Software Engineering

م .م ايناس إسماعيل عمران

Chapter One: Introduction to Software Engineering

1.1 The Computer Software Definition

1.2 Software Engineering Definition

1.3 The characteristic of software engineer

1.4 Software Characteristics

1.5 Software Applications

1.6 Software: A crisis on the horizon

1.7 The Characteristics of Well-Engineered Software

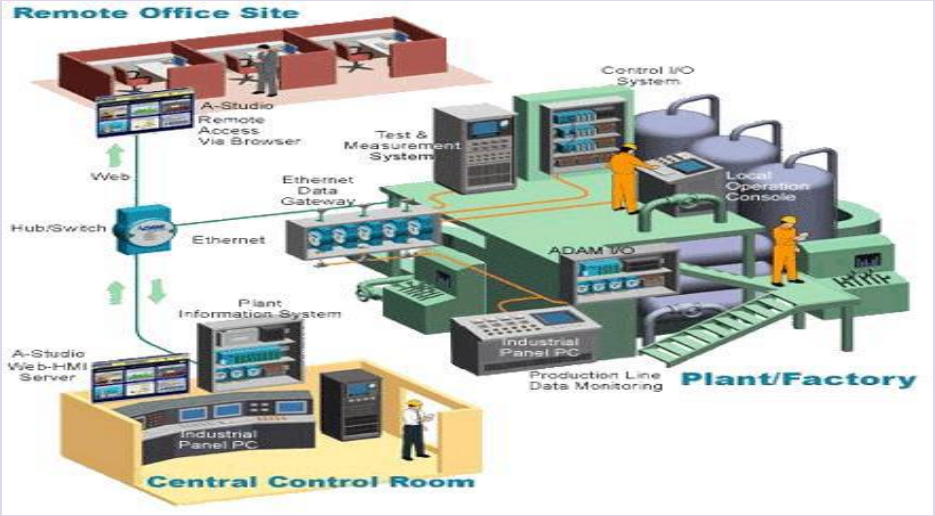
1.8 The Goals of Software Engineering

What is software?

The software might take the following forms:

- **Instructions:** Computer programs, that when executed provide desired function and performance.
- **Data structured:** That enable the programs to manipulate information.
- **Documents:** That describes the operation and use of programs.

Types of Software



Generic **Custom**

Generic Software Products

Developed to be sold to a range of different customers.

Examples:

—MS Office

—Photoshop

—games

Custom Software Products

Developed for a single customer according to their specification.

The user (or person paying for the software) controls the specification.

Example:

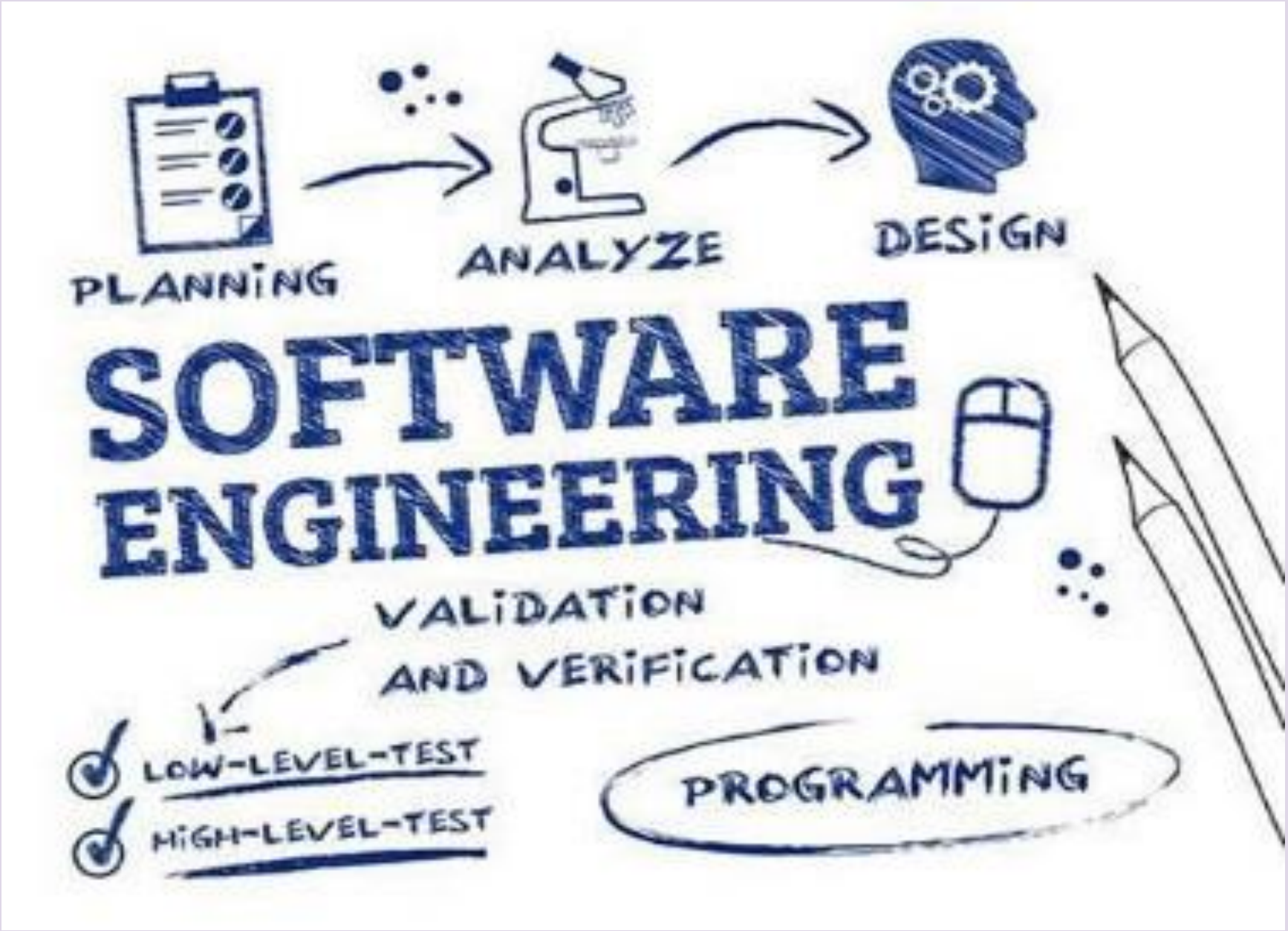
—Air traffic control systems

Computer Software Definition

It is the product that software engineers design and build. It contains **programs** that execute within a computer of any size and architecture, **documents** that include hard-copy and **data** that combine numbers and text, **video**, and **audio** information.

Software Engineering Definition

- The systematic of engineering approaches to the design and construction of computer programs.
- Software engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements.
- The **use of engineering principles** (methods) in order to obtain software that is reliable and works on real machines.


PLANNING
ANALYZE
DESIGN
SOFTWARE ENGINEERING
VALIDATION AND VERIFICATION
LOW-LEVEL-TEST
HIGH-LEVEL-TEST
PROGRAMMING

The characteristics of software engineer

1- Good programmer and fluent in one or more programming language.

2- Well knowledgeable in data structure and approaches.

3- Familiar with several designs approaches.

4-Be able to translate not clear requirements and desires into accurate specification (مواصفات).

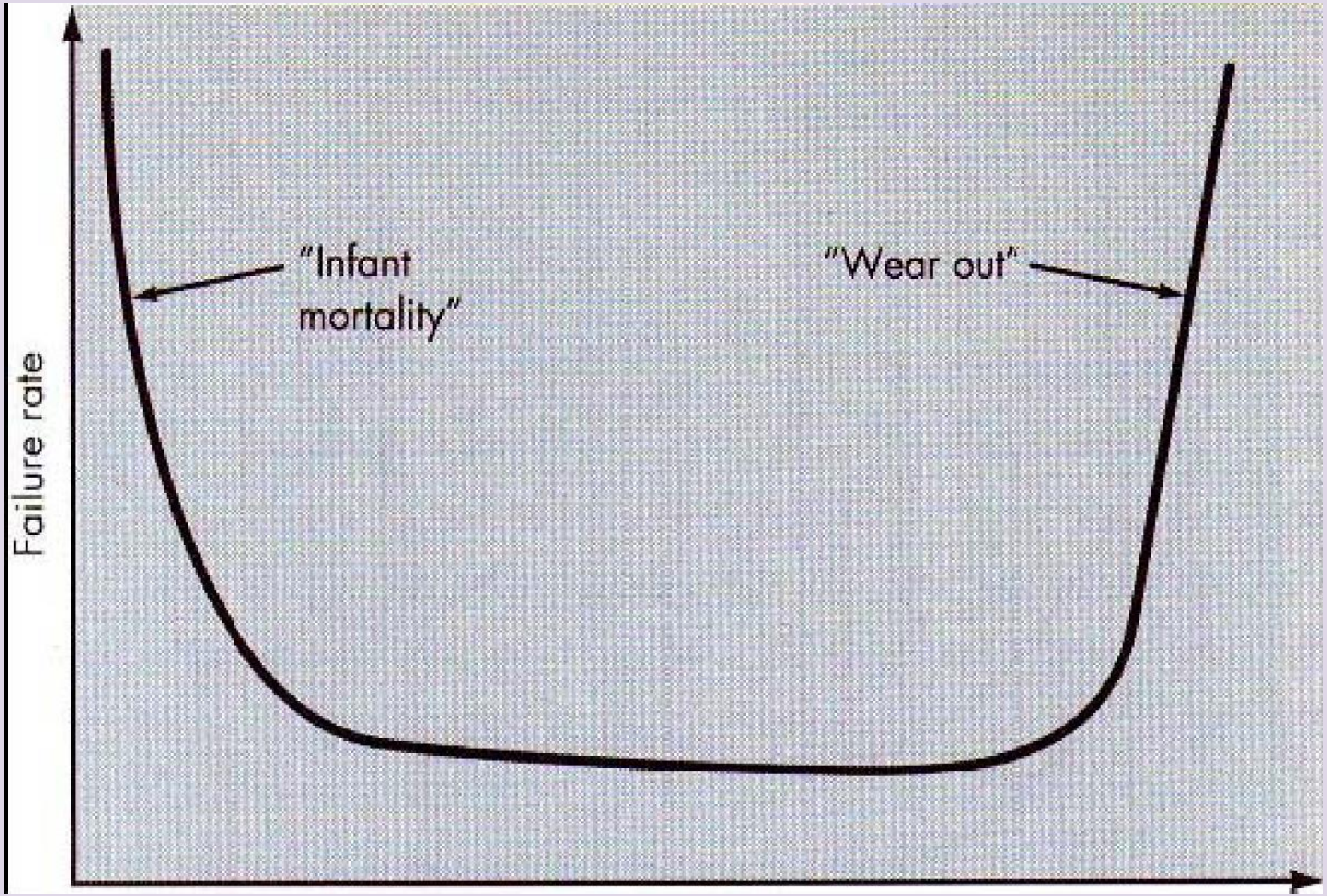
5- Able to a build a model.

6- Communication skills and interactive skills.

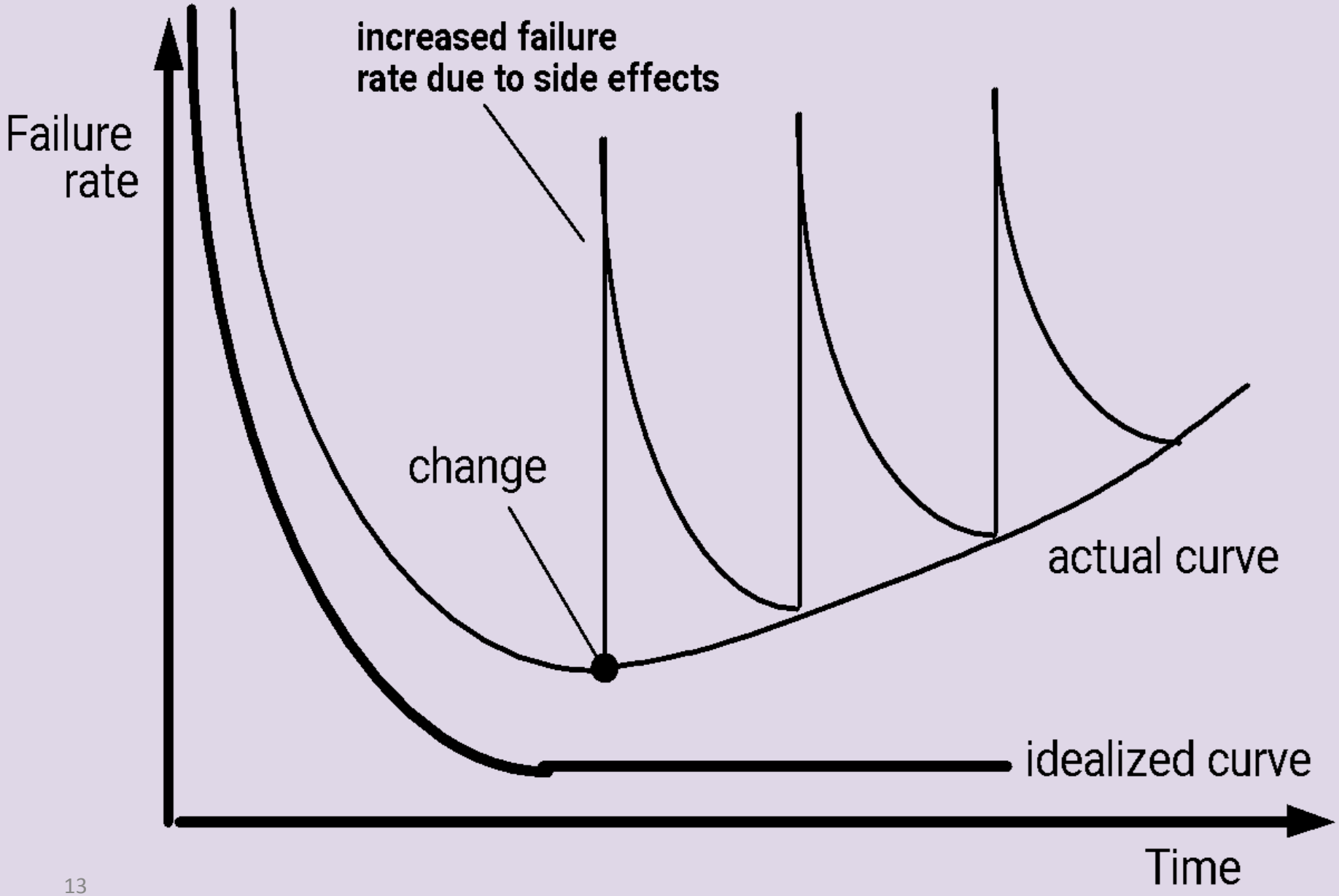
Software Characteristics

1. **Software is developed or engineered**; it is not manufactured in the classical sense. Some similarities exist between software development and hardware manufacture. In both activities, **high quality** is achieved through **good design**. Software costs are concentrated in engineering.
2. **Software doesn't "wear out“** يتآكل

Failure curve for hardware



Idealized and actual failure curves for software



Software Characteristics

When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves more complexity than hardware maintenance.

Software Characteristics

3. Software continues to be custom built.

A software component should be designed and implemented so that it can be **reused** in many different programs(algorithms, data structure and encapsulation).

Software Engineering

م .م ايناس إسماعيل عمران

Software Chapter One: Introduction to Engineering

1.1 The Computer Software Definition

1.2 Software Engineering Definition

1.3 The characteristic of software engineer

1.4 Software Characteristics

1.5 Software Applications

1.6 Software: A crisis on the horizon

1.7 The Characteristics of Well-Engineered Software

1.8 The Goals of Software Engineering

Software Applications

1.System software: : is software designed to provide a platform for other software or is a type of computer program that is designed to run a computer's hardware and application programs. Hence ;system software is the interface between the hardware and user applications) Examples of system software include operating systems like mac OS, Linux, Android and Microsoft Windows, computational science software, game engines, search engines.

Software Applications

2. Real-time software:

is a time-bound system which has well-defined, fixed time constraints. For example: Online gaming.

3. Business software: Business information processing is the largest single software application area. (e.g., payroll, accounts receive/pay).

4. Engineering and scientific software: Computer-aided design (CAD), system simulation, and other interactive applications have begun to take on real-time.

Software Applications

5. **Embedded software:** Intelligent products have become common place in nearly every industrial market. Example) keypad control for a microwave(.

6. **Personal computer software:** Such as (Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management).

7. **Web-based software:** The Web pages retrieved by a browser (e.g., HTML, Perl, or Java).

8. **Artificial intelligence software:** It makes use of non-numerical algorithms to solve complex problems. (e.g., Expert system games playing are representative of applications within this category).

Software: A crisis on the horizon الازمات

The term refers to a set of problems that are appears in the development of computer software.

Some "crisis" issues:

- cost of hardware/software
- "Wrong" products
- Poor quality
- Constant maintenance
- slow

Well-Engineered The Characteristics of Software

1- **Maintainability:** software should be written in such a way that it may evolve to meet the changing needs of customer. This is critical attribute because software change is an usual requirement of a changing business environment.

2- **Dependability:** software dependability has a range of characteristics, including reliability, security and safety.

Well-Engineered The Characteristics of Software

- 3- **efficiency:** software should not make wasteful use of system resources, such as memory and processor cycles. Therefore efficiency includes responsiveness, processing time, memory utilization etc.
- 4- **Usability:** software must be usable, without under effort by the type of user for whom it is designed. This means that it should have an appropriate user interface and adequate documentation.

