



جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

Software Engineering

LEC13

Presented
by Lecturer : Wafaa Ali

Modularity

- ▶ Modularity is the single attribute of software that allows a program to be intellectually manageable.
- ▶ Monolithic software (i.e., a large program composed of a single module) cannot be easily grasped by a reader.
- ▶ Meyer defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system:
- ▶ **Modular decomposability** : If a design method provides a systematic mechanism for decomposing the problem into sub problems, it will reduce the complexity of the overall problem, thereby achieving an effective modular solution.

- ▶ **Modular composability** : If a design method enables existing (reusable) design components to be assembled into a new system, it will yield a modular solution that does not reinvent the wheel.
- ▶ **Modular understandability** : If a module can be understood as a standalone unit (without reference to other modules), it will be easier to build and easier to change.
- ▶ **Modular continuity** : If small changes to the system requirements result in changes to individual modules, rather than systemwide changes, the impact of change-induced side effects will be minimized.
- ▶ **Modular protection** : If an aberrant condition occurs within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized.

Effective Modular Design

- ▶ In fact, modularity has become an accepted approach in all engineering disciplines. A modular design reduces complexity, facilitates change (a critical aspect of software maintainability) and results in easier implementation by encouraging parallel development of different parts of a system.

1- Functional Independence

- ▶ The concept of functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding. In landmark papers on software design Parnas and Wirth allude to refinement techniques that enhance module independence. Later work by Stevens, Myers and Constantine solidified the concept.
- ▶ Functional independence is achieved by developing modules with "single-minded" function and an "aversion" to excessive interaction with other modules.
- ▶ Independent modules are easier to maintain and test because secondary effects caused by design or code modification are limited, error propagation is reduced, and reusable modules are possible.

- ▶ To summarize, functional independence is a key to good design and design is the key to software quality.
- ▶ Independence is measured using two qualitative criteria: cohesion and coupling.

2- Cohesion

- Cohesion is a measure of the relative functional strength of a module.
- A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.
- Cohesion may be represented as a "spectrum". We always strive for high cohesion, although the mid-range of the spectrum is often acceptable. The scale for cohesion is nonlinear. In practice, a designer need not be concerned with categorizing cohesion in a specific module. Rather, the overall concept should be understood and low levels of cohesion should be avoided when modules are designed.

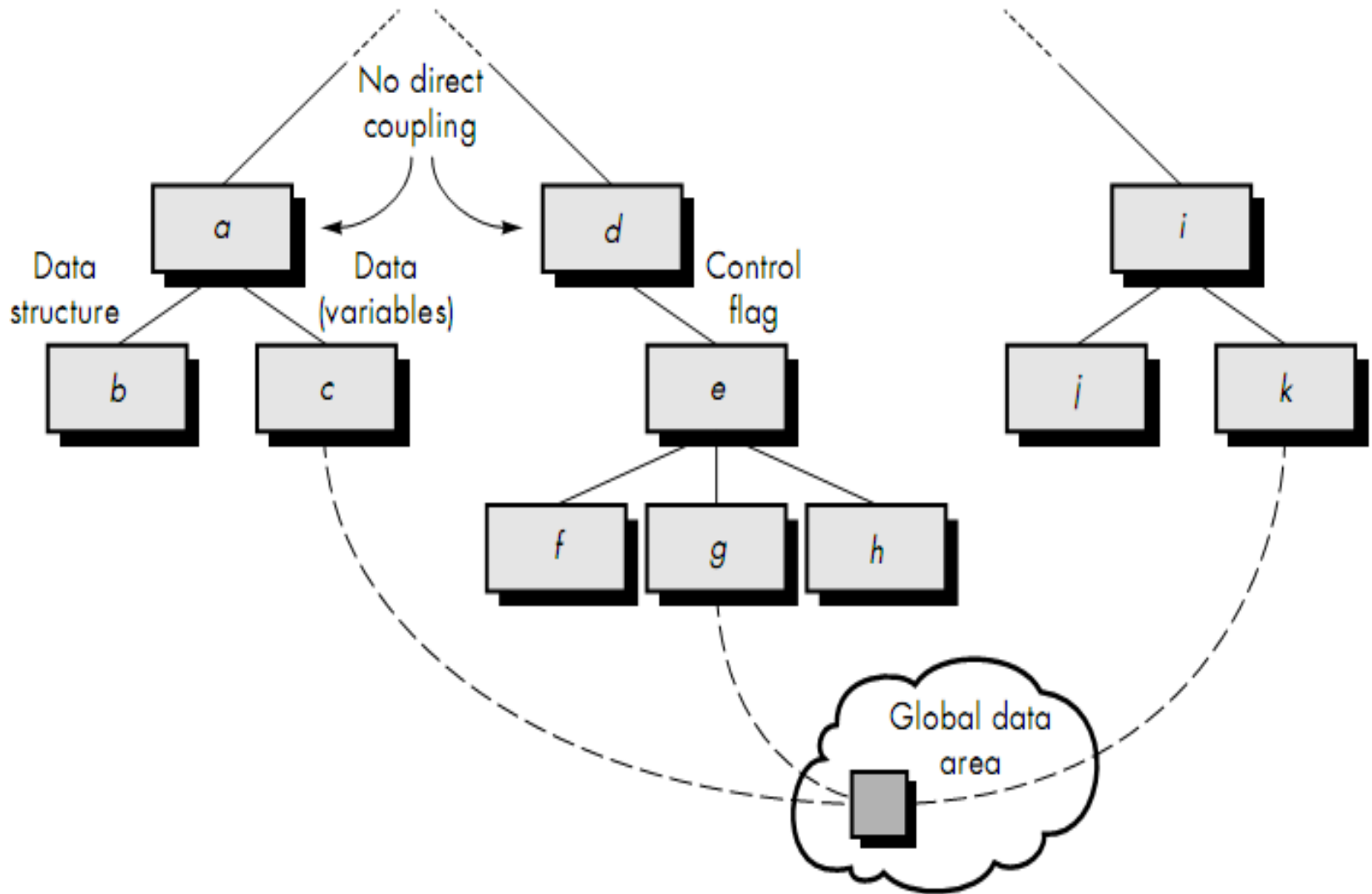
- A module that performs tasks that are related logically (e.g., a module that produces all output regardless of type) is logically cohesive.
- The module is called when computed data exceed prespecified bounds. It performs the following tasks:
 - (1) Computes supplementary data based on original computed data.
 - (2) Produces an error report (with graphical content) on the user's workstation.
 - (3) Performs follow-up calculations requested by the user.
 - (4) Updates a database.
 - (5) Enables menu selection for subsequent processing.

- ▶ Although the preceding tasks are loosely related, each is an independent functional entity that might best be performed as a separate module.
- ▶ As we have already noted, it is unnecessary to determine the precise level of cohesion. Rather it is important to strive for high cohesion and recognize low cohesion so that software design can be modified to achieve greater functional independence.

3 -Coupling

- ❖ Coupling is a measure of interconnection among modules in a software structure.
- ❖ Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.
- ❖ In software design, we strive for lowest possible coupling.
- ❖ Figure (1) provides examples of different types of module coupling. Modules a and d are subordinate to different modules. Each is unrelated and therefore no direct coupling occurs. Module c is subordinate to module a and is accessed via a conventional argument list, through which data are passed.

- ❖ A variation of data coupling, called stamp coupling, is found when a portion of a data structure is passed via a module interface. This occurs between modules b and a.
- ❖ At moderate levels, coupling is characterized by passage of control between modules. Control coupling is very common in most software designs and is shown in Figure (1) where a “control flag” (a variable that controls decisions in a subordinate or superordinate module) is passed between modules d and e.
- ❖ High coupling occurs when a number of modules reference a global data area.



Figure(1) Types of coupling

Introduction to Object Oriented Design

- ▶ Your design should be specific to the problem at hand but also general enough to address future problems and requirements.
- ▶ The design of object oriented software requires the definition of a multilayered software architecture.
- ▶ OOD is performed by a software engineer.
- ▶ OOD is divided into two major activities: system design and object design.
- ▶ System design creates the product architecture, defining a series of “layers” that accomplish specific system functions and identifying the classes that are encapsulated by subsystems that reside at each layer. In addition, system design considers the specification of three components: the user interface, data management functions, and task management facilities.

- ▶ Object design focuses on the internal detail of individual classes, defining attributes, operations and message detail.
- ▶ What is the work product?
An OO design model encompasses software architecture, user interface description, data management components, task management facilities and detailed descriptions of each class to be used in the system.
- The unique nature of object-oriented design lies in its ability to build upon four important software design concepts: abstraction, information hiding, functional independence and modularity.
- Object-oriented design, object-oriented programming and object-oriented testing are construction activities for OO systems.

Design For Object-Oriented Systems

- ▶ For object-oriented systems, we can also define a design pyramid. The four layers of the OO design pyramid are :
 - 1- The subsystem layer** contains a representation of each of the subsystems that enable the software to achieve its customer defined requirements and to implement the technical infrastructure that supports customer requirements.
 - 2-The class and object layer** contains the class hierarchies that enable the system to be created using generalizations and increasingly more targeted specializations. This layer also contains representations of each object.

- 3- The message layer** contains the design details that enable each object to communicate with its collaborators. This layer establishes the external and internal interfaces for the system.
- 4- The responsibilities layer** contains the data structure and algorithmic design for all attributes and operations for each object. The design pyramid focuses exclusively on the design of a specific product or system.

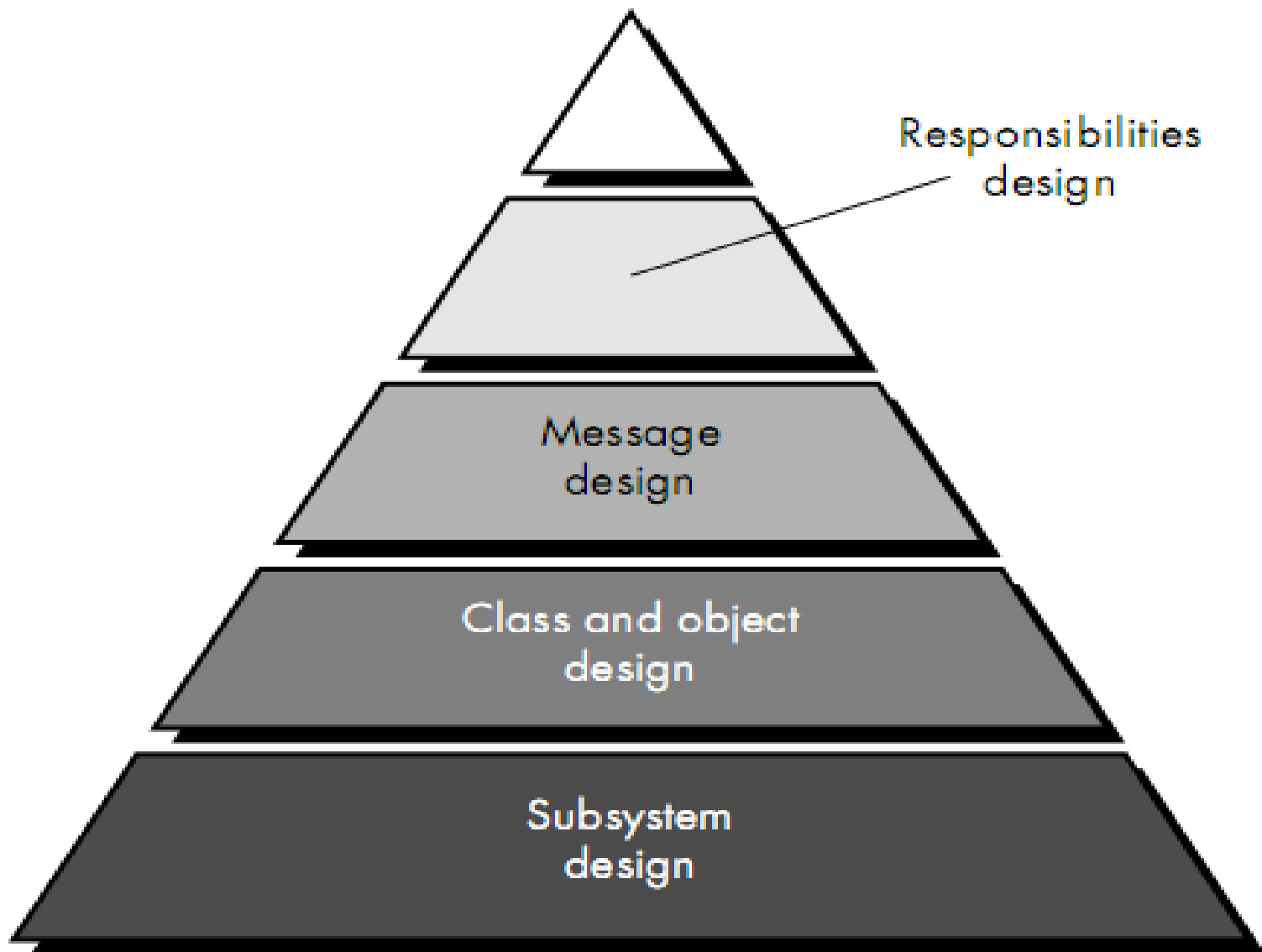


Figure (2) The OO design pyramid

THANKS



جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

*Software Engineering

LEC.4

Presented

by Lecturer : Wafaa Ali

2.5 Software Process Models:

A process model :is a simplified representation of a software process. It is a set of ordered tasks, involving activities, constraints and Resources

2.5.1 The Waterfall Model:

*One such approach/process used in Software Development is "The 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.**
- 5) Maintenance.**

*All these phases are cascaded to each other so that second phase is started as and when defined set of goals are achieved for first phase and it is signed off, so the name "Waterfall Model".

Waterfall Model Phases:

1- Requirements analysis and definition:

- All possible requirements of the system to be developed are captured in this phase.
- Requirements then are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.
- Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model

*** 2- System and software design:**

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

3- Implementation and 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 developed and tested for its functionality referred to as Unit Testing.
- Unit testing mainly verifies if the modules/units meet their specifications.

***4- Integration and system testing**

- The units are integrated into a complete system during Integration phase and tested to check if all units/modules coordinate between each other and the system as a whole behaves as per the specifications.
- After successfully testing the software, it is delivered to the customer

5- Operation and maintenance

- This phase of "The Waterfall Model" is virtually never ending phase (Very long).
- Generally, problems with the system developed (which are not found during the development life cycle) come up after its practical use starts, so 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- referred as Maintenance.

Waterfall Model Advantages:

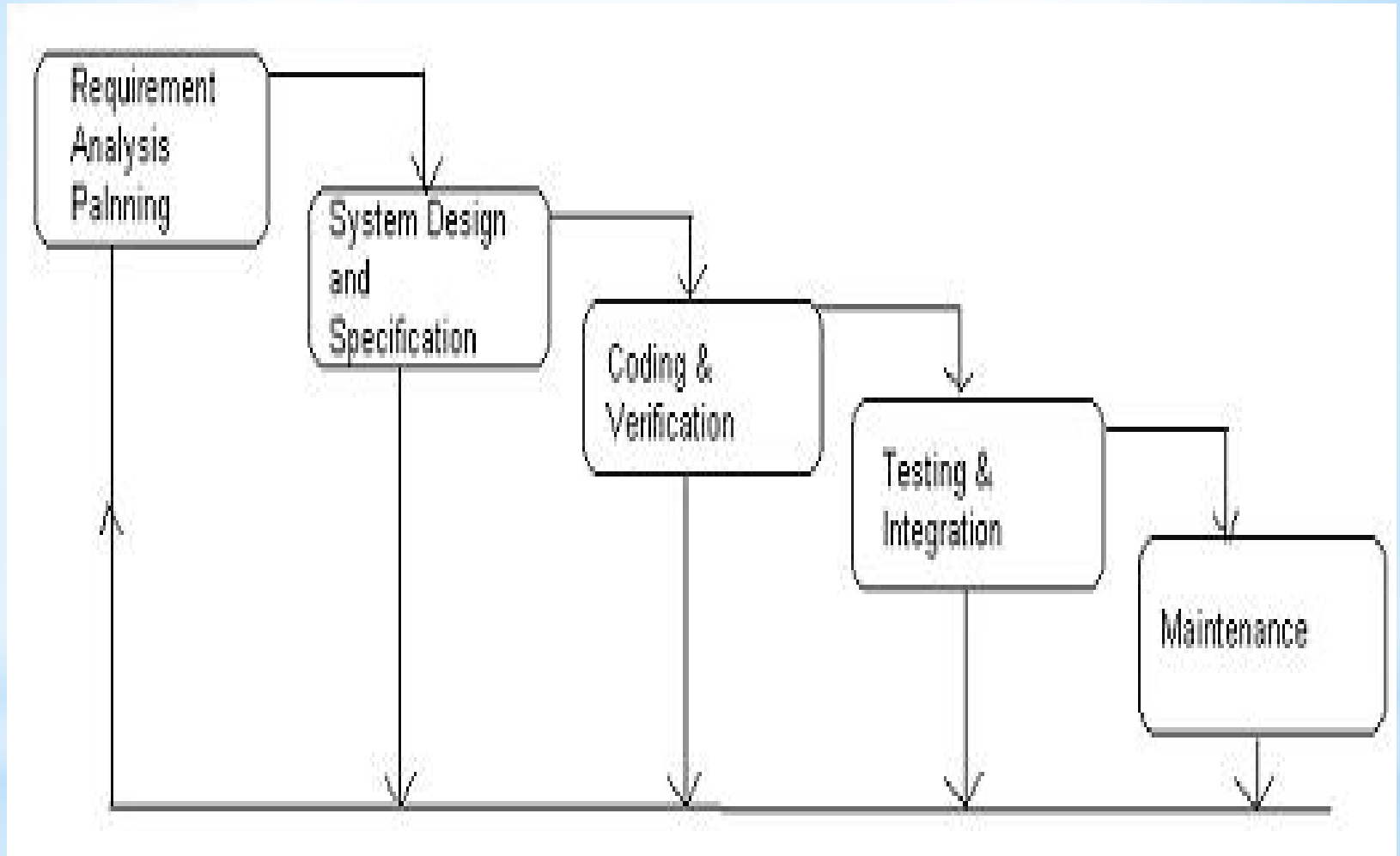
- * A waterfall model is easy to implementation.
- * It helps to find errors earlier
- * Easy to understand, easy to use.
- * Works well when quality is more
- * important than cost or schedule
- * Documentation is produced at every stage
- * of a waterfall model allowing people to understand what has been done.
- * Testing is done at every stage.

