.geeksforgeeks.org/coding-standards-and-guidelines/>
- <https://www.softwaretestingmaterial.com/software-testing/>





Module 4

Introduction and Database Modeling using ER Model - Part 1

Mr. Swapnil S Sontakke (Asst. Prof.)

Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli



Content

- Introduction to Database Systems
- Advantages and Applications of Database Systems
- Data Models
- Data Abstraction
- Database Schema and Instance
- Database Languages
- Database System Architecture
- Database Users and Administrator
- Entity-Relationship Model

Introduction to Database Systems



➤ What is Data?

“facts and statistics collected together for reference or analysis.”

Or

“the quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.”

- Oxford Dictionary



Introduction to Database Systems

➤ What is Data?

- *Data is a collection of facts, such as numbers, words, measurements, observations or just descriptions of things which in turn may be qualitative or quantitative.*
- Data can be in many formats such as numbers, text, images, audio, video, etc.
- Data is used to produce the information.

Introduction to Database Systems



➤ What is Database?

- *Database is a collection of related data stored electronically in a organized manner and used to insert, retrieve, update and delete the data efficiently.*
- Data in the database is organized in the form of tables, views, schemas, reports, etc.
- E.g. College database will have data related to students, faculty, departments, labs, etc. which are interrelated to each other.



Introduction to Database Systems

➤ What is Database Management System (DBMS)?

- *DBMS is an application (software) which is used to manage the databases.*
- The primary goal of a DBMS is to provide a way to store, retrieve, update and delete database information conveniently and efficiently.
- E.g. MySQL, Oracle, MariaDB, PostgreSQL, Microsoft SQL Server, Amazon Aurora, etc.



File Processing Systems

- Initially file based systems were used to store and retrieve the data.
- In this simple files were stored on the electronic devices such as computers.
- There are many issues with this system which are as follows:
- **Data Redundancy**
 - Same data is duplicated in many places (files).
 - E.g. Student Email ID may be present at many sections.
 - This may lead higher storage and access cost.

File Processing Systems



➤ Data Inconsistency

- Multiple copies of same data does not match with each other.
- E.g. If Phone number is different in contact information section and profile section or accounts section, it will be inconsistent.

➤ Difficulty in Accessing Data

- If a set of programs for performing a particular operation is not available then it becomes difficult to access or retrieve those data.
- E.g. Getting a list of students who live in particular district.
- No convenient or efficient way to access the data

File Processing Systems



➤ Data Isolation

- Data are stored in multiple files and files may be in different formats.
- Writing new programs to retrieve the appropriate data is difficult.

➤ Integrity Problems

- There can be many consistency constraints that data must satisfy.
- E.g. Account balance should not fall below Rs. 1000, etc.
- Whenever new constraints are to be added, changes in programs is necessary which is very difficult task.

File Processing Systems



➤ **Atomicity Problems**

- A computer system is subject to failure.
- In case of failure occurs, it is necessary that the data must be restored to the consistent state that existed prior to the failure.
- This means the operations should be atomic i.e. it must happen in its entirety or not at all.
- E.g. Transferring money from one account to another
- Achieving atomicity is difficult in file processing systems.

File Processing Systems



➤ Concurrent Access Anomalies

- For the overall performance and faster response, many systems allow multiple users to access and update the database concurrently.
- Concurrent updates may lead to inconsistency.
- E.g. If an account with ₹10, 000 is debited by two clerks concurrently with ₹500 and ₹800 resp. If two programs run concurrently, they may both read the value ₹10, 000 and write back ₹9500 and ₹9200. But the correct value is ₹8700.
- This should not happen for any of the programs.

File Processing Systems



➤ Security Problems

- Not every user should be given access to all the data.
- E.g. Accounts persons should not have access to personal information, Students should have access to only student portal to see their results and should not be able to change the data, etc.
- It is difficult to provide security in file processing systems.

➤ Backup and Recovery

- File processing systems does not incorporate any backup and recovery mechanism if the data is lost or file is corrupted.

Advantages of Database Systems



- All the issues with the file processing systems are tackled by the Database systems or DBMS.
- They follow ACID properties i.e. **A**tomicity, **C**oncurrency, **I**ntegrity and **D**urability.
- Control data redundancy
- Control concurrent access anomalies
- Provide security, backup and recovery.
- They also support multiple views for different users such as admin, user, manager, etc.
- Easy to maintain

Applications of Database Systems



- Database systems have very vast range of applications in every area.
- Some of the representative applications are mentioned on the next slides.

Applications of Database Systems



➤ Enterprise Information

- **Sales:** For customer, product, and purchase information.
- **Accounting:** For payments, receipts, account balances, assets, and other accounting information.
- **Human resources:** For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.

➤ Manufacturing:

For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.

Applications of Database Systems



➤ Banking and Finance

- **Banking:** For customer information, accounts, loans, and banking transactions.
- **Credit/Debit card transactions:** For purchases on credit cards and generation of monthly statements.
- **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.

Applications of Database Systems



- **Universities:** For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).
- **Airlines:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
- **Telecommunication:** For keeping records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

Applications of Database Systems



➤ Web-based services

- **Social-media:** For keeping records of users, connections between users (such as friend/follows information), posts made by users, rating/like information about posts, etc.
- **Online retailers:** For keeping records of sales data and orders as for any retailer, but also for tracking a user's product views, search terms, etc.,
- **Online advertisements:** For keeping records of click history to enable targeted advertisements, product suggestions, news articles, etc.

Data Models



- Underlying structure of a database is known as data model.
- It is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
- The data models can be classified into four categories:
 - Relational Model
 - Entity-Relationship (E-R) Model
 - Semi-structured Data Model
 - Object-Based Data Model

Data Models



➤ Relational Model

- It uses a collection of tables to represent both data and relationship between those data.
- Each table consists of many columns and each column has unique name.
- Tables are also known as *relations*.
- This model is record-based model with fixed format.
- It is one of the widely used model.

Data Models



➤ Entity-Relationship (E-R) Model

- It graphically represents the objects and their relationships.
- It uses a collection of basic objects called as *entities* and *relationships* among these objects.
- An entity is a thing or object in the real world that distinguishable from other objects.
- The E-R model is widely used in the database design.



Data Models

➤ **Semi-structured Model**

- It permits the specification of data where individual data items of the same type may have different set of attributes.
- It is different from previous data models.
- XML and JSON are widely used semi-structured models.

➤ **Object-Based Data Model**

- It is a object-oriented data model.
- It can be seen extending relational model with object oriented concepts such as encapsulation, object identity, methods, etc.



Data Abstraction

- For the system to be usable, it must retrieve data efficiently.
- To achieve the efficiency, the database developers use complex data structures to represent data in the database.
- As many database-system users are not trained, complexity needs to be hidden from them.
- Developers achieve this using several levels of ***data abstraction***.
- *“Data abstraction is the process of hiding the background implementation details and providing only essential information to the users (outside world).”*
- Data abstraction simplifies the users' interactions with the system.



Data Abstraction

- There are three levels of abstraction in the database system: Physical Level, Logical Level and View Level
- **Physical Level**
 - Lowest level of abstraction
 - Describes *how* the data are actually stored
 - Describes the complex low-level data structures in detail
 - E.g. data stored can be described as block of storage (bytes, gigabytes), type of files, indexes in memory
 - Users: Programmers/Developers



Data Abstraction

➤ Logical Level

- Next higher level of abstraction
- Describes *what* data are stored in the database and what relationships exists among those data
- Describes the entire database in terms of small number of relatively simple structures
- It may still involve complex physical-level structures, but complexity is kept hidden from its users



Data Abstraction

➤ Logical Level

- i.e. It hides the physical level and gives the logical view of the database.
- This is called as *Physical Data Independence*.
- Users: Database Administrator
- E.g.

Department	Budget
Civil	500000
Electrical	450000
Electronics	450000
Mechanical	500000



Data Abstraction

➤ View Level

- Highest level of abstraction
- Describes only part of the entire database
- It may still involve complexity, but complexity is kept hidden from its users
- Users don't need access to entire database; they need only part of it.
- The view level abstraction simplifies the interaction of end users with the system by providing many views of the same database.



Data Abstraction

➤ View Level

- Users: authentic end users
- E.g.
 - Checking only bank balance
 - Checking last 5 transactions
 - Deriving age of using date of birth
 - Employee can see only his/her salary and not the manager's salary, etc.
- Figure shows relationship among three levels of abstraction.

Data Abstraction

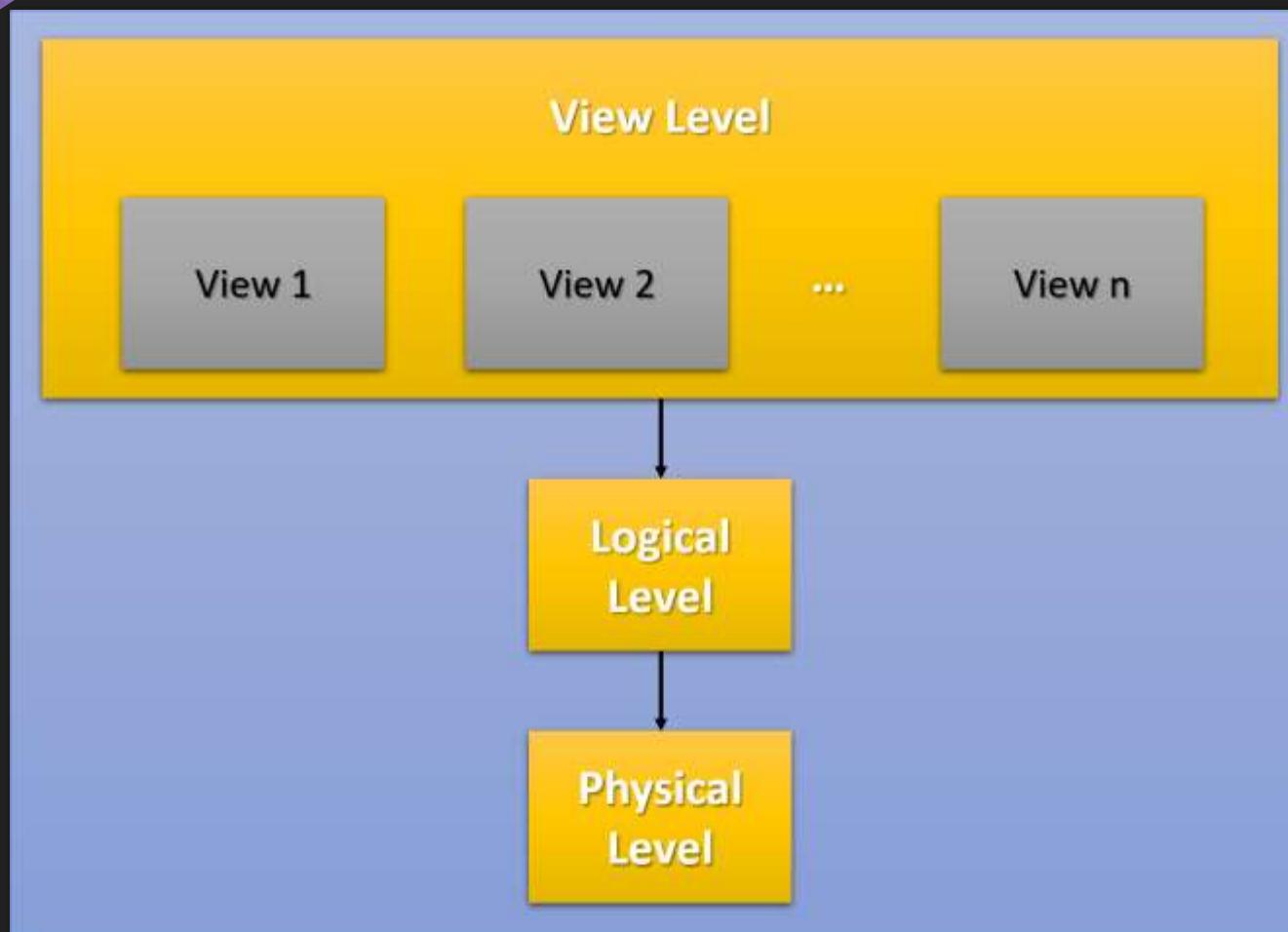


Fig. 1 Three levels of abstraction



Database Schema

- “*The overall design of a database is called as the schema.*”
- It defines a structure in formal language supported by a DBMS.
- It shows the logical view of the entire database.
- It shows the organization of data and relationship among them.
- Various schemas are defined at various abstraction level.
- **Physical Schema**
 - Defines database design at physical level
 - Defines how data is stored on disk storage in terms of files and indices



Database Schema

➤ **Logical Schema**

- Defines all the logical constraints that need to be applied on the data stored
- It defines tables, views and integrity constraints.

➤ **Subschemas**

- A database may also have several schemas at the view level.
- These are called as “subschemas”.
- They describe various views of the database.



Database Schema

```
CREATE TABLE student (
    studentExamNo int(11) NOT NULL,
    studentFullName varchar(70) NOT NULL,
    mobileNo varchar(50) NOT NULL,
    address varchar(50) NOT NULL,
    state varchar(50) DEFAULT NULL,
    country varchar(50) NOT NULL,
    PRIMARY KEY (StudentExamNo),
)
```

Fig. 1 Example of Database Schema in SQL



Database Instance

- Database is not a static, it is dynamic.
- This means it changes over time as new information is inserted, deleted and updated.
- *"The collection of information stored in the database at a particular moment is called as an **instance** of the database."*



Database Languages

- A database system provides two types of languages:
 - Data Definition Language (DDL)
 - Data Manipulation Language (DML)
- In practice, DDL and DML are not separate languages rather they are part of single database languages such as SQL.
- *SQL is employed in almost all relational databases.*



Database Languages

➤ Data Definition Language (DDL)

- *DDL is a language that specifies a database schema by a set of definitions.*
- DDL also specifies additional properties of data.
- The implementation details provided by DDL are in general hidden from the end users. E.g.

```
create table department  
    (dept name char (20),  
     building char (15),  
     budget numeric (12,2));
```



Database Languages

➤ Data Manipulation Language (DML)

- *DML is a language that enables users to manipulate or access the data as organized by the appropriate data model.*
- The types of access are:
 - Insertion of new information into the database
 - Retrieval of information stored in the database
 - Modification of information stored in the database
 - Deletion of information from the database



Database Languages

➤ Data Manipulation Language (DML)

- There are two types of DML: Procedural DML and Non-procedural (Declarative) DML
- Procedural DMLs require a user to specify **what** data are needed and **how** to get those data.
- Declarative DMLs require a user to specify **what** data are needed **without** specifying how to get those data.

Database Languages



➤ Data Manipulation Language (DML)

- A **query** is a statement requesting the retrieval of information.
- The portion of a DML that involves information retrieval is called a **query language**.
- SQL is widely used query language.

Database System Architecture



- The database system is divided into different modules having their set of responsibilities.
- These modules are
 - The storage manager
 - The query processor components and
 - The transaction management component

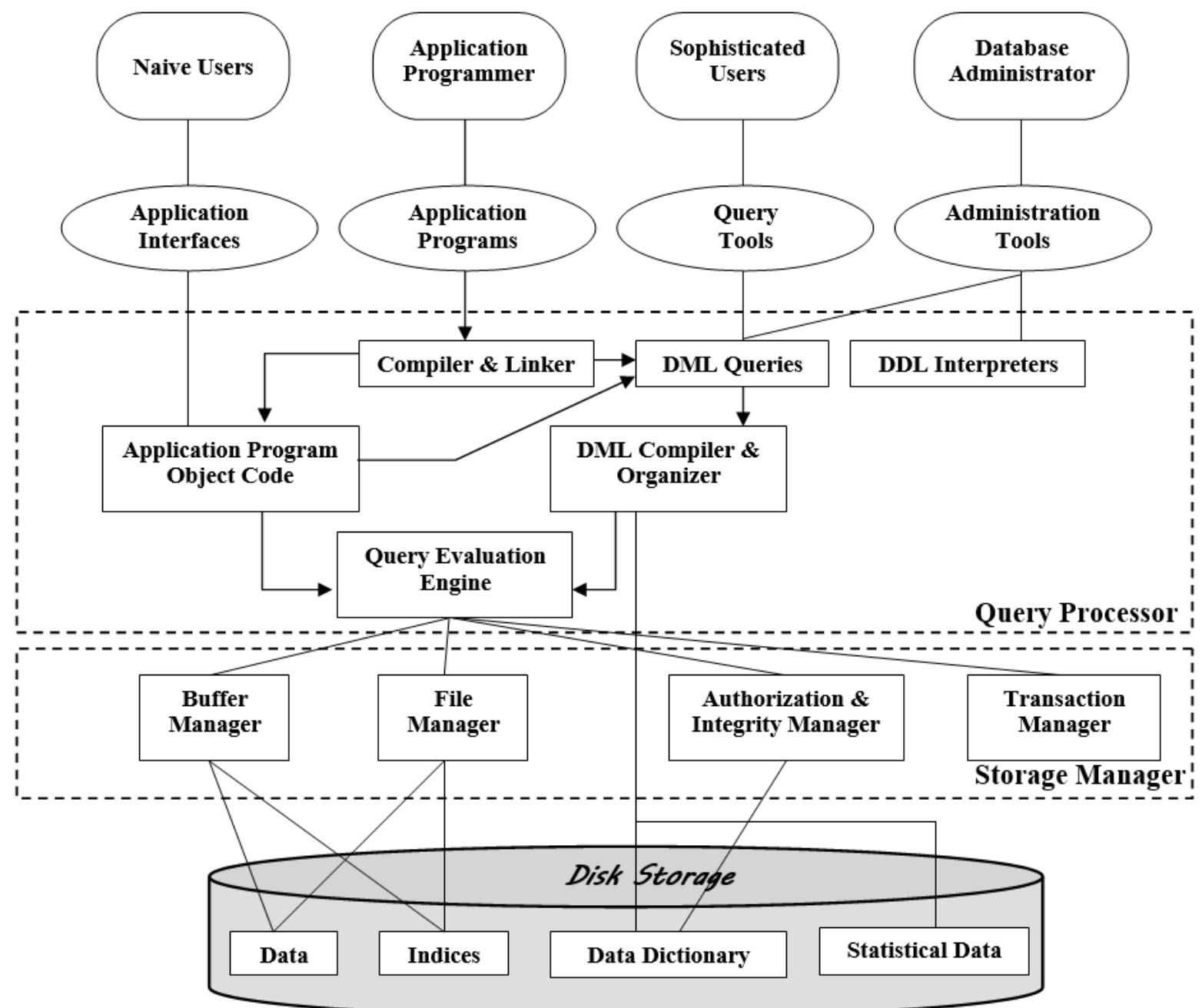


Figure: System Architecture

Database System Architecture



➤ The Storage Manager

- It provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- It is responsible for interaction with the file manager.
- It translates DML statements into low-level file-system commands, hence is responsible for storing, retrieving, updating and deleting the data in the database.