The Goals of Software Engineering

1.User Satisfaction:

This is the first goals of software engineering and also the most important goal while developing any software is focused about user satisfaction.

2. High reliability:

Code of software must be reliable so that cant have any mistake or bugs in final product which is going to release for end user.

The Goals of Software Engineering

3. Delivery on time

The delivery time matters while develop any software for client or customer. It is hard to determine the exact time to complete software but if the developing software is done by using a systematic order and breaking the whole project in parts with estimating time for each module. By accomplish this steps a proposed deadline can be given to completing a project for a client.

The Goals of Software Engineering

4-Low Production Cost:

software which is cost-effective for develop and maintain gets always the attention of users and if the software is succeeded to match the user requirement there is a big chance of sale or profit in either way.

How should you make software ?engineer goals

- Specific: Clear and well-defined.
- Measurable: Easy to track.
- Attainable: Feasible with your resources.
- Relevant: Appropriate for your ambitions, field, or company.
- Time-bound: Achievable within a preset time frame.

Software Development Model

م.م ايناس إسماعيل عمران

Chapter Two: Software Development Model

Topics:

2.1 Software Lifecycle

2.2 Linear Sequential Waterfall Model

2.3 Prototyping Model

2.4 Incremental Model

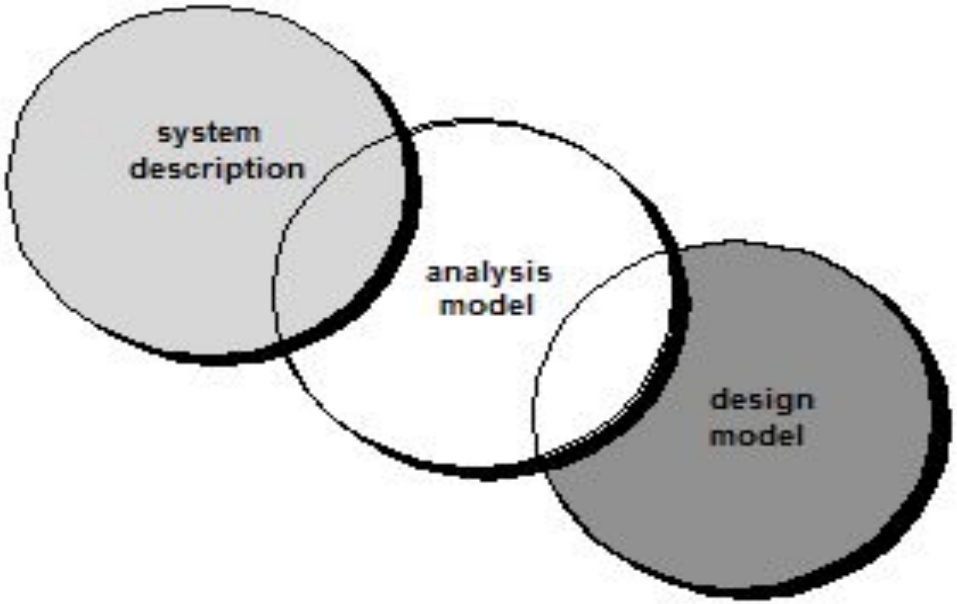
2.5 Spiral Model

2.5 Others Model

Software Lifecycle

Is a process that produces software with the highest quality and lowest cost in the shortest time possible.

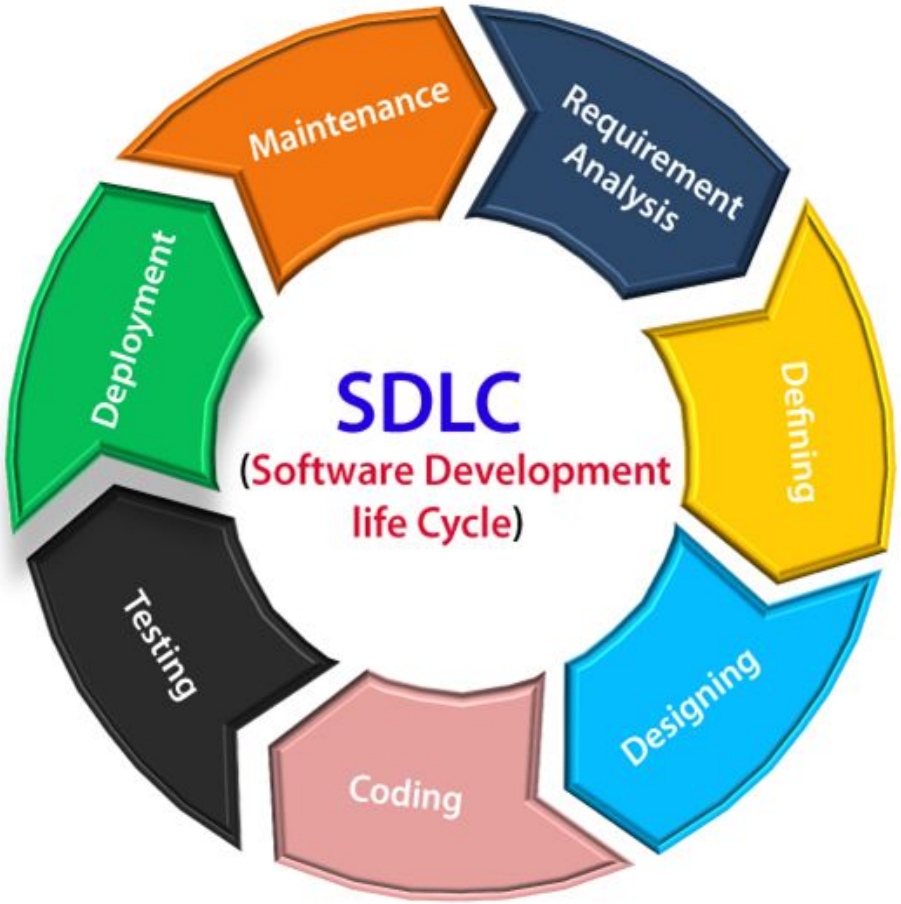
SDLC provides a well-structured flow of phases that help an organization to quickly produce high-quality software which is well-tested and ready for production use.



Software Lifecycle

Each software product proceeds to a number of stages, :these are

Requirements engineering •
Software design •
Software construction •
Software testing •
Software deployment •
Software maintenance •



Requirements Engineering Components

- **Requirements gathering**

  (also known as Requirements elicitation) is the process of generating a list of requirements (functional, system, technical, etc.) from the various stakeholders (customers, vendors, IT staff, etc.) that will be used as the basis for the formal Requirements Definition.

  - **Requirements analysis**

  refining and modifying the gathered requirements

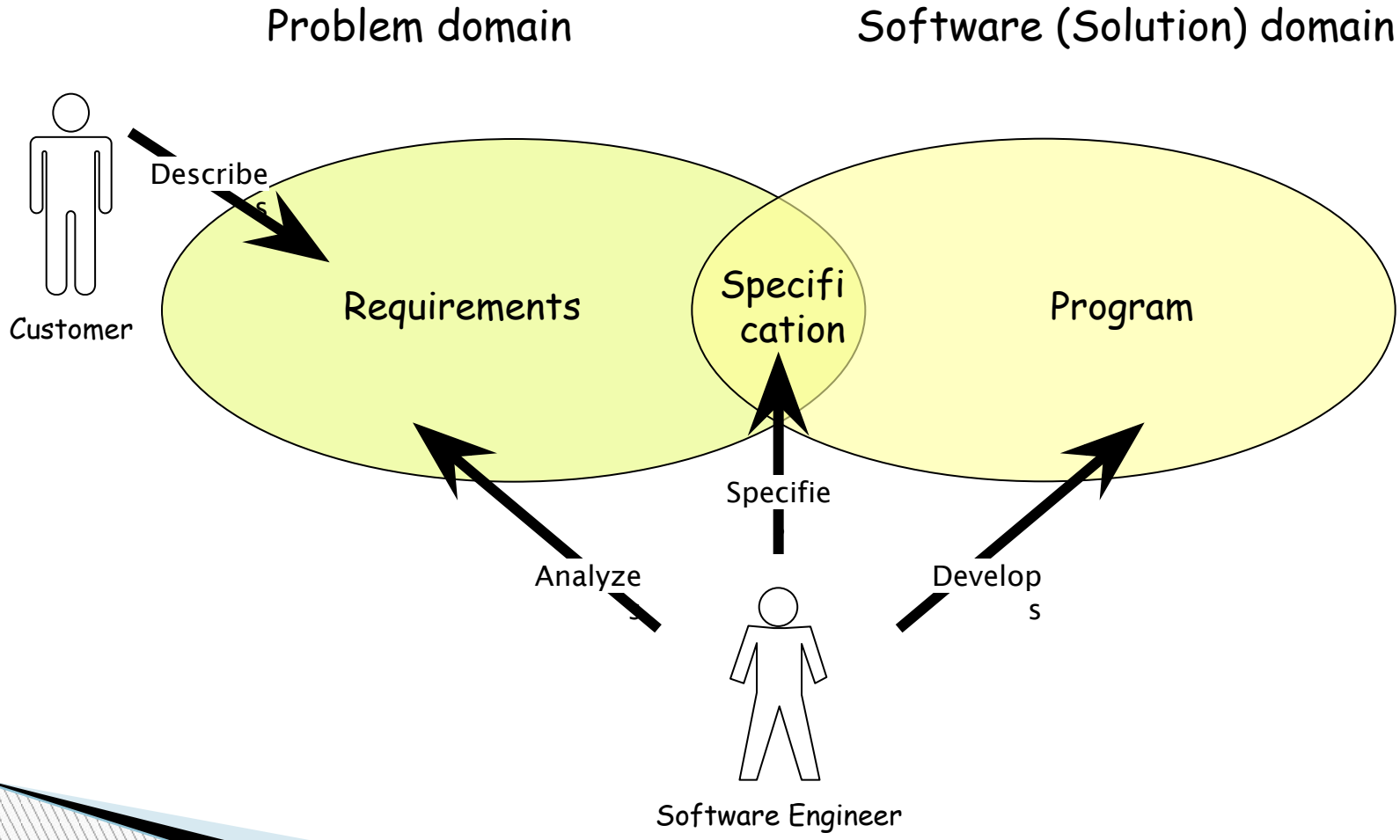
- **Requirements specification**

documenting the system requirements in a formal manner to ensure clarity, reliability, and completeness

STRUCTURE OF SRS

Chapter no. 1	**Preface**	**It briefly explains about project.**
Chapter no. 2	Introduction	Highlights the projects with its title and briefly describe the projects.
Chapter no. 3	Scope	What is the capability of the product?
Chapter no. 4	Glossary	Definition, acronyms and abbreviation.
Chapter no. 5	User requirement definition	Describes non-functional requirements
Chapter no. 6	Architecture	Specifies system architecture
Chapter no. 7	System requirements	System description with function and non-function requirement.
Chapter no. 8	System model	System model used to represent relationship.
Chapter no. 9	System evaluation	How system is evolved?
Chapter no. 10	Appendices	Annexure, application, data requirements.
Chapter no. 11	indexes	Indices of diagram, tables, functions.

Requirements and Specification



Software Lifecycle

1. **Requirements Engineering:** (requirement analysis and definition by using engineering approach) Requirements engineering is the interface between customers and developers on a software project. Requirements should make explicit the ideas of the customer about the system.

2. **Software Design:** the designers convert the software requirements from stage 1 into a software design by describe the software in such a way that programmers can write line of code that implement what the requirements specify.

3. **Software Construction:** is concerned with implementing the software design by means of programs in one or more programming languages. This stage content several steps, these are :

A- Software reuse: (writing units of source code, which can be reused in other projects). Building software systems from prefab software components is an old dream of software engineering .

b. Security and reliability: software must be dependable by making it reliable (work very well under any environments), secure and safety.

c. Software documentation: (User documentation, Technical documentation and Documentation generation).

d. Coding Standards: ensure code maintainable by others than the original developer.

4. Validation and Verification

a. Software Inspections

Are reviews of the code with the purpose of detecting defects. In an inspection someone other than the programmer reads a program unit to determine whether it satisfies the requirements and specification.

b. Software Testing:

Testing each unit founded in this software, follow by testing software integration.

5. Software Deployment:

After development, software should be put to use (available to users, who can then **download, install, and activate it**).

Activities make up the SW deployment process are:

- Packaging
- Release
- Installation
- Configuration
- Activation
- De-activation
- Update
- De-installation
- De-release

6. **Software Maintenance:** after first release, software maintenance is needed to improve it (repair defects), and to extend it (add new functionality).

Software Development Model

اعداد

ايناس إسماعيل عمران

The software process

A structured set of activities required to develop a software system

- Specification
- Design
- Construction
- Validation
- Evolution

A software process model is a representation of a process. It presents a description of a process from some particular view

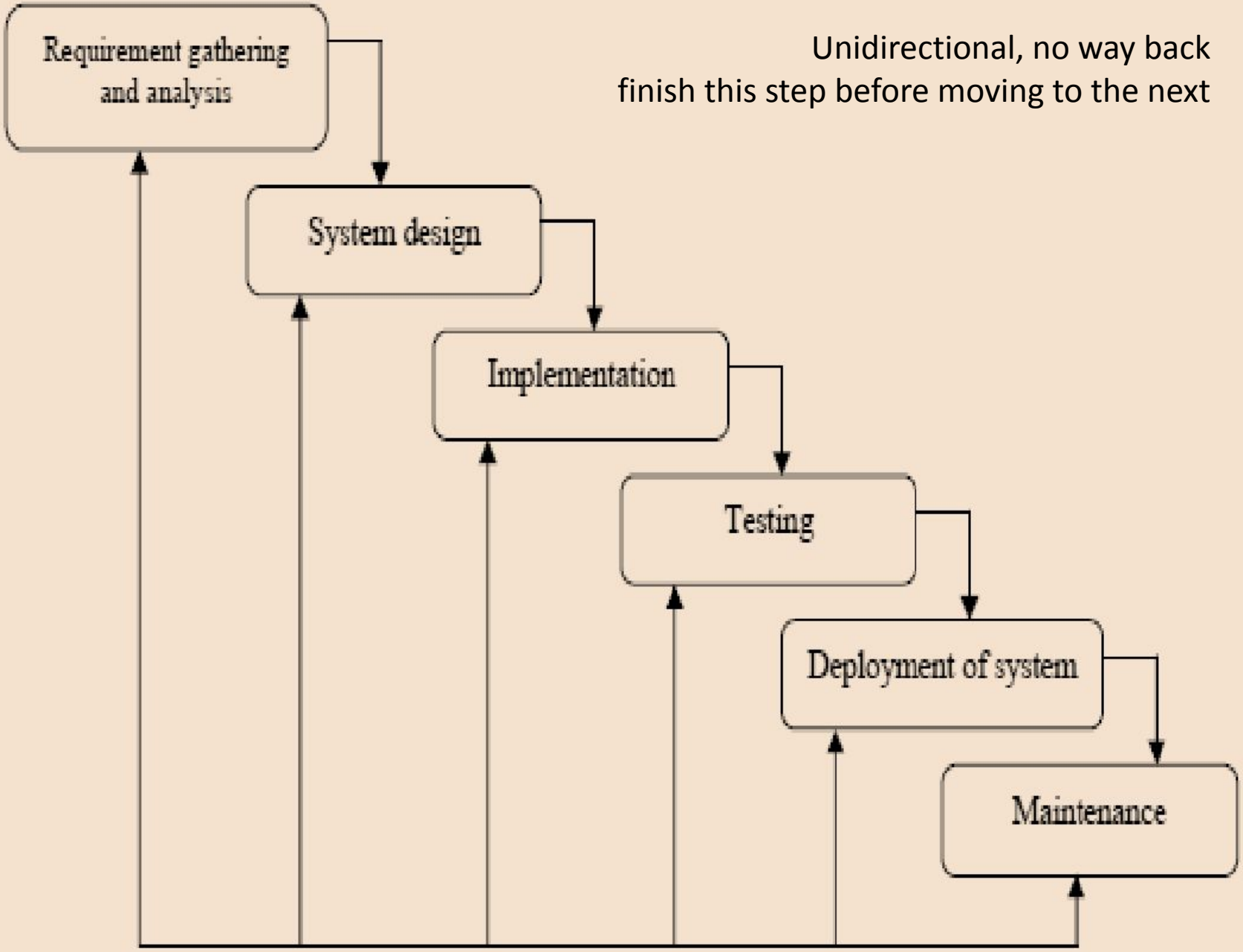
SW Process Models

1-Linear Sequential Waterfall Model

Waterfall approach was first Process Model to be introduced and followed widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, The whole process of software development is divided into **separate process** phases, these are:

1) Requirement Specifications (analysis and definition).
2) Software Design.
3) Implementation.
4) Testing.

.Maintenance (5



Unidirectional, no way back
finish this step before moving to the next

Waterfall Model process

1. Requirement Analysis & Definition:

All possible requirements of the system to be developed are captured in this phase. **Requirements** are set of functionalities and constraints that the end-user expects from the system.

The requirements are gathered from the end-user by meeting, these requirements are analyzed for their validity. Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

Waterfall Model process

2. System & Software Design:

Before a starting for actual coding, the requirement specifications from first phase are studied in this phase and system design is prepared.