Waterfall Model Disadvantages:

- * It is only suitable for the small size projects.
- * Constant testing of the design is needed.
- * If requirements may change the Waterfall model may not work.
- * Difficult to estimate time and cost for each stage of the development process.
- * Adjust scope during the life cycle can kill a project.
- * High amount of risk and uncertainty.

When to use the Waterfall Model:

- Requirements are very well known.
- Product definition is stable.
- Technology is understood.
- New version of an existing product.
- Porting an existing product to a new platform



The flow diagram of the waterfall model

GOOD LUCK



جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

Software Engineering

lec 9

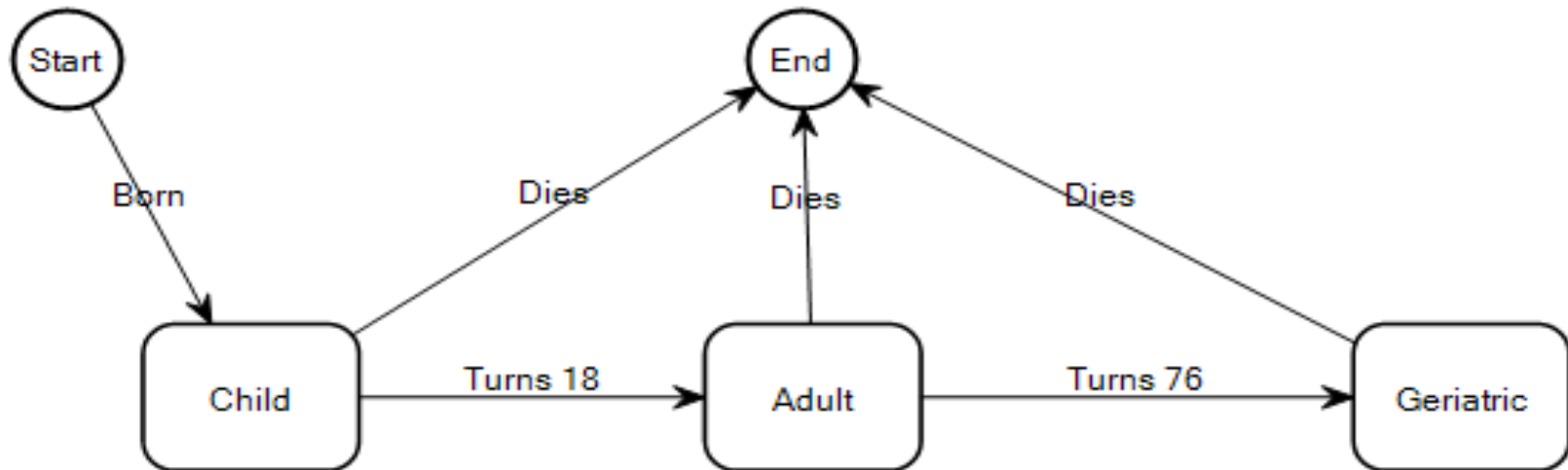
Presented

by Lecturer :

4.1 The Analysis Model

- The analysis model, actually a set of models, is the first technical representation of a system.
- Analysis modeling uses a combination of text and diagrams to represent software requirements (data, function and behavior) in an understandable way.
- Building analysis models helps make it easier to uncover requirement inconsistencies and omissions.
- Two types of analysis modeling are commonly used: structured analysis and object-oriented analysis.

- Data modeling uses entity-relationship diagrams to define data objects, attributes and relationships.
- Functional modeling uses data flow diagrams to show how data are transformed inside the system.
- Behavioral modeling uses state transition diagrams to show the impact of events.



4.2 Concepts of Structure Analysis

- Analysis products must be highly maintainable, especially the software requirements specification.
- Problems of size must be dealt with using an effective method of partitioning.
- Graphics should be used whenever possible.
- Differentiate between the logical (essential) and physical (implementation) considerations.
- Find something to help with requirements partitioning and document the partitioning before specification.
- Devise a way to track and evaluate user interfaces.
- Devise tools that describe logic and policy better than narrative text.

4.3 Analysis Model Objectives

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

4.4 The Elements of Analysis Model

- **4.4.1 Data dictionary:**
- I. A tool for recording and processing information (metadata) about the data that an organization uses.
- II. A central catalogue for metadata.
- III. Can be integrated within the DBMS or be separate.
- IV. May be referenced during system design, programming, and by actively-executing programs.
- V. Can be used as a repository for common code (e.g. library routines)..

DATA

employee_id	first_name	last_name	nin	dept_id
44	Simon	Martinez	HH 45 09 73 D	1
45	Thomas	Goldstein	SA 75 35 42 B	2
46	Eugene	Comelsen	NE 22 63 82	2
47	Andrew	Petculescu	XY 29 87 61 A	1
48	Ruth	Stadick	MA 12 89 36 A	15
49	Bary	Scardelis	AT 20 73 18	2
50	Sidney	Hunter	HW 12 94 21 C	6
51	Jeffrey	Evans	LX 13 26 39 B	6
52	Doris	Bemdt	YA 49 88 11 A	3
53	Diane	Eaton	BE 08 74 68 A	1

DATA DICTIONARY (METADATA)

Column	Data Type	Description
employee_id	int	Primary key of a table
first_name	nvarchar(50)	Employee first name
last_name	nvarchar(50)	Employee last name
nin	nvarchar(15)	National Identification Number
position	nvarchar(50)	Current position title, e.g. Secretary
dept_id	int	Employee department. Ref: Department
gender	char(1)	M = Male, F = Female, Null = unknown
employment_start_date	date	Start date of employment in organization.
employment_end_date	date	Employment end date.



Attribute Name	Required	Format	Max Field Size	Location
Client Name	Yes	Text	20	/main/main_info.xls
Birth Date	Yes	Date	10	/main/main_info.xls
Savings	Yes	Lookup	10	/main/financial/savings.xls
Account Number	Yes	Number	30	/main/main_info.xls
Credit Card ID	No	Lookup	20	/main/credit_cards.xls
Loans	No	Lookup	10	/main/financial/loan.xls

DDS should provide two sets of facilities:

- **To record** and analyze data requirements independently of how they are going to be met -
conceptual data models (entities, attributes, relationships).
 - **To record** and design decisions in terms of database or file structures implemented and the programs which access them - internal schema.

One of the main functions of a DDS is to show the relationship between the conceptual and implementation views. The mapping should be consistent - inconsistencies are an error and can be detected here.

Benefits of a DDS:

- improved documentation and control
- consistency in data use •
- easier data analysis •
- reduced data redundancy •
- simpler programming •
- the enforcement of standards •
- better means of estimating the effect of change. •

DDS Disadvantages:

- • The DDS 'project' may itself take two or three years.
- • It needs careful planning, defining the exact requirements designing its contents, testing, implementation and evaluation.
- • The cost of a DDS includes not only the initial price of its installation and any hardware requirements, but also the cost of collecting the information entering it into the DDS, keeping it up-to-date and enforcing standards.
- • The use of a DDS requires management commitment, which is not easy to achieve , particularly where the benefits are intangible and long term.

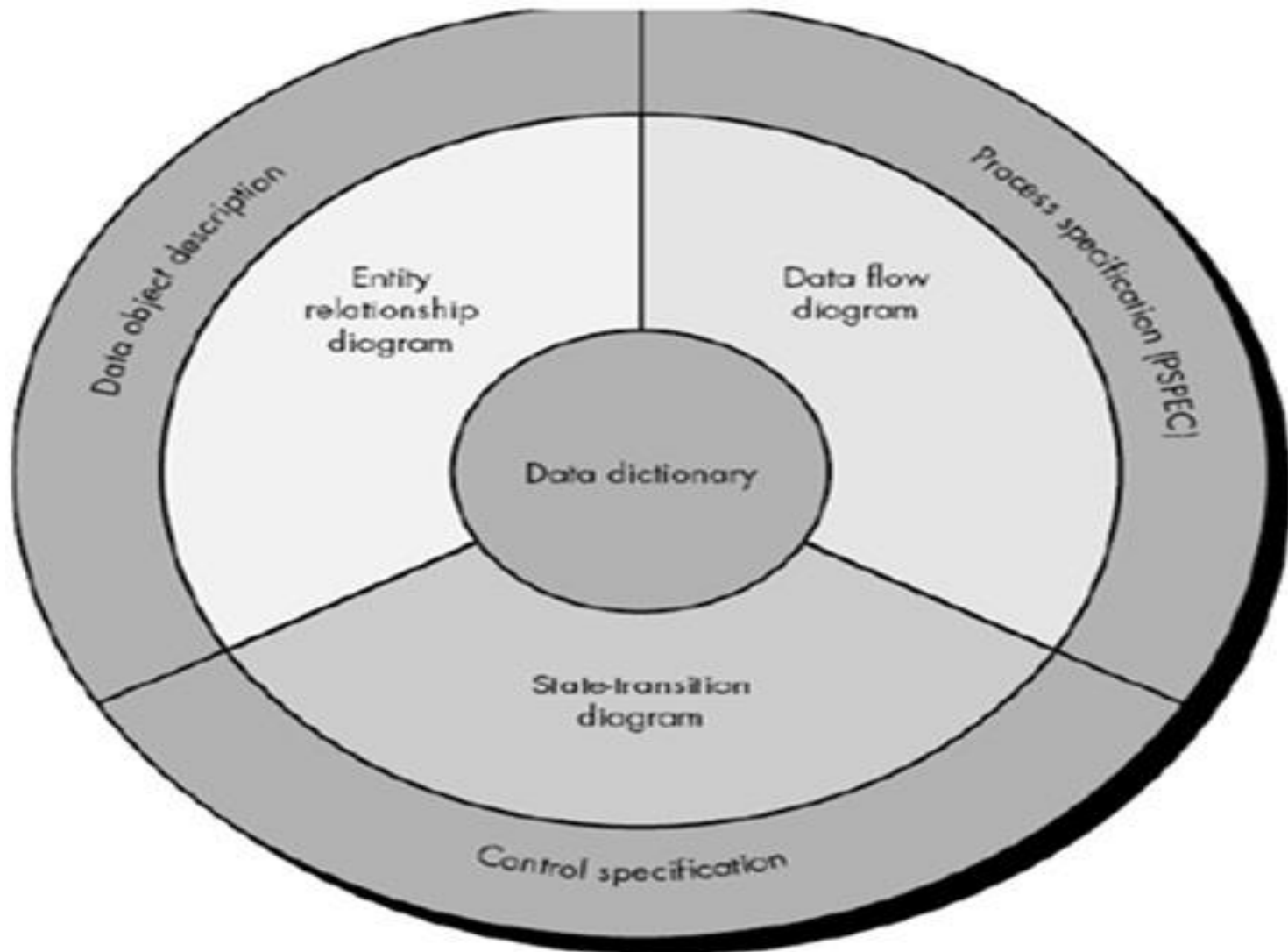


Figure 1 The structure of the analysis model

THANKS



جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

SOFTWARE ENGINEERING

Presented
by Lecturer : Wafaa Ali

CHAPTER ONE

INTRODUCTION TO SOFTWARE ENGINEERING

Topics

1.1 Software Definition

1.2 Software Characteristics

1.3 Software Applications

1.4 Software Crisis

1.5 The Attributes of Good Software

1.6 Software Engineering Definition

1.7 The Characteristics of Software Engineer

1.8 The Evolving Role of Software

1.9 The Goals of Software Engineering

1.1 SOFTWARE DEFINITION

Software is

- (1) Instructions (computer programs) that when executed provide desired function and performance,**
- (2) Data structures that enable the programs to adequately manipulate information**
- (3) Documents that describe the operation and use of the programs.**

There are two fundamental types of software product :

1- Generic products : developed to be sold to a range of different customers.

Ex: product include software for PCs such as drawing packages.

2- Customized (or bespoke) products : developed for a single customer according to their specification.

Ex: air traffic control system.

1.2 SOFTWARE CHARACTERISTICS

Software is a logical rather than a physical system element.

Therefore, software has characteristics that are considerably different than those of hardware:

- 1. Software is developed or engineered; it is not manufactured in the classical sense.**

Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.

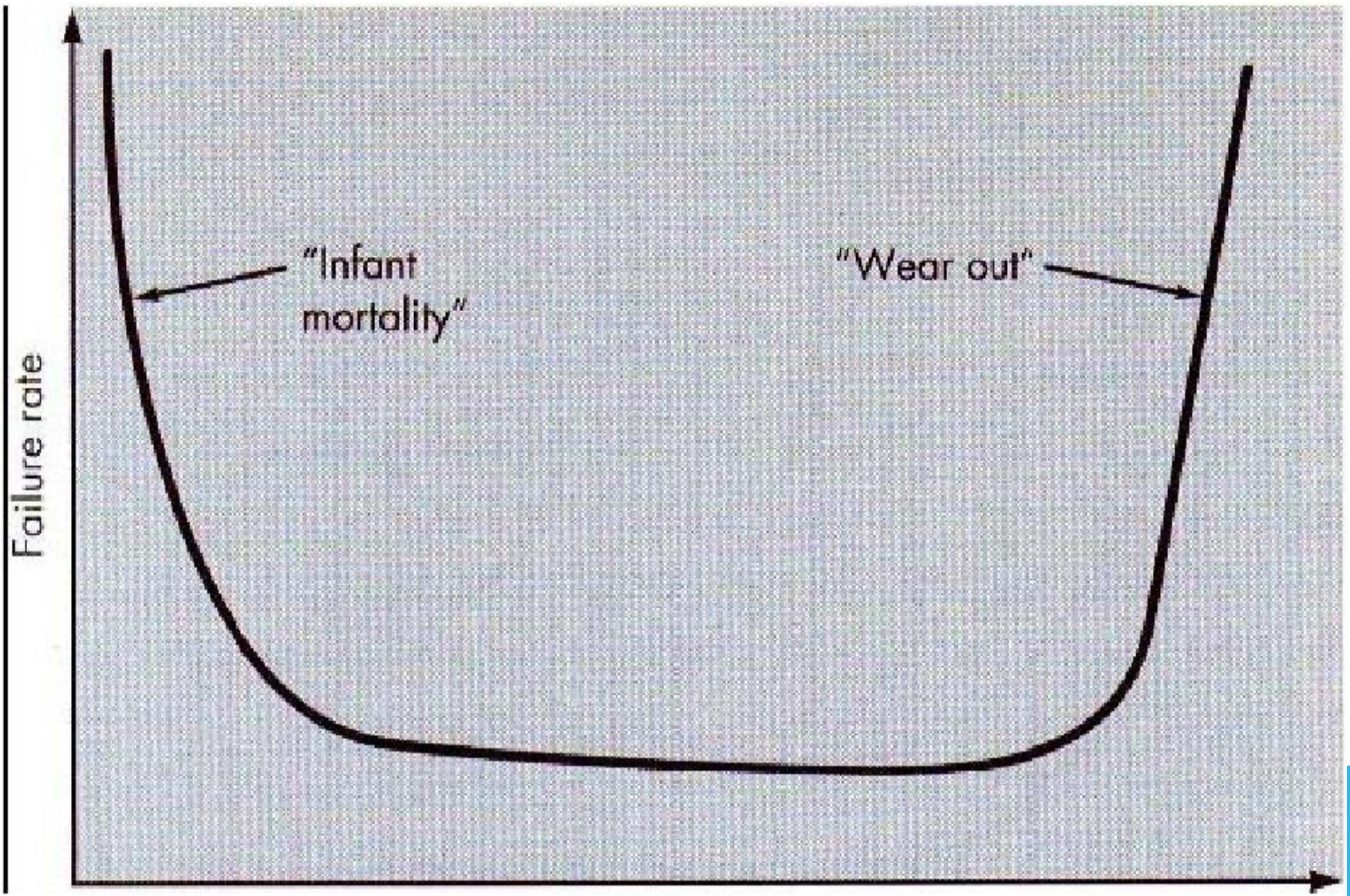


Figure (1) Failure curve for hardware

2. Software doesn't "wear out." or get tired

- **Figure (1) depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects).**
- **Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, therefore, the failure rate curve for software should take the form of the "idealized curve" shown in Figure (2).**