Database System Architecture



➤ The Storage Manager

- The components of storage manager are
 - Authorization and integrity manager
 - Transaction manager
 - File manager
 - Buffer manager
- The data structures implemented by the storage manager are *data files*, *data dictionary* and *indices*.

Database System Architecture



➤ The Query Processor

- It helps the database system to simplify and facilitate access to data.
- It allows database users to achieve a good performance while working at the view level without focusing on physical level details of implementations.
- The components of query processor are
 - DDL Interpreter
 - DML Compiler
 - Query Evaluation Engine

Database System Architecture



➤ The Transaction Manager

- *A transaction is a collection of operations that performs a single logical function in a database application.*
- Each transaction is a unit of both ***atomicity*** and ***consistency***.
- It allows application developers to treat a sequence of database accesses as if they were a single unit that either happens in its entirety or not at all.

Database System Architecture



- While database engines were traditionally ***centralized computer systems***, today ***parallel*** and ***distributed*** databases which run faster and at different locations are widely used.
- Architectures of database applications that use databases can be two tier or three tier.
- In ***two-tier architecture***, the application resides at the client machine, and invokes database system functionality (directly) at the server machine through query language statements.
- E.g. Desktop applications, games, music, etc.



Database System Architecture

- In **three-tier architecture**, the client machine acts as front end and there is no any database calls.
- User uses application clients such as web browsers, mobile applications, etc.
- The front end communicates with the application server (at another location).
- The application server then communicates with a database system to access data.
- The **business logic** of the application, which says what actions to carry out under what conditions, is embedded in the application server.

Database System Architecture



- E.g. Web-based applications, website, mobile applications, etc.

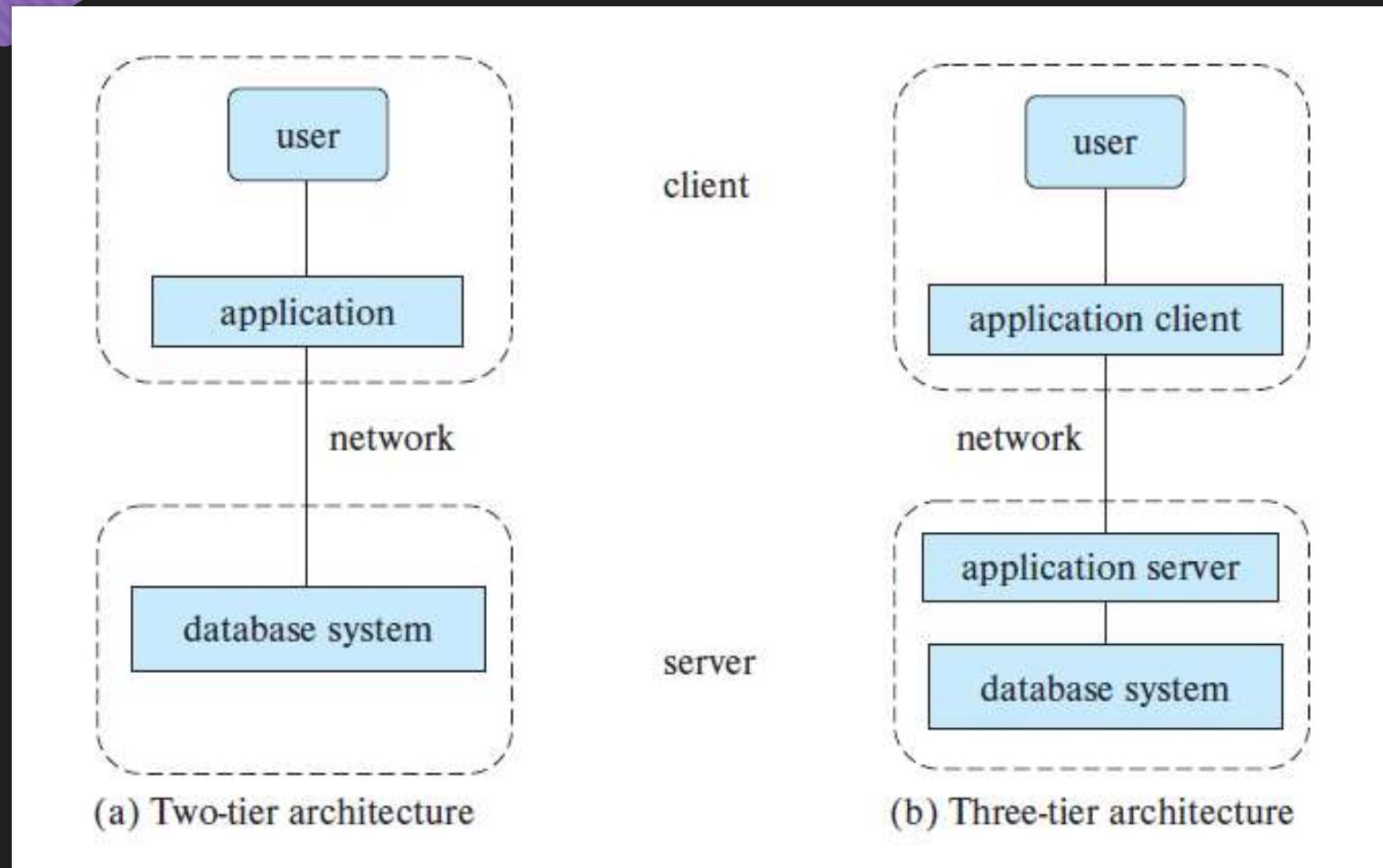


Fig. 3 Client-server 2-tier & 3-tier Architecture



Database Users and Administrator

- People who work with a database can be categorized as *database users* or *database administrators*.
- **Database Users and User Interface**
 - There are different types of database users, differentiated based on how they interact with the system.
 - For each of the user types different ***user interfaces (UI)*** have been designed.
 - These users are Naïve users, Application Programmers and Sophisticated users.

Database Users and Administrator



➤ Naïve users

- *Unsophisticated users* who interact with the system by using predefined user interfaces, such as web or mobile applications
- E.g. end users using any mobile or desktop applications or website users

➤ Application Programmers

- *Computer professionals* who write application programs.
- Application programmers can choose from many tools to develop user interfaces.
- E.g. Programmers/Developers

Database Users and Administrator



➤ Sophisticated users

- Sophisticated users interact with the system without writing programs.
- They form their requests either using a database query language or by using tools such as data analysis software.
- E.g. Analysts who submit queries to explore data in the database



Database Users and Administrator

➤ Database Administrator

- A person who has *central control over the system (both data and programs to access those data)* is called a database administrator (DBA).
- The functions of a DBA include:
 - Schema definition
 - Storage structure and access-method definition
 - Schema and physical-organization modification
 - Granting of authorization for data access
 - Routine maintenance such as backup, disk space requirements, etc.



Entity-Relationship Model

- **The Entity-Relationship (E-R) Model** is developed to design the structure of a database with the help of a diagram known as **The Entity-Relationship (E-R) diagram.**
- ER diagram expresses the overall logical structure of a database graphically.
- Here we discuss the main components of E-R model.
 - Entity Set
 - Relationship Set

Entity-Relationship Model



➤ Entity Set

- *An entity is a “thing” or “object” in the real word that is distinguishable from all other objects.*
- E.g. every student in a college is an entity, course in a college, a flight reservation, bank management system, etc.
- An entity has a set of properties, and the values for some set of properties must uniquely identify an entity.
- E.g. a student may have *examSeatNo* whose value uniquely identifies that person.

Entity-Relationship Model



➤ Entity Set

- *An entity set is a set of entities of same type that share the same properties, or attributes.*
- E.g. *student* can be an entity set that represents all students in the college, *teacher* can be an entity set that represents all teachers in the college, etc.
- An entity set do not need to be disjoint.
- E.g. *person* entity may represent a *student* entity, *teacher* entity, both or neither.

Entity-Relationship Model



➤ Weak Entity Set

- A **weak entity set** is an entity set which cannot be identified by its own attributes and whose existence is dependent on another entity set.
- The supporting entity set is called **identifying entity set** of weak entity set.
- An entity set that is not weak entity set is called as **strong entity set**.



Entity-Relationship Model

➤ Weak Entity Set

- E.g.
- *Room* is a weak entity set which cannot exist without a *building*
- *Bank account* cannot be uniquely identified without associating it to a particular *bank*
- *Module/Chapter* is a weak entity set which needs *book, paper*
- *Insurance Policy* requires an *employee*
- *Library* is a strong entity set as it can exist without *college/university*



Entity-Relationship Model

➤ **Attributes**

- An entity is represented by a set of attributes.
- **Attributes** are descriptive properties possessed by each member of an entity set.
- The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set.
- However, each entity may have its own value for each attribute.
- E.g. *student* can have *examSeatNo*, *name*, *dept*, *class*, etc. attributes

Entity-Relationship Model



➤ Value

- Each entity has a ***value*** for each of its attributes.
- E.g. *student* entity can have value 2050BTECS00001 for *examSeatNo*, value Ram for *name*, value AI & ML for *dept*, and value First Year for *class*.

➤ Primary Key

Entity-Relationship Model



➤ Types of Attributes

- An attribute in E-R model can be characterized by following types:
 - Simple and Composite
 - Single-valued and Multivalued
 - Derived



Entity-Relationship Model

➤ Types of Attributes

- **Simple and Composite**

- *An attribute which cannot be further subdivided into component attributes is called as simple attribute.*
- E.g. examSeatNo, employeeID, ISBN, accountNo, etc.
- *An attribute which can be further subdivided into component attributes is called as composite attribute.*
- E.g. address (houseNo, streetNo, city, state, country), name(first, middle, last), phoneNo(STDCode, Number), etc.



Entity-Relationship Model

➤ Types of Attributes

- **Single-valued and Multivalued**

- *An attribute which takes up only single value for a particular entity is called as single-valued attribute.*
- E.g. examSeatNo, age, height, weight, etc.
- *An attribute which takes up two or more values for a particular entity is called as multivalued instance.*
- E.g. mobileNo, emailID, address, course, hobby, etc.



Entity-Relationship Model

➤ Types of Attributes

- **Derived**

- *An attribute whose value can be derived from other related attributes or entities is called as derived attribute.*
- E.g. totalMarks, averageMarks, age, BMI, etc.

Entity-Relationship Model



➤ Relationship Set

- An association among several entities is called as ***relationship***.
- E.g. Mr. R. Sharma is a ***coach*** of Mr. Virat Kohli
- ***Relationship set*** is a set of relationships of the same type.
- E.g. we define relationship set ***coach*** to denote the association between ***instructor*** and ***player***.

Entity-Relationship Model



1	Ramakant Achrekar
2	Mr. R. Sharma

P1	Mr. Sachin Tendulkar
P2	Mr. Virat Kohli
P3	Mr. A. B. C
P4	Mr. X. Y. Z.



Entity-Relationship Model

➤ Degree of Relationship Set

- It is number of entity sets participating in a relationship set.
- There are following degrees exist:
- **Unary Relationship**
 - When there is only ONE entity set participating in a relation, the relationship is called as unary relationship.
 - For example, one **Person** is married to only one **Person**.
 - The degree of unary relationship is 1.



Entity-Relationship Model

➤ Degree of Relationship Set

- **Binary Relationship**

- When there TWO entity sets participating in a relation, the relationship is called as binary relationship.
- For example, ***Student*** is enrolled in ***Course***.
- The degree of binary relationship is 2.



Entity-Relationship Model

➤ Degree of Relationship Set

- **Ternary Relationship**

- When there THREE entity sets participating in a relation, the relationship is called as ternary relationship.
- For example, ***Student*** is working on a ***Project*** under the guidance of ***Guide***.
- The degree of ternary relationship is 3.

- **N-ary Relationship**

- When there ***n*** entity sets participating in a relation, the relationship is called as n-ary relationship.
- The degree of n-ary relationship is ***n***.

Entity-Relationship Model



➤ Mapping Cardinality/Cardinality Ratios

- *It expresses the number of entities to which another entity can be associated via a relationship set.*
- *The number of times an entity of an entity set participates in a relationship set is known as cardinality.*
- For a binary relationship set R between entity sets A and B, the mapping cardinality are as follows:



Entity-Relationship Model

➤ Mapping Cardinality/Cardinality Ratios

- For a binary relationship set R between entity sets A and B, the mapping cardinality are as follows:
- **One-to-One:**
 - An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.
 - The cardinality is one to one.



Entity-Relationship Model

➤ Mapping Cardinality/Cardinality Ratios

- **One-to-Many:**

- An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.
 - The cardinality is one to many.

- **Many-to-One:**

- An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.
 - The cardinality is many to one.



Entity-Relationship Model

➤ Mapping Cardinality/Cardinality Ratios

- **Many-to-Many:**

- An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.
- The cardinality is many to many.



Entity-Relationship Model

➤ Participation Constraints

- **Total Participation**

- The participation of an entity set E in a relationship set R is said to be ***total*** if every entity in E participates in at least one relationship in R.

- **Partial Participation**

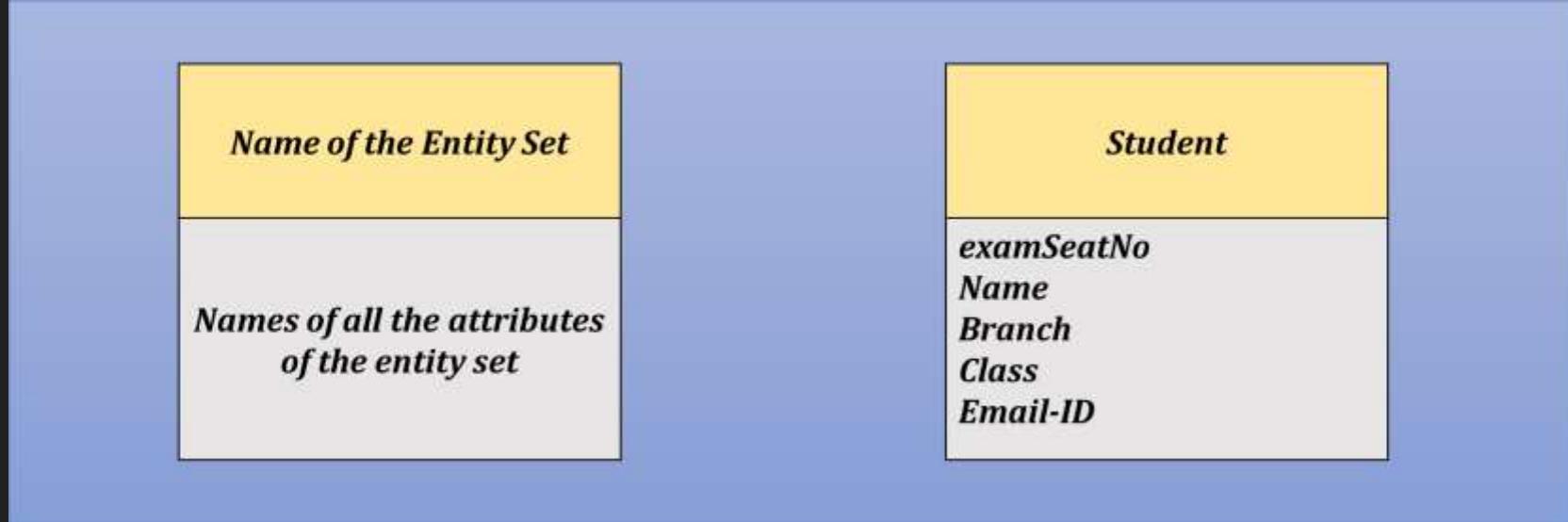
- If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be ***partial***.