System Design helps in specifying hardware and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

Waterfall Model process

3- Implementation & Unit Testing:

On receiving system design documents, the work is divided in modules/units and actual coding is started. The system is first developed in small programs called units, which are integrated in the next phase.

Each unit is tested for its functionality; this is referred to as **Unit Testing**. Unit testing mainly verifies if the modules/units meet their specifications.

Waterfall Model process

3- Implementation & Unit Testing:

On receiving system design documents, the work is divided in modules/units and actual coding is started. The system is first developed in small programs called units, which are integrated in the next phase. Each unit is tested for its functionality; this is referred to as **Unit Testing**. Unit testing mainly verifies if the modules/units meet their specifications.

Waterfall Model process

4- Integration & System Testing:

As specified above, the system is first divided in units which are developed and tested for their functionalities. These units are integrated into a complete system during Integration phase and tested to check if all modules/units coordinate between each other as a whole system. After success testing the software, it is delivered to the customer.

5) Maintenance: Generally, the issues related to the system are solved after deployment of the system. Not all the problems come in picture directly but they arise time to time and needs to be solved; hence this process is referred as Maintenance.

Disadvantages of the Waterfall Model

1. Not all requirements are received at once.
2. The problems with one phase are never solved completely during that phase.
3. The project is not separated in phases in flexible way.
4. As the requirements of the customer goes on getting added to the list, not all the requirements are satisfied, this results in development of almost unusable system.
5. difficulty to change after the process is started

Therefore, this model is only appropriate when the requirements are well-understood

SOFTWARE MODELS

م.م أيناس إسماعيل عمران

THE PROTOTYPE MODEL

Is a systems development method in which a prototype is built, tested and then rewritten as necessary until an acceptable outcome is achieved.

The prototype model is using for many reasons, such as:

1. A customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements.

2. The developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, In these situations, a prototyping model may offer the best approach.

THE PHASES OF THE PROTOTYPING :MODEL

1) **Requirements gathering (listen to customer):** developer and customer meet and define the requirements of the system in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

2) Quick design

The second phase is a primary design or a quick design. In stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype

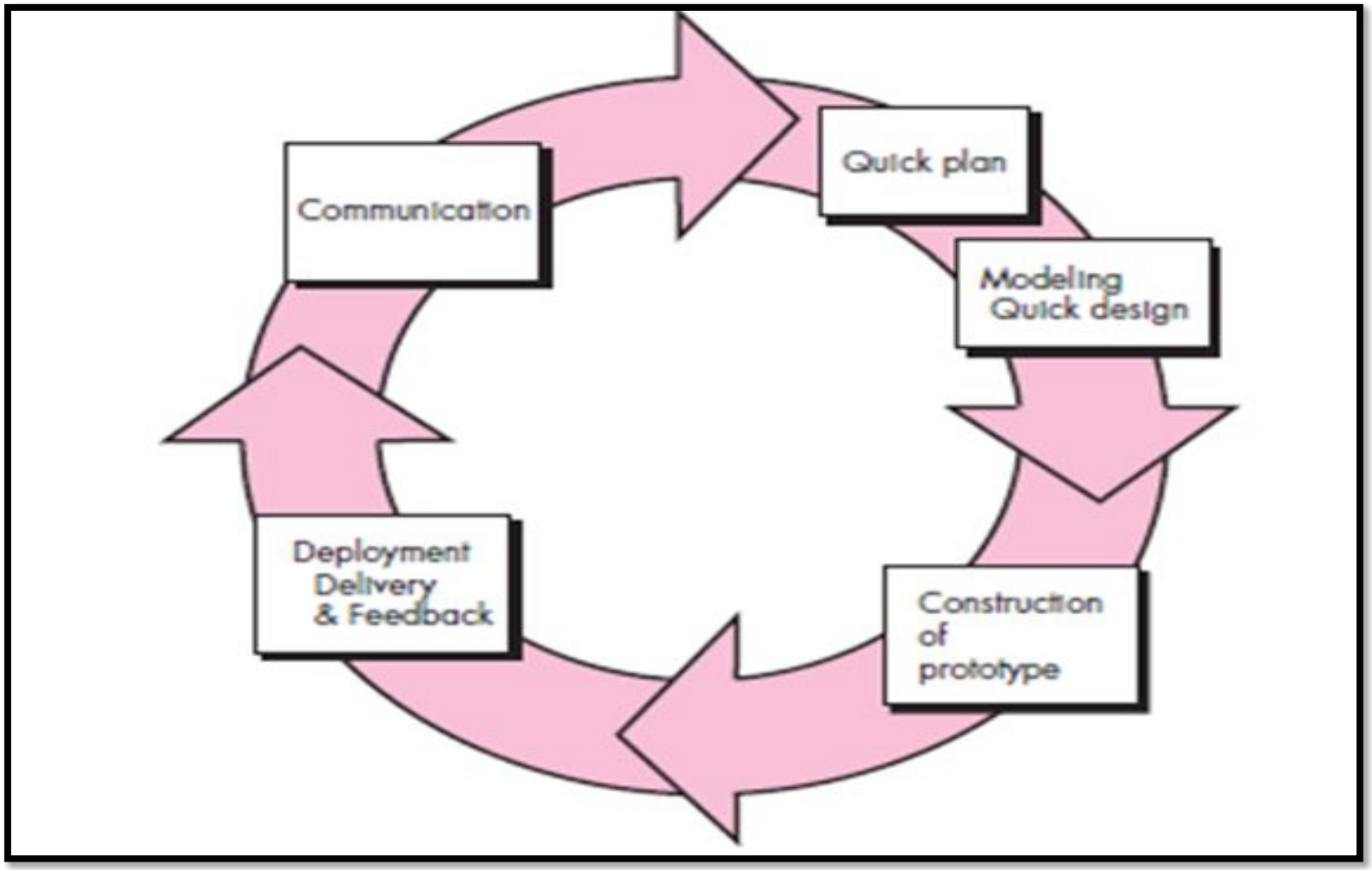
.

THE PHASES OF THE PROTOTYPING :MODEL

3) **Construction of a prototype**: writing the software code depending on the quick design information.

4) Evaluation: the prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is turned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what requirements to be done.

:Figure of prototyping model



ADVANTAGES OF THE PROTOTYPING MODEL

- Users are involved in development. Therefore, errors can be detected in the initial stage of the software development process.
- Missing functionality can be identified, which helps to reduce the risk of failure.
- Helps team member to communicate effectively
- Customer satisfaction exists because the customer can feel the product at a very early stage.
- There will be hardly any chance of software rejection.

ADVANTAGES OF THE PROTOTYPING MODEL

- Allows the client to compare if the software code matches the software specification.
- No need for specialized experts to build the model
- The prototype helps to gain a better understanding of the customer's needs.
- Prototypes can be changed and even discarded.
- Quicker user feedback helps you to achieve better software development solutions.

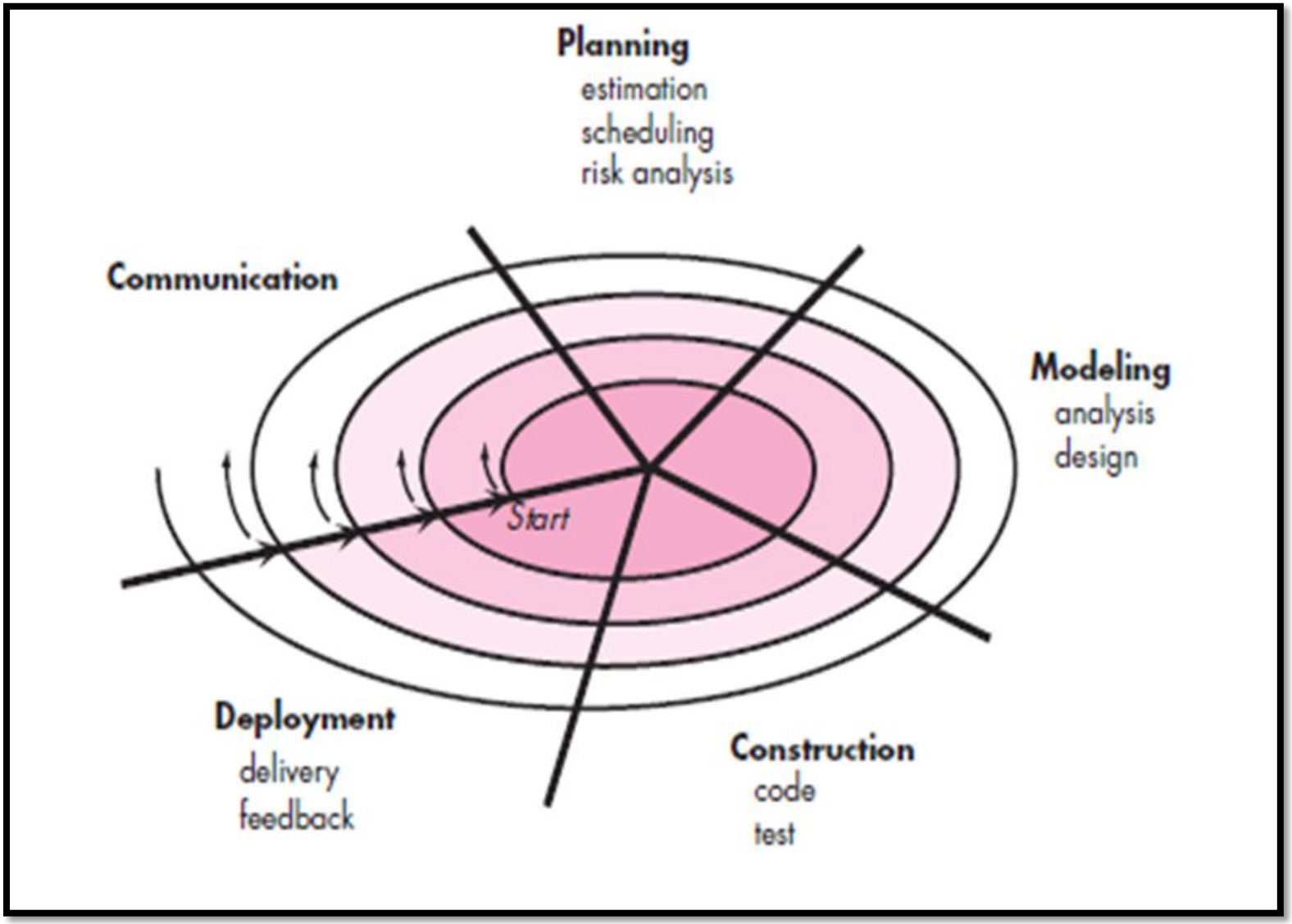
DISADVANTAGES OF THE PROTOTYPING MODEL

- Prototyping is a slow and time taking process.
- The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
- Prototyping may encourage too much change requests.
- Some times customers may not be willing to participate in the iteration cycle for the longer time duration.
- It is very difficult for software developers to accommodate all the changes demanded by the clients.

THE SPIRAL MODEL

Spiral model, is an evolutionary software process model that **couples** the **iterative nature of prototyping** with the controlled and **systematic aspects of the linear sequential model**. Using the spiral model, software is developed in a series of incremental releases. During **early** iterations, the incremental release might be a **paper model** or prototype. During **later** iterations, **more complete** versions of the engineered system are produced.

The Spiral Model Phases



Software Development Model

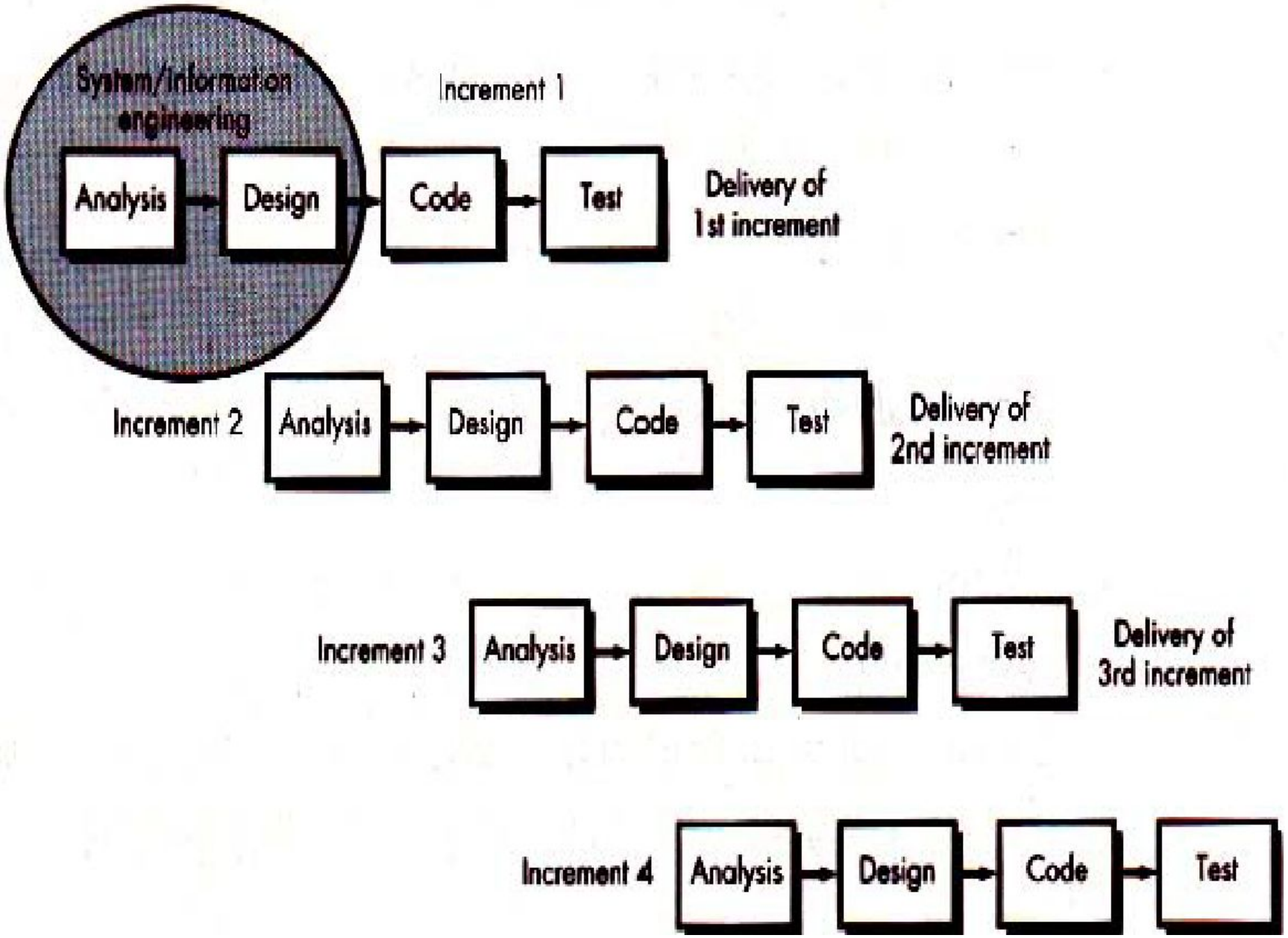
اعداد

ايناس إسماعيل عمران

The Incremental Model

The incremental model combines elements of the (waterfall model with the iterative philosophy of prototyping). Each linear sequence produces "increment" of the software. For example, word-processing software developed using the incremental might:

1) Deliver basic file , editing, and document production functions in the first increment.
2) More editing and document production capabilities in the second increment.
3) Spelling and grammar checking in the third increment.
4) Advanced page layout capability in the fourth increment.


System/information engineering
Increment 1
Analysis
Design
Code
Test
Delivery of 1st increment
Increment 2
Analysis
Design
Code
Test
Delivery of 2nd increment
Increment 3
Analysis
Design
Code
Test
Delivery of 3rd increment
Increment 4
Analysis
Design
Code
Test

- The first increment is often *a **core product***. That is, basic requirements are addressed, but many additional features remain undelivered. The core product is used by the customer. As a result of use and/or evaluation, a plan is developed for the next increment. But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment.

- User requirements are listed and the highest priority requirements are included in early increments
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development advantages

- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change requirements.
- Easier to test during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration
- Lower risk of overall project failure
- By using this model client is able to respond to every built

Problems

- For incremental model, required good designing and well planning.
- The complete cost of this model is higher than waterfall model.
- Special skills (e.g. in languages JavaScript) may be required.

Applicability

- For small or medium-size interactive systems
- For parts of large systems (e.g. the user interface)
- For short-lifetime systems

SOFTWARE MODELS

م.م ايناس إسماعيل
عمران

Spiral Model

Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. spiral model contain many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a **Phase of the software development process**. Cost and schedule are adjusted based on feedback derived from customer .In addition, the **project manager** adjusts the planned number of iterations required to complete the software. hence; the project manager has an important role to develop a product using spiral model.

A spiral model is divided into six task regions:

1. **Customer communication-tasks** required to establish effective communication between developer and customer.
2. **Planning-tasks** required defining resources, timelines, and other project-related information.
3. **Risk analysis-tasks** required to assess both technical and management risks.
4. **Engineering-tasks** required building one or more representations of the application.

5.**Construction and release-tasks** required to construct, test code , install, and provide user support (e.g., documentation and training).

6.**Customer evaluation-tasks** required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

:Advantages of Spiral Model

1. Software is produced early in the software life cycle.
2. Risk handling is one of important advantages of the Spiral model, it is best development model to follow due to the risk analysis and risk handling at every phase.
3. Flexibility in requirements. In this model, we can easily change requirements at later phases ,Also, additional Functionality can be added at a later date.
4. It is good for large and complex projects.
5. It is good for customer satisfaction. We can involve customers in the development of products at early phase of the software development.