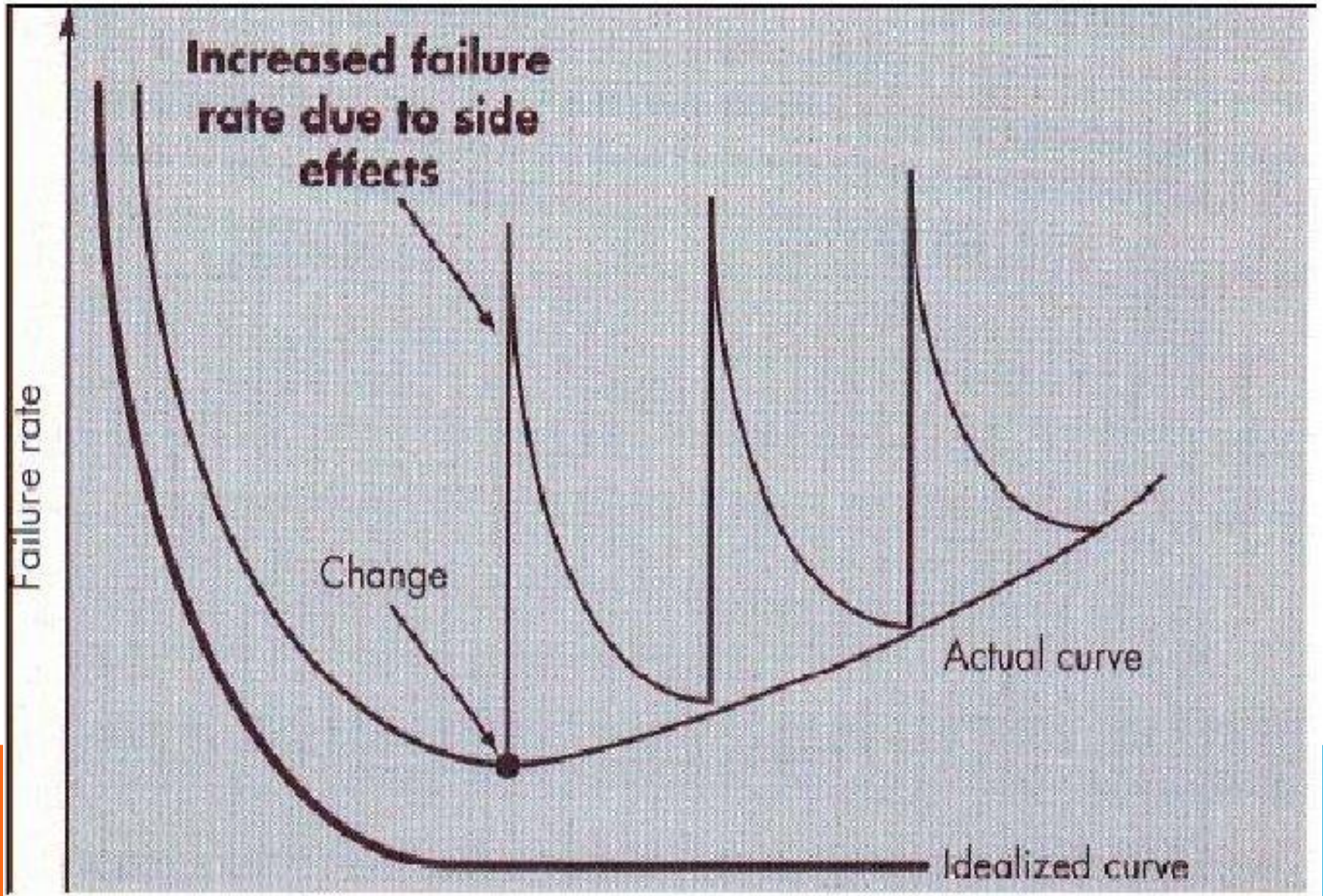


Figure (2) Idealized and actual failure curves for software

The idealized curve is a gross oversimplification of actual failure models for software. During its life, software will undergo change (maintenance).

Another aspect of wear illustrates the difference between hardware and software. 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 considerably more complexity than hardware maintenance.

3. Software continues to be custom built.

In the hardware world, component reuse is a natural part of the engineering process. In the software world, it is something that has only begun to be achieved on a broad scale.

A software component should be designed and implemented so that it can be reused in many different programs.

Modern reusable components encapsulate both data and the processing applied to the data, enabling the software engineer to create new applications from reusable parts.

For example, today's graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms.

1.3 SOFTWARE APPLICATIONS

The following software areas indicate the breadth of potential applications:

- 1. System software:** It is a collection of programs written to service other programs. Some system software (e.g. compilers, editors, and file management utilities) .
- 2. Real-time software:** Software that monitors/analyzes controls real world events as they occur is called real time.
real-time system must respond within strict time constraints.
- 3. Business software:** Business information processing is the largest single software application area.

- 4. Engineering and scientific software: modern applications within the engineering/scientific area are moving away from conventional numerical algorithms. Computer aided design, system simulation, and other interactive applications have begun to take on real-time.**

- 5. Embedded software: Intelligent products have become commonplace in nearly every consumer and industrial market (e.g., keypad control for a microwave oven or digital functions in an automobile such as fuel control, and braking systems).**

- 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 are software that incorporates executable instructions (e.g., HTML, Perl, or Java), and data (e.g., hypertext and a variety of visual and audio formats).**

- 8. Artificial intelligence software: It makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Expert systems, also called knowledge-based systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing are representative of applications within this category.**

1.4 SOFTWARE CRISIS

- ❖ **Whether we call it a software crisis or affliction, the term alludes to a set of problems that are encountered in the development of computer software. The problems are not limited to software that “doesn’t function properly”. Rather, the affliction encompasses problems associated with how we develop software, how we support a growing volume of existing software, and how we can expect to keep pace with growing demand for more software.**

1.5 THE ATTRIBUTES OF GOOD SOFTWARE

The specific set of attributes which you might expect from a software system obviously depends on its application :

- 1- Maintainability:** software should be written in such a way that it may evolve to meet the changing needs of customer.
- 2- Dependability:** software dependability has a range of characteristics, including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure.

Dependability can be achieved in three ways:

Error-free software development.

- **Develop exceptional software that enables software to perform work.**

Detecting errors.

- 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.**

1.6 SOFTWARE ENGINEERING DEFINITION

Software engineering is an engineering discipline which is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.

software engineering is the application of principles used in the field of engineering, which usually deals with physical systems, to the design, development, testing, deployment and management of software systems.

What is The Deference Between Software Engineering and Computer Science?

- ❖ **Computer science is concerned with theory , software engineering is concerned with the practicalities of developing useful software.**
- ❖ **Computer science is a discipline that involves the design and understanding of computers and computational processes. It is a broad scientific topic. It includes the study of how data is processed, the security of networks, organizing databases, Software Engineering is a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements.**

The Difference Between Computer Science and Software Engineering

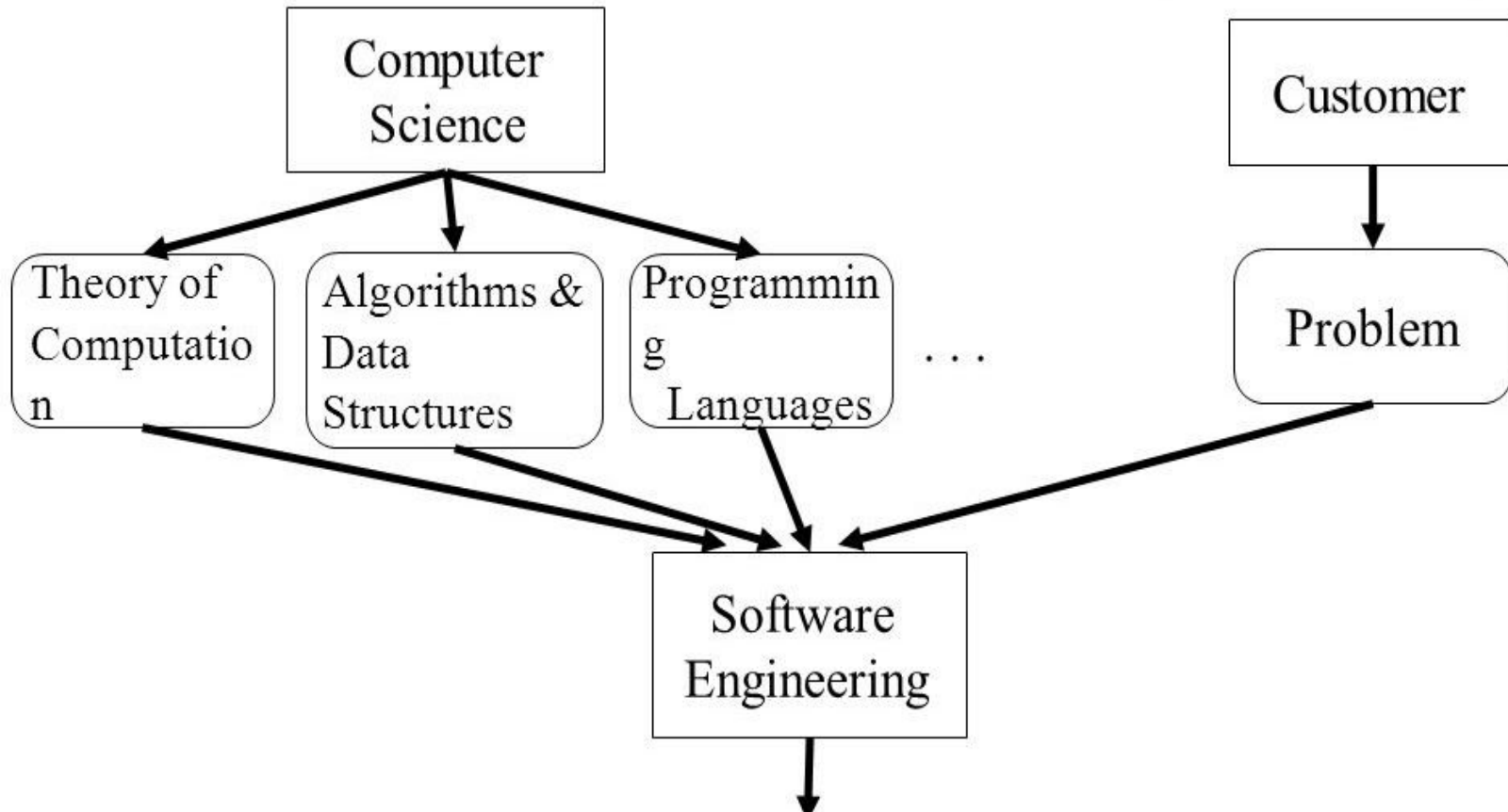


Figure (3) The relationship between computer science and software engineering

1.7 THE CHARACTERISTICS OF SOFTWARE ENGINEER

- 1- Good programmer and fluent in one or more programming language.**
- 2- Well versed data structure and approaches.**
- 3- Familiar with several designs approaches.**
- 4- Be able to translate vague (not clear) requirements and desires into precise specification.**
- 5- Be able to converse with the user of the system in terms of application not in “computer”.**
- 6- Able to a build a model. The model is used to answer questions about the system behavior and its performance.**
- 7- Communication skills and interpersonal skills.**

1.9 THE GOALS OF SOFTWARE ENGINEERING

- ❖ **To produce software that is absolutely correct.**
- ❖ **To produce software with minimum effort.**
- ❖ **To produce software at the lowest possible cost.**
- ❖ **To produce software in the least possible time.**
- ❖ **To produce software that is easily maintained and modified.**
- ❖ **To maximize the profitability of the software production effort.**

GOOD LUCK



جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

Software Engineering

LEC.10

2-Entity relationship diagram (ERD)

- An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes.

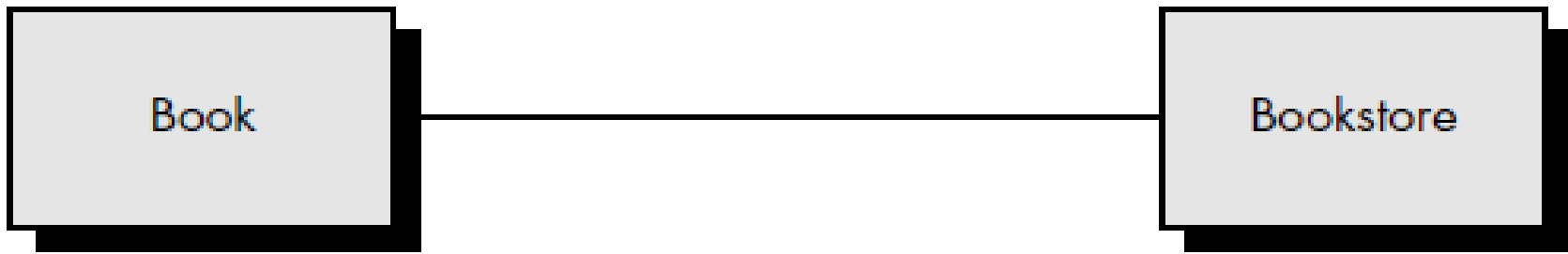
- The data model consists of three interrelated pieces of information:
 - The Entity:, the attributes that describe the data object and the relationships that connect data objects to one another.
1. **Entity:** important individual thing, object, concept in the real world of interest (e.g.the person called John, my red car, ...)

2-Attributes

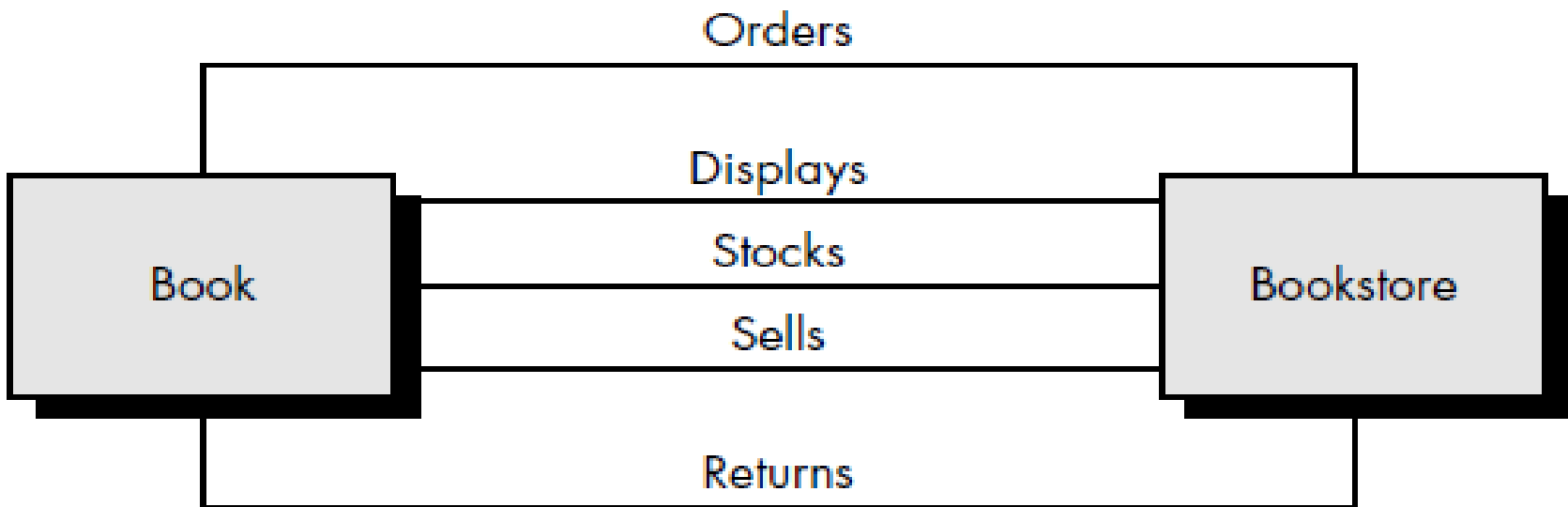
- Attributes define the properties of a data object and take on one of three different characteristics. They can be used to
 - (1) name an instance of the entity.
 - (2) describe the instance.
 - (3) make reference to another instance in another table. In addition, one or more of the attributes must be defined as an identifier-that is, the identifier attribute becomes a "key" when we want to find an instance of the data object.

3- Relationships

- Data objects are connected to one another in different ways. Consider two data objects, **book and bookstore**. These objects can be represented using the simple notation illustrated in Figure (1). A connection is established between book and bookstore because the two objects are related. **But what are the relationships?** To determine the answer, we must understand the role of books and bookstores within the context of the software to be built. We can define a set of object/relationship pairs that define the relevant relationships. For example,
 - A bookstore orders books.
 - A bookstore displays books.
 - A bookstore stocks books.
 - A bookstore sells books.
 - A bookstore returns books.