Entity-Relationship Model

➤ E-R Diagrams – Basic Structure

- An E-R diagram consists of following components:
- **Rectangles divided into two parts:** Represent **entity sets**

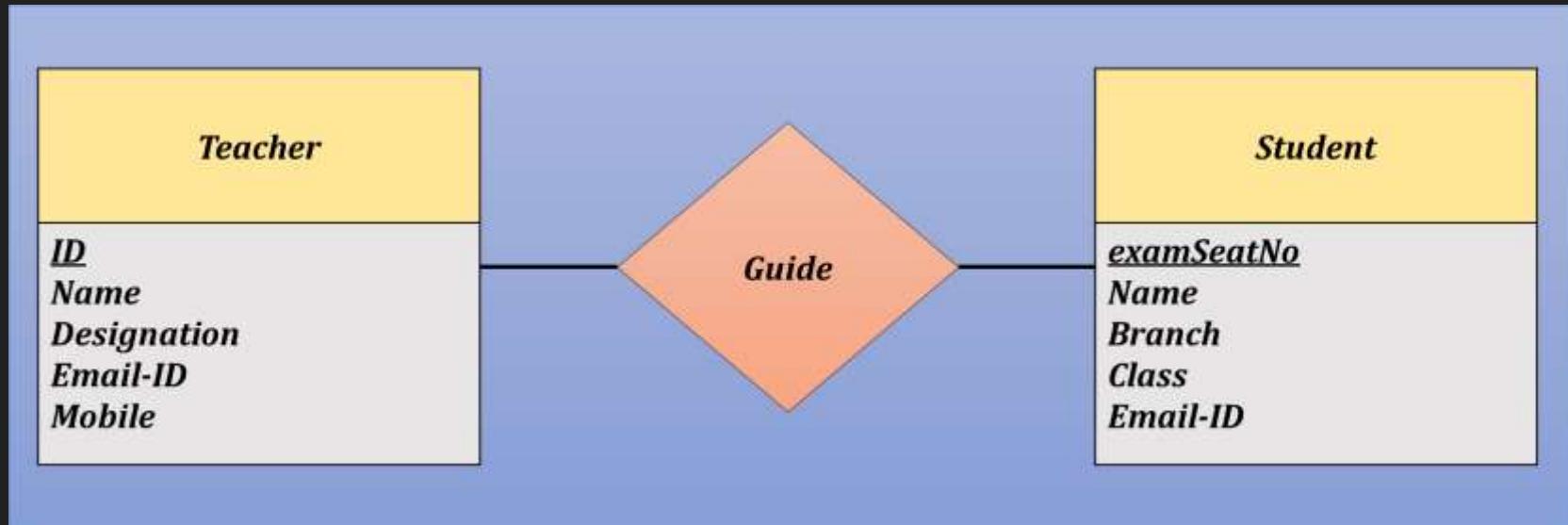




Entity-Relationship Model

➤ E-R Diagrams - Basic Structure

- **Lines:** link entity sets to relationship sets.
- **Diamonds:** Represent **Relationship sets**

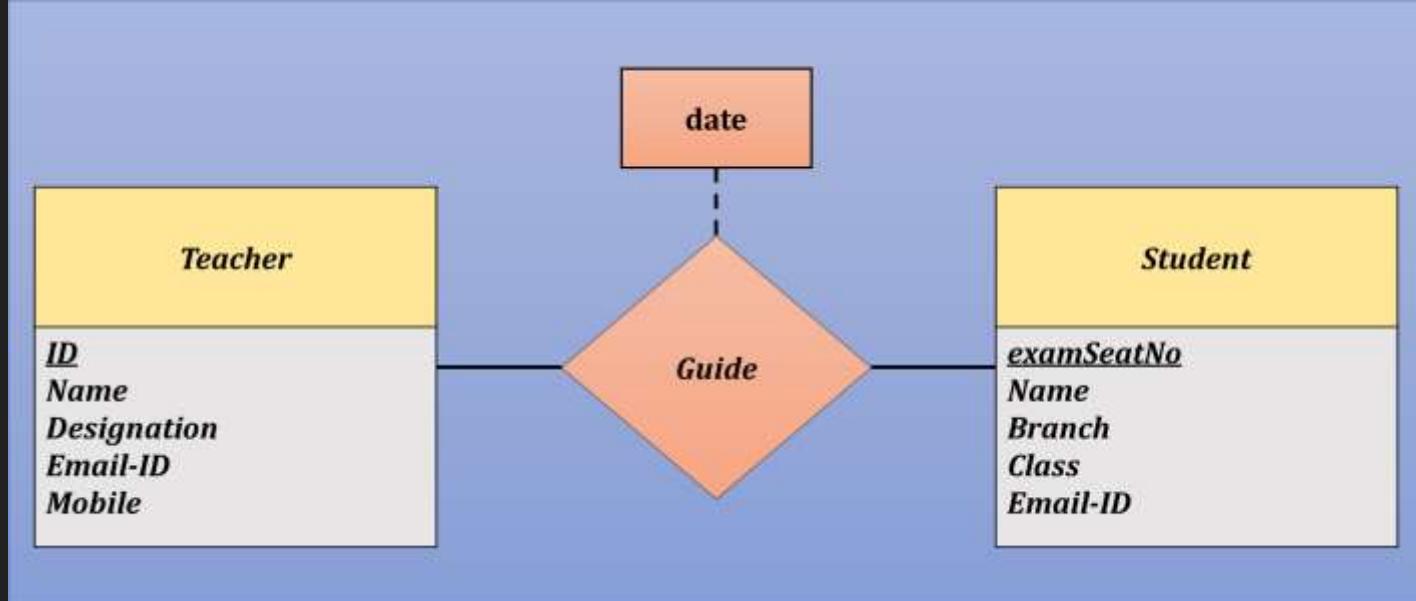




Entity-Relationship Model

➤ E-R Diagrams – Basic Structure

- **Undivided rectangles:** Represent **the attributes of a relationship set.**
- **Dashed lines** link attributes of a relationship set to the relationship set.

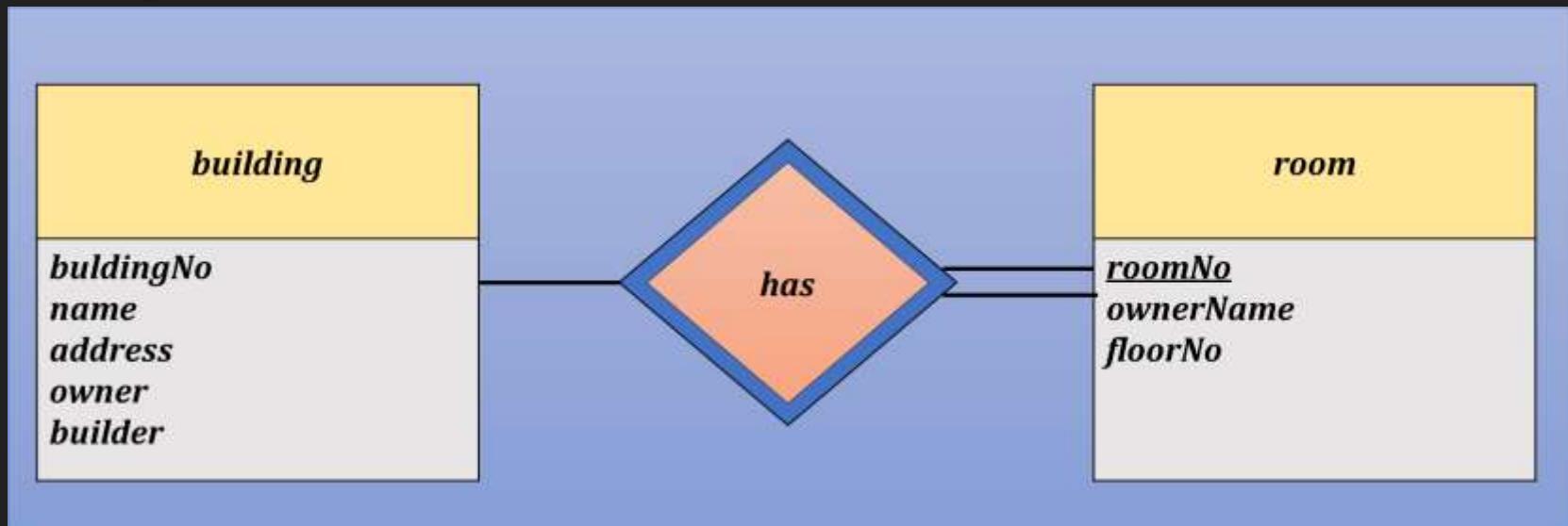




Entity-Relationship Model

➤ E-R Diagrams - Basic Structure

- **Double lines** indicate total participation of an entity in a relationship set.
- **Double diamonds** represent identifying relationship sets linked to weak entity sets





Entity-Relationship Model

➤ E-R Diagrams – Basic Structure

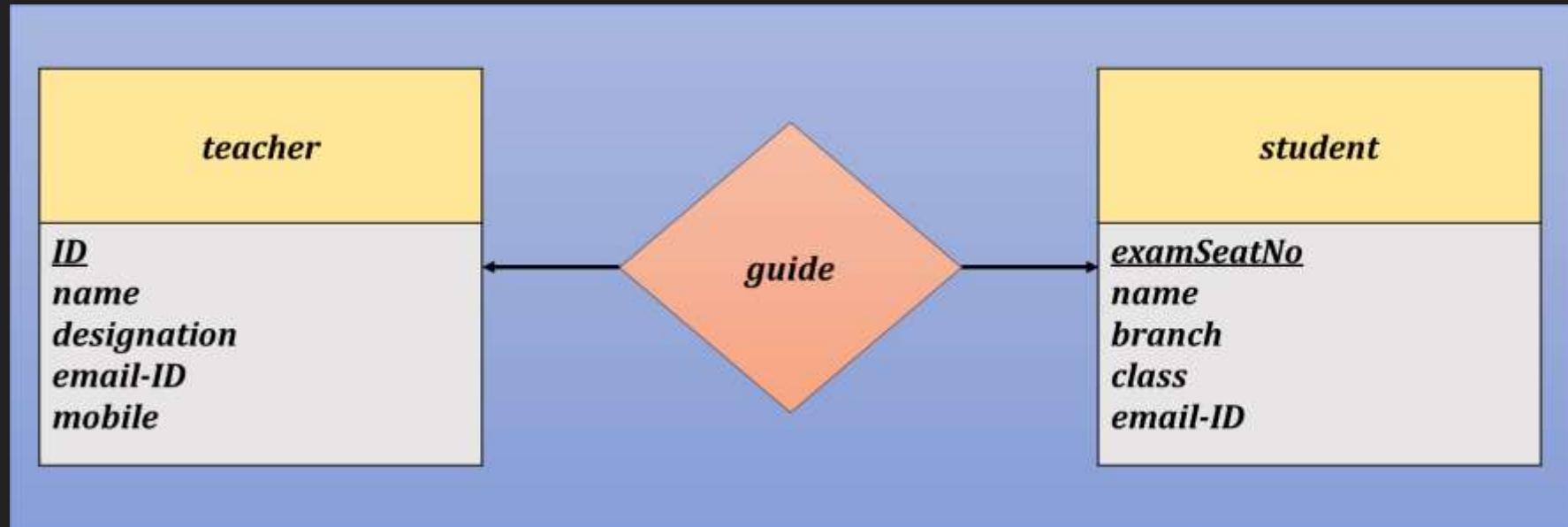
- **Double rectangle** represent weak entity
- **Primary key attribute** is always underlined. It helps identify each of the member in an entity uniquely.



Entity-Relationship Model

➤ E-R Diagrams – Basic Structure

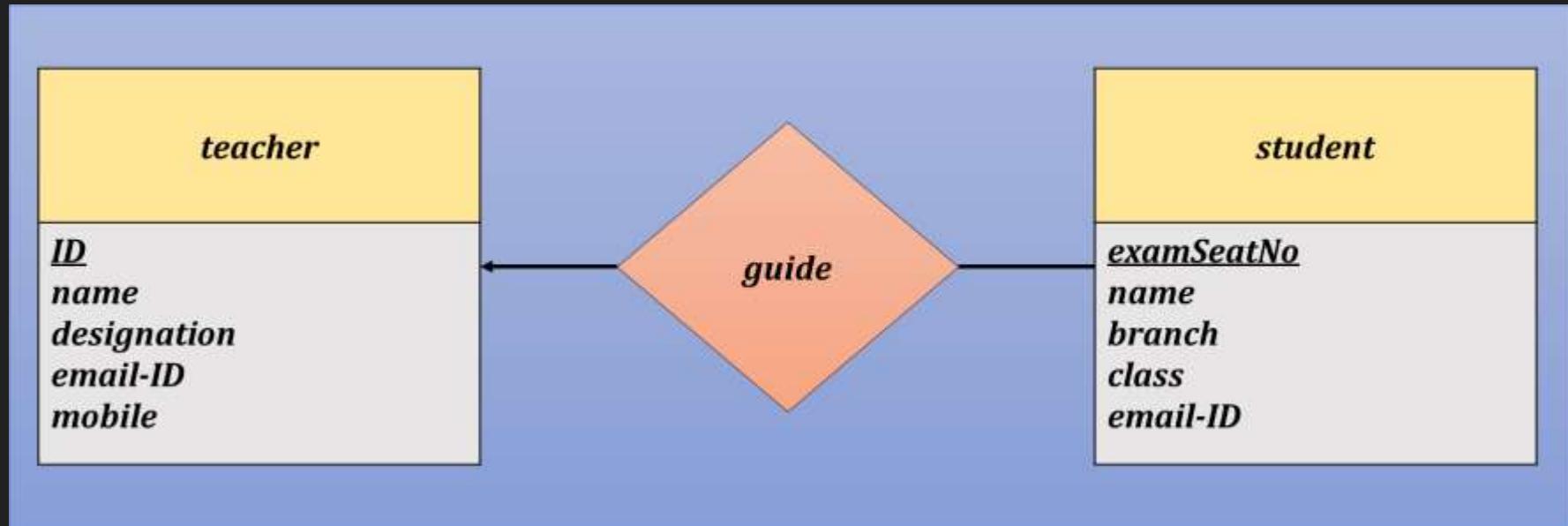
- One-to-One Relationship





Entity-Relationship Model

- E-R Diagrams – Basic Structure
 - One-to-Many Relationship

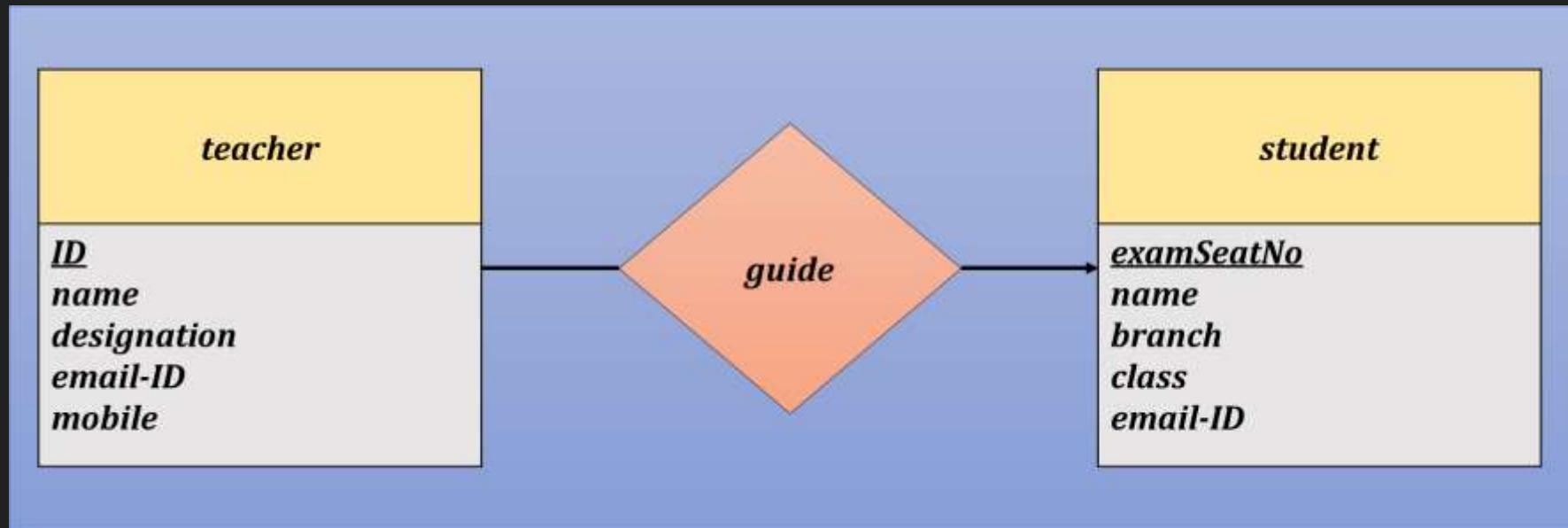




Entity-Relationship Model

➤ E-R Diagrams - Basic Structure

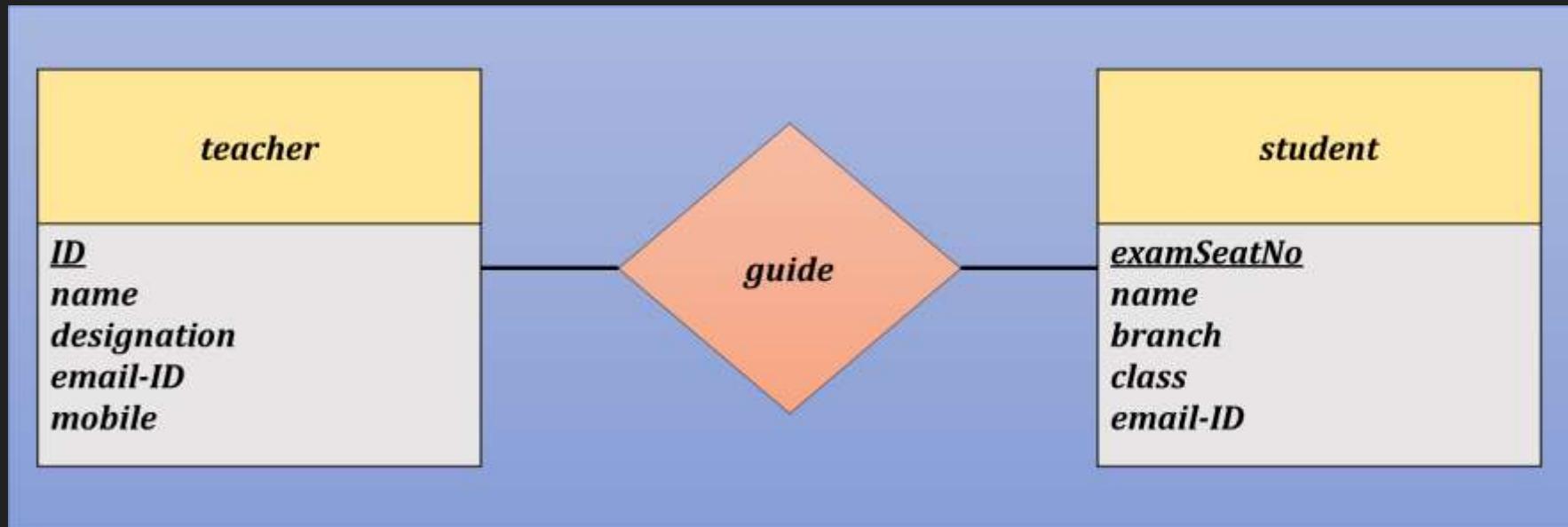
- Many-to-One Relationship





Entity-Relationship Model

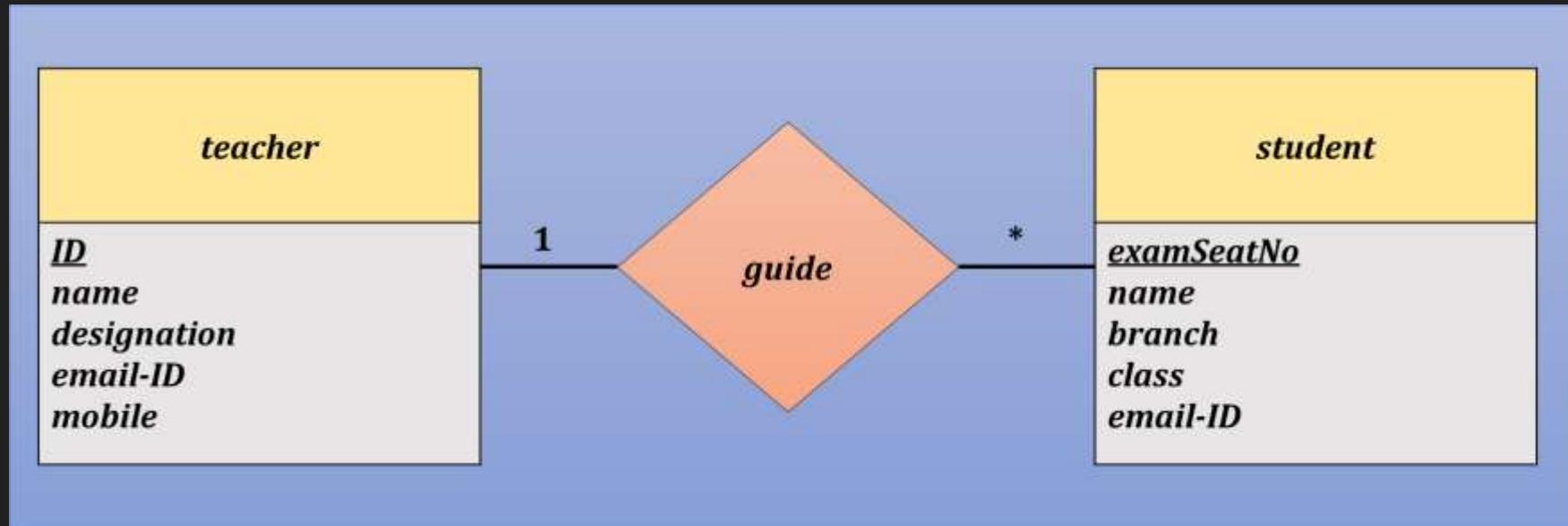
- E-R Diagrams – Basic Structure
 - Many-to-Many Relationship