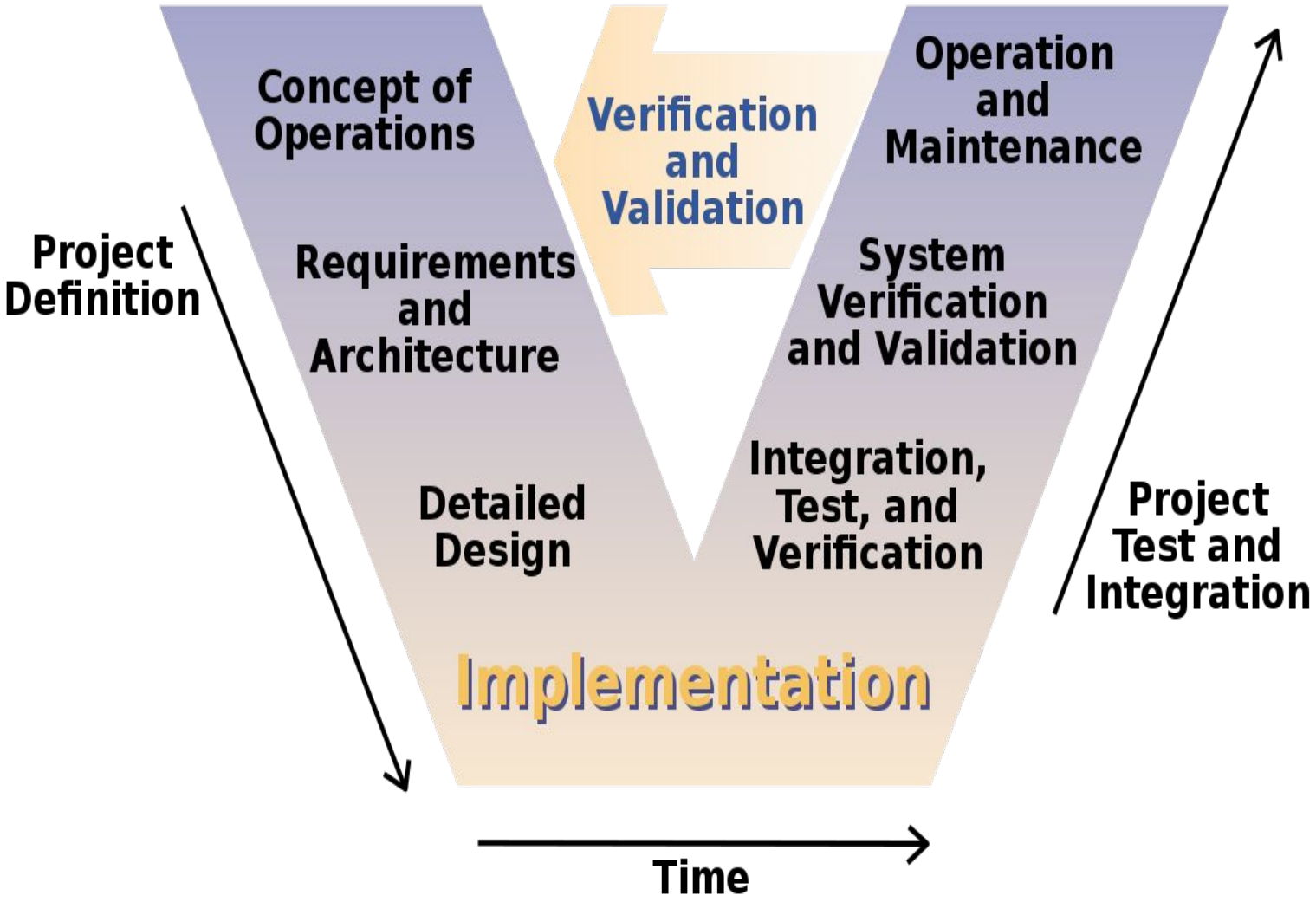
:Disadvantages of Spiral Model

1. It is not suitable for small projects as it is expensive.
2. It is much more complex than other SDLC models. Process is complex.
3. Too much dependable on Risk Analysis and requires highly particular skill.
4. Difficulty in time management. As the number of phases is unknown at the start of the project, so time estimation is very difficult.
5. The model has not been used as widely as the linear sequential or prototyping model hence, it is difficult to predict the phase (iteration) exactly

V-Shaped Model

1. Also known as the Verification and Validation model, Validation is the process of checking whether the specification captures the customer's needs, while verification is the process of checking that the software meets the specification
2. the V-shaped model is develop model of Waterfall and is characterized by a testing phase for each development stage. Like Waterfall, each stage begins only after the previous one has ended.
3. This model is useful when there are no unknown requirements, as it’s still difficult to go back and make changes.

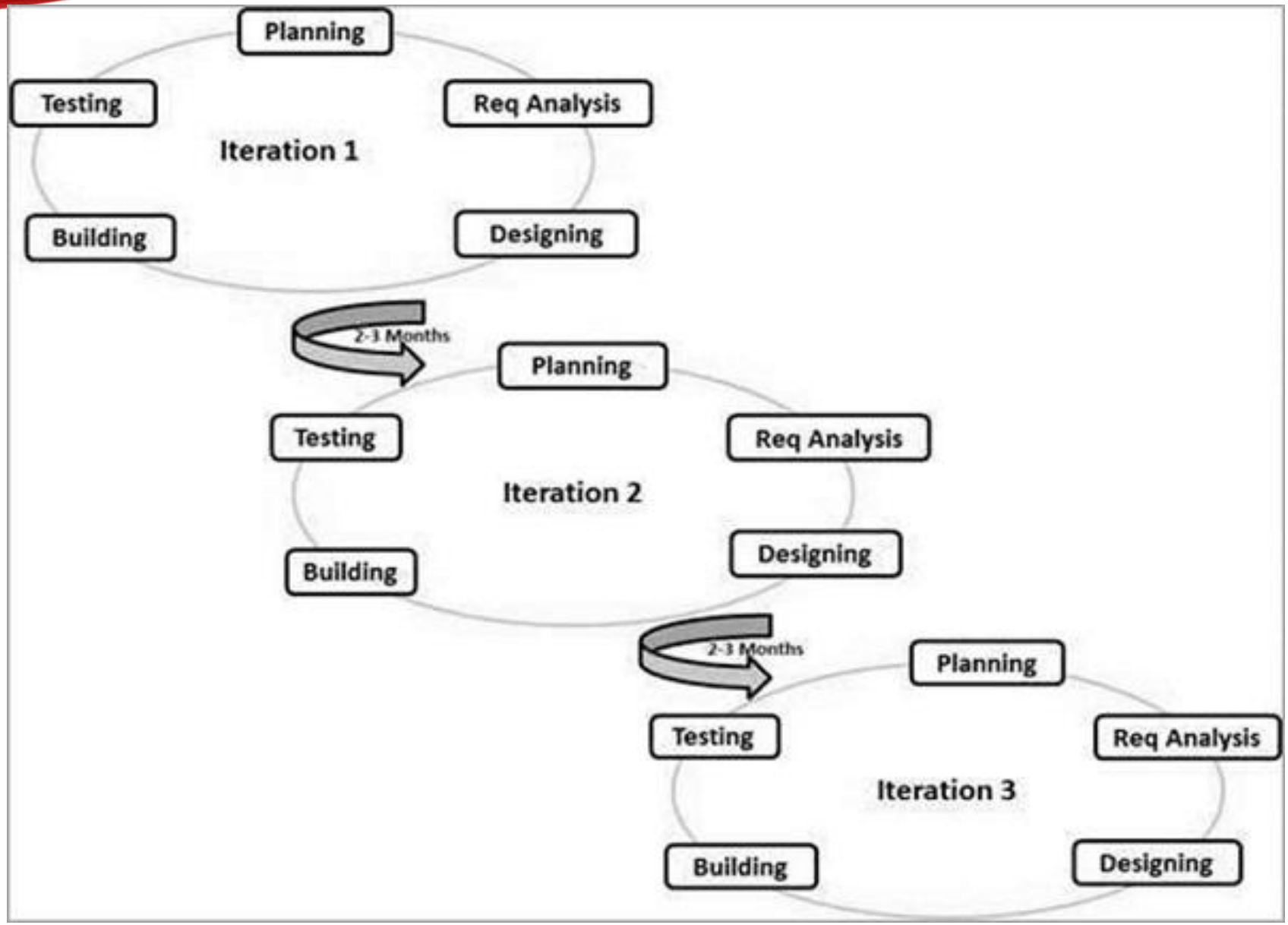
V-Shaped Model



Agile Model

1. Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

2. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three months. The model produces uncompleted releases, so the incremental changes from the previous release. At each iteration, the product is tested. Every iteration need functional teams (customers, developers and testers) work together throughout the project


Planning
Testing
Req Analysis
Iteration 1
Building
Designing
2-3 Months
Planning
Testing
Req Analysis
Iteration 2
Designing
Building
2-3 Months
Planning
Testing
Req Analysis
Iteration 3
Building
Designing

Software Requirements

م.م ايناس إسماعيل عمران

Topics :

3.1 Requirements Analysis and Definition

3.2 Requirements Specification

3.3 Software Requirements

3.4 Software Specification

3.5 Software Requirements Document

Requirements

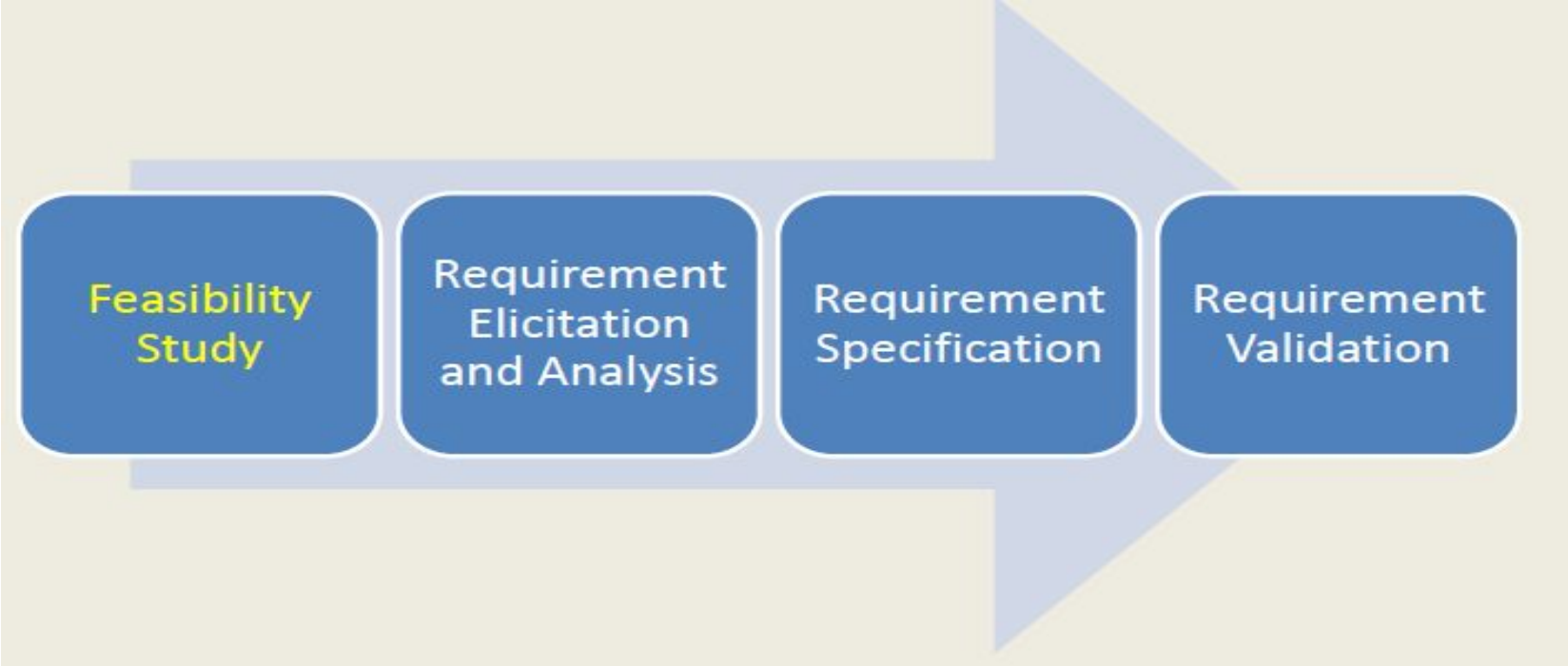
- Features of system or system function used to fulfill system purpose.

- Focus on **customer's needs not** on **solutions**:
  - Requirements **definition** document

    (written for customer).
  - Requirements **specification** document

    (written for programmer; technical staff).

Requirements Engineering Process



Feasibility studies

- A feasibility study decides whether or not the proposed system is worthwhile.
- A short (2-3 weeks) focused study that checks

—If the system contributes to organizational objectives;

—If the system can be engineered using current technology and within budget;

—If the system can be integrated with other systems that are used.

Elicitation and analysis

- Involves **technical staff working with customers** to find out about:

  —the application domain,

  —the services that the system should provide

  —and the system's operational constraints.

- May involve end-users, managers, engineers involved in maintenance, experts, etc. These are called *stakeholders* Domain .

Requirements Analysis and Definition

Software requirements analysis is necessary to avoid creating software product that fails to meet the customer's needs.

Software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be known or unknown, expected or unexpected from client's point of view.

Writing Requirements Definitions

Requirements definitions usually consist of *natural language*, supplemented by (e.g., UML: Unified Modeling Language) *diagrams and tables*.

Three types of problems can arise:

- **Lack of clarity**: It is hard to write documents that are both ***exact and easy-to-read.***
- **Requirements confusion**: ***Functional and non-functional requirements*** tend to be intertwined.
- **Requirements combination**: Several ***different requirements*** may be expressed together.

User requirements

- Statements in natural language **plus** diagrams of the services that the system provides **and** its operational constraints.
- Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge.
- Written for customers

System requirements

- A structured document setting out detailed descriptions of the system services.
- Written for developers

System Req.: Functional and Non-functional Requirements

Functional requirements describe system *services* or *functions* *that a software must perform.*

- Calculation (ex. calculate sales tax).
- data manipulation (ex. update the database on the server) .

Non-functional requirements are *constraints* on the system or the development process. Non-functional requirement is essential to ensure effectiveness of the entire software system .

- constraints in design of the system (**e**x the site should load in 3 seconds when the number of simultaneous users are > 10000)

Software Requirements Analysis

م.م ايناس إسماعيل عمران

Requirements Analysis

- Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must suitable and detailed. In software engineering, such requirements are often called functional specifications. Requirements analysis is an important aspect of project management.
- Provides software designer with a representation of system information and function that can be translated to architectural design.

Software Requirements Analysis Phases

- Problem recognition
- Specification
- Modeling
- Evaluation
- Review

Analysis Principles

- The information domain of the problem must be represented and understood.
- The functions that the software is to perform must be defined.
- Models describe information and function in detail.
- The analysis process should move from the essential information toward implementation details.

Software Requirement Specification (SRS)

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

- SRS defines hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Limitations etc.

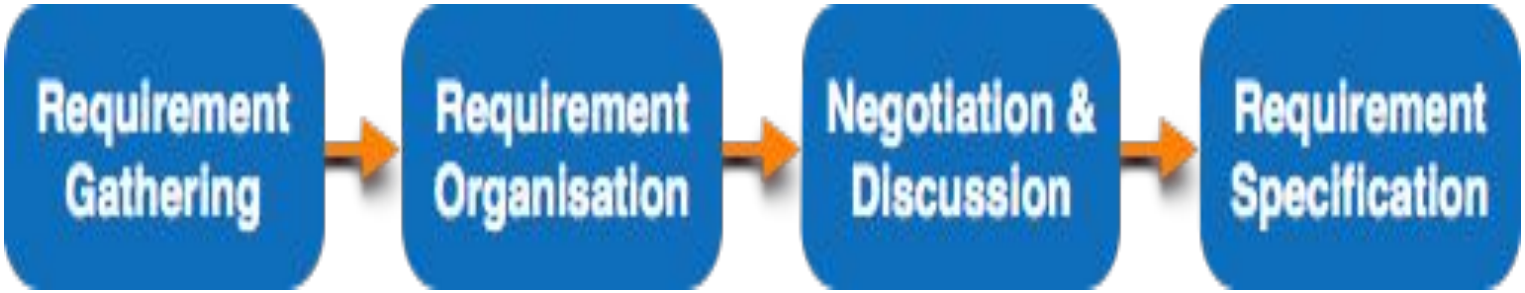
- The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be useful by the software development team.

SRS should come up with following features:

- **User Requirements** are expressed in natural language, (ex:Twenty users can use System at the same time as without noticeable system delays, print on-screen data to the printer).
- **Technical requirements** are expressed in structured language, which is used inside the organization (ex performance, security).
- **Format of Forms** and GUI screen prints.

Requirement Elicitation Process

Requirement elicitation process can be illustrated using the following diagram:



1-Requirements gathering -The developers discuss with the client and know their expectations from the software.

2-Organizing Requirements - The developers arrange the requirements in order of importance, ease of use and suitability.

- **3-Negotiation & discussion** - If there are some conflicts in requirements of various stakeholders, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.
- **4-Documentation** - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

REQUIREMENTS DOCUMENT

م.م ايناس إسماعيل عمران

The Requirements Document

- statement of what is required of the system developers.