(a) A basic connection between objects



(b) Relationships between objects

Figure (1) Relationships

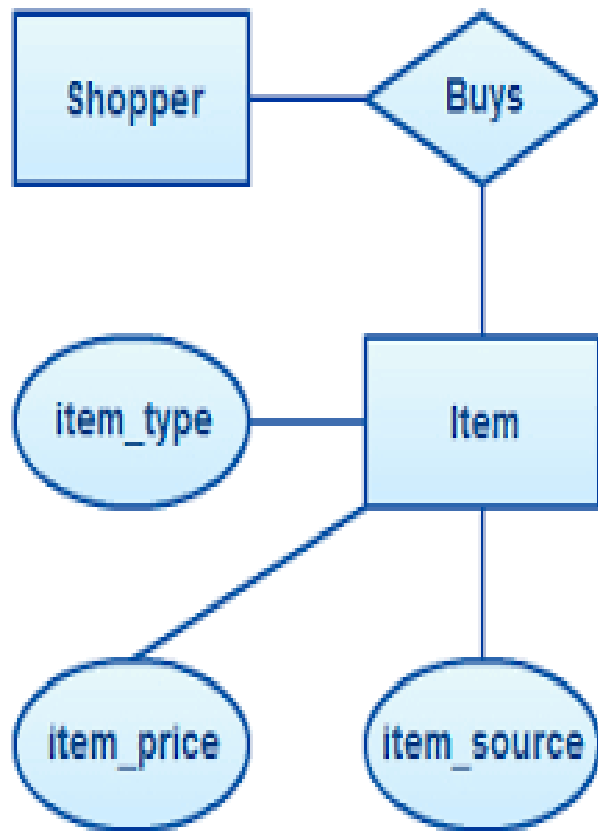
Creation of ERD

- The entity relationship diagram enables a software engineer to fully specify the data objects that are input and output from a system, the attributes that define the properties of these objects and their relationships. Like most elements of the analysis model, the ERD is constructed in an iterative manner. The following approach is taken:
 1. During requirements elicitation, customers are asked to list the “things” that the application or business process addresses. These “things” evolve into a list of input and output data objects as well as external entities that produce or consume information.

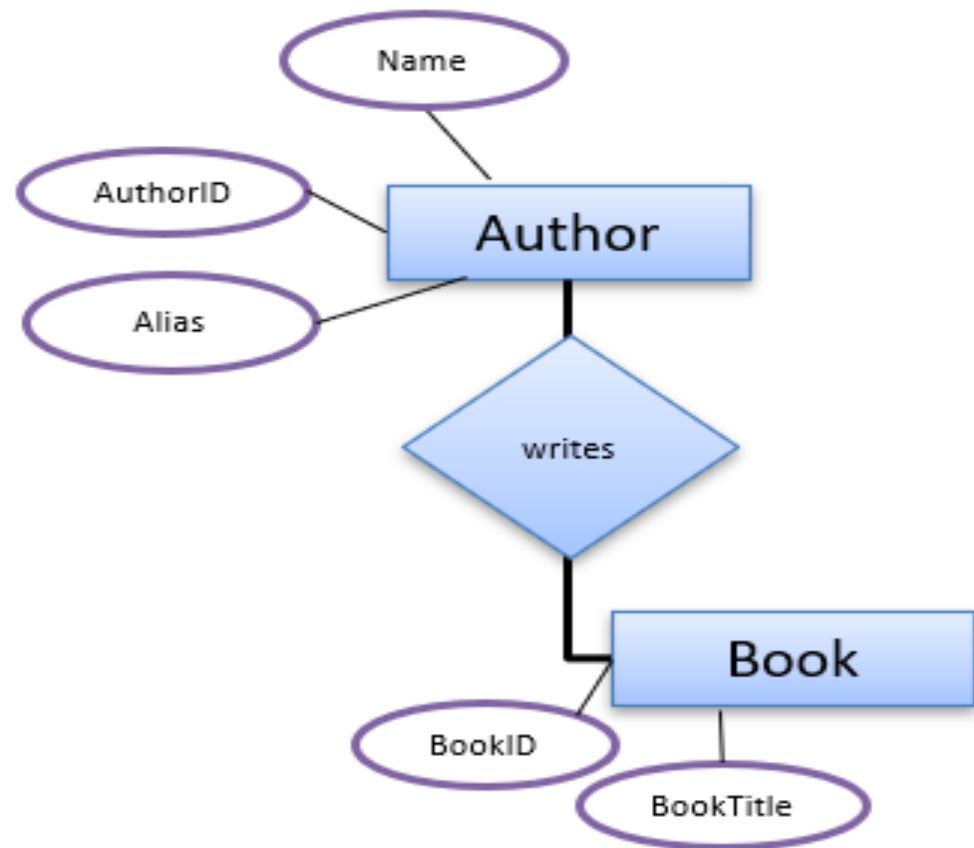
2. Wherever a connection exists, the analyst and the customer create one or more object/relationship pairs.
3. The attributes of each entity are defined.
4. An entity relationship diagram is formalized and reviewed.

Tips for Effective ER Diagrams

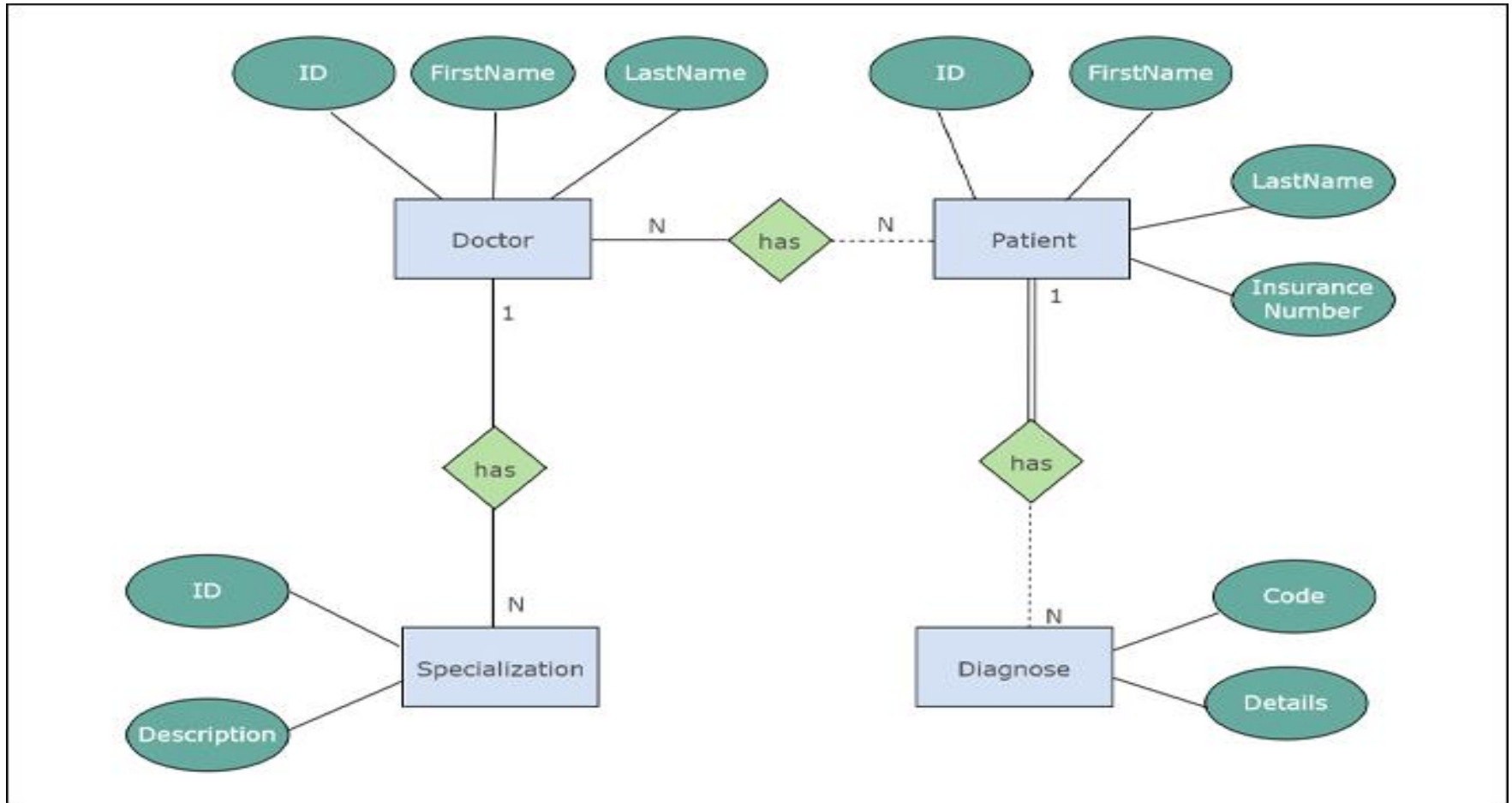
- Make sure that each entity only appears once per diagram.
- Name every entity, relationship, and attribute on your diagram.
- Examine relationships between entities closely. Are they necessary? Are there any relationships missing? Eliminate any redundant relationships.
- Don't connect relationships to each other.
- Use colors to highlight important portions of your diagram.



S



Entity Relationship Diagram Examples



THANKS



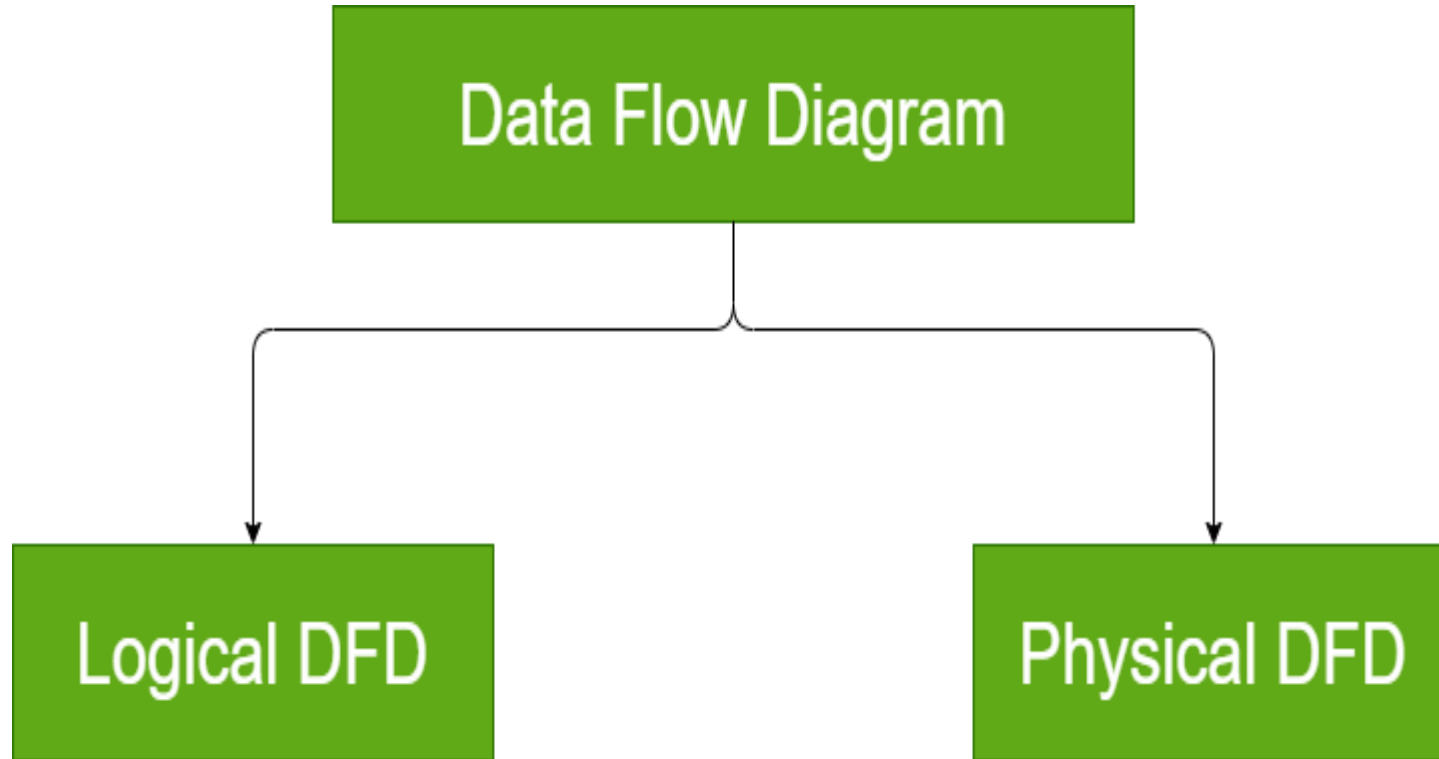
جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

SOFTWARE ENGINEERING LEC.11

. DATA FLOW DIAGRAM (DFD) :

- ◉ Data refers to information, flow refers to move, and diagram refers to a picture to represent something. So, DFD is basically the graphical representation of the flow of data or information. It is a framework or pattern of the data systems. It includes storing data, data input, data output. It is describes as the process of taking the data as input, storing the data, giving the data as output. It describes the path of data that completes the process.

TYPES OF DFD :



TYPES OF DFD

- ◉ Logical data flow diagram mainly focuses on the system process. It illustrates how data flows in the system. Logical DFD is used in various organizations for the smooth running of system. Like in a Banking software system, it is used to describe how data is moved from one entity to another.
- ◉ Physical data flow diagram shows how the data flow is actually implemented in the system. Physical DFD is more specific and close to implementation.

COMPONENTS OF DATA FLOW DIAGRAM:

- ◉ **Entities:**

Entities include source and destination of the data. Entities are represented by rectangle with their corresponding names.

- ◉ **Process:**

The tasks performed on the data is known as process. Process is represented by circle. Somewhere round edge rectangles are also used to represent process.

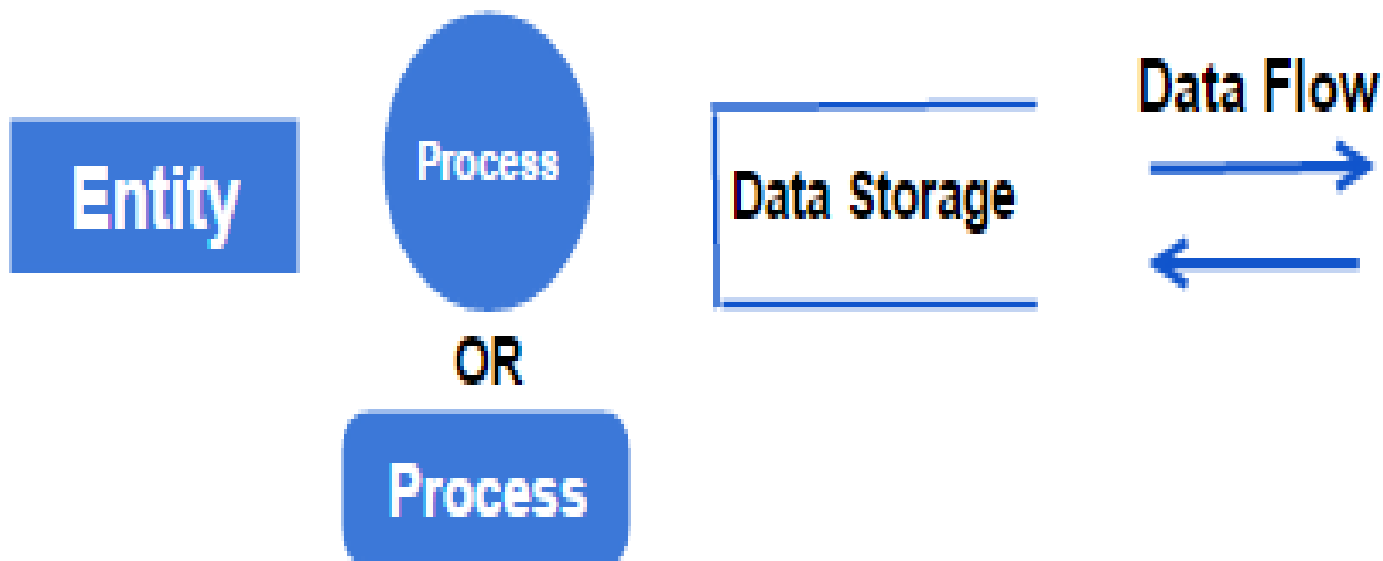
- ◉ **Data Storage:**

Data storage includes the database of the system. It is represented by rectangle with both smaller sides missing or in other words within two parallel lines.

- ◉ **Data Flow:**

The movement of data in the system is known as data flow. It is represented with the help of arrow. The tail of the arrow is source and the head of the arrow is destination.

Components of DFD



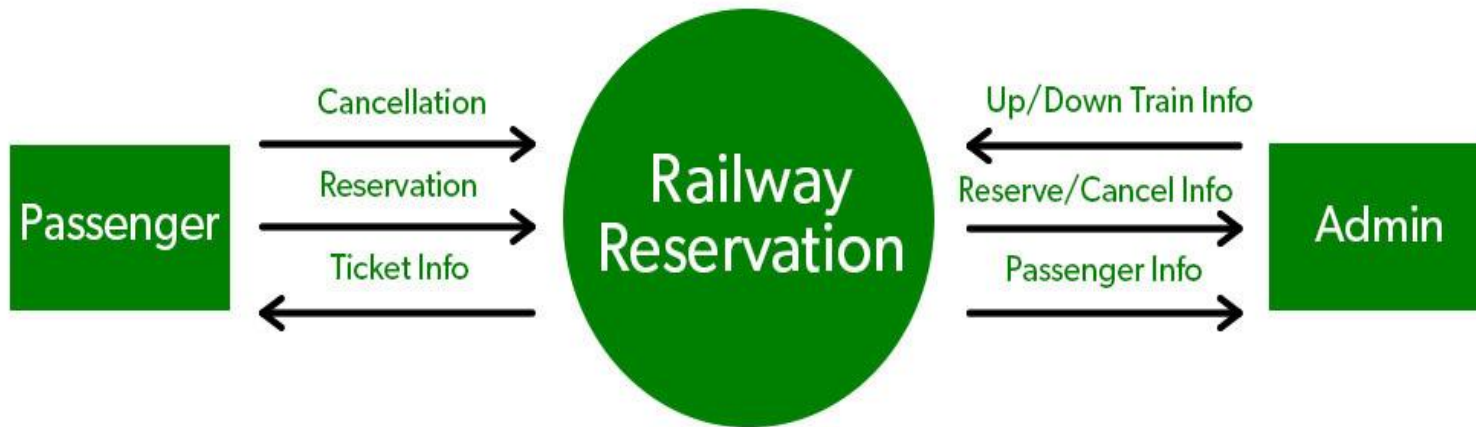
ADVANTAGES OF DFD

- it helps us to understand the functioning and the limits of a system.
- It is a graphical representation which is very easy to understand as it helps visualize contents.
- Data Flow Diagram represent detailed and well explained diagram of system components.
- It is used as the part of system documentation file.
- Data Flow Diagrams can be understood by both technical or nontechnical person because they are very easy to understand.