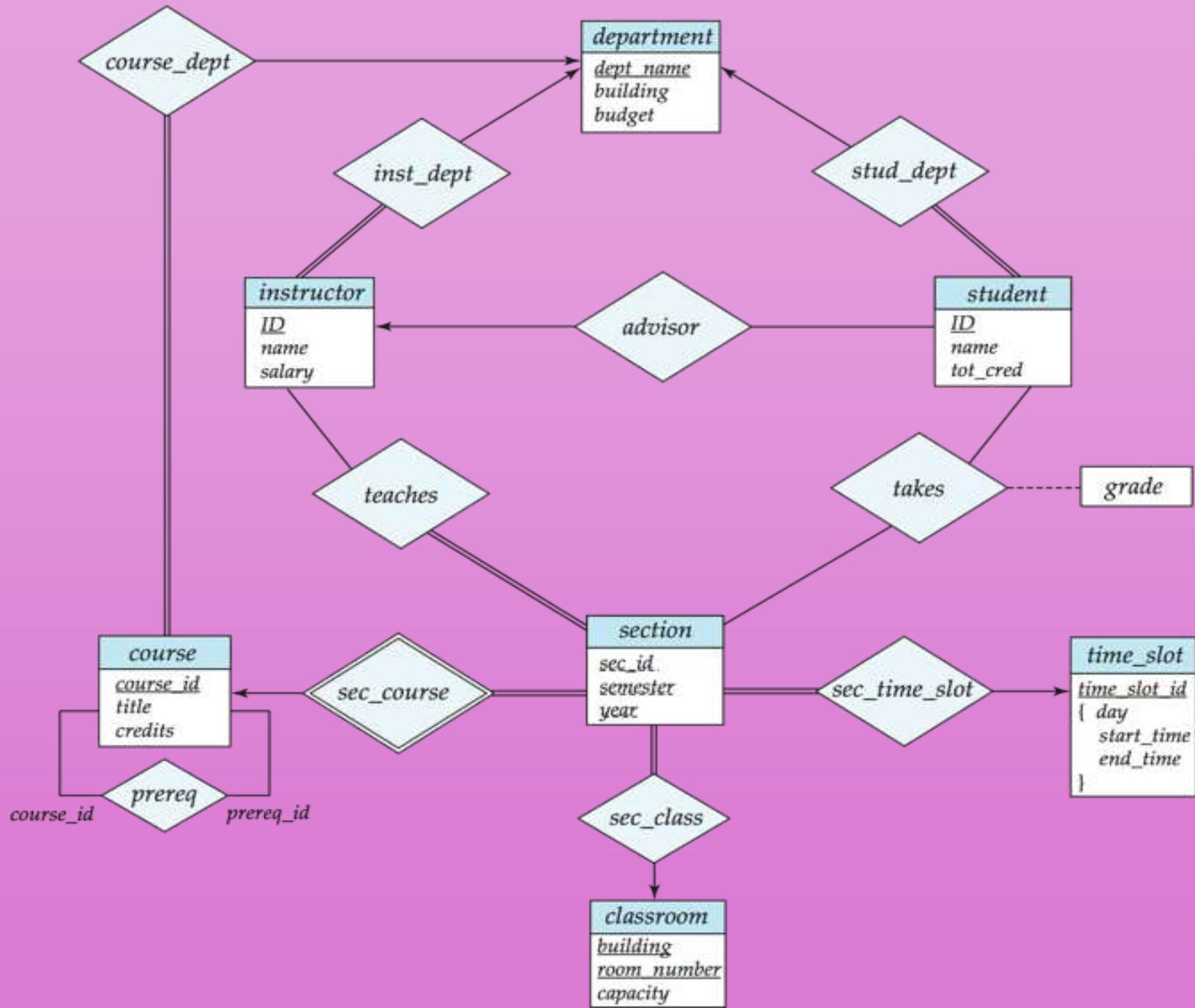




Entity-Relationship Model

- E-R Diagrams – Basic Structure
 - Mapping Cardinality Limit (Another way)







References

- Abraham Silberschatz, Henry F. Korth and S. Sudarshan, “Database System Concepts”, Mc-Graw Hill, 7th Edition.



Module 5

Relational Model and SQL - Part 1

Mr. Swapnil S Sontakke (Asst. Prof.)

Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli

Content



- Introduction to Relational Model
- Structure of Relational Database
- Keys
- Relational Algebra



Introduction to Relational Model

- Proposed by Dr. E. F. Codd in 1970
- Received The Turing Award in 1981
- Relational model forms the basis for relational databases i.e. it represents how data is stored in the relational databases.
- In relational model all data is represented in terms of **tuples** and grouped into **relations**.
- A database organized in terms of relational model is **relational database**.



Introduction to Relational Model

- After designing the conceptual model of Database using ER diagram, we need to convert it in the relational model which can be implemented using any RDMBS languages like Oracle SQL, MySQL, etc.
- It is one of the primary data model used for commercial data processing applications because of its simplicity.



Structure of Relational Databases

- A database organized in terms of relational model is ***relational database***.
- **Relations**
 - A relational database consists of a collection of **relations** each having a unique name.
 - Relations are also known as tables.
 - Data is stored in rows and columns.



Structure of Relational Databases

➤ **Tuple**

- Each row in the table is known as a tuple.
- It represents data related to an entity or object.
- It represents the relationship between the values.

➤ **Attribute**

- Each column in the table represents the attribute.
- Each column name must have unique attribute name.



Structure of Relational Databases

exam_seat_no	name	Department	mobile
2050BTECS001	Ram	Computer Science	1111111111
2050BTEEN001	Shyam	Electronics	2222222222
2050BTEEL001	Ganesh	Electrical	3333333333
2050BTECV001	Ravi	Civil	4444444444
2050BTEME001	Mahesh	Mechanical	5555555555

Table 1. Student relation (table)



Structure of Relational Databases

➤ Domain

- For each column/attribute of table, there is a set of permitted values called as domain of that attribute.
- E.g. For the attribute *exam_seat_no*, domain is set of all exam seat numbers, for *name*, it is a set of all possible names.
- A domain is said to be ***atomic*** if elements of the domain are considered to be indivisible units.
- E.g. *phone_number*: it can have multiple values and even its single value can be sub divided (into *country code*, *area code*, and *local number*)

Structure of Relational Databases



➤ Domain

- **null value** is a special values that signifies that the value is unknown or does not exist.
- E.g. It may be possible that value for *phone_number* for *student* relation may not exist or is unlisted.



Structure of Relational Databases

➤ Database Schema

- It is a logical design of a database.
- In general, it is physical level, logical level and view level.

➤ Database Instance

- It is a snapshot of the data in the database at a given instant in time.



Structure of Relational Databases

➤ Relation Schema

- It is a programming language notion of type definition.
- It consists of a list of attributes and their corresponding domains.
- In general, it does not change with time.

➤ Relation Instance

- It is a programming language notion of a value of a variable.
- It refers to a specific instance of a relation that contains a specific set of rows.
- They do not contain duplicate values.
- In general, it changes with time as the relation is updated.



Structure of Relational Databases

```
student (
    studentExamNo int(11) NOT NULL,
    studentFullName varchar(70) NOT NULL,
    mobileNo varchar(50) NOT NULL,
    address varchar(50) NOT NULL,
    state varchar(50) DEFAULT NULL,
    country varchar(50) NOT NULL,
    PRIMARY KEY (StudentExamNo),
)
```

Fig. 1. Sample Relation Schema

Structure of Relational Databases



➤ Relation Instance

- E.g.

student (st_id, st_nm, st_email, st_dept, st_course)

teacher (tr_id, tr_nm, tr_email, tr_dept, tr_course)

teaches (tr_id, course_id, dept, sem, year)

Keys



- A relation consists of large number of tuples.
- Each tuple has a set of values for given set of attributes/columns.
- To uniquely identify every tuple, attribute values should be uniquely specified.
- These attributes that are used to uniquely identify tuples in the relation are called as *keys*.
- There are following types of keys: Superkey, candidate key, primary key and foreign key



Keys

➤ Superkey

- *A superkey is a set of one or more attributes that, taken collectively, allows us to identify uniquely a tuple in a relation.*
- E.g. In STUDENT (*examSeatNo, stName, stMobile, stBranch*) relation, an attribute *examSeatNo* is a superkey as it is sufficient to identify one student record (tuple) from another.

However, *stName* or *stBranch* cannot be a superkey as values for both the attributes may be same.

{*examSeatNo, stName*} or {*examSeatNo, stMobile*}, {*examSeatNo, stBranch*} are also superkeys for STUDENT relation



Keys

➤ Candidate key

- *A candidate key is a minimal set of attributes which can uniquely identify a tuple in a relation.*
- In general, superkey may contain extraneous attributes like *stName* in $\{examSeatNo, stName\}$
- If K is a superkey, any of its superset is also a superkey.
- But, *candidate key is a minimal superkeys for which no proper subset is a superkey.*

Keys



➤ Candidate key

- E.g. In STUDENT (*examSeatNo, stName, stMobile, stBranch*) relation.

Some Superkeys are: {*examSeatNo*}, {*examSeatNo, stName*}, {*examSeatNo, stMobile*} and {*examSeatNo, stBranch*}.

{*stName, stMobile*} is also sufficient to uniquely identify the tuple in STUDENT.

So, Candidate keys are: {*examSeatNo*} and {*stName, stMobile*}.

Remaining keys are only superkeys and not candidate keys as they contain {*examSeatNo*} which alone is a candidate key.



Keys

➤ Candidate key

- E.g. In EMPLOYEE (*employeeID*, *empName*, *empSalary*, *empPAN*, *empAadharNo*) relation.

So, Candidate keys are: {*employeeID*}, {*empPAN*} and {*empAadharNo*}.

All other combinations of attributes may form superkeys such as {*employeeID*, *empName*}, {*empName*, *empPAN*}, etc.

Keys



➤ Points to remember

- A superkey is a superset of candidate keys, but vice versa is not true.
- There can be one or more superkeys in a relation.
- There can be one or more candidate keys in a relation.
- Superkey and candidate key cannot be null.
- Adding 0 or more attributes in a candidate key generates superkey.
- Superkey and candidate can be simple or composite.



Keys

➤ Primary key

- *A primary key is a candidate key that is chosen by the database designer as a principal means of identifying tuples in within a relation.*
- There can be **only one** primary key for a relation (chosen out of many candidate keys).
- *The candidate key other than primary key is called as alternate key.*
- E.g. In STUDENT relation, *examSeatNo* is a primary key.
In EMPLOYEE relation, *employeeID* is a primary key.
- Primary key cannot be **null** and **duplicate**.

Keys

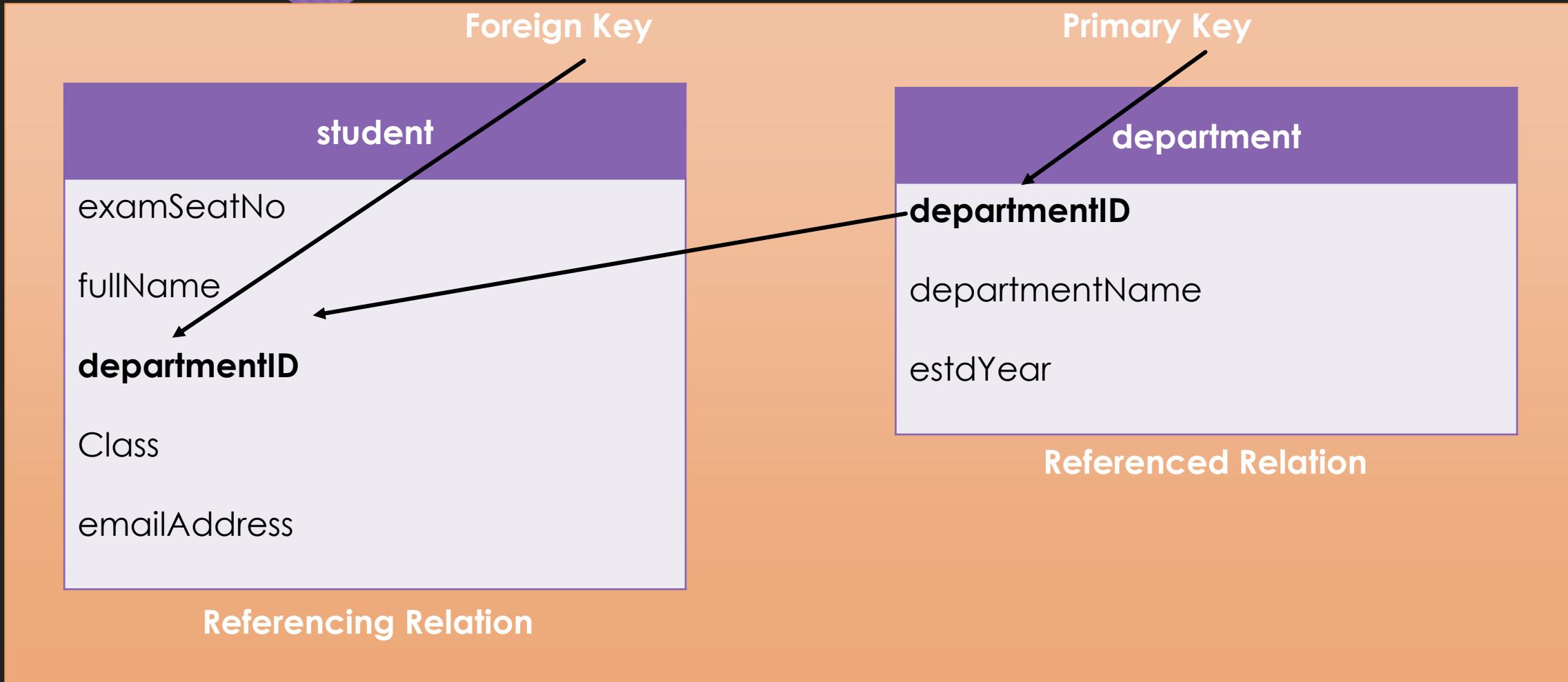


➤ Foreign key

- A *foreign-key constraint* from attribute(s) A of relation r_1 to the primary-key B of relation r_2 states that on any database instance, the value of A for each tuple in r_1 must also be the value of B for some tuple in r_2 .
- Attribute set A is called a *foreign key* from r_1 , referencing r_2 .
- The relation r_1 is also called the *referencing relation* of the foreign-key constraint, and r_2 is called the *referenced relation*.
- In simple words, foreign keys are the column of one table which is used to point to the primary key of another table.



Keys



Keys



➤ Points to Remember

- Primary key cannot be **null** and **duplicate**.
- A foreign key can be null as well as may contain duplicate values.
- Primary keys are also referred to as *primary key constraints*.
- Primary key attributes are underlined.
- A *referential integrity constraint* requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.



Relational Query Languages

➤ **Query Language**

- *It is a language in which a user requests information from the database.*
- There are three types of query languages:
 - Imperative
 - Functional
 - Declarative

Relational Query Languages



➤ Imperative Query Language

- In imperative query language, the user instructs the system to perform a specific sequence of operations on the database to compute the desired result.
- It uses the state variables, which are updated in the course of the computation.
- Requires deep technical and language knowledge.
- They are not user friendly and may prone to human error.
- E.g. No pure imperative query languages, Gremlin and JavaAPI (for Neo4j)

Relational Query Languages



➤ Functional Query Language

- In functional query language, the computation is expressed as the evaluation of functions that may operate on data in the database or on the results of other functions.
- The functions are side-effect free, and they do not update the program state.
- E.g. Relational Algebra which forms the basis for SQL language

Relational Query Languages



➤ Declarative Query Language

- In declarative query language, the user describes the desired information without giving a specific sequence of steps or function calls for obtaining that information.
- The desired information is typically described using some form of mathematical logic.
- It is the job of the database system to figure out how to obtain the desired information.
- E.g. Tuple relational calculus, domain relational calculus and SQL

Relational Query Languages



➤ Declarative Query Language

- SQL actually includes the elements of imperative, functional and declarative approaches.