- **Should include both a definition and a specification of requirements**
- **Should:**
  - **-- specify implementation constraints**
  - **- be easy to change (but changes must be managed)**
  - **- serve as a reference tool for maintenance**
  - **- record about the life cycle of the system**
  - **- characterize responses to unexpected events**

- **It is not a design document**

REQUIREMENTS DOCUMENT STRUCTURE

- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

USERS OF REQUIREMENTS DOCUMENTS

- **System customers:** Read them back to check that they meet their **needs**; specify **changes** to the requirements
- **Managers:** Use the requirements document to **plan** the system development process
- **System Engineers:** Use the requirements to **understand** what system is to be developed
- **Test Engineers:** Use the requirements to **develop** validation tests for the system
- **Maintenance Engineers:** Use the requirements to help **understand** the system and the relationships between its parts

SOFTWARE REQUIREMENTS CHARACTERISTICS

- Clear
- Correct
- Coherent
- Logical
- Modify
- Arranged
- Unambiguous
- Traceable
- Credible source

Requirement Elicitation Techniques

- Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, system users and others who have a stake in the software system development.
- There are various ways to discover requirements

REQUIREMENT ELICITATION TECHNIQUES

1. Interviews
2. Surveys
3. Questionnaires
4. Task analysis
5. Domain Analysis
6. Brainstorming
7. Prototyping
8. Observation

INTERVIEWS-1

- Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:
- **Structured (closed) interviews,** where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.

INTERVIEWS

- **Non-structured (open) interviews**, where information to gather is not decided in advance, more flexible and less biased.
- **Written interviews**
- **One-to-one interviews** which are held between two persons across the table
- **Group interviews** which are held between groups of participants. They help to uncover any missing requirement as many people are involved.

REQUIREMENT ELICITATION TECHNIQUES

م.م ايناس إسماعيل عمران

REQUIREMENT ELICITATION TECHNIQUES

1. Interviews
2. Surveys
3. Questionnaires
4. Task analysis
5. Domain Analysis
6. Brainstorming
7. Prototyping
8. Observation

2.SURVEYS

- Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

3.QUESTIONNAIRES

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

- A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be ignored.

4.Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

5. Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

6. Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

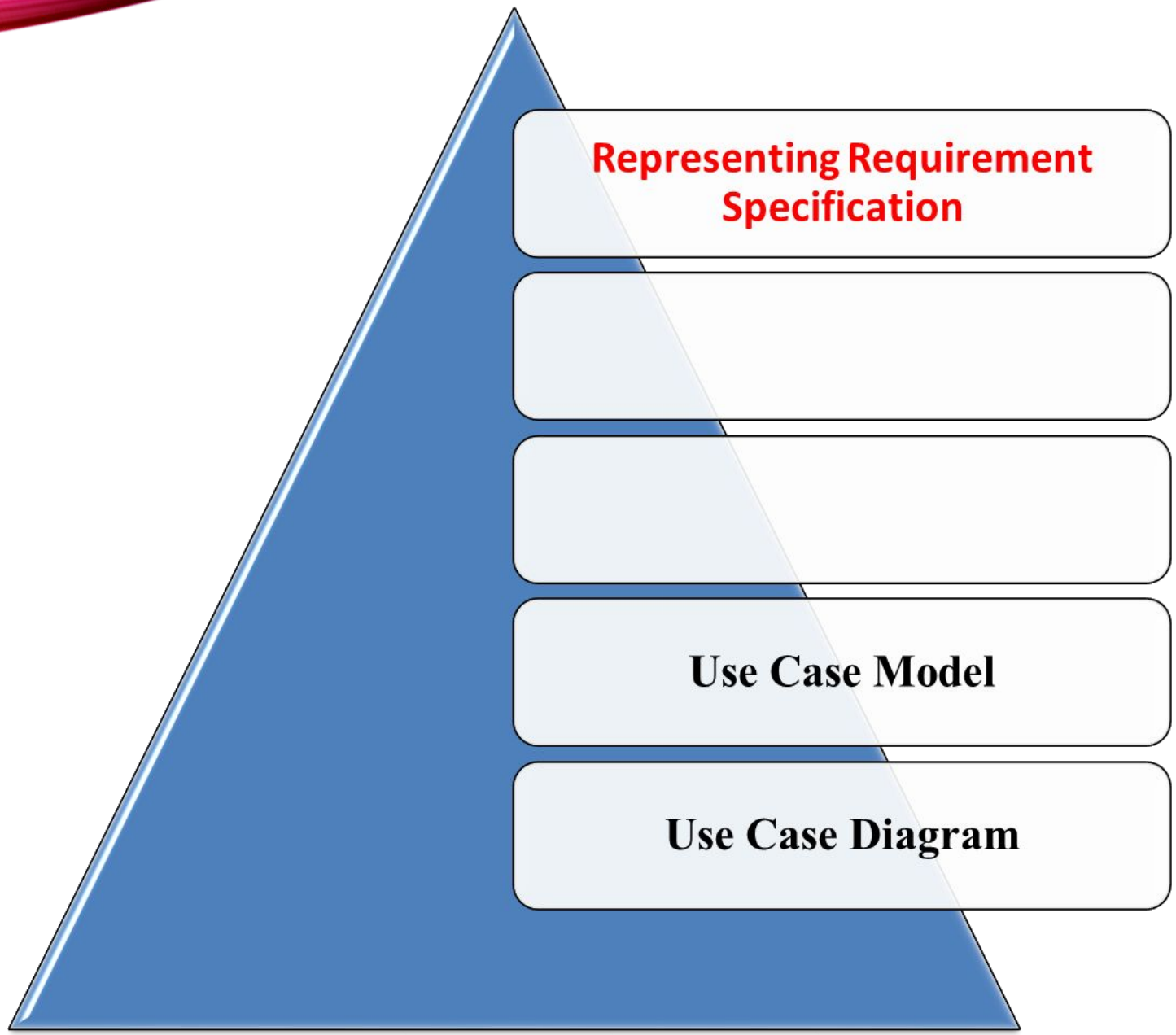
7. Prototyping

Prototyping is building user interface without adding detail functionality for user to understand the features of intended software product.

If there is no software installed and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

8. Observation

Team of experts visits the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.


Representing Requirement Specification
Use Case Model
Use Case Diagram

Use Cases and Scenarios

- A use case is the specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system".
- e.g., buy a DVD through the internet
- A scenario is a particular trace of action occurrences, starting from a known initial state.
- e.g., connect to myDVD.com, go to the "search" page

Use case model

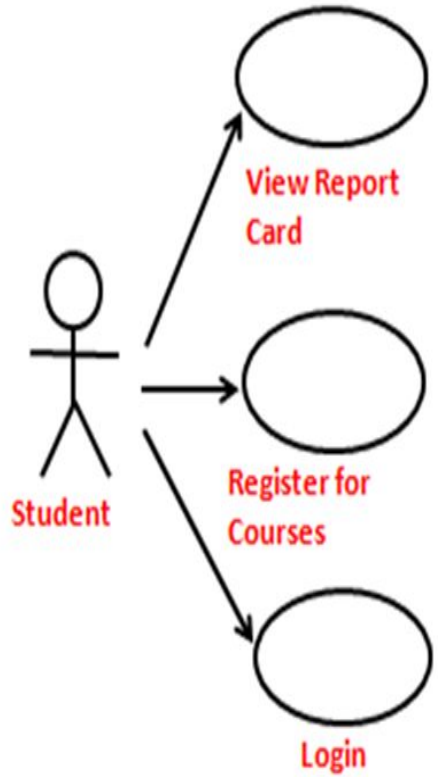
Use-cases are a scenario based technique in the UML (Unified Modeling Language)

—identify the actors in an interaction

—describe the interaction itself

A set of use cases should describe all possible interactions with the system.

use case = a named collection of scenarios



Use – Cases ,Structure Analysis

اعداد

م.م ايناس إسماعيل عمران

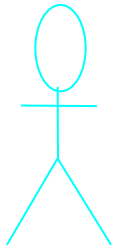
Ex: Use case : Register for courses Description:

Actors: Student, Billing System.

Main success scenario:

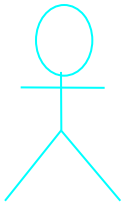
1. The student identifies himself/herself.
2. The system verifies student identity.
3. The student selects a valid semester.
4. The student creates, reviews, or changes a schedule.
5. The systems prints a notification.
6. The system sends billing information to the Billing System

Practice: Solution



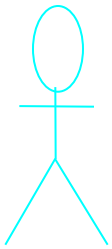
Billing System

The external system responsible for student billing



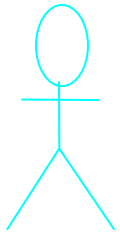
Student

A person who is registered to take courses at the University



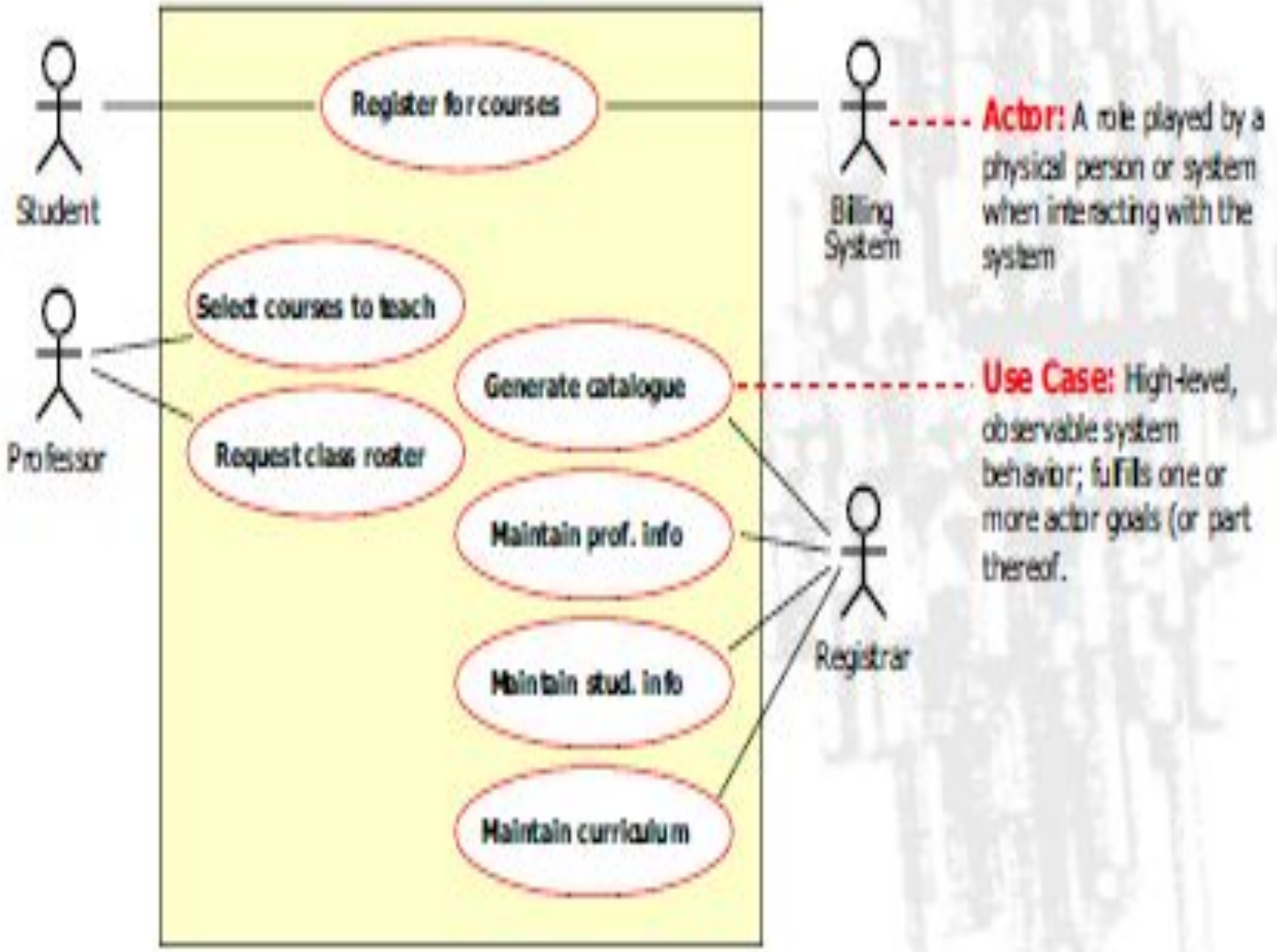
Professor

A person who is teaching classes at the University



Registrar

The person who is responsible for the maintenance of the course registration system


Register for courses
Student
Billing System
Actor: A role played by a physical person or system when interacting with the system
Select courses to teach
Professor
Request class roster
Generate catalogue
Use Case: High-level, observable system behavior; fulfills one or more actor goals (or part thereof.
Maintain prof. info
Registrar
Maintain stud. info
Maintain curriculum

Structure Analysis

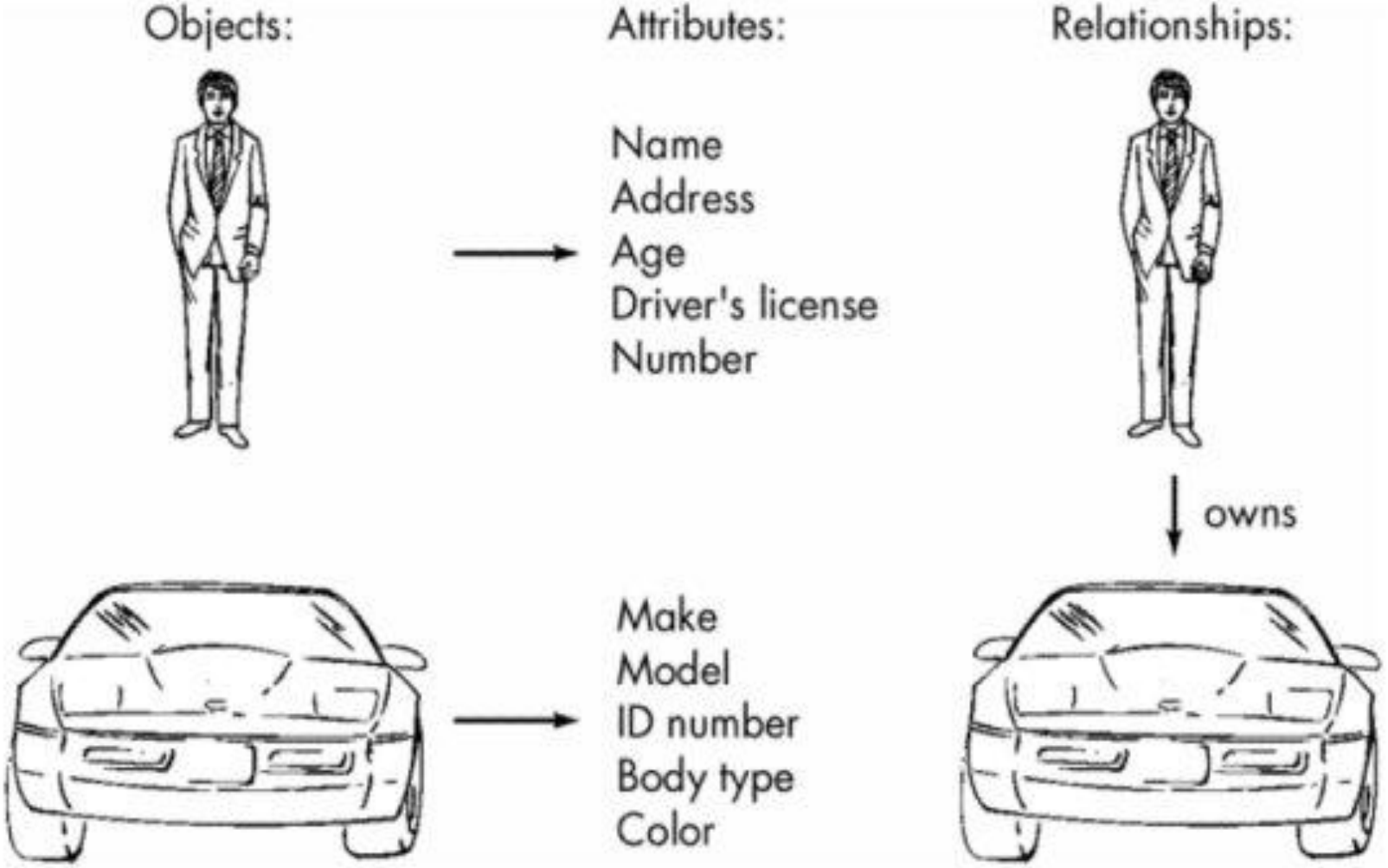
Structured analysis is a software engineering technique that uses **graphical diagrams** to portray system **specifications** that are easily understood by users. These diagrams describe the steps that need to occur and the data required to meet the design function of particular software. This type of analysis mainly **focuses** on logical systems and functions, and **aims** to convert requirements into computer programs and hardware specifications.

Analysis Model Objectives

- Describe what the customer requires.
- Establish a basis for the creation of a software design.
- create a set of requirements that can be validated once the software is built.

The Elements of Analysis Model

- **Data dictionary** - contains the descriptions of all data objects used or produced by the software.
- **Entity relationship diagram (ERD)** –describe relationships between data objects.
- **Data flow diagram (DFD)** - provides an indication of how data are transformed as they move through the system; also describe functions that transform the data flow.