DISADVANTAGES OF DFD

- ⦿ At times DFD can confuse the programmers regarding the system.
- ⦿ Data Flow Diagram takes long time to be generated, and many times due to this reasons analysts are denied permission to work on it.

EXAMPLE:



0-LEVEL DFD



جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

Software Engineering

LEC12

Presented
by Lecturer : Wafaa Ali

Software Design Definition

- Design is a meaningful engineering representation of something that is to be built. In the software engineering context, design focuses on four major areas of concern: data, architecture, interfaces and components.
- designer's goal is to produce a model or representation of an entity that will later be built.
- The process by which the design model is developed is described by belady:

There are two major phases to any design process:

1. Diversification is the acquisition of a repertoire of alternatives, the raw material of design: components, component solutions, and knowledge, all contained in catalogs, textbooks and the mind.

2. Convergence, the designer chooses and combines appropriate elements from this repertoire to meet the design objectives, as stated in the requirements document and as agreed to by the customer.

Activities of Software Design:-

- Software design sits at the technical kernel of software engineering and is applied regardless of the software process model that is used. Beginning once software requirements have been analyzed and specified, software design is the first of three technical activities-design, code generation, and test-that are required to build and verify the software. Each of the elements of the analysis model provides information that is necessary to create the four design models required for a complete specification of design.

- The design task produces a data design, an architectural design, an interface design and a component design.
- The flow of information during software design is illustrated and design levels in Figure (1)

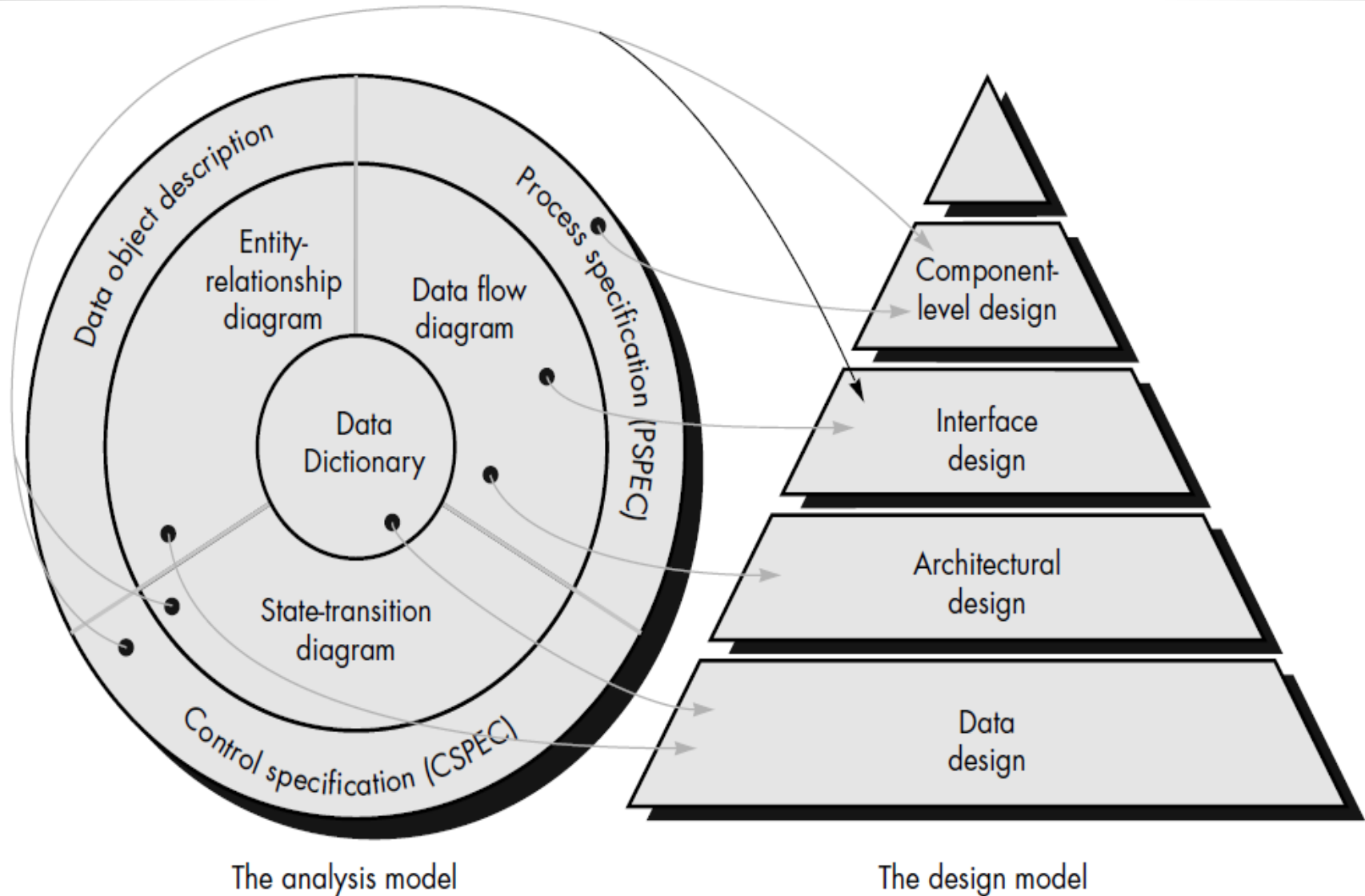


Figure (1) Translating the analysis model into a software design

1- Data Design

- The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software. The data objects and relationships defined in the entity relationship diagram and the detailed data content depicted in the data dictionary provide the basis for the data design activity.
- Part of data design may occur in conjunction with the design of software architecture.
- More detailed data design occurs as each software component is designed.

2-Architectural Design

- The architectural design defines the relationship between major structural elements of the software, the “design patterns” that can be used to achieve the requirements that have been defined for the system and the constraints that affect the way in which architectural design patterns can be applied.
- The architectural design representation-the framework of a computer-based system-can be derived from the system specification, the analysis model and the interaction of subsystems defined within the analysis model.

3- Interface Design

- The interface design describes how the software communicates within itself, with systems that interoperate with it and with humans who use it.
- An interface implies a flow of information (e.g., data and/or control) and a specific type of behavior.
- Therefore, data and control flow diagrams provide much of the information required for interface design.

4- The component-level design

- The component-level design transforms structural elements of the software architecture into a procedural description of software components. Information obtained from the PSPEC, CSPEC and STD serve as the basis for component design.

why is design so important?

The importance of software design can be stated with a single word—quality.

- Design is the place where quality is fostered in software engineering.
- Design is the only way that we can accurately translate a customer's requirements into a finished software product or system.

- **Three characteristics that serve as a guide for the evaluation of a good design:**
- The design must implement all of the explicit requirements contained in the analysis model
- The design must be a readable, understandable guide for those who generate code and for those who test software.
- The design should provide a complete picture of the software, addressing the data, functional and behavioral domains from an implementation perspective.

THANKS



جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

Software Engineering

LEC14

Presented
by Lecturer : Wafaa Ali

Architectural Style

Style describes a system category that encompasses:

- 1- A set of components (e.g., a database, computational modules) that perform a function required by a system;
- 2- A set of connectors that enable “communication, coordinations and cooperation” among components.
- 3- Constraints that define how components can be integrated to form the system.
- 4- Semantic models that enable a designer to understand the overall properties of a system.

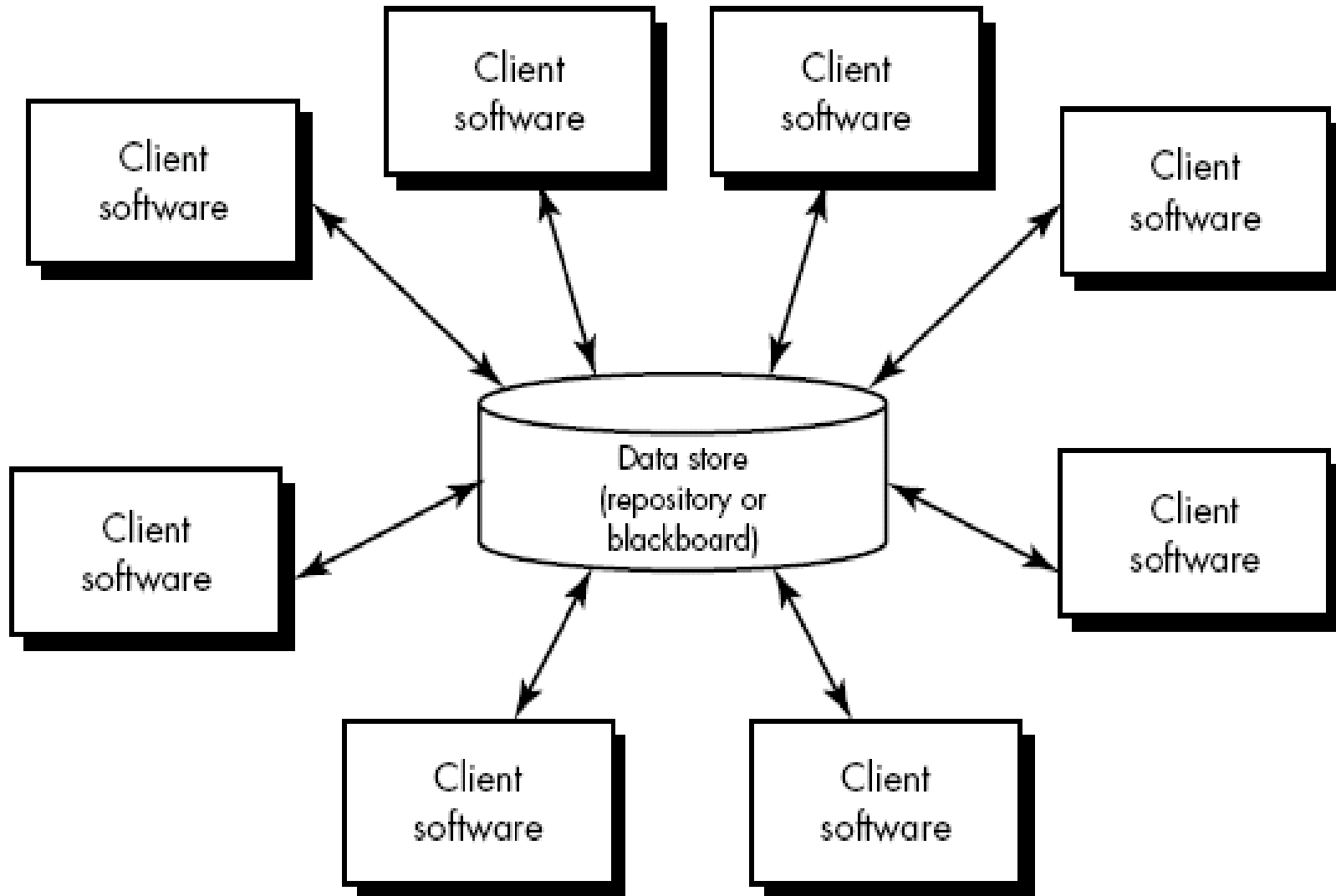
It can be represent by

- Data-centered architecture.
- Data flow architecture.
- Call and return architecture.
- Object oriented architecture.
- Layered architecture.

Data-centered architecture

- ▶ A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete or otherwise modify data within the store.
- ▶ Client software accesses a central repository which is in passive state (in some cases). client software accesses the data independent of any changes to the data or the actions of other client software. So, in this case transform the repository into a “Blackboard”. A blackboard sends notification to subscribers when data of interest changes, and is thus active.
- ▶ Existing components can be changed and new client components can be added to the architecture without concern about other clients. Data can be passed among clients using the blackboard mechanism. So Client components independently execute processes.

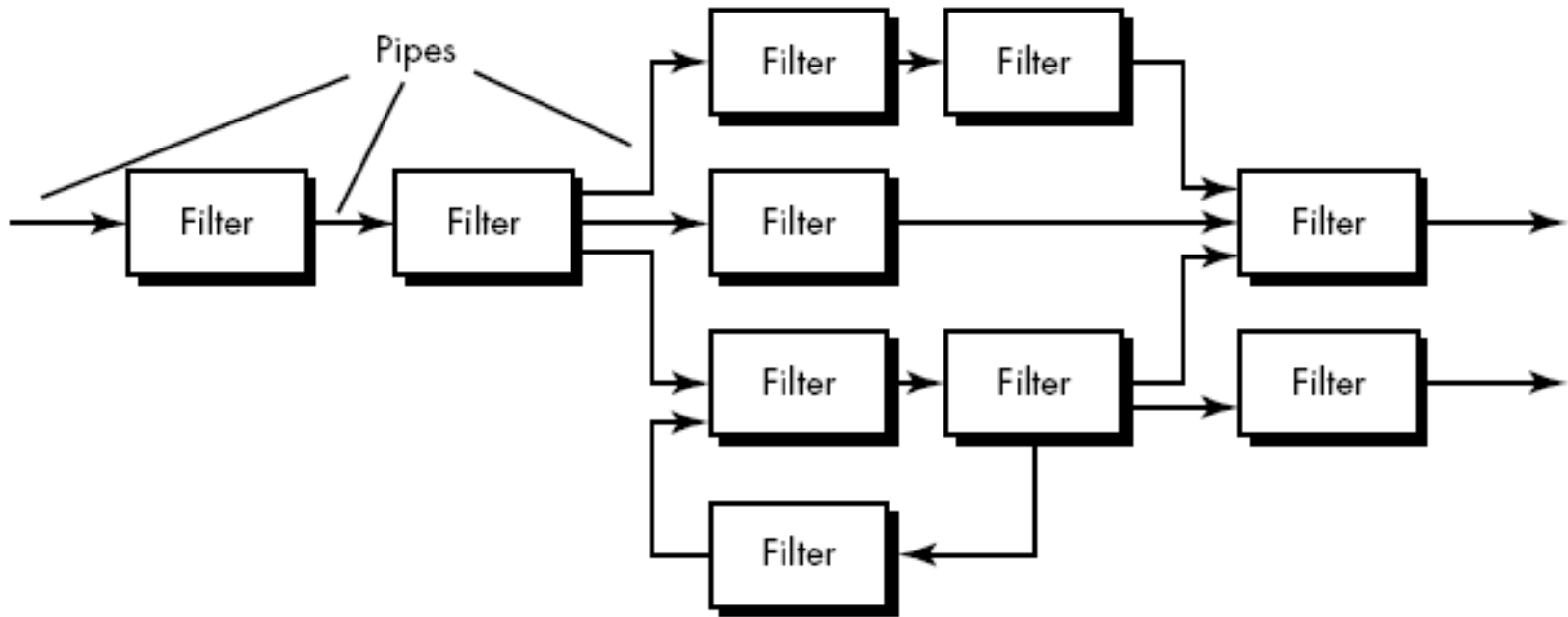
Data-Centered Architecture



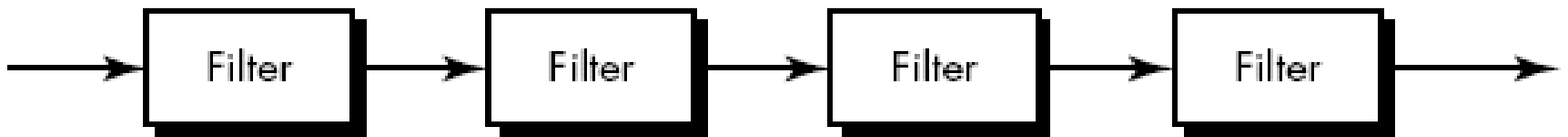
Data Flow Architecture

- ▶ This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.
- ▶ A pipe and filter pattern has a set of components, called filters, connected by pipes that transmit data from one component to the next.
- ▶ Each filter works independently (i.e. upstream, downstream) and is designed to expect data input of a certain form, and produces data output (to the next filter) of a specified form. the filter does not require knowledge of the working of its neighboring filters. If the data flow degenerates into a single line of transforms, it is termed batch sequential.

Data Flow Architecture



Pipes and filters



Batch Sequential

Call and Return Architecture

Architecture style enables a software designer to achieve a program structure that is relatively easy to modify and scale.