Relational Algebra

- The relational algebra consists of a set of operations that take one or two relations as inputs and produce a new relation as their result.
- The operations can be either unary or binary.
- Unary operations are those which operate on only one relation to produce a desired result.
- Select, project and rename operations are unary operations.
- Binary operations are those which operate on a pair of relations to produce a desired result.
- Union, Cartesian-Product and set difference are binary operations.

Relational Algebra



➤ The Select operation

- This is a unary operation used *select tuples* that satisfy the given predicate.
- It is denoted by lowercase Greek letter *sigma* (σ).
- The predicate appears as a subscript to σ and the argument relation is in the parenthesis after the σ .
- In general, comparisons are allowed using $=$, \neq , $<$, \leq , $>$ and \geq .
- Several predicates can be combined into larger one using *and*(\wedge), *or*(\vee) and *not* (\neg) operators.



Relational Algebra

➤ The Select operation

- E.g. 1. Select all the tuples of *instructor relation* where instructor is in Physics department.

$$\sigma_{\text{dept_name}=\text{"Physics"}} (\textit{instructor})$$

- E.g. 2. Select all the tuples of instructor relation where instructor has salary greater than 90000

$$\sigma_{\text{salary}>90000} (\textit{instructor})$$



Relational Algebra

➤ The Select operation

- E.g. 3. Select all the tuples of *instructor relation* where instructor is in Physics department *and* has salary greater than 90000.

$$\sigma_{\text{dept_name}=\text{"Physics"} \wedge \text{salary} > 90000} (\text{instructor})$$

- E.g. Select all the tuples of *department relation* where department name is same as building name.

$$\sigma_{\text{dept_name}=\text{building}} (\text{department})$$

Relational Algebra



ID	name	dept_name	salary
1	A	Mechanical	90000
2	B	Electrical	80000
3	C	Physics	75000
4	D	Physics	150000
5	E	Civil	200000
6	F	Electronics	300000

Table 2. instructor relation



Relational Algebra

ID	name	dept_name	salary
3	C	Physics	75000
4	D	Physics	150000

Table 2-1. Output of the E.g.1



Relational Algebra

ID	name	dept_name	salary
4	D	Physics	150000
5	E	Civil	200000
6	F	Electronics	300000

Table 2-2. Output of the E.g.2

Relational Algebra



ID	name	dept_name	salary
4	D	Physics	150000

Table 2-3. Output of the E.g.3

Relational Algebra



➤ The Project operation

- This is a unary operation used *return its argument relation with certain attributes left out.*
- That is this operation shows the list of those attributes that we wish to appear in the result.
- It is denoted by uppercase Greek letter *pi* (Π).
- The predicate appears as a subscript to Π and the argument relation is in the parenthesis after the Π .

Relational Algebra



➤ The Project operation

- E.g. 1. (Refer Table 2) Return ID, name and salary from *instructor* relation.

$$\Pi_{ID, \text{name}, \text{salary}} (\textit{instructor})$$

- A basic version of project operation $\Pi_L (E)$ allows only attributes names to be present in the list (L) but generalised version allows expressions involving attributes to appear in the list (L).
- E.g. To get monthly salary of instructor:

$$\Pi_{ID, \text{name}, \text{salary}/12} (\textit{instructor})$$



Relational Algebra

ID	name	salary
1	A	90000
2	B	80000
3	C	75000
4	D	150000
5	E	200000
6	F	300000

Table 2-4. Output of the E.g. 1 of Project Operation



Relational Algebra

➤ The Composition of Relational Operations

- Fact: The result of a relational operation is itself a relation.
- Because of this fact, these operations can be composed together into a relational algebra expression.
- For e.g., To find the names of all instructors in the Physics department:

$$\Pi_{\text{name}} (\sigma_{\text{dept_name}=\text{"Physics"}} (\text{instructor}))$$

Here there is no relation name given as an argument of project operation. Instead, expression is given that evaluates to a relation.



Relational Algebra

➤ The Rename operation

- In general, results of relational-algebra expressions do not have name.
- But, if you want to give the name to the result rename operation is used.
- This is unary operation and is denoted by lowercase Greek letter *rho* (ρ).
- E.g. Given relational algebra expression E,

$$\rho_x(E)$$

returns the result of expression E under the name x



Relational Algebra

➤ The Rename operation

- Another form includes a relational algebra expression E with arity n .

$$\rho_{x(A_1, A_2, \dots, A_n)}(E)$$

returns the result of expression E under the name x and with the attributes names renamed to A_1, A_2, \dots, A_n .

E.g. Find the ID and name of those instructors who earn more than the instructor whose ID is 1.

$$\Pi_{i.ID, i.name} ((\sigma_{i.salary > w.salary} (\rho_i(instructor) \times \sigma_{w.ID=1} (\rho_w(instructor)))))$$



Relational Algebra

➤ The Cartesian-Product operation

- The Cartesian-Product operation allows us to combine information from any two relation.
- It is denoted by a Cross (\times).
- E.g. Cartesian-Product of relation $r1$ and $r2$ is $r1 \times r2$.
- A Cartesian-Product of set produces pairs but a Cartesian-Product of database relations concatenates two tuples $t1$ and $t2$ into a single tuple.
- This is a binary operation.



Relational Algebra

➤ The Cartesian-Product operation

- As the same attribute name may appear in the both relations, we use following naming schema:
- E.g. The relation schema $r = \text{instructor} \times \text{teaches}$ is $(\text{instructor.ID}, \text{instructor.name}, \text{instructor.dept_name}, \text{instructor.salary}, \text{teaches.ID}, \text{teaches.course_ID}, \text{teaches.sec_ID}, \text{teaches.semester}, \text{teaches.year})$
- Relation name is dropped for the attributes that appear in only one relation.
- $(\text{instructor.ID}, \text{name}, \text{dept_name}, \text{salary}, \text{teaches.ID}, \text{course_ID}, \text{sec_ID}, \text{semester}, \text{year})$

Relational Algebra



➤ The Cartesian-Product operation

- What tuples appear in r , if $r = \text{instructor} \times \text{teaches}$?
- r contains each possible pair of tuples: one from *instructor* relation and another from *teaches* relation.
- i.e. if *instructor* has n_1 tuples and *teaches* has n_2 tuples then resulting relation r will have $n_1 * n_2$ tuples in it.



Relational Algebra

ID	name	dept_name	salary
1	A	Mechanical	90000
2	B	Electrical	80000
3	C	Physics	75000
4	D	Physics	150000
5	E	Civil	200000
6	F	Electronics	300000

Table 2. instructor relation

Relational Algebra



ID	course_ID	sec_ID	semester	year
1	CV1	A1	1	First
2	ME2	A1	2	First
3	EL1	B1	1	Second
4	ET2	B2	2	Third
5	CS1	C1	1	Final
5	CS2	C2	2	Final

Table 3. teaches relation



Relational Algebra

ID	name	dept_name	salary	ID	course_ID	sec_ID	semester	year
1	A	Mechanical	90000	1	CV1	A1	1	First
1	A	Mechanical	90000	2	ME2	A1	2	First
1	A	Mechanical	90000	3	EL1	B1	1	Second
1	A	Mechanical	90000	4	ET2	B2	2	Third
1	A	Mechanical	90000	5	CS1	C1	1	Final
1	A	Mechanical	90000	5	CS2	C2	2	Final
2	B	Electrical	80000	1	CV1	A1	1	First
2	B	Electrical	80000	2	ME2	A1	2	First
...

Table 4. $r = \text{instructor} \times \text{teaches}$

Relational Algebra



➤ The Join operation

- Cartesian-product operation associates every tuple of one relation with every tuple of another relation regardless of their actual association.
- A join operation is binary operation which combines related tuples from different relations, if and only if the given condition is specified.
- A join operation allows us to *combine* a *selection* and a *Cartesian-Product* operation into a single operation.



Relational Algebra

➤ The Join operation

- If $r1$ and $r2$ are two relations, then join operation can be represented as

$$r1 \bowtie_{\theta} r2$$

and it is defined as

$$r1 \bowtie_{\theta} r2 = \sigma_{\theta}(r1 \times r2)$$

where, Θ is a predicate and \times is Cartesian-Product of $r1$ and $r2$

It can also be written as

$$r1 \bowtie_{\theta} r2$$

Relational Algebra



➤ The Join operation

- E.g. find the information about all instructors together with the course_id of all courses they have taught.
- It is represented as

$$\sigma_{\text{instructor.ID}=\text{teaches.ID}}(\text{instructor} \times \text{teaches})$$

- This states that take Cartesian-Product of *instructor* and *teaches*, and then select all tuples from the resulting relation where *instructor.ID matches with the teaches.ID*

Relational Algebra



ID	name	dept_name	salary
1	A	Mechanical	90000
2	B	Electrical	80000
3	C	Physics	75000
4	D	Physics	150000
5	E	Civil	200000
6	F	Electronics	300000

Table 5. instructor relation

Relational Algebra



ID	course_ID	sec_ID	semester	year
1	CV1	A1	1	First
1	CV2	A1	2	First
3	EL1	B1	1	Second
4	ET2	B2	2	Third
5	CS1	C1	1	Final
5	CS2	C2	2	Final

Table 6. teaches relation



Relational Algebra

ID	name	dept_name	salary	ID	course_ID	sec_ID	semester	year
1	A	Mechanical	90000	1	CV1	A1	1	First
1	A	Mechanical	90000	1	CV2	A1	2	First
1	A	Mechanical	90000	3	EL1	B1	1	Second
1	A	Mechanical	90000	4	ET2	B2	2	Third
1	A	Mechanical	90000	5	CS1	C1	1	Final
1	A	Mechanical	90000	5	CS2	C2	2	Final
2	B	Electrical	80000	1	CV1	A1	1	First
2	B	Electrical	80000	1	CV2	A1	2	First
...

Table 7. instructor × teaches



Relational Algebra

ID	name	dept_name	salary	ID	course_ID	sec_ID	semester	year
1	A	Mechanical	90000	1	CV1	A1	1	First
1	A	Mechanical	90000	1	CV2	A1	2	First
3	C	Physics	75000	3	EL1	B1	1	Second
4	D	Physics	150000	4	ET2	B2	2	Third
5	E	Civil	200000	5	CS1	C1	1	Final
5	E	Civil	200000	5	CS2	C2	2	Final

Table 8. $r = \text{instructor} \bowtie \text{teaches}$



Relational Algebra

➤ The Set operations - Union

- Union operation combines results of two or more relational algebra operations and provides a single relation as its output.
- It is denoted as U .
- If r and s are two relations then their union is

$$r \cup s$$

- It displays the attributes or tuples that are present in either or both of the relations.

Relational Algebra



➤ The Set operations - Union

- E.g. Find the set of all courses taught in the Fall of 2017 semester, Spring 2018 semester or both semesters.
- To find the set of all courses taught in the Fall of 2017, we write

$$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Fall"} \wedge \text{year}=2017} (\text{section}))$$

- To find the set of all courses taught in the Spring of 2018, we write

$$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Spring"} \wedge \text{year}=2018} (\text{section}))$$

Relational Algebra



➤ The Set operations - Union

- E.g. Find the set of all courses taught in the Fall of 2017 semester, Spring 2018 semester or both semesters.
- Finally their ***union*** can be written as

$$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Fall"} \wedge \text{year}=2017} (\text{section})) \cup \\ \Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Spring"} \wedge \text{year}=2018} (\text{section}))$$

Relational Algebra



ID	course_id	sec_id	semester	year
1	CV1	1	Fall	2017
1	CV2	1	Spring	2017
2	CS1	2	Summer	2018
3	EL1	2	Winter	2017
3	EL2	1	Fall	2017
4	ET1	1	Spring	2018
4	ET2	2	Spring	2018
5	ME1	2	Summer	2018
6	ME2	1	Fall	2017

Table 9. section relation

Relational Algebra



course_id
CV1
EL2
ET1
ET2
ME2

Table 10. union of given statement



Relational Algebra

➤ The Set operations - Intersection

- Intersection operation allows us to find the tuples that are present in both input relations.
- It is denoted as \cap .
- E.g. If r and s are two relations then their intersection is

$$r \cap s$$

Relational Algebra



➤ The Set operations - Intersection

- E.g. Find the set of all courses taught in the Fall of 2017 semester and Spring 2018 semester.
- To find the set of all courses taught in the Fall of 2017, we write r as

$$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Fall"} \wedge \text{year}=2017} (\text{section}))$$

- To find the set of all courses taught in the Spring of 2018, we write s as

$$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Spring"} \wedge \text{year}=2018} (\text{section}))$$

Relational Algebra



➤ The Set operations - Intersection

- E.g. Find the set of all courses taught in the Fall of 2017 semester and Spring 2018 semester.
- Finally their *intersection* $r \cap s$ can be written as

$$\begin{aligned} & \Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Fall"} \wedge \text{year}=2017} (\text{section})) \cap \\ & \Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Spring"} \wedge \text{year}=2018} (\text{section})) \end{aligned}$$

Relational Algebra



ID	course_id	sec_id	semester	year
1	CV1	1	Fall	2017
1	CV2	1	Spring	2017
2	CS1	2	Summer	2018
3	EL1	2	Winter	2017
3	EL2	1	Fall	2017
4	ET1	1	Spring	2018
4	CV1	2	Spring	2018
5	ME1	2	Summer	2018
6	ME2	1	Fall	2017

Table 11. section relation

Relational Algebra



course_id
CV1

Table 12. intersection of given statement

Relational Algebra



➤ The Set operations – set-difference

- Set-difference operation allows us to find the tuples that are present in one relation but are not in the other relation.
- It is denoted as $-$.
- E.g. If r and s are two relations then their set-difference is

$$r - s$$

which produces a relation containing those tuples in r but not in s .

Relational Algebra



➤ The Set operations – set-difference

- E.g. Find all the courses taught in the Fall 2017 semester but not in Spring 2018 semester.
- To find the set of all courses taught in the Fall of 2017, we write r as

$$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Fall"} \wedge \text{year}=2017} (\text{section}))$$

- To find the set of all courses taught in the Spring of 2018, we write s as

$$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Spring"} \wedge \text{year}=2018} (\text{section}))$$



Relational Algebra

➤ The Set operations – set-difference

- E.g. Find all the courses taught in the Fall 2017 semester but not in Spring 2018 semester.
- Finally their *set-difference* $r - s$ can be written as

$$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Fall"} \wedge \text{year}=2017} (\text{section})) - \\ \Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Spring"} \wedge \text{year}=2018} (\text{section}))$$

Relational Algebra



ID	course_id	sec_id	semester	year
1	CV1	1	Fall	2017
1	CV2	1	Spring	2017
2	CS1	2	Summer	2018
3	EL1	2	Winter	2017
3	EL2	1	Fall	2017
4	ET1	1	Spring	2018
4	CV1	2	Spring	2018
5	ME1	2	Summer	2018
6	ME2	1	Fall	2017

Table 13. section relation

Relational Algebra



course_id
EL2
ME2

Table 13. set-difference of given statement

Relational Algebra



➤ The Set operations – Points to remember

- For union, intersection and set difference operations to make sense:
 - We must ensure that the input relations to the union operation have the same number of attributes; the number of attributes of a relation is referred to as its *arity*.
 - When the attributes have associated types, the types of the i^{th} attributes of both input relations must be the same, for each i .
- Such relations are referred to as ***compatible relations***.



Relational Algebra

➤ The assignment operation

- For ease of understanding and convenience, parts of a relational algebra expression can be assigned to temporary relation variables.
- This can be done with the help of *assignment operator* which is denoted as \leftarrow .
- It works like assignment operator in programming language.



Relational Algebra

➤ The assignment operation

- E.g. Find the set of all courses taught in the Fall of 2017 semester, Spring 2018 semester or both semesters. We could write it as:

$\text{courses_fall_2017} \leftarrow \Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Fall"} \wedge \text{year}=2017} (\text{section}))$

$\text{courses_spring_2018} \leftarrow \Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{"Spring"} \wedge \text{year}=2018} (\text{section}))$

And

$\text{courses_fall_2017} \cup \text{courses_spring_2018}$



References

- Abraham Silberschatz, Henry F. Korth and S. Sudarshan, “Database System Concepts”, Mc-Graw Hill, 7th Edition.
- https://en.wikipedia.org/wiki/Edgar_F._Codd
- <https://www.geeksforgeeks.org/relational-model-in-dbms/>
- <https://www.javatpoint.com/dbms-relational-model-concept>
- <https://www.geeksforgeeks.org/relation-schema-in-dbms/>



Module 5

Relational Model and SQL - Part 2

Mr. Swapnil S Sontakke (Asst. Prof.)

Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli



Content

- Introduction to SQL
- SQL Data Definition
- SQL Operators
- Aggregate Functions
- Modification of the Database
- Basic Structure of SQL Queries
- Set Operations
- Nested Subqueries



Introduction to SQL

- Original version called '*Sequel*' was developed by IBM as a part of System R project in 1974.
- Later the name changed to **SQL (Structured Query Language)**
- SQL was standardized in 1986 by ANSI and ISO, called SQL-86.
- Recent version of standard is SQL:2016 (stable)
- SQL has clearly established itself as the standard relational database language.



Introduction to SQL

➤ SQL language has several parts:

- **Data Definition Language (DDL)**
 - Has commands for defining relation schemas, deleting relations and modifying relation schemas
- **Data Manipulation Language (DML)**

- Provides commands to retrieve the information from the database and to insert tuples into, delete tuples from and modify tuples in the database



Introduction to SQL

➤ SQL language has several parts:

- **Integrity**

- Provides commands for specifying the integrity constraints that the data stored in the database must satisfy
- Operations that violates integrity constraints are not allowed.