Objects:
Attributes:
Relationships:
Name
Address
Age
Driver's license
Number
owns
Make
Model
ID number
Body type
Color

The Elements of Analysis Model

- **State transition diagram (STD)** - indicates how the system behaves as a consequence of external events, states are used to represent behavior modes. Arcs are labeled with the events triggering the transitions from one state to another .

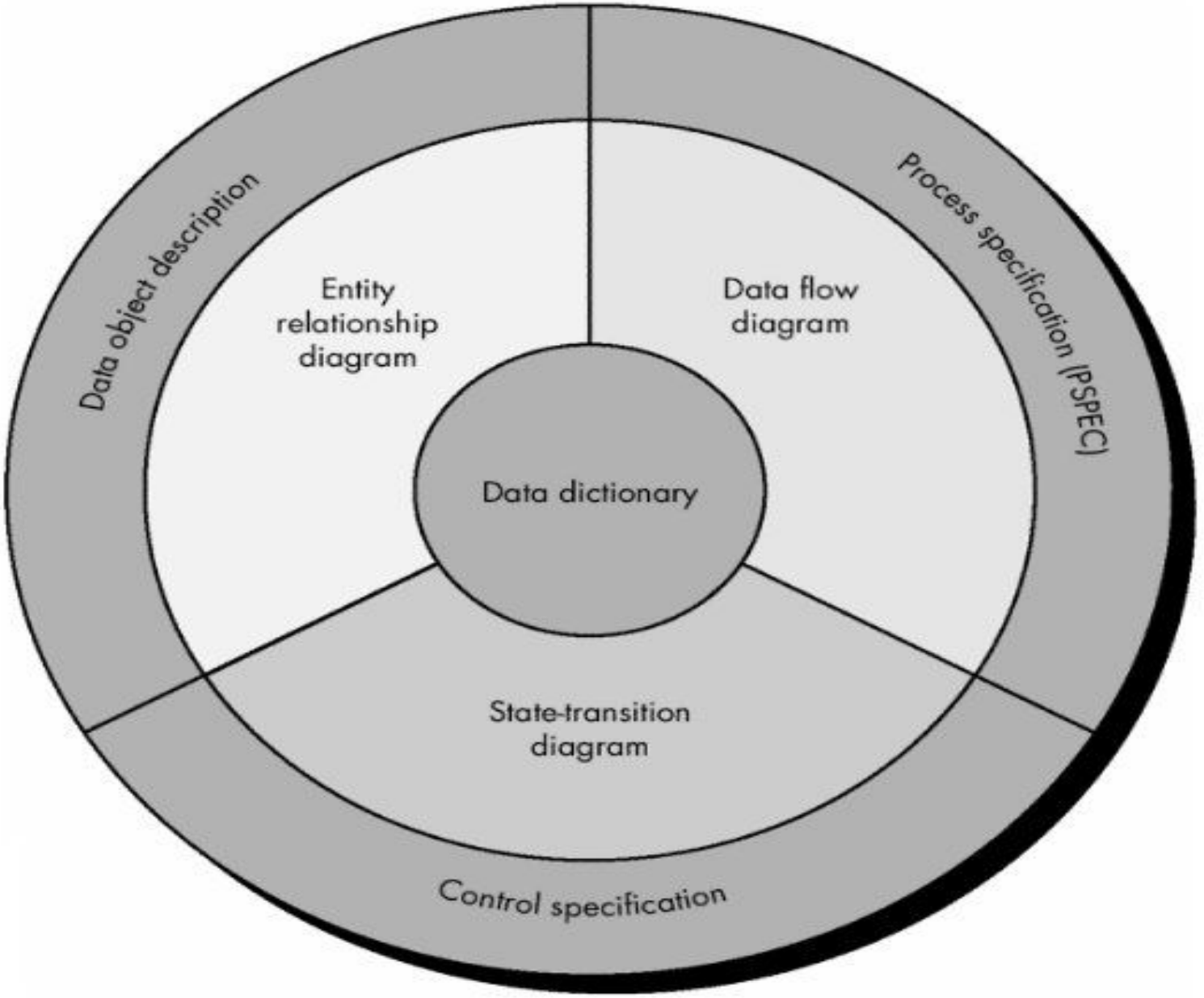


Figure (4.1): The structure of the analysis model

Use- Case- Example

اعداد

م.م ايناس اسماعيل عمران

Use-Case Diagrams: Example [1]

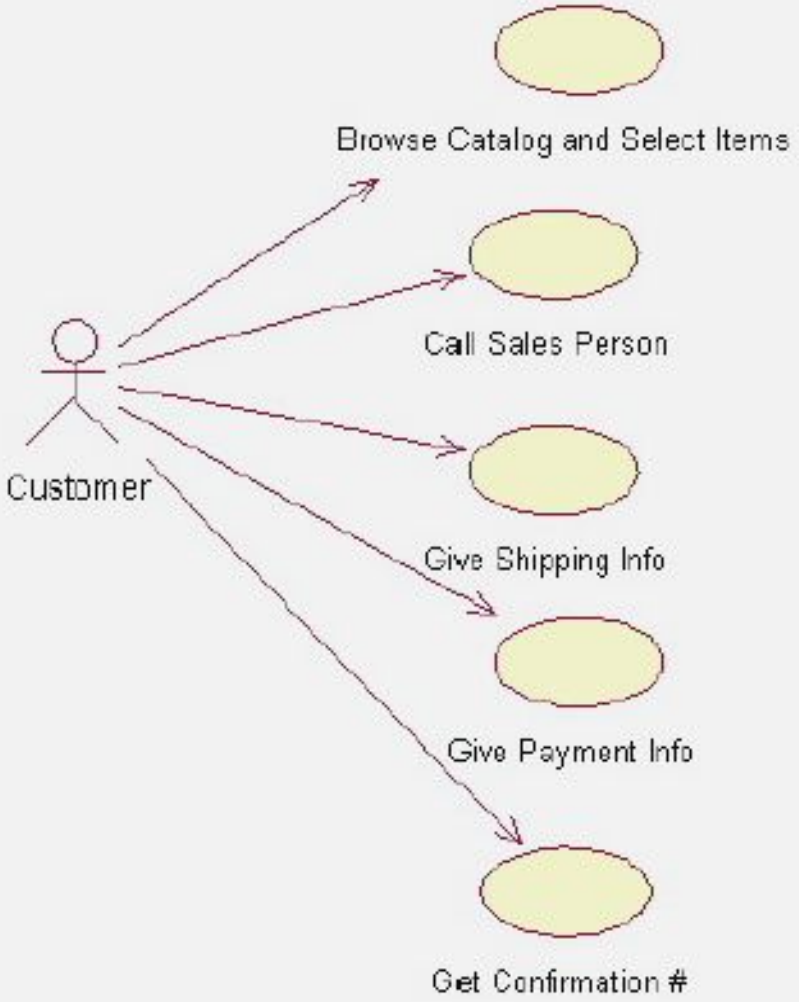
I. Begin with a Use Case!

A user placing an order with a sales company might follow these steps :

1. Browse catalog and select items.
2. Call sales representative.
3. Supply shipping information.
4. Supply payment information.
5. Receive conformation number from salesperson.

II. Then translate Use Case sequence into Diagram

Use-Case Diagrams: Example [2]



The salesperson could also be included in this use case diagram because the salesperson is also interacting with the ordering system.

Outline of Take Exam Use Case

Name of Use Case: Take Exam

Actor(s): ExamTaker

Flow of events:

1. ExamTaker connects to the Exam server.
2. Exam server checks whether ExamTaker is already authenticated and runs authentication process if necessary.
3. ExamTaker selects a exam from a list of options.
4. ExamTaker repeatedly selects a question and either types in a solution, attaches a file with a solution, edits a solution or attaches a replacement file.

Outline Specification of Use Case (continued)

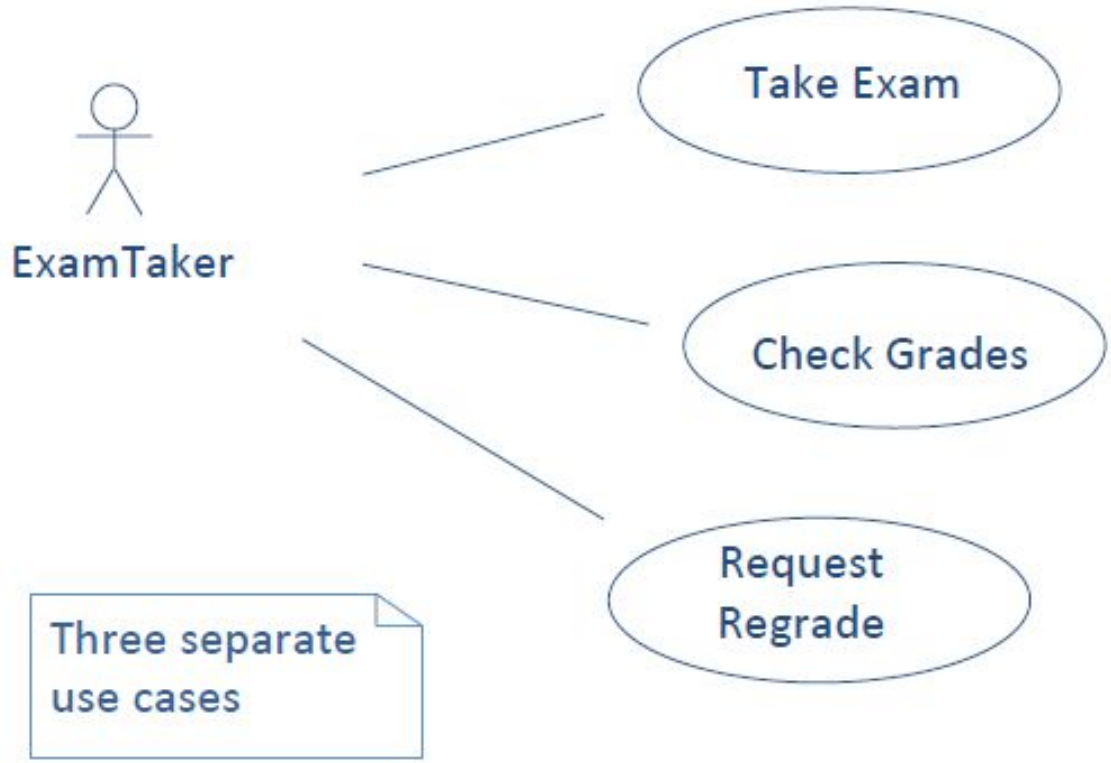
Flow of events (continued):

5. ExamTaker either submits completed exam or saves current state.
6. When a completed exam is submitted, Exam server checks that all questions have been attempted and either sends acknowledgement to ExamTaker, or saves current state and notifies ExamTaker of incomplete submission.
7. ExamTaker logs out.

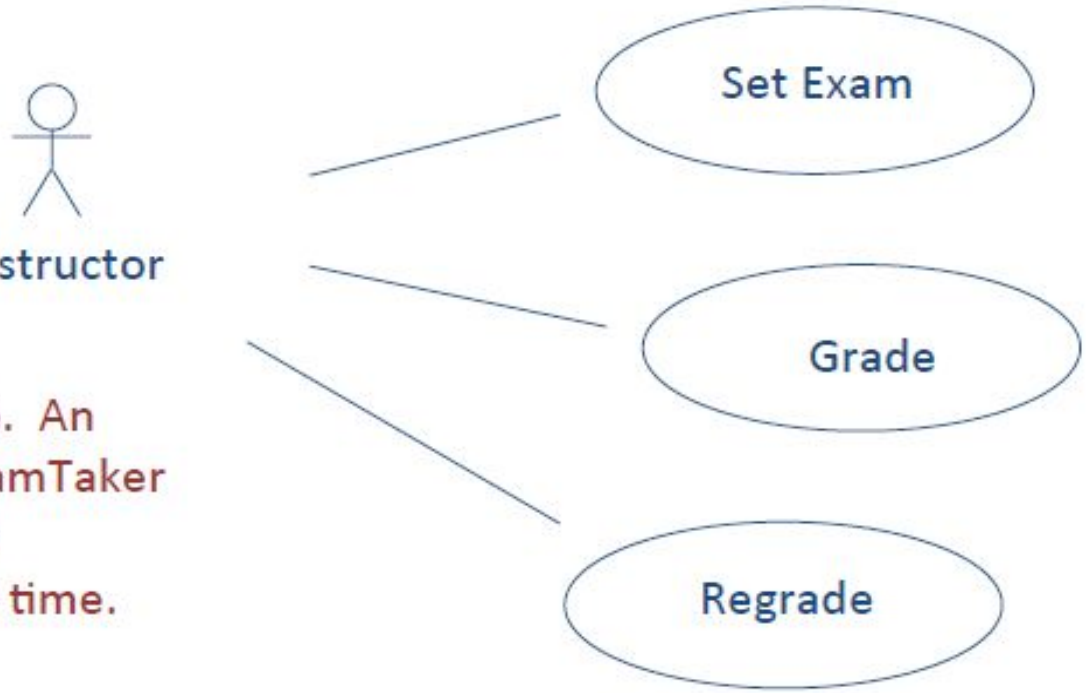
Entry conditions:

1. ExamTaker must have authentication credentials.
2. Computing requirements: supported browser.

Use Cases for Exam System

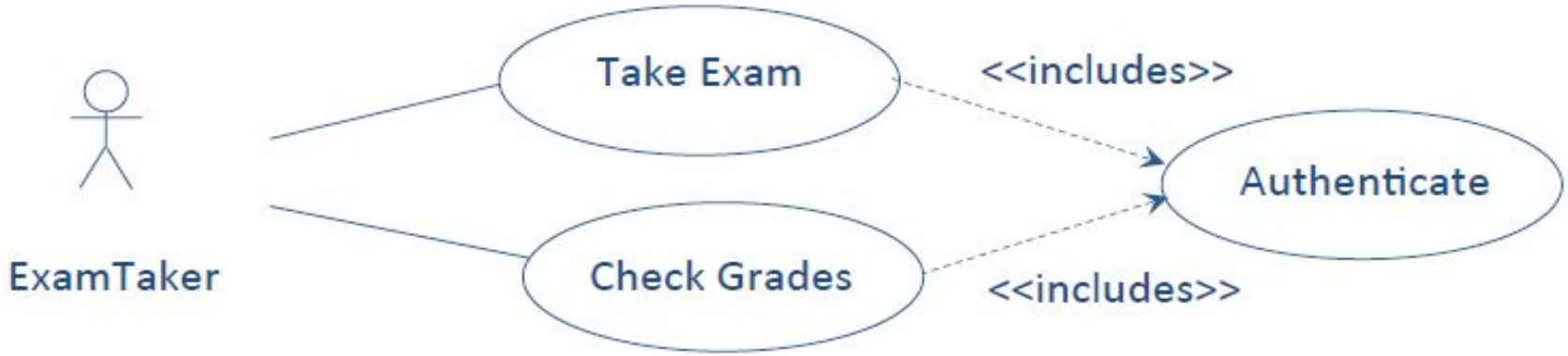


Use Cases for Exam System (continued)



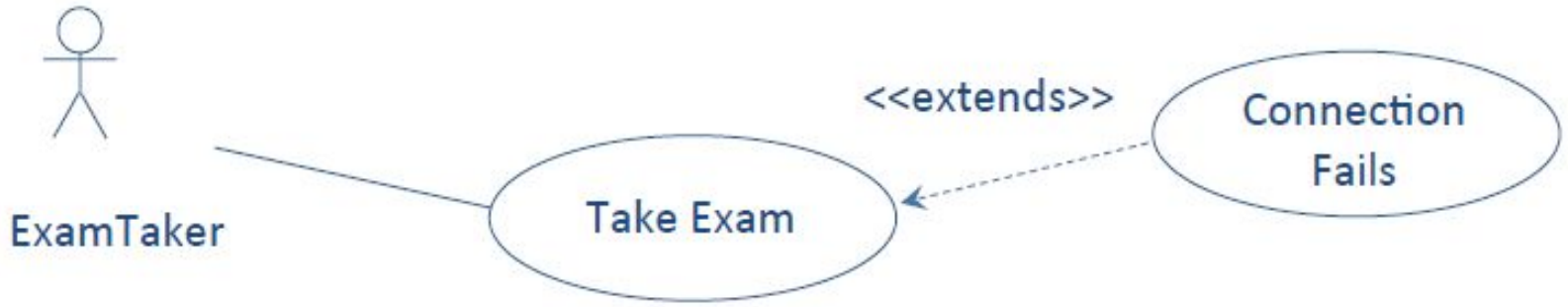
Note that actor is a role. An individual can be an ExamTaker on one occasion and an Instructor at a different time.

Relationships Between Use Cases: <<includes>>



The Authenticate use case may be used in other contexts

Relationships Between Use Cases: <<extends>>



<<includes>>	is used for use cases that are in the flow of events of the main use case.
<<extends>>	is used for exceptional conditions, especially those that can occur at any time.

Data Modeling Elements

اعداد

م.م ايناس إسماعيل

عمران

Data Modeling Elements (ERD)

Attributes - name a data object instance, describe its characteristics, or make reference to another data object

Instance

Naming attributes

Ties one data object to another, in this case, owner

Identifier

Descriptive attributes

Referential attributes

Make	Model	ID#	Body type	Color	Owner
Lexus	LS400	AB123. . .	Sedan	White	RSP
Chevy	Corvette	X456. . .	Sports	Red	CCD
BMW	750iL	XZ765. . .	Coupe	White	LJL
Ford	Taurus	Q12A45. . .	Sedan	Blue	BLF

Relationships - indicate the manner in which data objects are connected to one another .