Two sub-styles exist within this category:

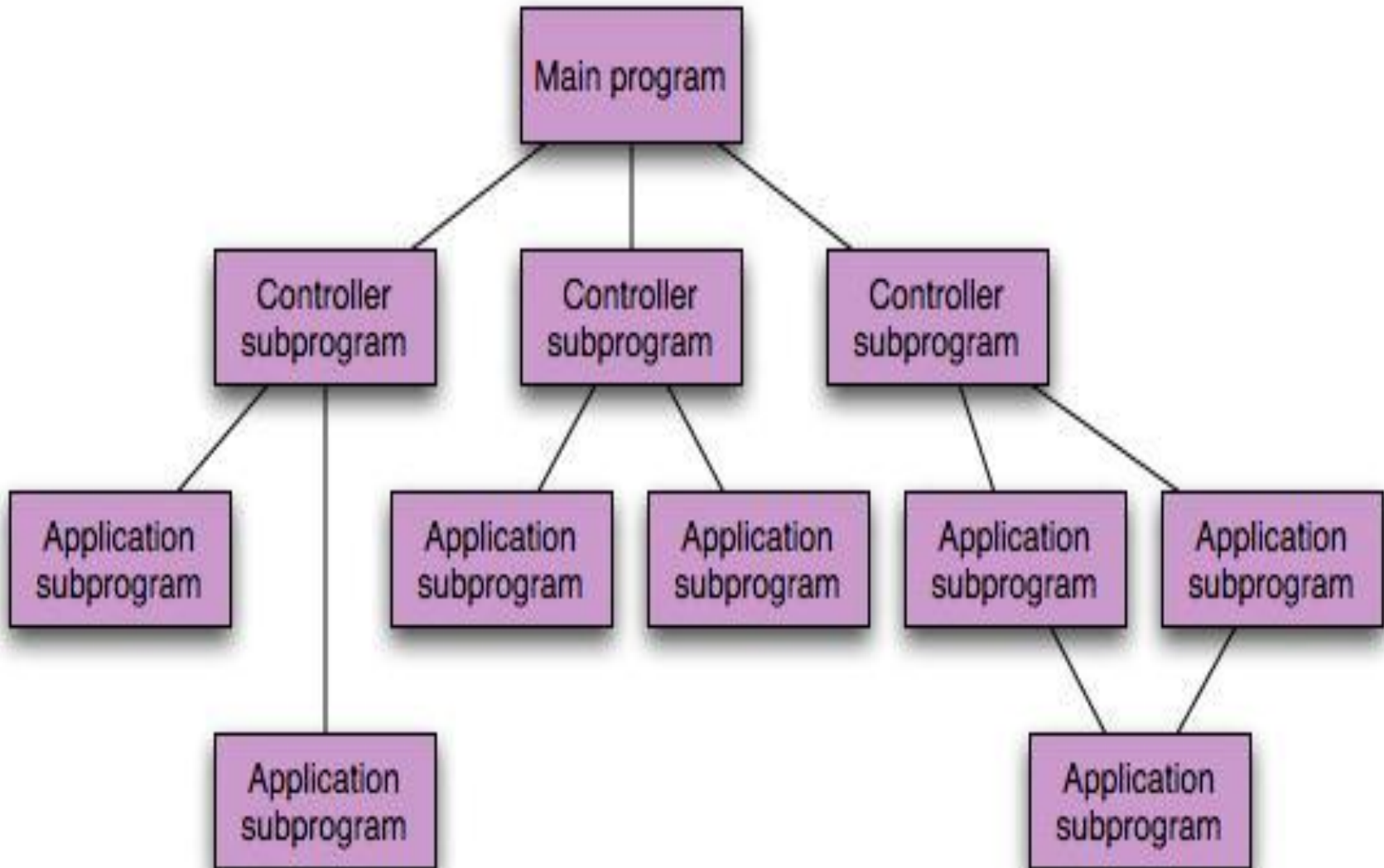
1- **Main/sub program architecture:**

Program structure decomposes function into a control hierarchy where a “main” program invokes a number of program components, which in turn may invoke still other components.

2- **Remote procedure Call architecture:**

The components of a main program/subprogram architecture are distributed across multiple computers on a network.

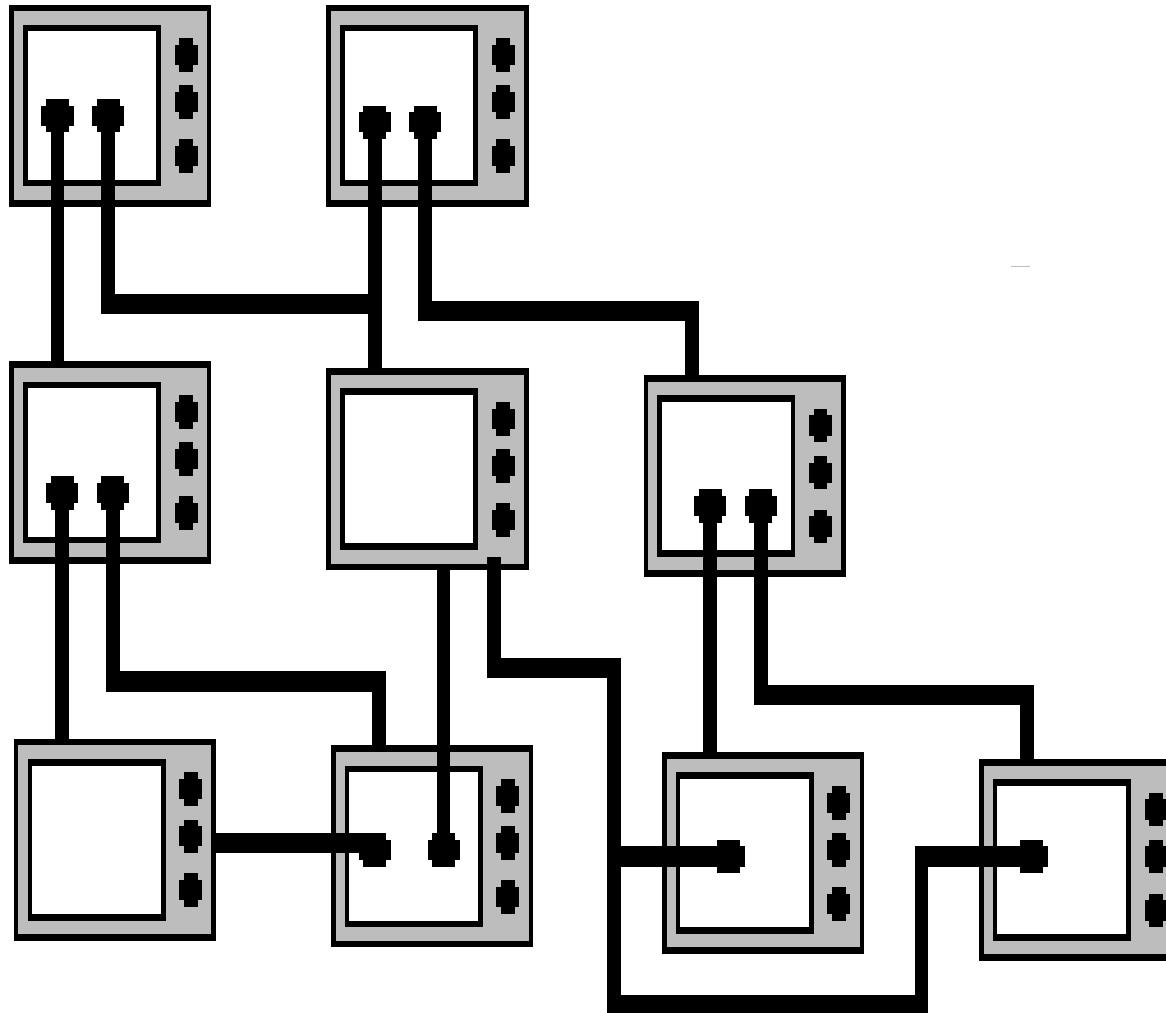
Call and Return Architecture



Object-Oriented Architecture

- ▶ The object-oriented paradigm, like the abstract data type paradigm from which it evolved, emphasizes the bundling of data and methods to manipulate and access that data (Public Interface).
- ▶ Components of a system summarize data and the operations that must be applied to manipulate the data. Communication and coordination between components is accomplished via message passing.

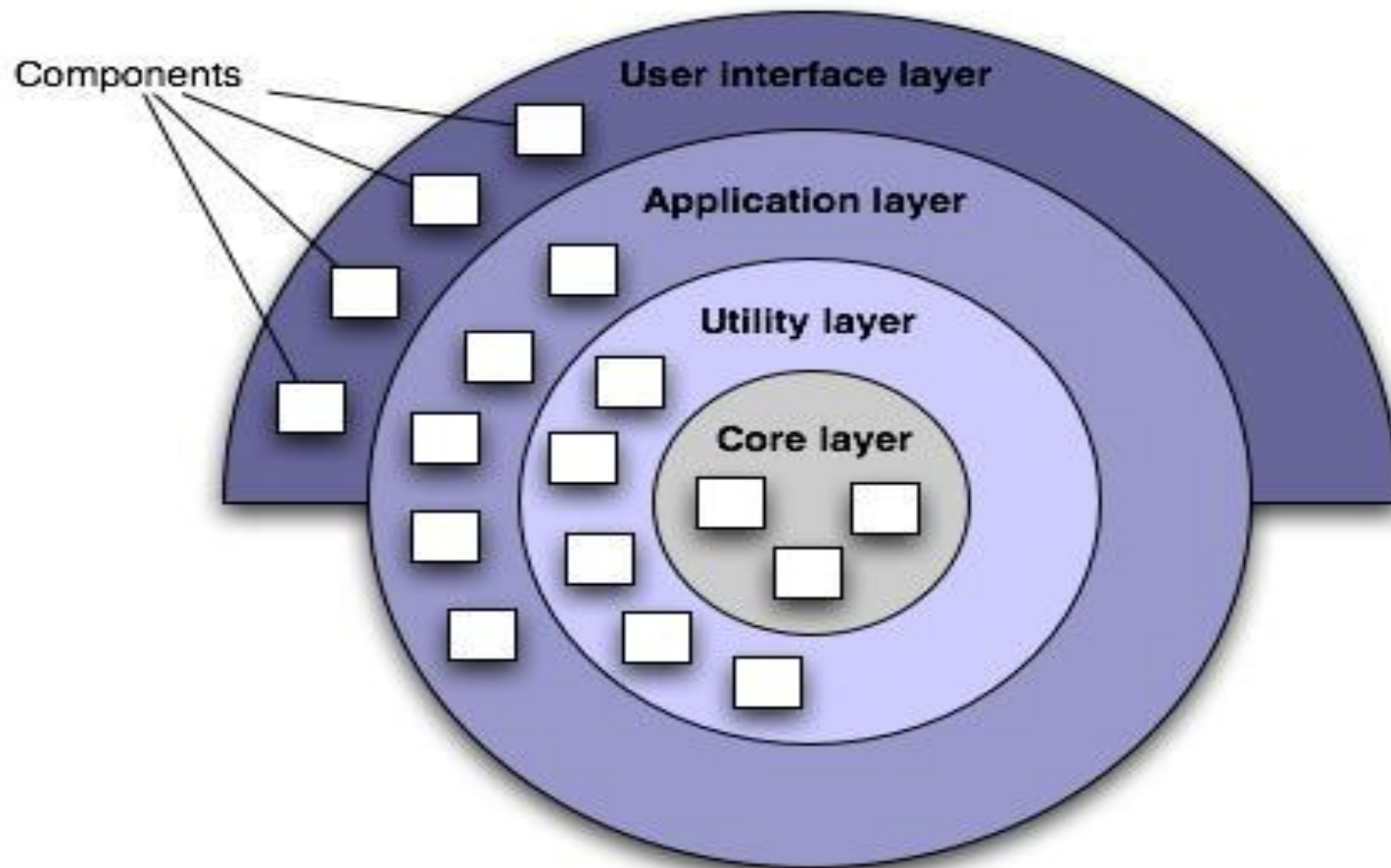
Object-Oriented Architecture



Layered Architecture

- ▶ A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.
- ▶ At the outer layer, components examine user interface operations. At the inner layer, components examine operating system interfacing. Intermediate layers provide utility services and application software functions.

Layered Architecture



Evaluation criteria can be applied during early design reviews:

- 1-** The length and complexity of the written specification of the system and its interface provide an indication of the amount of learning required by users of the system.
- 2-** The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.
- 3-** The number of actions, tasks and system states indicated by the design model imply the memory load on users of the system.
- 4-** Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

THANKS



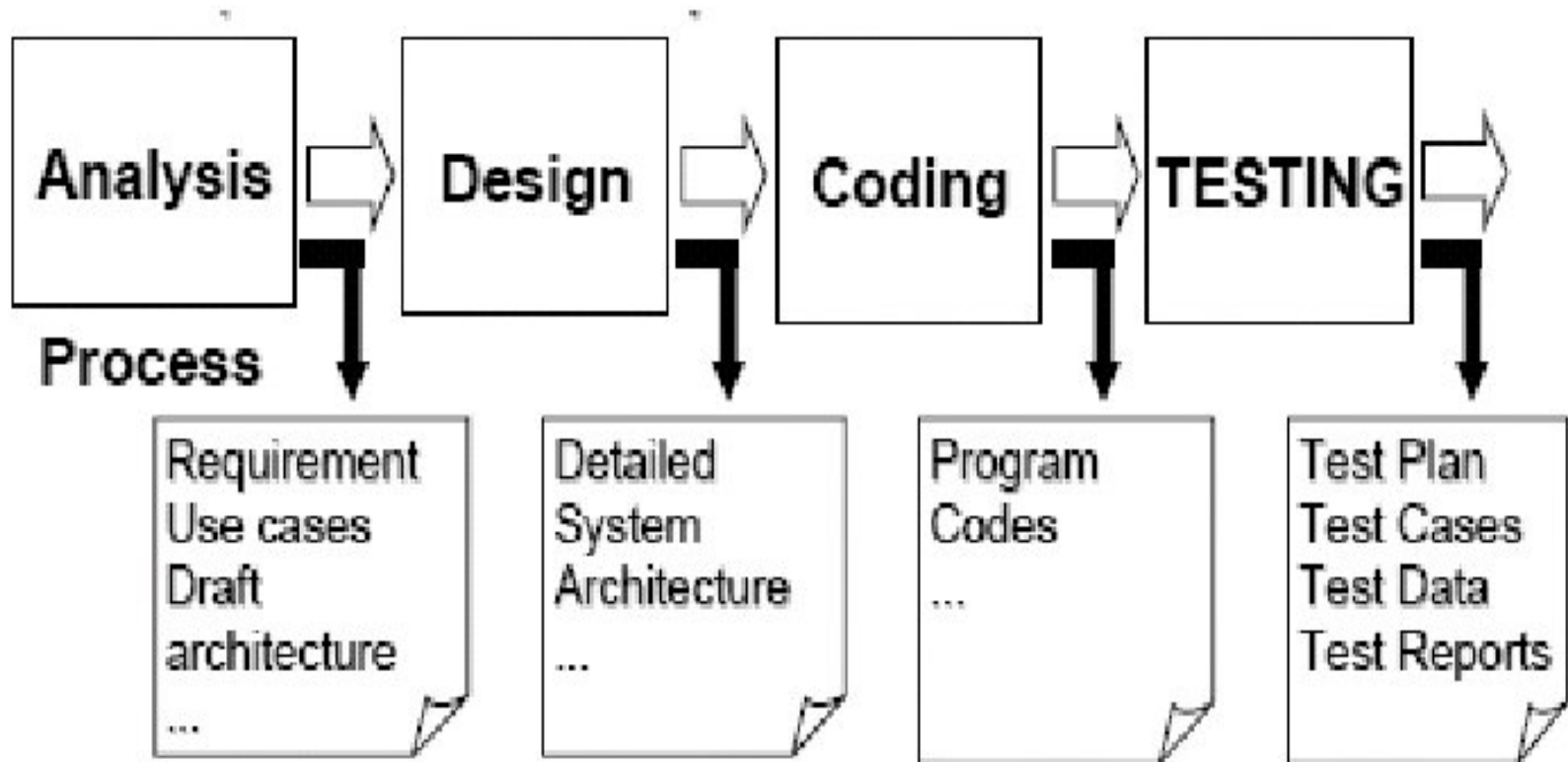
جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

Software Engineering

LEC.15

1 -Introduction

- ▶ Revisit the Software Life Cycle
- ▶ Classical Life Cycle or (Linear Sequential Software, Process Model or Waterfall model).



2- Why Testing?

- ❖ Two objectives verification and validation.
- ❖ To uncover errors 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.

3- What Testing?

- ▶ At a lower level, testing involves designing a series of “TEST CASES” or (“TEST SUITE”) to uncover errors and validate conformance to requirements.
- ▶ At a higher level, testing involves formulating a test plan and test strategy for the execution of the testing process.

4 -Work Products or (Deliverables) of Testing Stage

- ❖ Test Plan (Test Strategies).
- ❖ Test Report (includes Test Cases, Test Data, etc..).

5- Testing is Important

- ❖ “Testing is as important, if not more important than coding“. Your clients will not accept programs that are full of bugs, or worse still, don't meet the requirements.