- **View Definition**

- Provides commands for defining views



Introduction to SQL

- SQL language has several parts:
 - **Transaction Control**
 - Provides commands for specifying the beginning and end points of transactions
 - **Authorization**
 - Provides commands for specifying access rights to relations and views



Introduction to SQL

- SQL language has several parts:
 - **Embedded SQL and Dynamic SQL**
 - Embedded and Dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, Java, Python, etc.



SQL Data Definition

- The set of relations in the database are specified using Data Definition Language (DDL).
- DDL also specifies the following information about each relation:
 - The schema for the relation
 - The types of values associated with each attribute
 - The integrity constraints
 - The set of indices to be maintained for each relation.
 - The security and authorization information for each relation.
 - The physical storage structure of each relation on disk.



SQL Data Definition

- **Basic Types**
- SQL supports a variety of built-in types. Some of them are:
 - **char(n)**: character: A fixed-length character string with user-defined length n
 - **varchar(n)**: character varying: A variable-length character string with user-defined maximum length n
 - **int**: integer: a finite subset of the integers that is machine dependent
 - **smallint**: A small integer: a machine-dependent subset of the integer type



SQL Data Definition

- **Basic Types**
- SQL supports a variety of built-in types. Some of them are:
 - **Numeric(p,d)**: A fixed-point number with user-defined length precision. The number consists of p digits in total (plus a sign) and d of the p digits are to the right of the decimal points.
 - **Real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision
 - **float(n)**: A floating-point number with precision of at least n digits



SQL Data Definition

- **Basic Schema Definition**
- SQL relation can be defined using **create table** command.
- The general form of create command is

```
create table table_name
(
    A1 D1,
    A2 D2,
    ...
    An Dn,
    <integrity-constraint1k>
);
```

**Fig. 1 General form of
create table**



SQL Data Definition

- **Basic Schema Definition**
- Here,
 - Each A_i is an attribute name,
 - Each D_i specifies the domain of attribute A_i
- Some of the integrity constraints are
 - primary key ($A_{j_1} \dots A_{j_n}$)
 - Foreign key ($A_{k_1} \dots A_{k_n}$) references s
 - not null



SQL Data Definition

- Basic Schema Definition
- Example: instructor relation

```
create table instructor
(
    ID          varchar(5),
    name        varchar(50) not null,
    dept_name   varchar(20),
    salary      numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department (dept_name)
);
```

Fig. 2 create *instructor* table



SQL Data Definition

- Basic Schema Definition
- Example: department relation

```
create table department
(
    dept_name      varchar(20),
    building       varchar(20),
    budget         numeric(12,2),
    primary key (dept_name)
);
```

Fig. 3 **create department table**



SQL Data Definition

➤ Basic Schema Definition

- A newly created relation is empty initially.
- Inserting tuples into a relation, updating them, and deleting them are done by data manipulation statements insert, update, and delete

SQL Data Definition



Demonstration



SQL Data Definition

➤ Basic Schema Definition

- **drop table**
- To remove a relation from an SQL database drop table command is used.
- This command deletes all information about the dropped relation from the database.
- This deletes all tuples from table as well the table.
- E.g. **drop table table_name;**
drop table instructor;



SQL Data Definition

➤ Basic Schema Definition

- **delete from**
- This command deletes all tuples from the relation but do not delete table.
- Table becomes empty
- E.g. **delete from table_name;**
delete from instructor;



SQL Data Definition

➤ Basic Schema Definition

- **alter table**
- This command is used to alter the table i.e. to add new attributes to existing relation or drop attributes from a relation.
- E.g. **alter table *table_name* add A D;**
This will add attribute A with its domain D in the specified table.
- E.g. **alter table *table_name* drop A;**
This will drop an attribute A from specified table.

SQL Data Definition



Demonstration

SQL Data Definition



- **Demonstration**
- Offline Tool
 - XAMPP
- Online Compiler
 - https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in



SQL Operators

- SQL supports following operators that can be combined with queries.
 - Arithmetic (+,-,*,/ and %)
 - Bitwise (&, | and ^)
 - Comparison (=, >, >=, <, <= and <>)
 - Compound (+=, -=, etc.)
 - Logical
 - ALL - TRUE if all of the subquery values meet the condition
 - AND - TRUE if all the conditions separated by AND is TRUE
 - ANY - TRUE if any of the subquery values meet the condition



SQL Operators

- SQL supports following operators that can be combined with queries.
 - Logical
 - BETWEEN - TRUE if the operand is within the range of comparisons
 - EXISTS - TRUE if the subquery returns one or more records
 - IN - TRUE if the operand is equal to one of a list of expressions
 - LIKE - TRUE if the operand matches a pattern
 - NOT - Displays a record if the condition(s) is NOT TRUE
 - OR - TRUE if any of the conditions separated by OR is TRUE
 - SOME - TRUE if any of the subquery values meet the condition



SQL Aggregate Functions

- **Aggregate functions** are functions that take a collection (a set or multiset) of values as input and return a single value.
- SQL offers five standard built-in aggregate functions:
 - Average (**avg**)
 - Minimum (**min**)
 - Maximum (**max**)
 - Total (**sum**)
 - Count (**count**)



Modification of the Database

- Modification of database includes the commands to add, remove and update the database information.
- **Insertion**

- To insert data into a relation, we need to specify a tuple to be inserted.
- The attribute values for inserted tuples must be members of the corresponding attribute's domain.
- Command used for inserting a tuple is

insert into *table_name* values ('value 1', 'value 2', ..., 'value N');



Modification of the Database

```
create table department
(
    dept_name      varchar(20),
    building       varchar(20),
    budget         numeric(12,2),
    primary key (dept_name)
);
```

Fig. 4 department table



Modification of the Database

- To insert a tuple into ‘department’ relation we write

insert into department values ('Civil', 'Wing-A', 2500000.00);

Insert into department values ('Mechanical', 'Wing-B', 2500000.00);

- Here order of values is important.
- Values should be specified in the order in which the corresponding attributes are listed in the relation schema.
- To ease this process, another way to write insert query is



Modification of the Database

```
insert into department (dept_name, building, budget)
values ('Civil', 'Wing-A', 2500000.00);
```

```
Insert into department (dept_name, building, budget)
values ('Mechanical', 'Wing-B', 2500000.00);
```



Modification of the Database

- **Deletion**
- We can delete whole tuple from the relation using **delete command**.
- We cannot delete values on only particular attributes.
- The general form of deletion in SQL is

**delete from *r*
where *P*;**



Modification of the Database

- **Deletion**
- Here, P represents a predicate and r represents a relation.
- The delete statement first finds all the tuples t in r for which $P(t)$ is true and then deletes them from r .



Modification of the Database

- **Deletion**
- **from clause**
 - The from clause is a list of the relations to be accessed in the evaluation of the query.
- **where clause**
 - The where clause is a predicate involving attributes of the relation in the from clause.
 - In deletion, if where clause is omitted then whole relation i.e. all tuples from the specified relation will be deleted.



Modification of the Database

- **Deletion**
- E.g. To delete all tuples from *department* relation, we write

delete from *department*;

To delete tuples where dept_name is Civil, we write

**delete from *department*
where *dept_name*='Civil';**



Modification of the Database

- **Deletion**
- E.g. To delete tuples where *dept_name* is Civil, we write

delete from *department*
where *dept_name*=‘Civil’;

To delete tuples where *budget* is between 100000 and 200000, we write

delete from *department*
where *budget* between 100000 AND 200000;



Modification of the Database

- **Updates**
- To update/change the particular value of a tuple without changing all values of that tuple, **update** statement is used.
- Similar to the **insert** and **delete**, we can update any tuple in a relation using a query.
- The general form of **update** is

```
update table_name  
set A = <new value>;
```



Modification of the Database

- **Updates**
- E.g. To update budget of all departments by 5%,
**update department
set budget=budget*1.05;**
To update budget of departments by 5% where current budget is 150000.00 or less
**update department
set budget=budget*1.05
where budget <=150000.00**

SQL Data Definition



Demonstration



Basic Structure of SQL Queries

- A basic structure of an SQL query consists of three clauses: select, from and where.
- A query takes as its input the relations listed in the ***from*** clause, operates on them as specified in the ***where*** and ***select*** clauses, and then produces a relation as the result.
- ***Select*** clause will return relation based on the predicate specified in the ***where*** clause.
- It performs the select and project operations of relational algebra.



Basic Structure of SQL Queries

➤ **Queries on a Single Relation**

- Here, only one relation will be specified in the query.
- The general form is

```
select A1, A2, ..., An  
from table_name  
where P;
```



Basic Structure of SQL Queries

➤ **Queries on a Single Relation**

- E.g. Find the department names of all instructors

```
select dept_name  
      from instructor;
```

- If we don't want duplicates, then distinct keyword is used after select.
- E.g. For the above query, if we want department name only once then

```
select distinct dept_name  
      from instructor;
```



Basic Structure of SQL Queries

➤ **Queries on a Single Relation**

- If we want duplicates, then **all** keyword is used after select which *explicitly* specifies that include duplicates in the resulting relation.
- E.g. For the above query, if we want all department names then we write

**select all *dept_name*
from *instructor*;**

- By default SQL allows duplicates, so we don't need to use **all** keyword.
- But, ***distinct*** keyword is mandatory if we don't want duplicates.



Basic Structure of SQL Queries

➤ **Queries on a Single Relation**

- *Select* clause may also contain *arithmetic expressions* involving +,-,* and / operators
- E.g. Following query returns a relation that is same as the instructor relation, except the salary is incremented by 10%.

```
select ID, name, dept_name, salary*1.1  
from instructor;
```



Basic Structure of SQL Queries

➤ Queries on a Single Relation

- The ***where*** clause allows us to select only those rows in the resulting relation of the ***from*** clause that satisfy a specified predicate.
- E.g. To find the names of all instructors in the Electrical department who have salary greater than \$70,000

```
select name
      from instructor
     where dept_name='Electrical' and salary > 70000
```



Basic Structure of SQL Queries

➤ **Queries on a Single Relation**

- The ***where*** clause may also include comparison and logical operators as you can see in the above example.



Basic Structure of SQL Queries

➤ **Queries on a Multiple Relations**

- A single SQL query can specify two or more relations.
- The general form is

```
select  $A_1, A_2, \dots, A_n$   
from  $r_1, r_2, \dots, r_m$   
where  $P;$ 
```

- Each A_i represents an attribute, and each r_i a relation. P is a predicate. If the where clause is omitted, the predicate P is true.



Basic Structure of SQL Queries

➤ Queries on a Multiple Relations

- E.g. To retrieve the names of all instructors, along with their department names and department building name, we write

select name, instructor.dept name, building

from instructor, department

where instructor.dept name=department.dept name;



Basic Structure of SQL Queries

➤ **Queries on a Multiple Relations**

- In the similar manner, we can retrieve the Cartesian-Product of any two(or more) relations.
- E.g. Cartesian-Product of *instructor* and *department* can be written as

```
select ID, name, instructor.dept name, salary, department.dept_name,  
building, budget  
from instructor, department
```



Basic Structure of SQL Queries

➤ **Queries on a Multiple Relations**

- To retrieve all the attributes of a relation we use *.
- E.g. To retrieve the all attributes of instructor and department relation we write

select * from *instructor, department*;

Or

select *instructor.*, department.
from *instructor, department*;**



Basic Structure of SQL Queries

➤ Ordering the Resulting Relation

- The *order by* clause causes the tuples in the result of a query to appear in sorted order.
- E.g. To retrieve the all attributes of department relation sorted based on the *dept_name*, we write

```
select * from department
          order by dept_name;
```



Basic Structure of SQL Queries

➤ The Rename Operation

- The *rename operation* is used to rename the attributes or relations in the query.
- To rename we use *as* clause.
- It can be used in both *select* and *from* clause.
- E.g. To retrieve the *name* as *instructor_name* from instructor relation, we write

```
select name as instructor_name
      from instructor;
```



Basic Structure of SQL Queries

➤ The Rename Operation

- E.g. To retrieve the *name* as *instructor_name* and *dept_name* from *instructor* as T relation, we write

```
select T.name as instructor_name, T.dept_name  
from instructor as T;
```

SQL Data Definition



Demonstration



Basic Structure of SQL Queries

➤ Set Operations

- The mathematical set operations union, intersection and set difference can be represented in SQL with the help of ***union***, ***intersection*** and ***except***.

➤ Union

- This operation combines results of two ***select*** statements and provides a single result.
- It by default, automatically removes all the duplicates.
- To retain duplicates, use ***union all***.



Basic Structure of SQL Queries

➤ Union

- E.g. To find the set of all courses taught in the Fall 2017, Spring 2018 or in both, we write

```
(select course id  
      from section  
    where semester = 'Fall' and year= 2017)  
        union  
(select course id  
      from section  
    where semester = 'Spring' and year= 2018);
```



Basic Structure of SQL Queries

➤ **Union**

- Some databases allow parenthesis for each separate part of statements and some don't.
- Parenthesis are useful for ease of reading.



Basic Structure of SQL Queries

➤ **Intersect**

- This operation combines results of two select statements and provides a single result.
- It returns the common tuples from two (or more) relations.
- It by default, automatically removes all the duplicates.
- To retain duplicates, use *intersect all*.



Basic Structure of SQL Queries

➤ Intersect

- E.g. To find the set of all courses taught in both the Fall 2017 and Spring 2018,

(select course id

from section

where semester = 'Fall' and year= 2017)

intersect

(select course id

from section

where semester = 'Spring' and year= 2018);



Basic Structure of SQL Queries

➤ **except**

- This operation combines results of two select statements and provides a single result.
- It returns the tuples from that are present in one relation but not present in another relation.
- It by default, automatically removes all the duplicates.
- To retain duplicates, use ***except all***.



Basic Structure of SQL Queries

➤ except

- E.g. To find the set of all courses taught in the Fall 2017 but not in Spring 2018,

(select course id

from section

where semester = 'Fall' and year= 2017)

except

(select course id

from section

where semester = 'Spring' and year= 2018);



References

- Abraham Silberschatz, Henry F. Korth and S. Sudarshan, “Database System Concepts”, Mc-Graw Hill, 7th Edition.

Module 6

Database Architectures

Mr. Swapnil S Sontakke (Asst. Prof.)

**Department of Computer Science and Engineering,
Walchand College of Engineering, Sangli**





Content

- Overview
- Centralized Database Systems
- Server System Architectures
- Parallel Systems
- Distributed Database Systems
- Cloud Service Models



Overview

- The architecture of a database system is greatly influenced by the underlying computer system on which it runs.
- It includes processor, memory architecture, networking, and requirements of parallelism and distribution.
- Based on these components the database systems are developed to run smoothly on these systems.
- We discuss some of these database systems here.



Centralized Database Systems

- *Centralized database systems are those that run on a single computer system.*
- Such database systems span a range from single-user database systems running on mobile devices or personal computers to high-performance database systems running on a server with multiple CPU cores and disks and a large amount of main memory that can be accessed by any of the CPU cores.
- Centralized database systems are widely used for enterprise-scale applications.



Centralized Database Systems

- These systems are generally located, stored, and maintained in a single location such as desktop, a server CPU, or a mainframe computer.
- It is generally used by organization or an institute.
- Users access a centralized database through a computer network which is able to give them access to the central CPU, which in turn maintains to the database itself.
- Basically, there are two ways in which computer systems are used: Single User and Multiuser.

Centralized Database Systems



➤ Single User System

- *It is a system used by a single user at a time, usually with one processor (with multiple cores) and one or two storage disks.*
- E.g. Personal Computers, Laptops and Smartphones
- The features such as transaction management, concurrency control, security, backup, crash recovery may be absent or provided at the basic level.
- They may not have SQL support and instead may provide API for data access. These are called ***embedded database systems***.

Centralized Database Systems



➤ Single User System

- Embedded systems are usually intended to perform a dedicated function, either as an independent system or as a part of large system.
- E.g. MP3 players, Mobile Phones, Video Game Consoles, Digital Cameras, DVD Players, GPS, Household Appliances such as Washing Machines, etc.
- Fig. 1A shows single user system with the database.

Centralized Database Systems



Fig. 1A. Centralized Database System Architectures (Single User)

Centralized Database Systems



➤ Multi-User System

- *It is a system used by two or more users simultaneously.*
- *A typical multiuser system has multiple disks, a large amount of memory and multiple processor.*
- A large number of users are connected to the single system remotely, called as **server**.
- E.g. Mainframes, Minicomputers and other powerful computers
- The features such as transaction management, concurrency control, security, backup, crash recovery are provided in such systems.