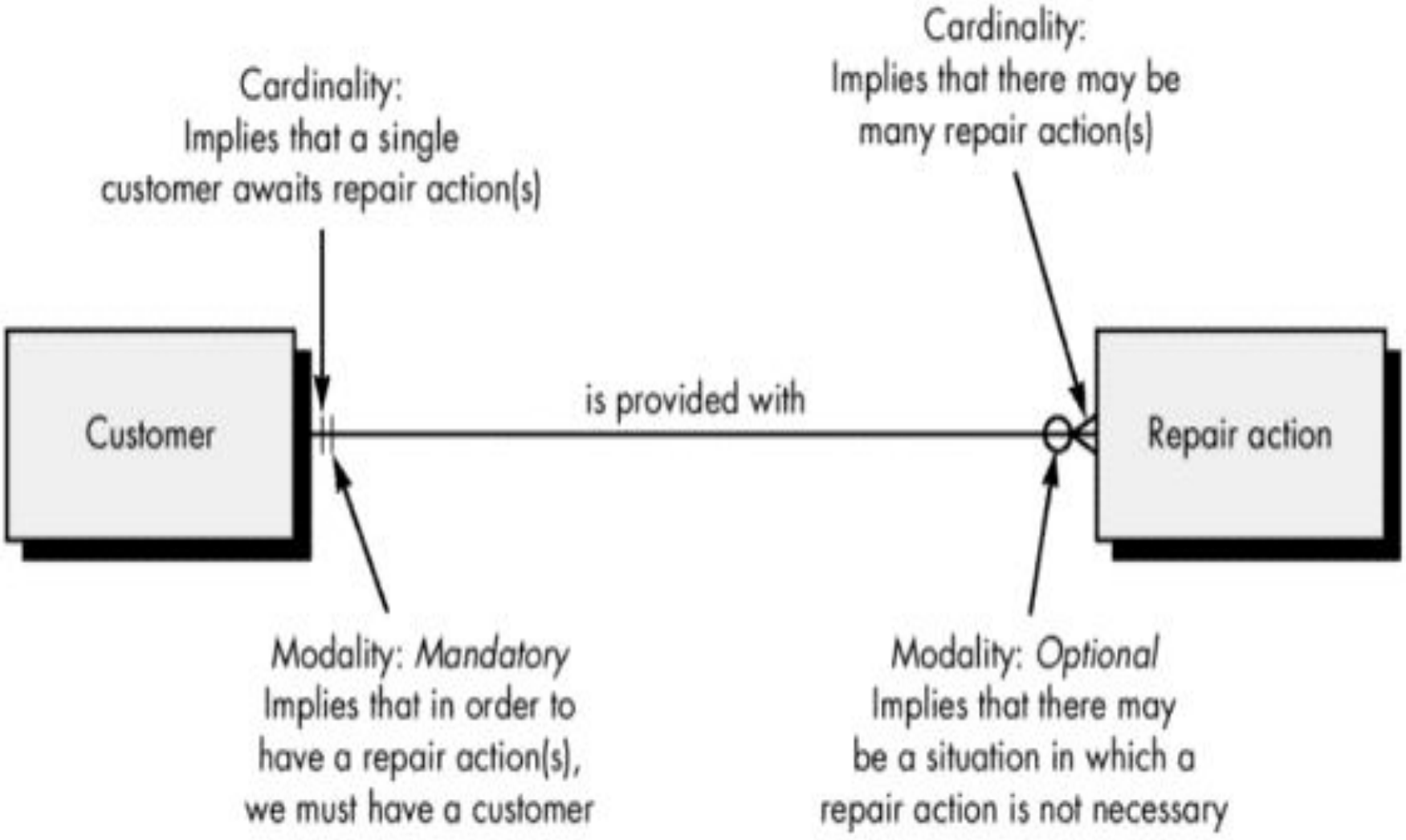
(a) A basic connection between objects

(b) Relationships between objects

Cardinality and Modality (ERD)

- ***Cardinality*** - in data modeling, cardinality specifies how the number of occurrences of one object are related to the number of occurrences of another object (1:1, 1:N, M:N)
- ***Modality*** - zero (0) for an optional object relationship and one (1) for a mandatory relationship

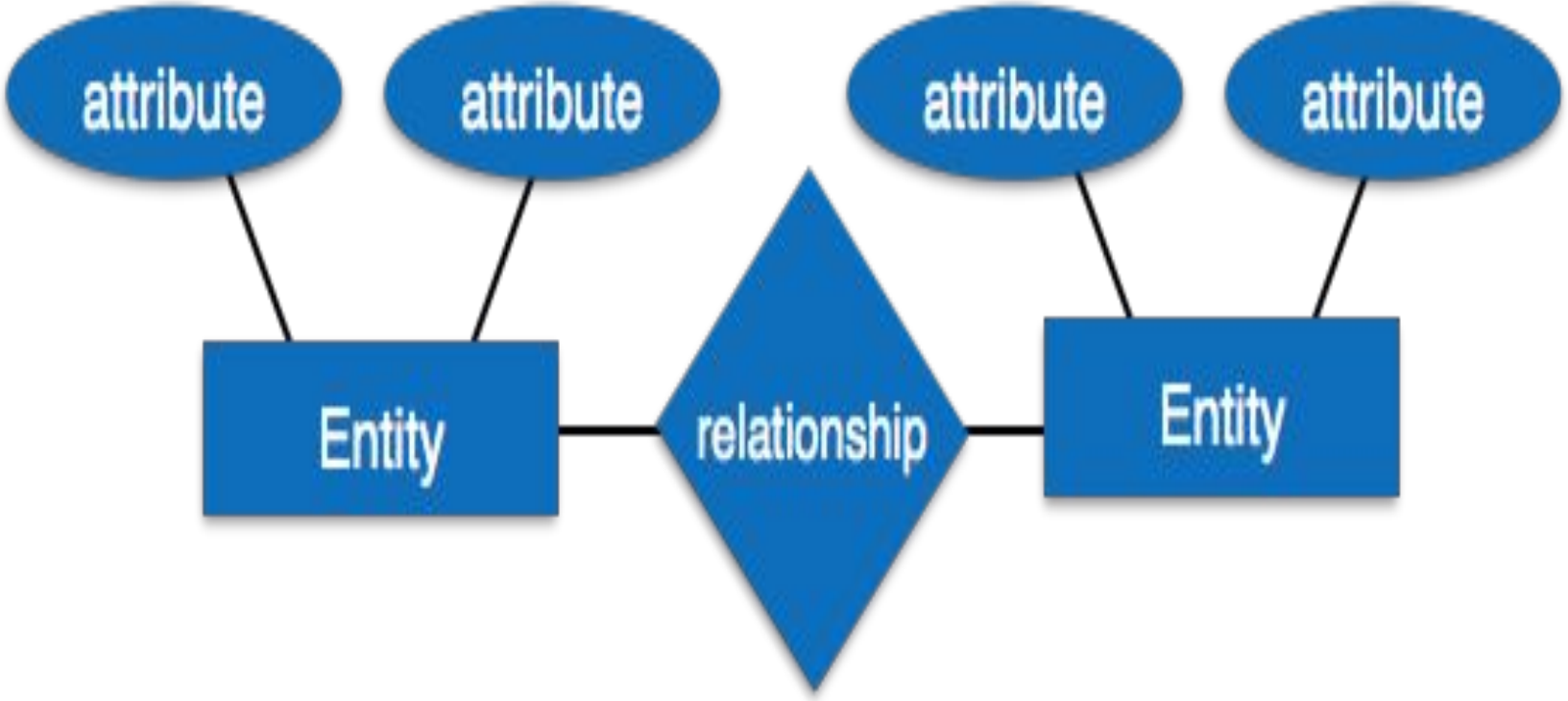
Cardinality and Modality (ERD)



Entity-Relationship Model:

- Entity-Relationship model is a type of database model based on the idea of real world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of relation among them.

ER Model is best used for the conceptual design of database. ER Model can be represented as follows :



Entity - An entity in ER Model is a real world being, which has some properties called ***attributes***. Every attribute is defined by its set of values, called ***domain***.

For example, Consider a school database. Here, a student is an entity. Student has various attributes like name, id, age and class etc.

Relationship - The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.

Mapping cardinalities:

- one to one
- one to many
- many to one
- many to many

Entity/Relationship Diagrams (ERD)

- **Primary components identified for the ERD:**
  - data objects
  - attributes
  - relationships
  - type indicators
- **Primary purpose**: represent data objects and their relationships
- **Iconography:**
  - Data objects are represented by a labeled rectangle
  - Relationships are indicated with a labeled line connecting objects
  - Connections between data objects and relationships are established using a variety of special symbols that indicate cardinality and modality

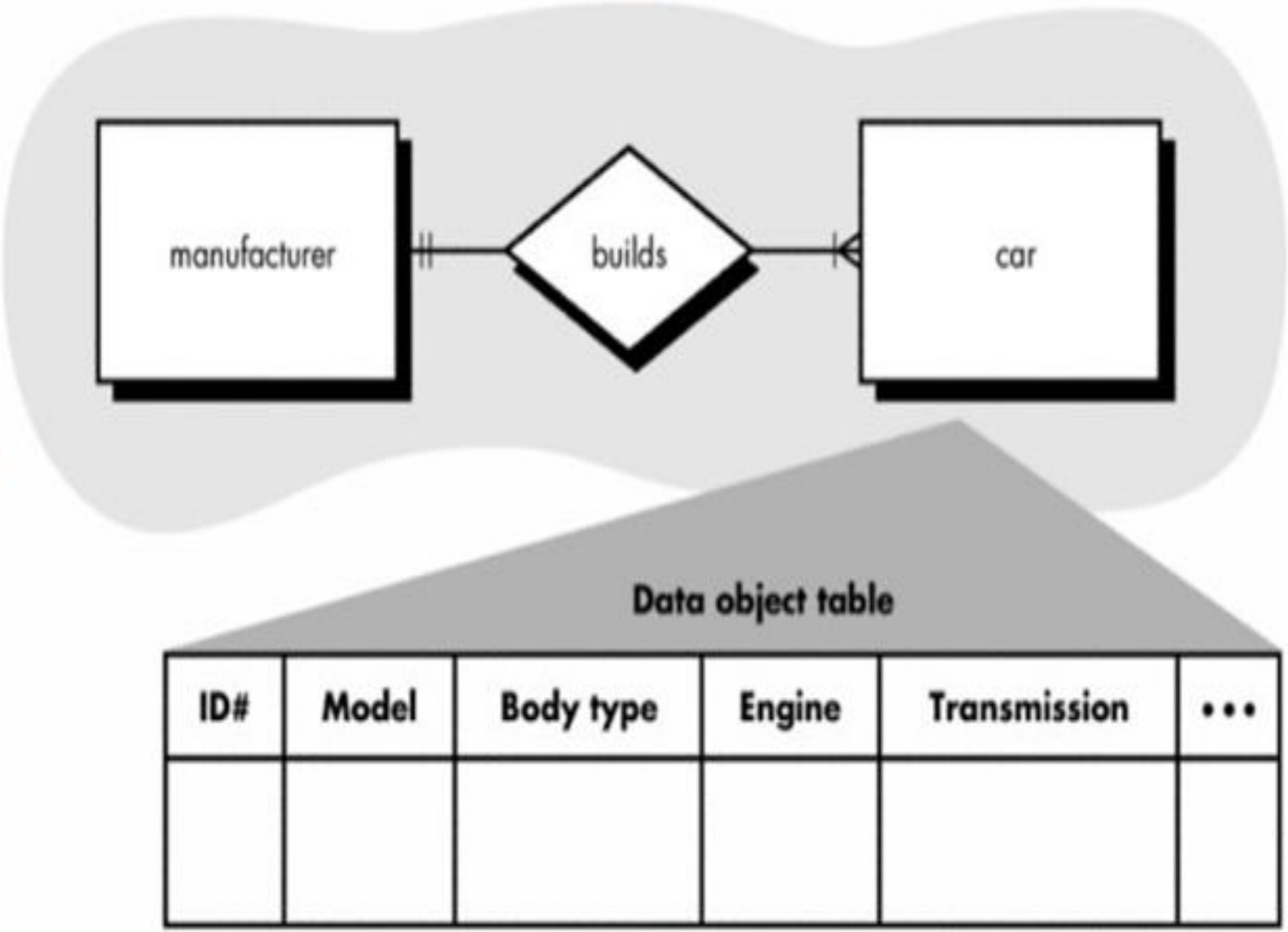


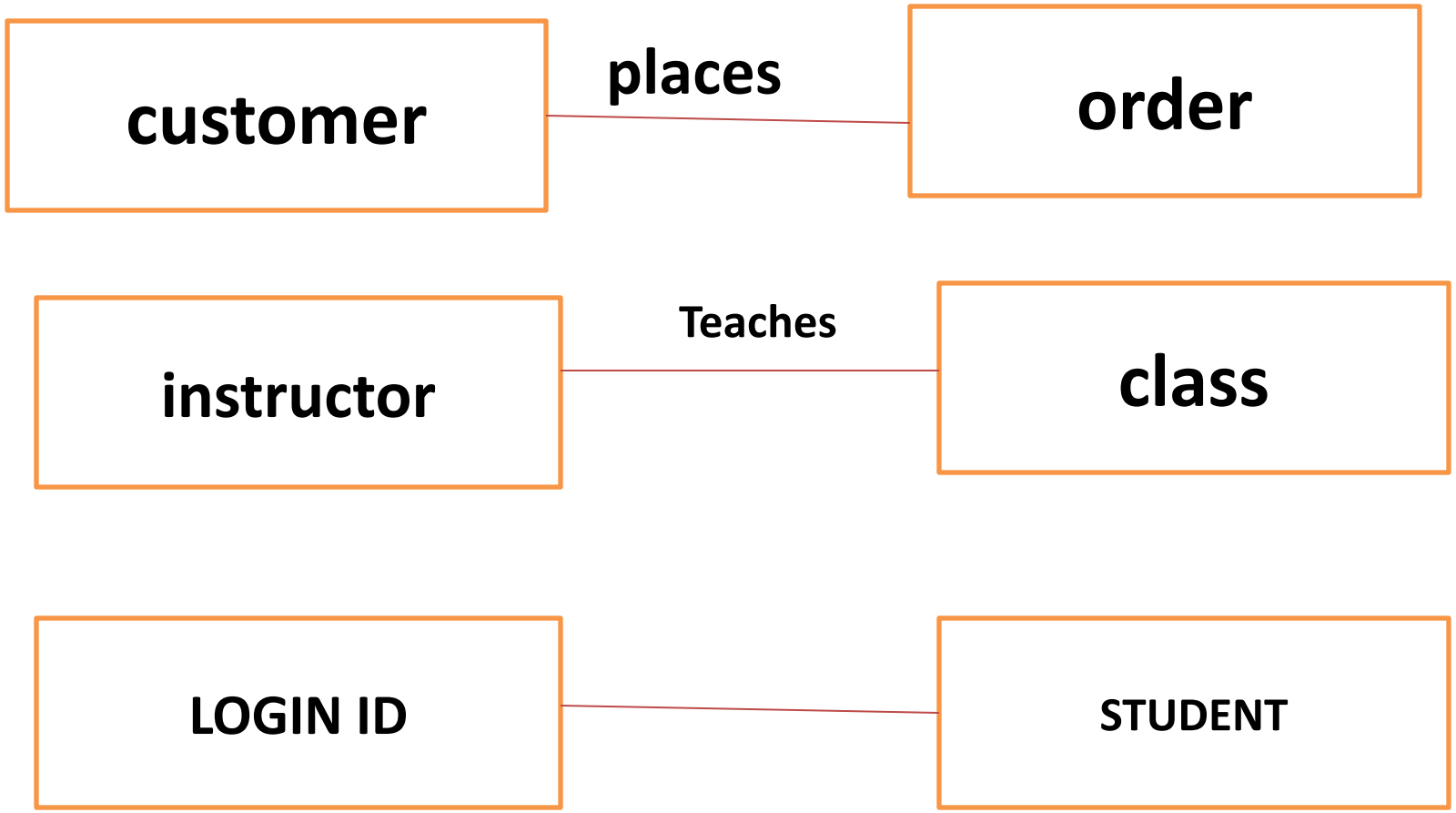
Figure 4.6: A simple ERD and data object table