❖ Example on Testing

❖ Requirement:

Write a program to assign an alphabetic grade to raw marks as follows:

Marks	Grade
0 - 49	'F'
50 - 59	'E'
60 - 69	'D'
70 - 79	'C'
80 - 89	'B'
90 - 100	'A'

❖ Program without Testing

```
if (marks >= 90 && marks < 100) return 'A';  
else if (marks >= 80 && marks < 90) return 'B';  
else if (marks >= 70 && marks < 80) return 'C';  
else if (marks >= 60 && marks < 70) return 'D';  
else if (marks >= 50 && marks < 60) return 'E';  
else return 'F';
```

- ❖ This code can compile and run (Compiler only catches syntax errors, not semantics errors or logical errors).
- ❖ Is the program working? (May be!) (Verification).
- ❖ Is the program correct? Does the program meet its specification? (NO!) (Validation)
- ❖ Is the program efficient? (This is an issue on Software Quality Assurance (SQA), not testing-There are 10 comparisons in the code, many of them are redundant!).

❖ Test Cases

- ❖ To verify and validate the program, we design “a series of test cases”. Each test case contains a specific input and the expected output. For examples:

Test Case No	Input	Expected Output	Purpose of Test
1	100	'A'	Grade 'A' upper limit
2	49	'F'	Grade 'F' upper limit
3	1	'F'	Grade 'F'
4	2	'F'	Grade 'F'
5

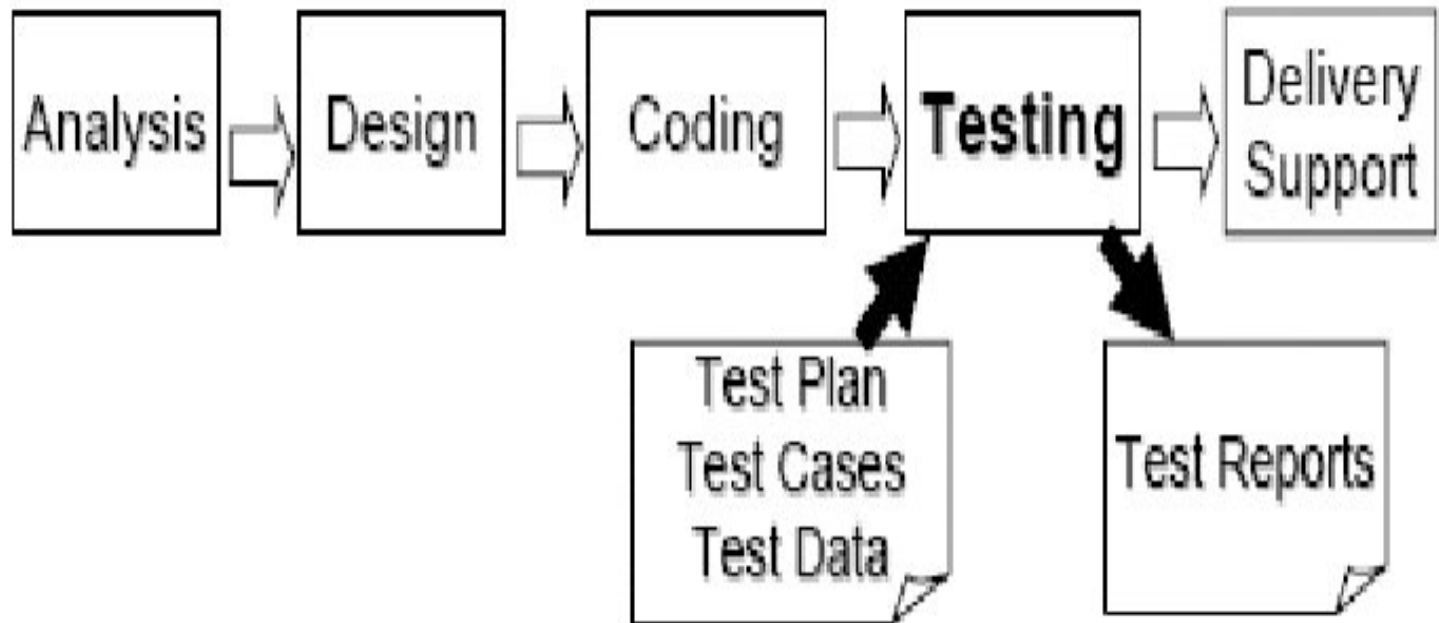
- ❖ How many test cases is “necessary and sufficient”? (101)
- ❖ How about numbers like 200 or 155? (Countable Infinity)

6- What Testing Shows?



7- Testing Stage of the Software Process

- ▶ The goal of testing is to “design a series of test cases” that has “a high likelihood of finding errors”.
- ▶ Design test cases systematically by “applying engineering principles and methods”.



8- Testing Principles

1. All tests should be traceable to customer requirements - to ensure that the software meets its intended use.
2. Tests should be planned long before testing begins -write tests first, before the coding.
3. Testing should begin “in the small” and progress toward testing “in the large” - perform unit tests, then integration tests, then validation test, and then system tests.
4. The “80-20 rule” applied - 80% of the errors are located in 20% of the software modules, isolate them and test them thoroughly.
5. To be more effective, testing should be conducted by an independent third -party testing specialist (ITG or Independent Testing Group).
- 6 Exhaustive test is not possible.

9- Who tests the system?



Developer

Understands the system,
but will test “gently”,
and is driven by “delivery”



Independent Tested

Must learn about the system,
but will attempt to “break” it,
and is driven by “quality”.

- ❖ 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.

THANKS

Testing Types

I-Manual Testing::

This type includes the testing of the Software manually i.e. without using any automated tool or any script. In this type the tester takes over the role of an end user and test the Software to identify any un-expected behavior or bug. There are different stages for manual testing like unit testing, Integration testing, System testing and User Acceptance testing

-

2-Automation Testing::

Automation testing which is also known as “Test Automation”, is when the tester writes scripts and uses another software to test the software. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly and repeatedly



Testing Methods:

I-Black Box Testing ::

The technique of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.



Advantages:

- 1- Well suited and efficient for large code segments.
- 2- Code Access not required.
- 3- Clearly separates user's perspective from the developer's perspective through visibly defined roles.
- 4- Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.

Disadvantages:

1- Limited Coverage since only a selected number of test scenarios are actually performed.

2- Inefficient testing, due to the fact that the tester only has limited knowledge about an application.

3- Blind Coverage, since the tester cannot target specific code segments or error prone areas.

4-The test cases are difficult to design.

2-White Box Testing::

White box testing is the detailed investigation of internal logic and structure of the code.

White box testing is also called glass testing or open box testing. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

Advantages:

As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.

- 1- It helps in optimizing the code.
- 2-Extra lines of code can be removed which can bring in hidden defects.
- 3-Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.

Disadvantages:

1- Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased.

2-Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested.

3- It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

3-Grey Box Testing::

Grey Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term “the more you know the better” carries a lot of weight when testing an application. Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application’s user interface, in grey box testing, the tester has access to design documents and the database. Having this knowledge, the tester is able to better prepare test data and test scenarios when making the test plan.



Advantages:

- 1-Offers combined benefits of black box and white box testing wherever possible.
- 2- Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.
- 3- Based on the limited information available, a grey box tester can design excellent test scenarios especially around communication protocols and data type handling.
- 4-The test is done from the point of view of the user and not the designer.




- **Disadvantages:**

1. Since the access to source code is not available, the ability to go over the code and test coverage is limited.
2. The tests can be redundant if the software designer has already run a test case.
3. Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

Levels of Testing

Levels of testing include the different methodologies that can be used while conducting Software Testing. Following are the main levels of Software Testing:

- **Functional Testing.**
 1. Unit Testing
 2. Integration Testing
 3. System Testing
 4. Acceptance Testing
 5. Regression testing
- **Non- functional Testing.**



The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional Testing of the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. There are five steps that are involved when testing an application for functionality.

Step 1 - The determination of the functionality that the intended application is meant to perform.

Step 2- The creation of test data based on the specifications of the application.

Step 3 - The output based on the test data and the specifications of the application.

Step 4 - The writing of Test Scenarios and the execution of test cases.

Steps 5 - The comparison of actual and expected results based on the executed test cases.

I-Unit Testing

- A unit is smallest testable piece of software
 - can be compiled, linked, loaded
 - functions/procedures, classes, interfaces
 - normally done by programmer
 - Test cases written after coding

2- Integration testing

- Test for correct interaction between system units
 - systems - built by merging existing libraries
 - modules coded by different people
 - Mainly tests the interfaces among units
 - Bottom up integration testing
 - Top down integration testing

3- System Testing

- Done by the test team
- Test of overall interaction of components
- Find disparities between implementation and specification
- Usually where most resources go to
Involves – load, performance, reliability and security testing

4- Acceptance Testing

- • Demonstrates satisfaction of user
- • Users are essential part of process
- • Usually merged with System Testing
- • Done by test team and customer
- • Done in simulated environment/real environment

5- Regression Testing

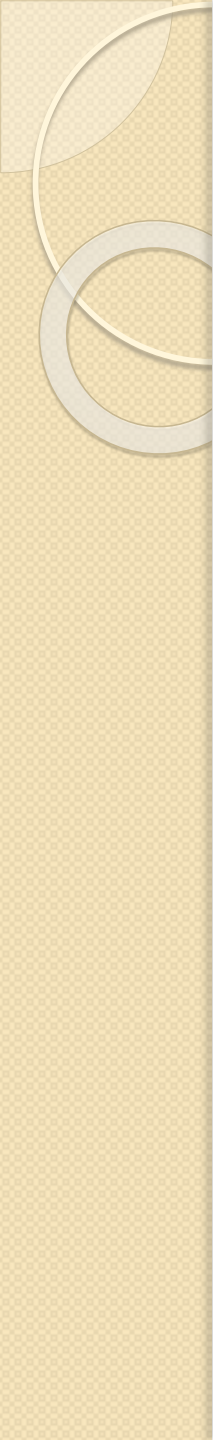
- The intent of Regression testing is to ensure that a change, such as a bug fix did not result in another fault being uncovered in the application.
 - On going process throughout testing lifecycle
 - New bug-fix breaks previously tested units?
 - Perform regression test whenever program changes

Non-Functional Testing

- Non-Functional Testing is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.
- An excellent example of non-functional test would be to check how many people can simultaneously login into a software.
- Non-functional testing is equally important as functional testing and affects client satisfaction.











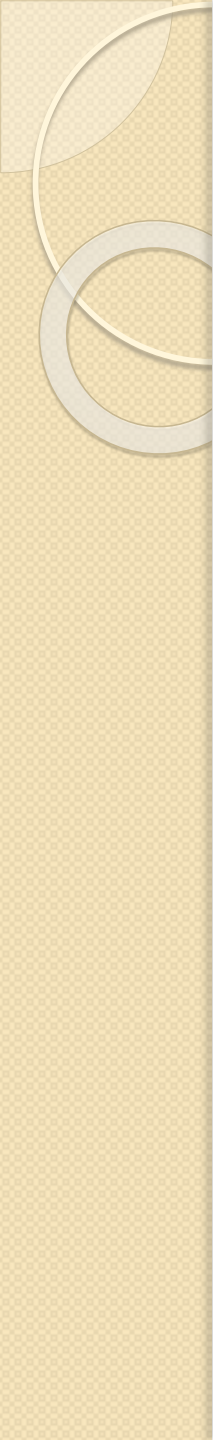
















جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

Software Engineering

lec. 17



Software Quality Assurance

- What is Quality?

Quality – developed product meets it's specification

Problems:

- Development organization has requirements exceeding customer's specifications (added cost of product development)
- Certain quality characteristics can not be specified in unambiguous terms (i.e. maintainability)
- Even if the product conforms to it's specifications, users may not consider it to be a quality product (because users may not be involved in the development of the requirements)



Software Quality Attributes

- Safety
- Security
- Reliability
- Resilience
- Robustness
- Understandability
- Testability
- Adaptability
- Modularity
- Complexity
- Portability
- Usability
- Reusability
- Efficiency
- Learnability

Software Quality Assurance

- To ensure quality in a software product, an organization must have a three-prong approach to quality management:
 - Organization-wide policies, procedures and standards must be established.
 - Project-specific policies, procedures and standards must be tailored from the organization-wide templates.
 - Quality must be controlled; that is, the organization must ensure that the appropriate procedures are followed for each project
- Standards exist to help an organization draft an appropriate software quality assurance plan.
 - ISO 9000-3
 - ANSI/IEEE standards
- External entities can be contracted to verify that an organization is standard-compliant.

What is SQA:

SQA includes all 4 elements...

- n **Software Quality Assurance** – establishment of network of organizational procedures and standards leading to high-quality software
2. **Software Quality Planning** – selection of appropriate procedures and standards from this framework and adaptation of these to specific software project
3. **Software Quality Control** – definition and enactment of processes that ensure that project quality procedures and standards are being followed by the software development team
4. **Software Quality Metrics** – collecting and analyzing quality data to predict and control quality of the software product being developed

SQA Activities

- Applying technical methods
 - To help the analyst achieve a high quality specification and a high quality design
- Conducting formal technical reviews
 - A stylized meeting conducted by technical staff with the sole purpose of uncovering quality problems
- Testing Software
 - A series of test case design methods that help ensure effective error detection
- Enforcing standards
- Controlling change
 - Applied during software development and maintenance
- Measurement
 - Track software quality and asses the ability of methodological and procedural changes to improve software quality
- Record keeping and reporting
 - Provide procedures for the collection and dissemination of SQA information



Advantages of SQA

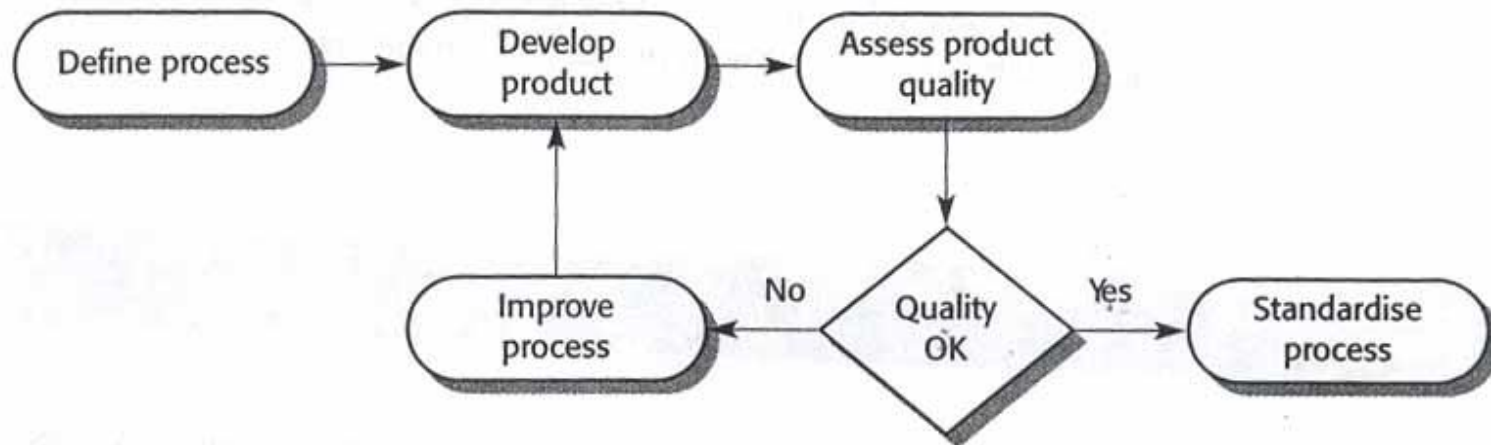
- Software will have fewer latent defects, resulting in reduced effort and time spent during testing and maintenance
- Higher reliability will result in greater customer satisfaction
- Maintenance costs can be reduced
- Overall life cycle cost of software is reduced



Disadvantages of SQA

- It is difficult to institute in small organizations, where available resources to perform necessary activities are not available
- It represents cultural change - and change is never easy
- It requires the expenditure of dollars that would not otherwise be explicitly budgeted to software engineering or QA

Process and Product Quality





جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

Software Engineering

chapter two

Software Development Models

Topics

2.1 The Software Lifecycle

2.2 Software Development

2.3 What is a Software Process ?

2.4 Software Engineering - A Layered Technology

2.5 Software Process Models

2.5.1 The Waterfall Model

2.5.2 The Prototype Model

2.5.3 Evolutionary Software Process Models

2.5.3.a The Incremental Model

2.5.3.b The Spiral Model

2.6 Component - Based Development

2.1 The Software Lifecycle

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

1. Requirements Engineering

2. Software Design

3. Software Construction

4. Validation and Verification

5. Software Testing

6. Software Deployment

7. Software Maintenance

Depending the software process used for the development of the software product, these stages may occur in different orders, or frequency.

2.1.1. 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 prospective system.

2.1.2. Software Design

- ⦿ The designers converts the logical software requirements from stage 1 into a technical software design by describe the software in such a way that programmers can write line of code that implement what the requirements specify.

2.1.3. Software Construction

- ⦿ Software construction is concerned with implementing the software design by means of programs in one or more programming languages and setting up a build management system for compiling and linking the programs.
- ⦿ **This stage content several steps, these are :**
- ⦿ **a. Software Reuse**
- ⦿ **b. Security and Reliability**
- ⦿ **c. Software Documentation**
- ⦿ **d. Coding Standards**

a. Software Reuse

The goal of software engineering is to achieve many features with little effort and few defects. Software reuse is believed to play an important role in achieving this goal by encapsulating effort in units of source code, which can be reused in other projects. ⦿

- ⦿ **b. Security and Reliability**

Software must be dependable by making it reliable (software should work very well under any environments), secure and safety (by verifying from user authentication to using any system).

- ⦿ **c. Software Documentation**

User documentation?