Centralized Database Systems



➤ **Multi-User System**

- Servers fulfill the requests which are received from application programs running on client/user's machines.
- The requests can be any SQL query or request to manipulate (insert, delete and update) the database stored on the server.
- The database may exist on a single machine or may be distributed.
- Fig. 1B shows multiuser system architecture



Centralized Database Systems

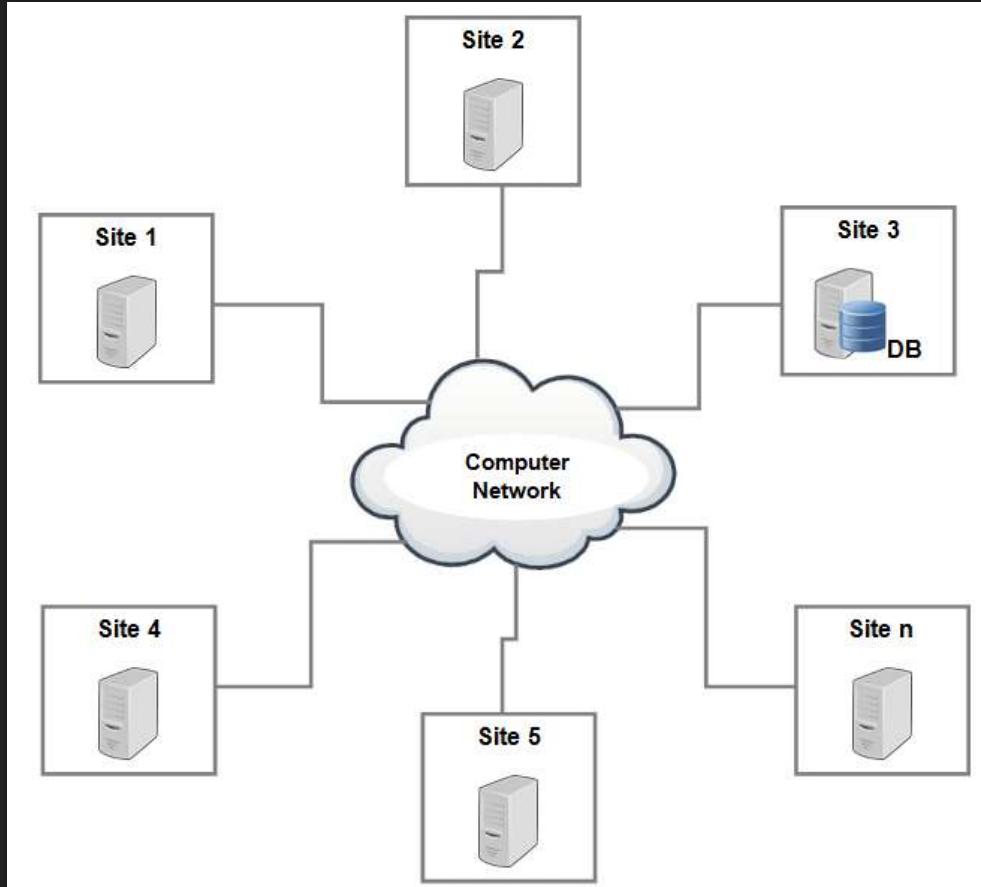


Fig. 1B. Centralized Database System Architectures (Multiuser)

Server System Architectures



- Server systems can be broadly categorized as transaction servers and data servers.
- **Transaction-server systems (query-server systems)**
 - Provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
 - Usually, client machines ship transactions to the server systems, where those transactions are executed, and results are shipped back to clients that are in charge of displaying the data.

Server System Architectures



➤ **Transaction-server systems (query-server systems)**

- Requests may be specified through the use of SQL or through a specialized application program interface.
- Fig. 2 shows transaction server system with shared memory and process structure

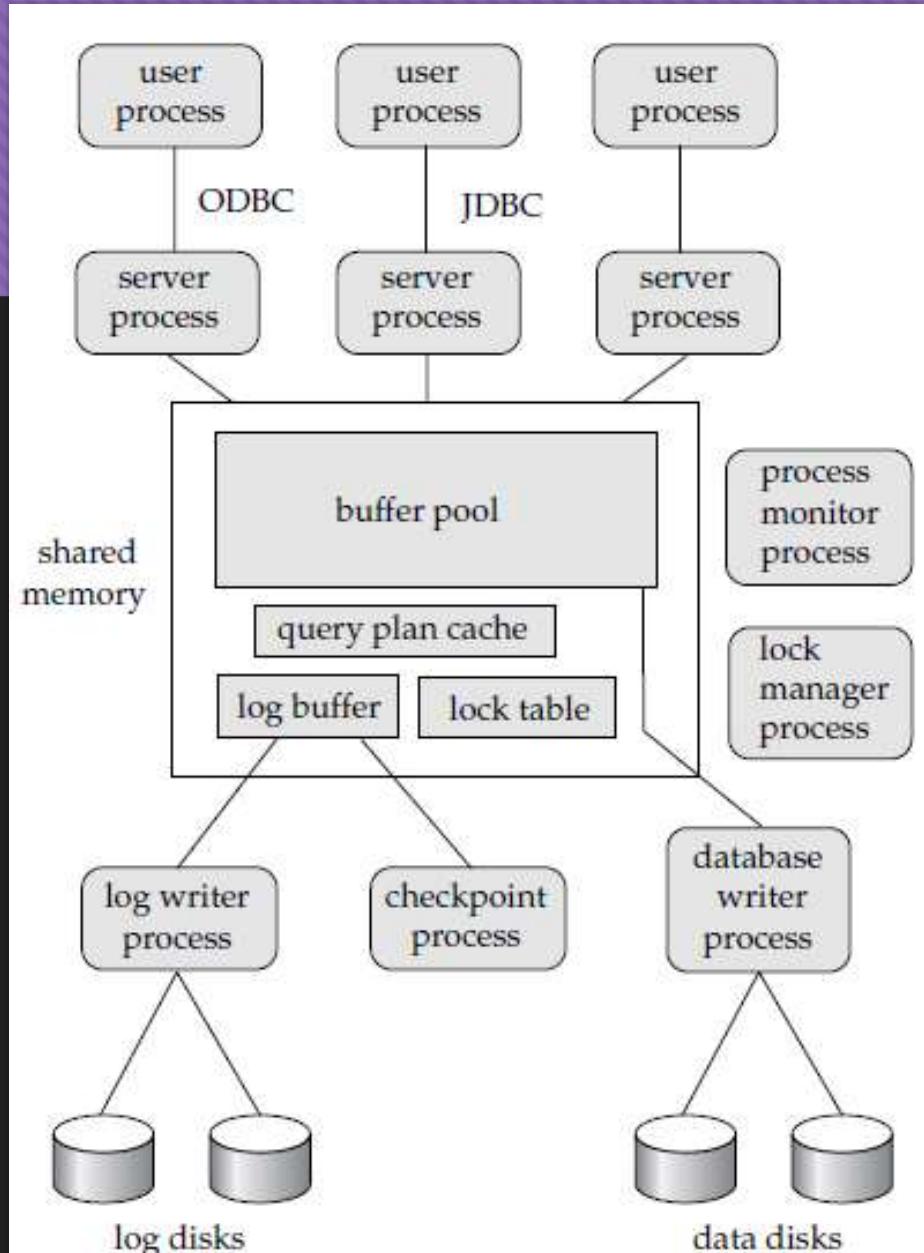


Fig. 2. Transaction server system with shared memory and process structure

Server System Architectures



➤ **Transaction-server systems (query-server systems)**

- Following processes form a part of database in transaction-server system:
- **Server Process**
 - Receive user queries (requests/transactions), execute them and sends the result back.
 - Queries may be submitted to the server process from user process running embedded SQL or via JDBC, ODBC or similar protocols



Server System Architectures

➤ Transaction-server systems (query-server systems)

- **Lock Manager Process**
 - Implements lock manager functionality which includes lock grant, lock release and deadlock detection
- **Database Writer Process**
 - Writes the modified buffer blocks to a disk on continuous basis
- **Log Writer Process**
 - Outputs log records from the log record buffer to the stable storage

Server System Architectures



➤ Transaction-server systems (query-server systems)

- **Checkpoint Process**
 - This process performs periodic checkpoints
- **Process Monitor Process**
 - This process monitors other processes, and if any of them fails, it takes recovery actions for the process, such as aborting any transaction being executed by the failed process and then restarting the process.

Server System Architectures



➤ Transaction-server systems (query-server systems)

- **Shared Memory**
 - The shared memory contains all shared data, such as:
 - Buffer pool
 - Lock table
 - Log buffer, containing log records waiting to be output to the log on stable storage
 - Cached query plans, which can be reused if the same query is submitted again

Server System Architectures



- Server systems can be broadly categorized as transaction servers and data servers.
- **Data-server systems**
 - Data-server systems allow clients to interact with the servers by making requests to read or update data, in units such as files, pages, or objects.
 - For example, file servers provide a file-system interface where clients can create, update, read, and delete files.

Server System Architectures



- Server systems can be broadly categorized as transaction servers and data servers.
- **Data-server systems**
 - They support units of data—such as pages, tuples, or objects—that are smaller than a file.
 - They provide indexing facilities for data, and they provide transaction facilities so that the data are never left in an inconsistent state if a client machine or process fails.



Parallel Systems

- Parallel systems improve *processing and I/O speeds* by using a large number of computers in *parallel*.
- In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.
- A parallel system can be *coarse-grain or fine-grain*.
- *A coarse-grain parallel machine consists of a small number of powerful processors.*
- A high-end server may offer coarse-grain parallelism where with 2-4 processors and 20-40 cores per processor.



Parallel Systems

- *A massively parallel or fine-grain parallel machine uses thousands of smaller processors.*
- In massively parallel computers memory sharing is impractical, so such systems are built using large number of computers each of which having separate memory and disks.



Interconnection Networks

- Parallel systems consist of a set of components (processors, memory, and disks) that can communicate with each other via an interconnection network.
- Fig 3. shows several commonly used types of interconnection networks:

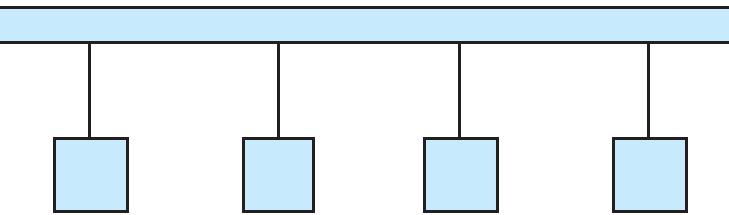


Interconnection Networks

➤ Bus

- All the system components can send data on and receive data from a single communication bus.
- No longer used for bigger networks
- Still used for connecting multiple CPUs and memory units within a single node, and they work well for small numbers of processors.
- Fig. 3a. shows bus network

Interconnection Networks



(a) bus

Fig. 3a. Bus Network

Interconnection Networks



➤ Ring

- The components are nodes arranged in a ring (circle), and each node is connected to its two adjacent nodes in the ring.
- Each link can transmit data concurrently with other links in the ring, leading to better scalability.
- Fig. 3b. shows ring network

Interconnection Networks

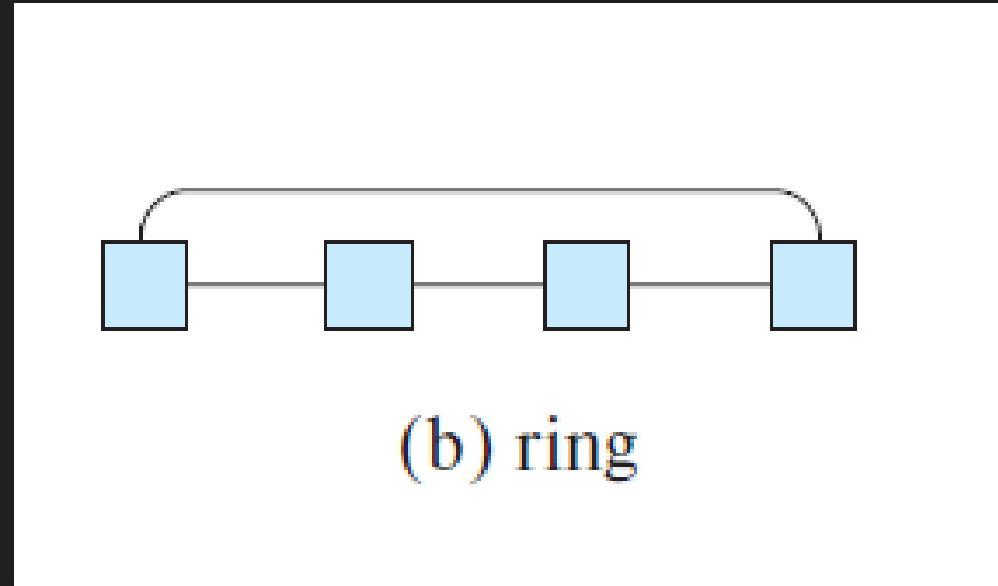


Fig. 3b. Ring Network

Interconnection Networks



➤ Mesh

- The components are nodes in a grid, and each component connects to all its adjacent components in the grid.
- In a two-dimensional mesh, each node connects to (up to) *four* adjacent nodes, while in a three-dimensional mesh, each node connects to (up to) *six* adjacent nodes.
- Fig. 3c. shows a two-dimensional mesh.

Interconnection Networks

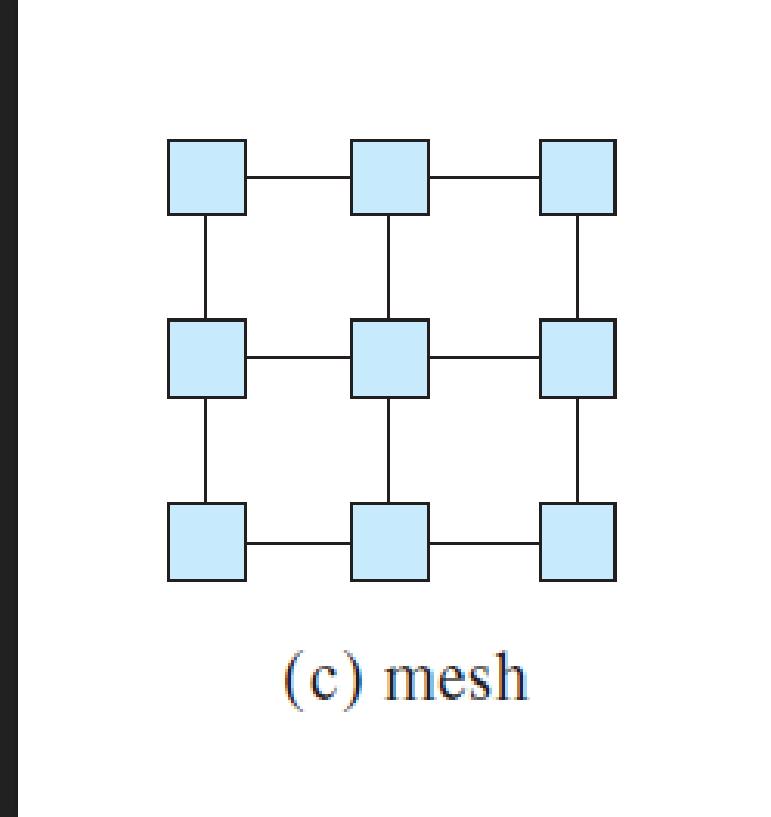


Fig. 3c. 2-D Mesh Network

Interconnection Networks



➤ Hypercube

- The components are numbered in binary, and a component is connected to another if the binary representations of their numbers differ in exactly one bit.
- Thus, each of the n components is connected to $\log(n)$ other components.
- Fig. 3d. shows a hypercube network with eight nodes

Interconnection Networks

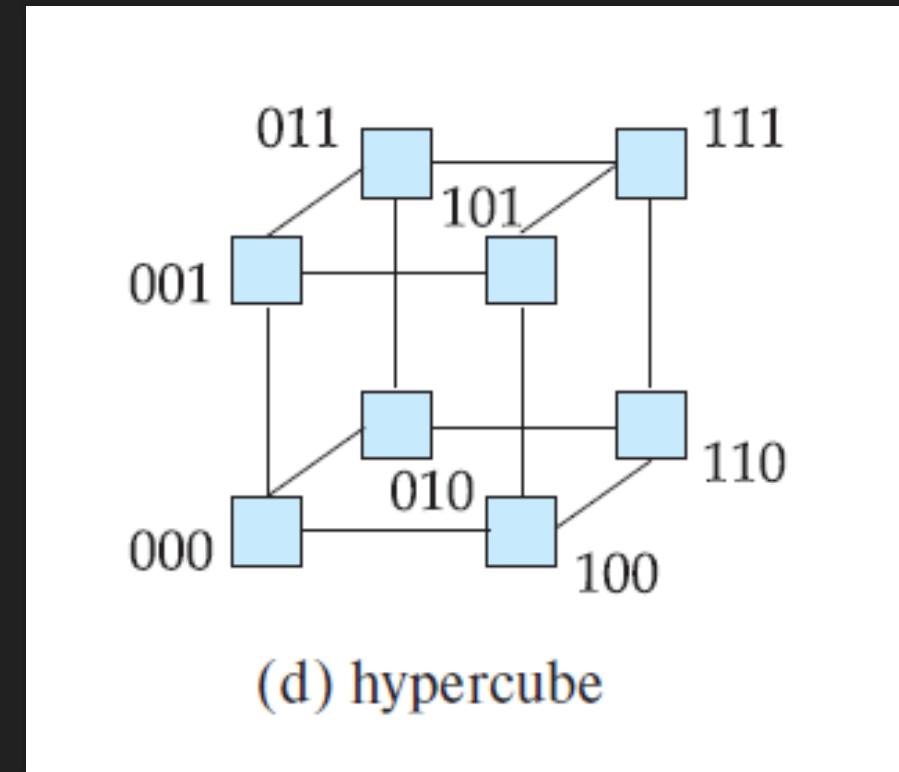


Fig. 3d. Hypercube Network

Interconnection Networks



➤ Tree-like

- Server systems in a *data center* are typically mounted in racks, with each rack holding up to about 40 nodes.
- Multiple racks are used to build systems with larger numbers of nodes.
- Multiple top-of-rack switches (also referred to as *edge switches*) can in turn be connected to another switch, called an *aggregation switch*, allowing interconnection between racks.
- Multiple aggregation switches can be connected to a *core switch*.
- *Such a network is called as tree-like network with three-tiers.*

Interconnection Networks

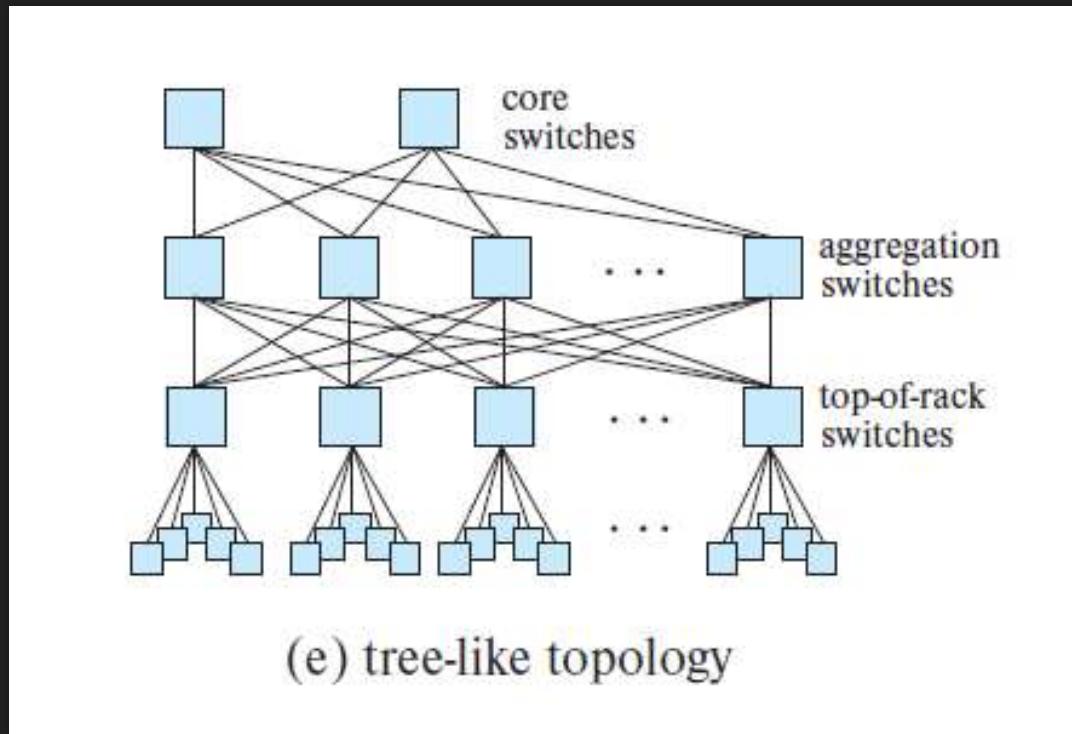


Fig. 3e. Tree-like Network

Parallel Database Architectures



- There are several architectural models for parallel machines.
- Among the most prominent ones are
 - Shared Memory
 - Shared Disk
 - Shared Nothing
 - Hierarchical
- In the figure 4a-4d, M denotes memory, P denotes a processor, and disks are shown as cylinders.