Use case –cardinality and modality example

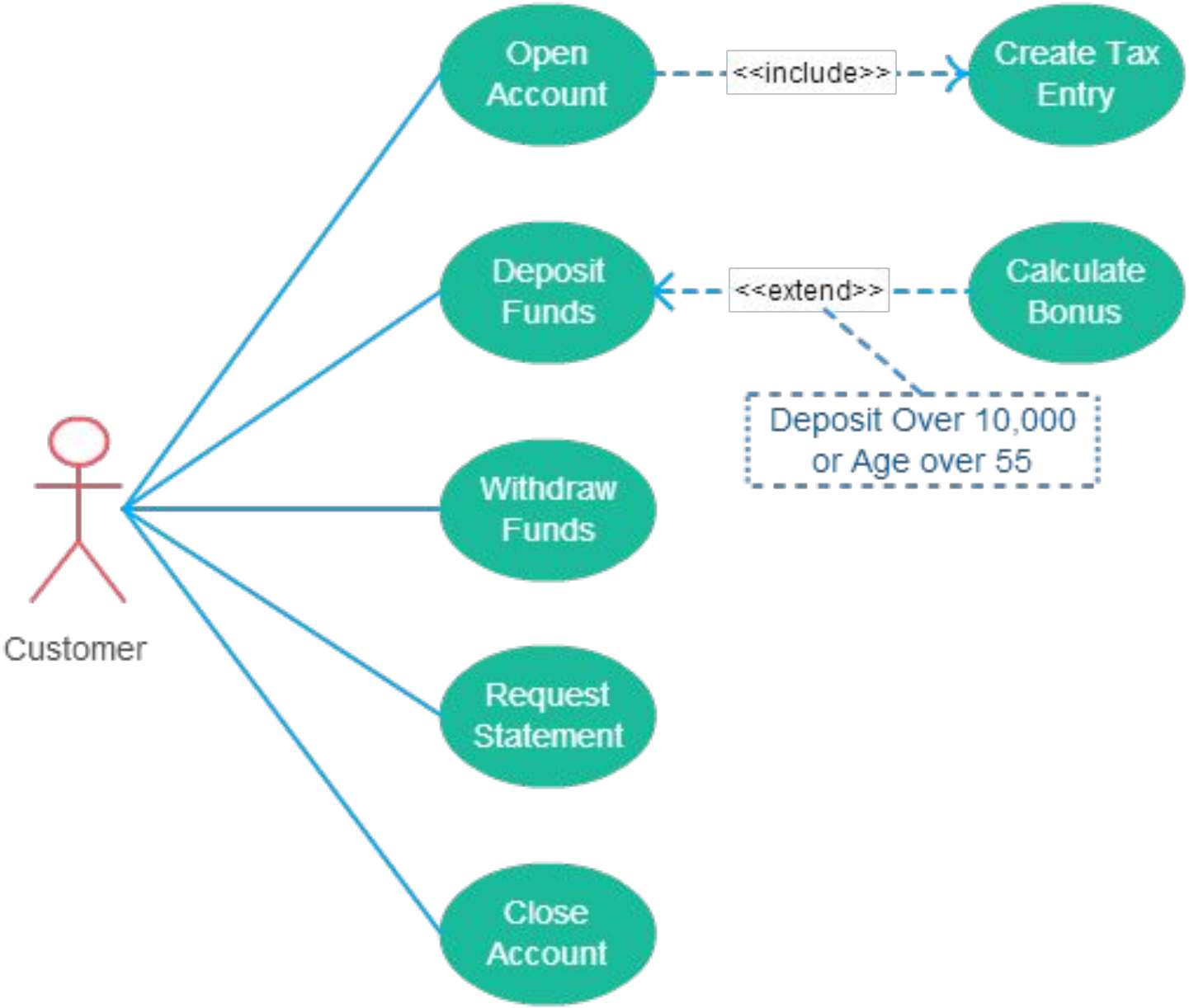
اعداد

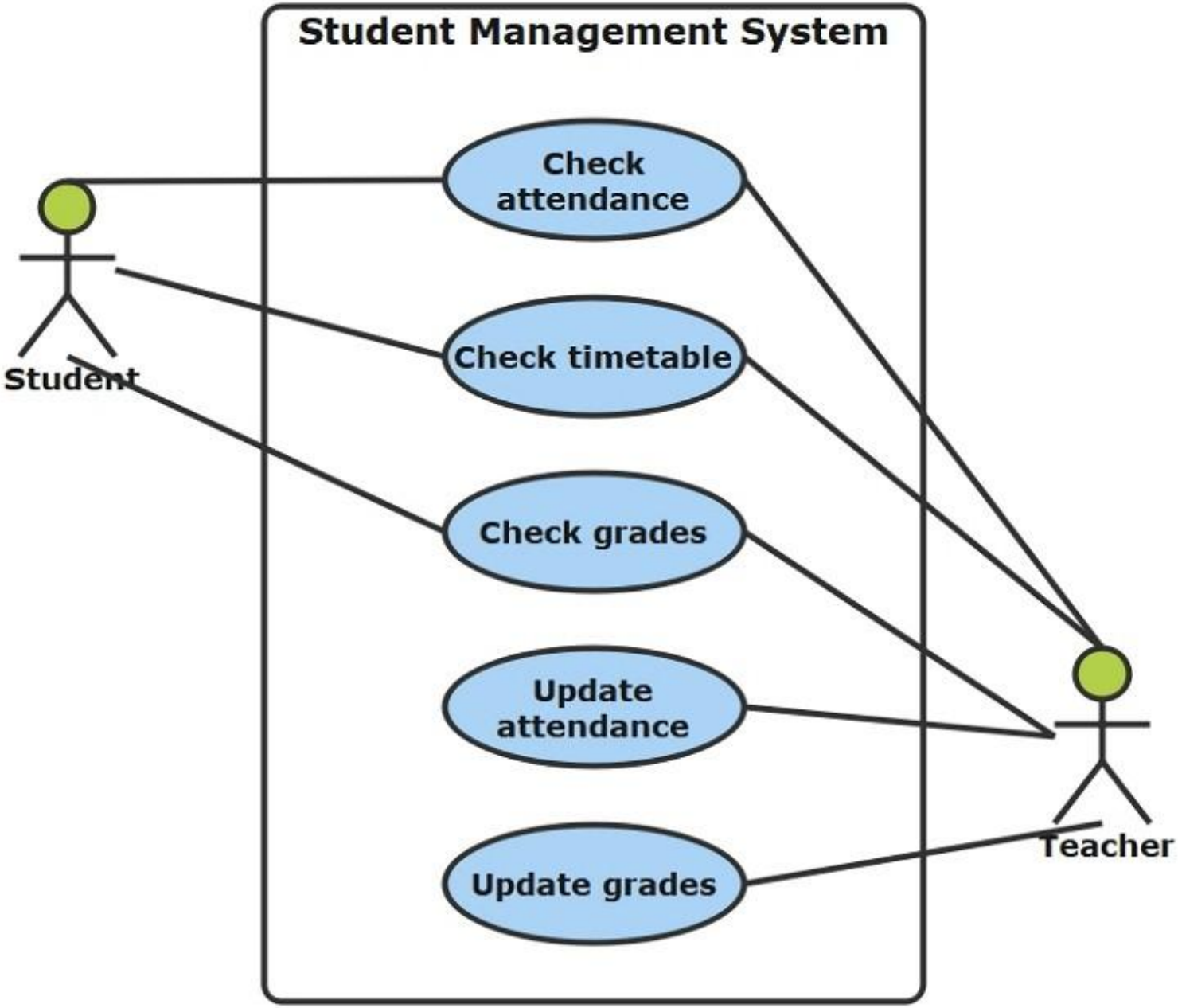
م.م ايناس اسماعيل عمران

cardinality and modality example



bank system


Open Account
<<include>>
Create Tax Entry
Deposit Funds
<<extend>>
Calculate Bonus
Deposit Over 10,000 or Age over 55
Withdraw Funds
Request Statement
Close Account
Customer


Student Management System
Check attendance
Check timetable
Check grades
Update attendance
Update grades
Student
Teacher

Software Design

اعداد

م .م ايناس إسماعيل عمران

Software Design

- Software Design Definition
- Activities of Software Design
- Effective Modular Design
- Introduction to Object Oriented Design
- Top Down and Bottom up Design

Software Design Definition

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

Software design is the first step in SDLC (**Software Design Life Cycle**), which moves the concentration from problem domain to solution domain. It tries to specify how to accomplish the requirements mentioned in **SRS**.

Activities of Software Design

Software design has three levels of results:

Architectural Design - The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

High-level Design- The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other.

Activities of Software Design

High-level design focuses on how the system along with all of its components can be implemented in forms of modules.

Detailed Design- Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

Effective Modular Design

Modularization

is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently.

These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- separate execution can be made possible
- Desired from security aspect

Coupling and Cohesion

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

Co-incidental cohesion - It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.

Coupling and Cohesion

Logical cohesion - When logically categorized elements are put together into a module, it is called logical cohesion.

Temporal Cohesion - When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.

- **Procedural cohesion** - When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.

Coupling and Cohesion

Communicational cohesion - When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.

Sequential cohesion - When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.

Functional cohesion - It is considered to be the highest degree of cohesion. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

Is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely –

Content coupling - When a module can directly access or modify or refer to the content of another module, it is called content level coupling.

Common coupling- When multiple modules have read and write access to some global data, it is called common or global coupling.

Coupling

Stamp coupling- When multiple modules share common data structure and work on different part of it, it is called stamp coupling.

Data coupling- Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Cohesion - grouping of all functionally related elements.

Coupling - communication between different modules.

Ideally, no coupling is considered to be the best.

Software Design & Test

اعداد

م.م ايناس إسماعيل

عمران

Object Oriented Design

Objects - All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it.

Classes - A class is description of an object. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.

In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

Object Oriented Design

Inheritance – IN OOD, sub-classes can import, so that re-use allowed for variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.

Polymorphism - OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism.

Software Design Approaches

1. Top Down Design

We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their own set of sub-system and components and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics.

Top Down Design .1

Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

2. Bottom-up Design

The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic components. It keeps creating higher level components until the desired system as one single component.

Both, top-down and bottom-up approaches are not practical individually. Instead, a good combination of both is used.

Chapter Six: Software Validation and Verification

- Software Testing is evaluation of the software beside requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing include Validation and Verification.

Why Testing?

Two objectives – Verification and Validation (V&V):

To uncover errors (or bugs) in the software before delivery to the client. This is called **Verification** -Verify that the program is working.

To ascertain that the software meet its requirement specification. This is called **Validation** – Validate that the software meets its requirements.

Chapter Six: Software Validation and Verification

Software Validation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software?"
- Validation depend on user requirements.

Software Verification

Verification is the process of confirming if the software is meeting the business requirements, and is developed according to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications?"
- Verifications concentrate on the design and system specifications.

: Target of the test are

- **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.

- **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail to perform its required function.

- **Failure** - failure is said to be the inability of the system to perform the desired function according to its specification. Failure occurs when fault exists in the system.

Software Test

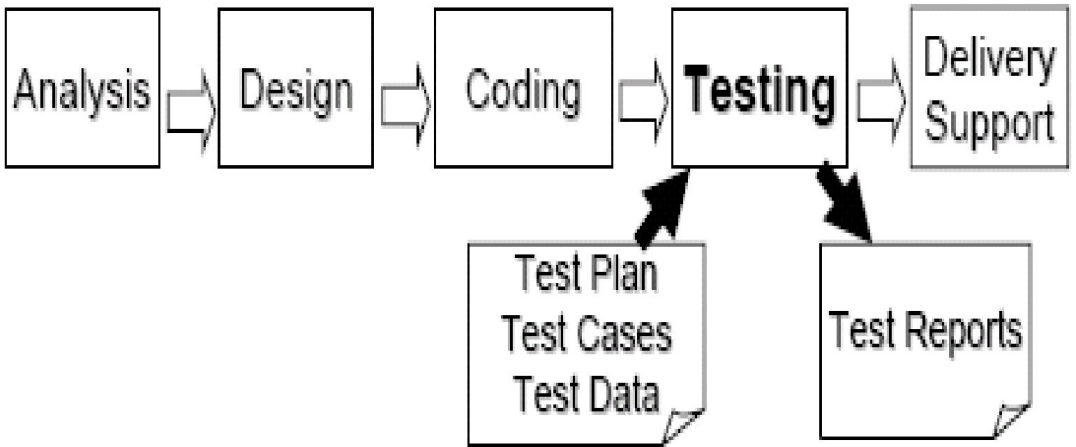
اعداد

م.م ايناس إسماعيل

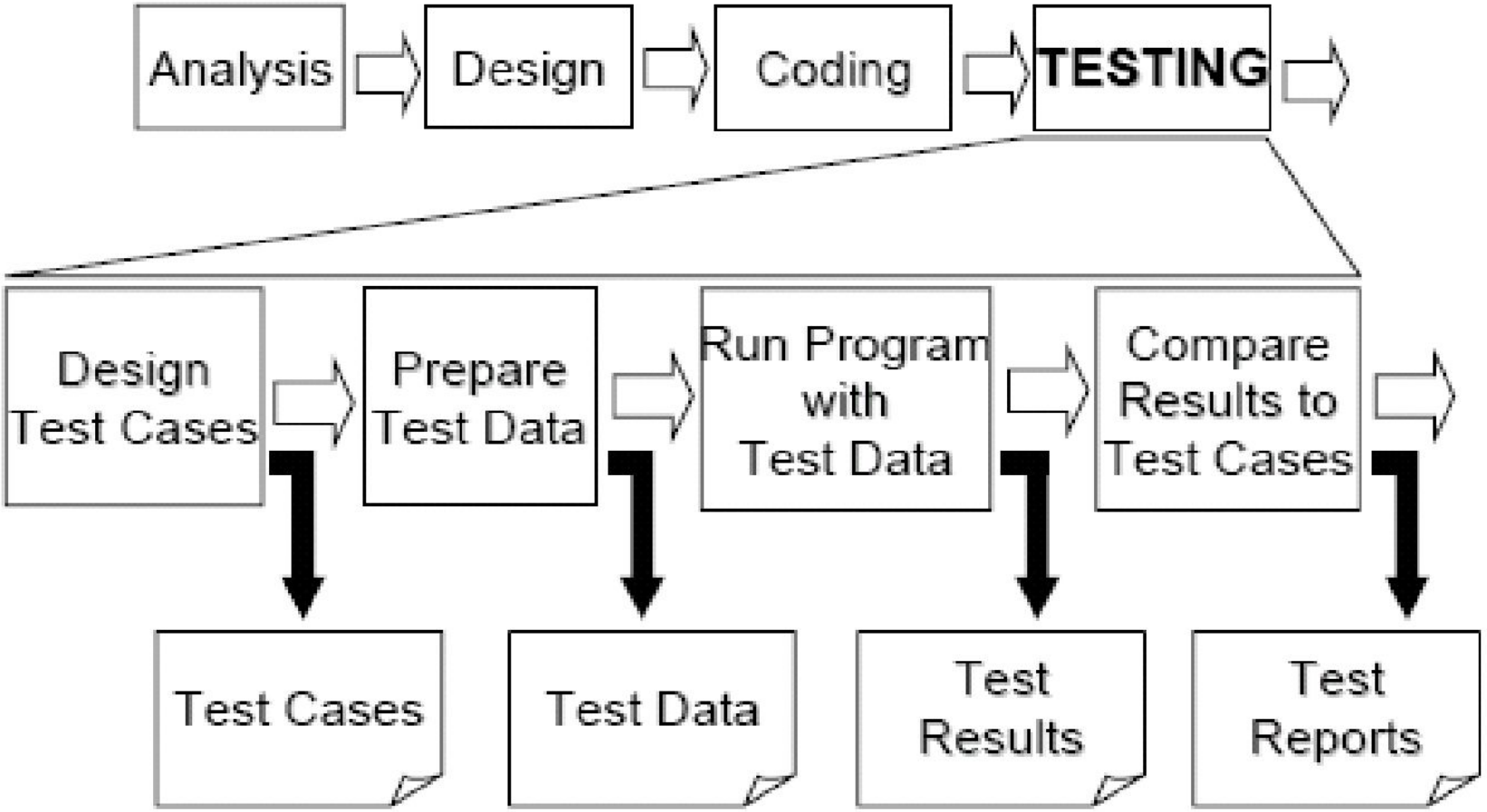
عمران

Testing Stage of the Software Process

- The goal of testing is to “design a series of test cases” that has “a high probability of finding errors”.
- How? Design test cases systematically by “applying engineering principles and methods”.
- The work product of the testing stage is a “test report” that documents all the test cases run, i.e., the test input, the expected output, the actual output, the purpose of the test and etc.



Testing Stage Details



Who tests the system?

- During the early stages of testing, the developer performs the tests. As the testing progresses, independent test specialist may involved.
- “Open–source” software like Linux, Java, Apache are known to be more secure and less buggy because many independent parties have “tested” the source code.

Software Testing

Testing Approaches

Tests can be conducted based on two approaches: –

-Functionality testing

- Implementation testing

When **functionality** is being tested without taking the actual implementation in concern it is known as **black-box** testing. The other side is known as **white-box** testing where not only functionality is tested but the way it is implemented is also analyzed.

Black-box testing

It is carried out to test functionality of the program. It is also called 'Behavioral' testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested 'ok' and problematic otherwise.

In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

Black-box testing techniques:

Equivalence class - The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.

Boundary values - The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.

Cause-effect graphing - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.

State-based testing - The system changes state on terms of input. These systems are tested based on their states and input.

White-box testing

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as 'Structural' testing. The design and structure of the code are known to the tester. Programmers conduct this test on the code.

The below are some White-box testing techniques:

Control-flow testing - The purpose of it is to set up test cases which cover all statements and branch conditions. The **branch** conditions are tested for both being true and false, so that all statements can be covered.

- **Data-flow testing** – which cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

Testing Levels

Unit Testing

The programmer performs some tests on unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Testing Levels

Integration Testing

There is a need to find out if the units if integrated together would also work without errors.

System Testing

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

Functionality testing - Tests all functionalities of the software against the requirement.

Performance testing - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task.

Acceptance Testing

Alpha testing - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.

Beta testing - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.