Parallel Database Architectures



➤ Shared Memory

- All the processors share a common memory typically through an interconnection network.
- Disks are also shared by the processors.
- Benefits
 - Data in shared memory can be accessed by any process without being moved with software.
 - A process can send messages to other processes much faster by using memory writes
- Fig. 4a shows shared memory architecture

Parallel Database Architectures

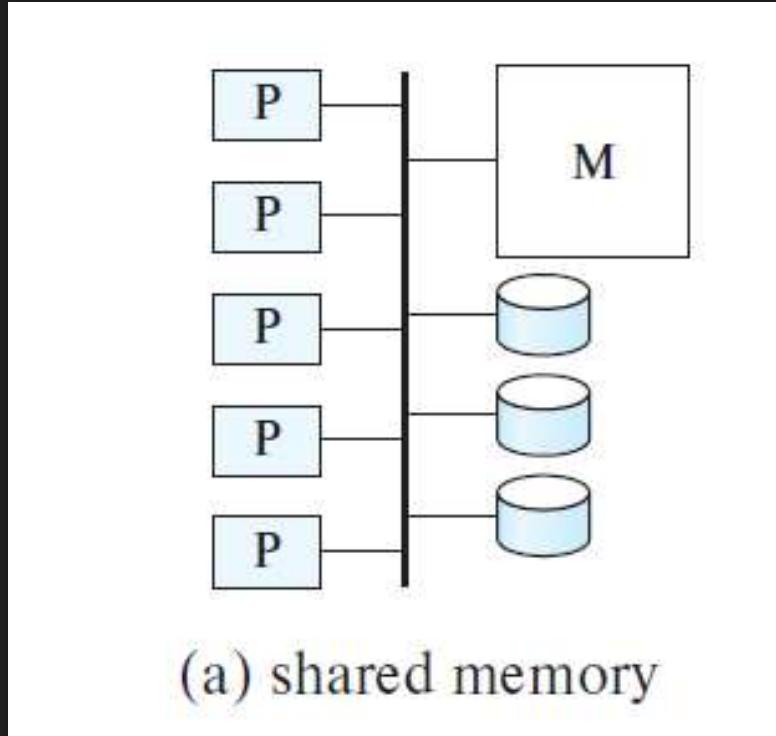


Fig. 4a. Shared Memory

Parallel Database Architectures



➤ Shared Disk (Clusters)

- Each node has its own processors and memory, but all nodes can access all disks directly via an interconnection network.
- Benefits
 - Shared-disk system can scale to a larger number of processors than a shared-memory system.
 - Offers a cheap way to provide a degree of *fault tolerance*: If a node fails, the other nodes can take over its tasks, since the database is resident on disks that are accessible from all nodes.
- Fig. 4b. shows shared disk architecture

Parallel Database Architectures

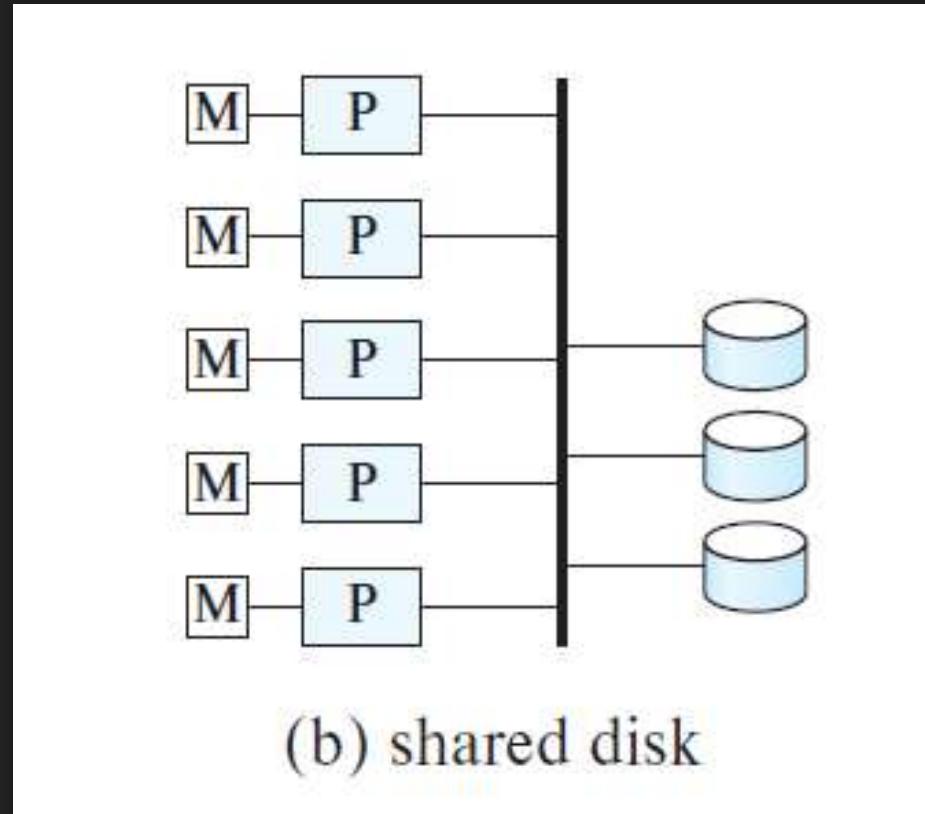


Fig. 4b. Shared Disk

Parallel Database Architectures



➤ **Shared Nothing**

- Each node consists of a processor, memory, and one or more disks.
- The nodes communicate by a high-speed interconnection network.
- A node functions as the server for the data on the disk or disks that the node owns.
- Benefit
 - Overcomes the disadvantage of requiring all I/O to go through a single interconnection network.
- Fig. 4c. shows shared nothing architecture

Parallel Database Architectures

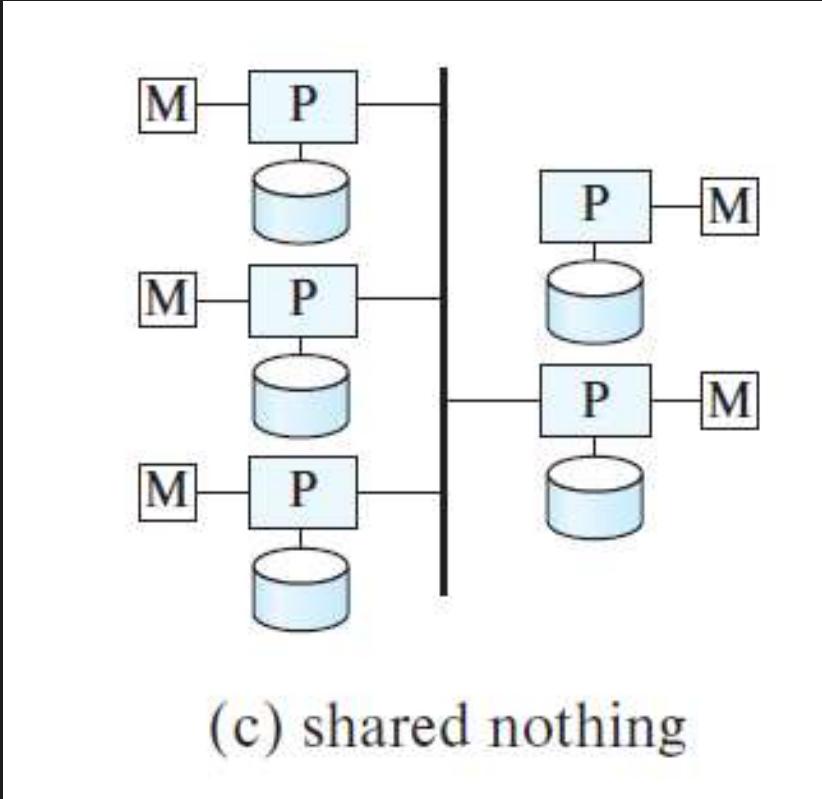


Fig. 4c. Shared Nothing

Parallel Database Architectures



➤ Hierarchical

- Combines the characteristics of shared-memory, shared disk, and shared-nothing architectures
- The top level is a shared-nothing architecture.
- Each node of the system could actually be a shared-memory system with a few processors.
- Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.

Parallel Database Architectures



➤ **Hierarchical**

- Thus, a system could be built as a hierarchy, with shared-memory architecture with a few processors at the base, and a shared-nothing architecture at the top, with possibly a shared-disk architecture in the middle.
- Fig. 4d. shows hierarchical architecture

Parallel Database Architectures

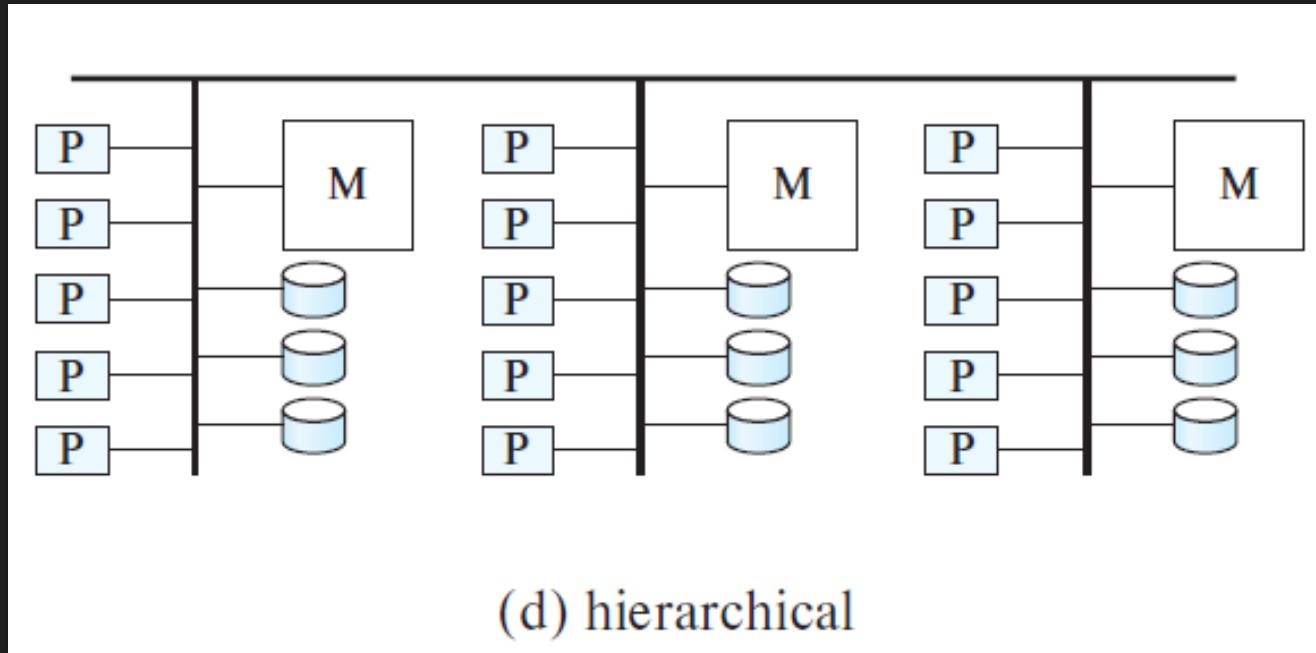


Fig. 4d. Hierarchical

Distributed Database Systems



- The database is stored on nodes located at geographically separated sites.
- The nodes in a distributed system communicate with one another through various communication media, such as high-speed private networks or the internet.
- They do not share main memory or disks.
- The general structure of a distributed system appears in Fig. 5

Distributed Database Systems

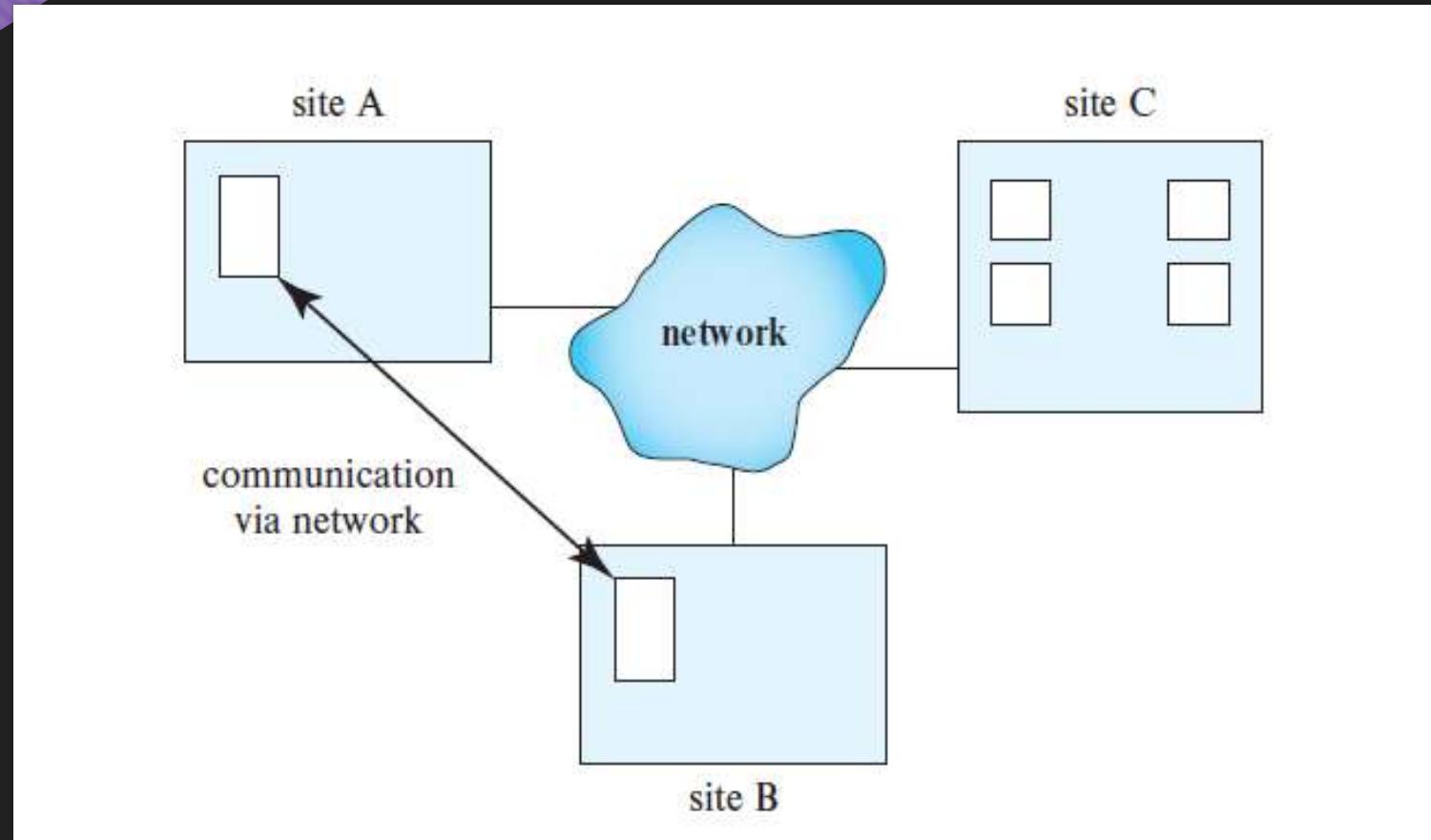


Fig. 5 A Distributed System



Distributed Database Systems

- The main difference between distributed database systems and shared nothing parallel systems is that in distributed systems, nodes/sites are *geographically separated*.
- Distributed database systems need to continue working even in the event of failure of an entire data center, to *ensure high availability*.
- A *local transaction* is one that accesses data only from nodes where the transaction was initiated.
- A *global transaction*, on the other hand, is one that either accesses data in a node different from the one at which the transaction was initiated, or accesses data in several different nodes.



Distributed Database Systems

- Distributed databases may be *separately administered*, with each site retaining some degree of autonomy of operation which is not present in parallel database systems.
- Nodes in a distributed database tend to *vary more in size and function*, whereas parallel databases tend to have nodes that are of similar capacity.



Distributed Database Systems

- Two types of distributed database systems
- **Homogeneous distributed database system**
 - Nodes share a common global schema, all nodes run the same distributed database-management software, and the nodes actively cooperate in processing transactions and queries.
- **Heterogenous distributed database system**
 - databases are constructed by linking together multiple already-existing database systems, each with its own schema and possibly running different database-management software.
 - Limited cooperation in processing transactions and queries.



References

- Abraham Silberschatz, Henry F. Korth and S. Sudarshan, “Database System Concepts”, Mc-Graw Hill, 7th Edition.