Technical documentation?

Documentation generation?

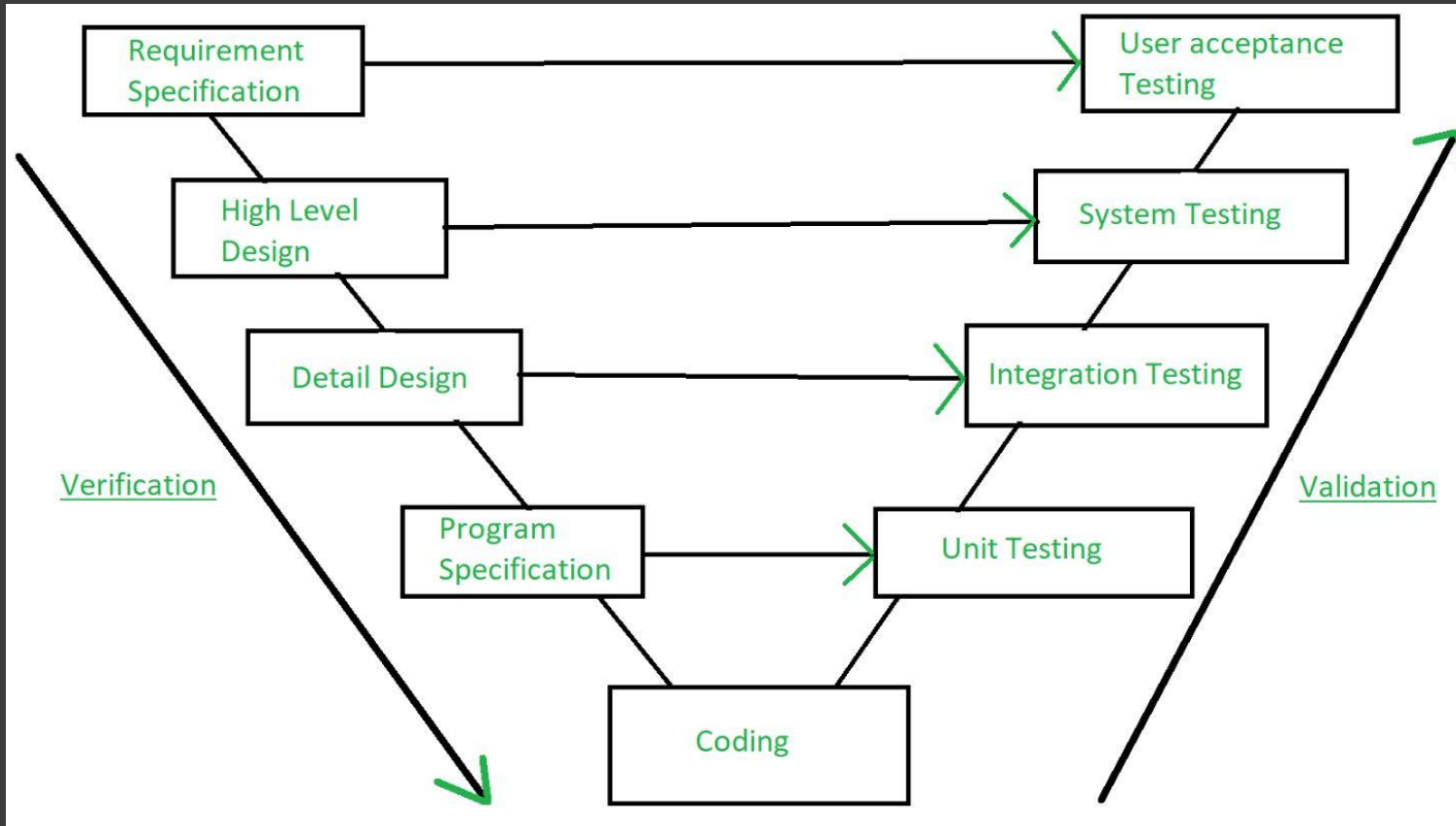
- ⦿ **d. Coding Standards**

Coding standards are important to ensure portability and make code maintainable by others than the original developer.

2.1.4. Validation and Verification

- ◎ **Validation** is the process of checking whether the software product is up to the mark or in other words product has high level requirements.
- ◎ **Verification** is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not.

V_Model



2.1.5. Software Testing :

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free

- The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

2.1.5 Software Deployment

- ⦿ After development, software should be put to use. That is, it should be released and made available to users, who can then download, install and activate it. These activities are captured under the common term software deployment.
- ⦿ The following deployment activities make up the software deployment process:
 - ⦿ Release , Packaging , Transfer , Installation
 - ⦿ Configuration , Activation
 - ⦿ De-activation , Update , Adapt , De-installation , De-release

© 2.1.6 Software Maintenance

As software evolves after its first release, software maintenance is needed to improve it, repair defects, and to extend it, and add new functionality.

2.2 Software development

- ◎ Three phases to develop the software

- 1- definition

- 2- design

- 3- maintenance

- 1- Definition**

- 1- What information to be processed?

- 2- What design constraints exist?

- 3- What function and performance desired?

- 4- What interfaces are desired?

- 5- What validation criteria are required?

- 6- What is modeling?

2- Design

- 1- How data structures to be designed.
- 2- How procedural details to be implemented.
- 3- How design to be translated into language.
- 4- How testing is performed.

3- Maintenance

- 1- error
- 2- Adaptation.
- 3- Modification

2.3 What is a Software Process ?

- ⦿ These set of activities whose goal is the development or evolution of software ,Generic activities in all software processes are:

1- Specification: the process of establishing what services are required and identifying the constraints (e.g. cost and time) on the operation of the system and its development.

2- Development: (design and implementation): the process of converting the system specification into an executable system.

3- Validation: Where the software is checked to ensure that it is what the customer required.

4- Evolution: Where the software is modified to adapt it to changing customer and market requirement.

2.4 Software Engineering - A Layered Technology

- Software engineering is a layered technology figure (2.1).

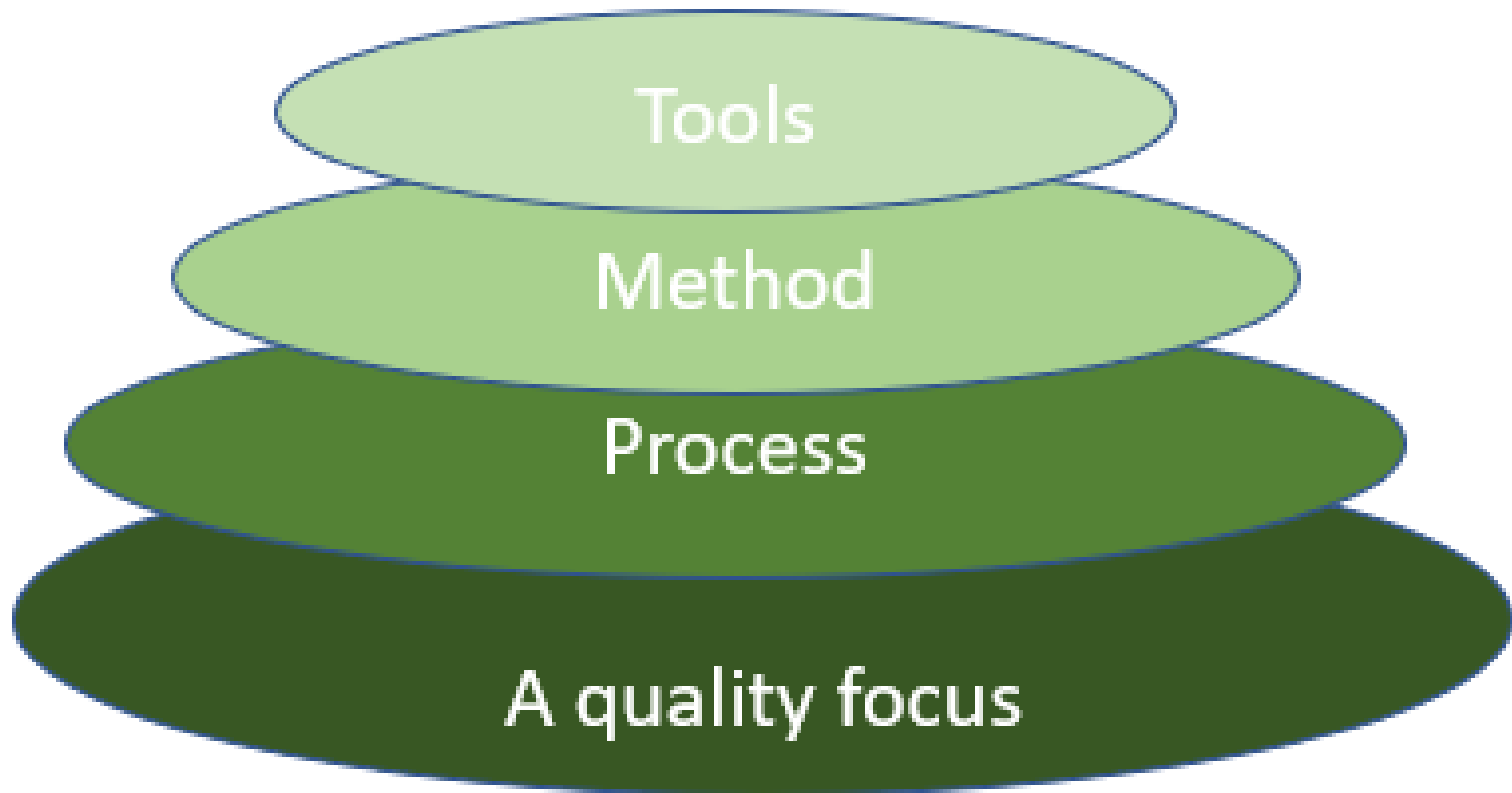
These layers are:

1- A quality focus: any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management and similar philosophies foster a continuous process improvement culture, and it is this culture that ultimately leads to the development of increasingly more mature approaches to software engineering. The bedrock that supports software engineering is a focus on quality.

2- Process: the foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework for a set of key process areas that must be established for effective delivery of software engineering technology.

3- Methods: software engineering methods provide the technical for building software methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing and maintenance. Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

4- Tools: software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer - aided software engineering (CASE), is established. CASE combines software, hardware, and software engineering database (repository containing important information about analysis, design program construction and testing) to create a software engineering environment.



GOOD LUCK



SOFTWARE ENGINEERING LEC5

Presented
by Lecturer : Wafaa Ali

2.5.2 The Prototype Model

The prototyping model is a systems development method in which a prototype is built, tested and then reworked as necessary until an acceptable outcome is achieved from which the complete system or product can be developed.

The Prototyping Model should be used when:

- ❖ the requirements of the product are not clearly understood or are unstable.
- ❖ if requirements are changing quickly

Approaches for this model

186

- 1. Rapid Throwaway Prototyping – This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of a better quality.

- 2. Evolutionary Prototyping –
In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.



Steps of Prototype Model:

188

- Requirement Gathering and Analyst
- Quick Decision
- Build a Prototype
- Assessment or User Evaluation
- Prototype Refinement
- Engineer Product

The stages of the prototyping model:

1) Step 1: Requirements gathering and analysis

A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

Step 2: Quick design

The second phase is a preliminary design or a quick design. In this 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.

. Step 3: Build a Prototype

In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

Step 4: Initial user evaluation

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

Step 5: Refining prototype

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

Step 6: Implement Product and Maintain

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

Advantages of using Prototype Model:

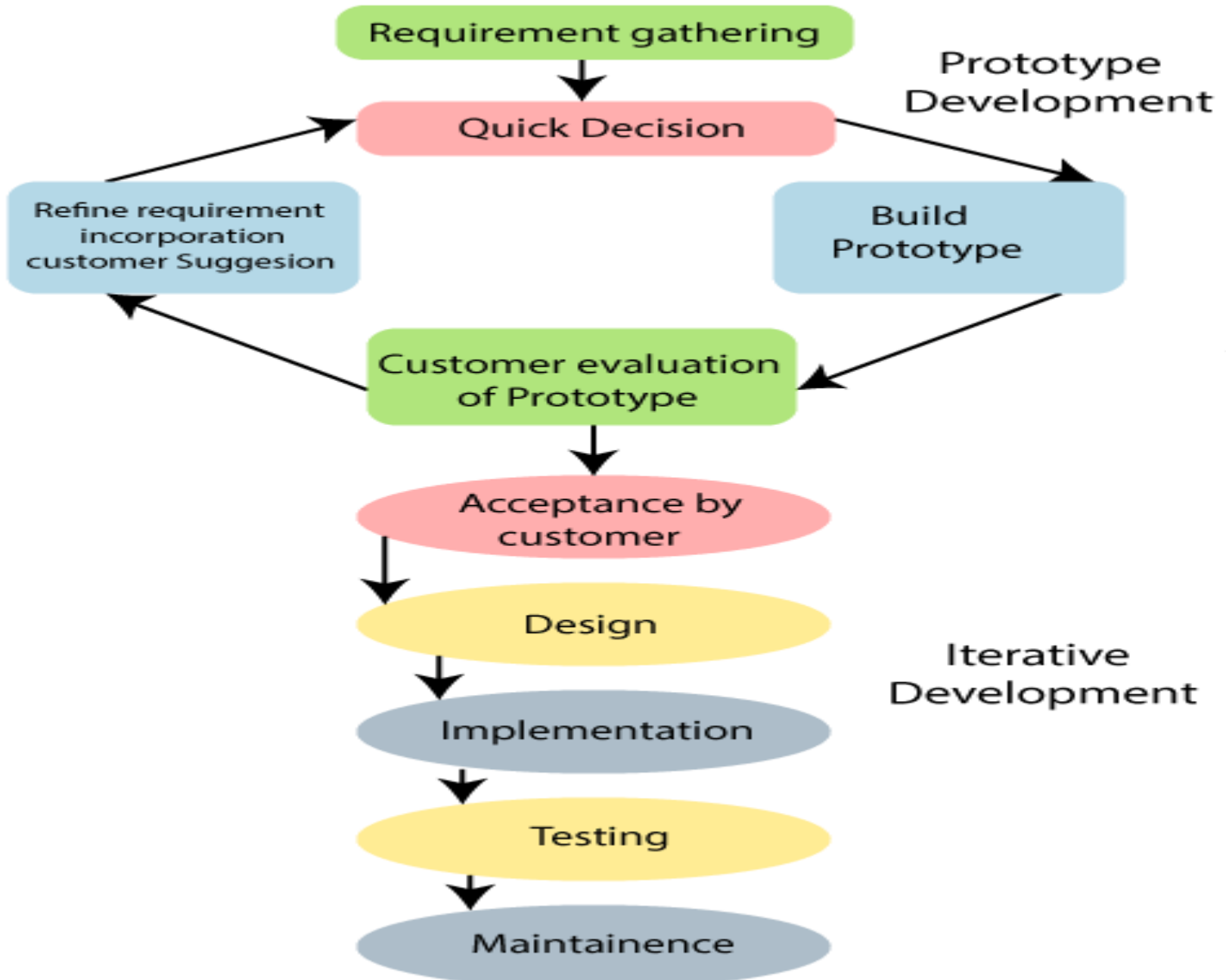
- This model is flexible in design.
- It is easy to detect errors.
- We can find missing functionality easily.
- There is scope of refinement, it means new requirements can be easily accommodated.
- It can be reused by the developer for more complicated projects in the future.
- It ensures a greater level of customer satisfaction and comfort.
- It is ideal for online system.
- It helps developers and users both understand the system better.
- Integration requirements are very well understood and deployment channels are decided at a very early stage.
- It can actively involve users in the development phase.

Disadvantages of using Prototype

Model :

- 1- An unstable/badly implemented prototype often becomes the final product.
- 2- Require extensive customer collaboration
- Costs customer money
- Needs committed customer
- Difficult to finish if customer withdraw
- May be too customer specific, no broad market
- 3- Difficult to know how long the project will last.
- 4- Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
- 5- Prototyping tools are expensive.
- 6- Special tools & techniques are required to build a prototype.
- 7- It is a time-consuming process.

Prototype Model



Conclusion:

- Prototyping helps for developing user interfaces,
- high technology software and systems with complex algorithms and interfaces.



جامعة ذي قار
كلية التربية للعلوم الصرفة
قسم علوم الحاسبات

SOFTWARE ENGINEERING

LEC 6,7

Presented

by Lecturer : Wafaa
Ali

2.5.3 EVOLUTIONARY SOFTWARE PROCESS MODELS

- Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software.

There are several types of that model, these are:

- a. The Incremental Model
- b. The Spiral Model

2.5.3.A. THE INCREMENTAL MODEL

- In incremental model, the software is divided into separate modules(components)/increments and each of these modules has a separate set of SDLC activities including requirements gathering, analysis, design, coding, Testing, deployment, and maintenance.
- When any component is ready, then the component is delivered to the customer and when remaining components become ready than delivered to the customer one by one by integrating new components with old once..
- For example, word-processing software developed using the incremental paradigm might:
 - 1) Deliver basic file management, editing, and document production functions in the first increment.
 - 2) More sophisticated 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.

EXAMPLE

- In this daily life example, we want to draw a picture, first, we draw the first part of the picture as shown in increment 1, Similarly after completing the first part of the picture we have to add one another part of the picture labeled as increment 2 in the above picture. Similarly, we complete this picture in four increments



www.t4tutorials.com

Increment 1

Increment 2

Increment 3

Increment 4

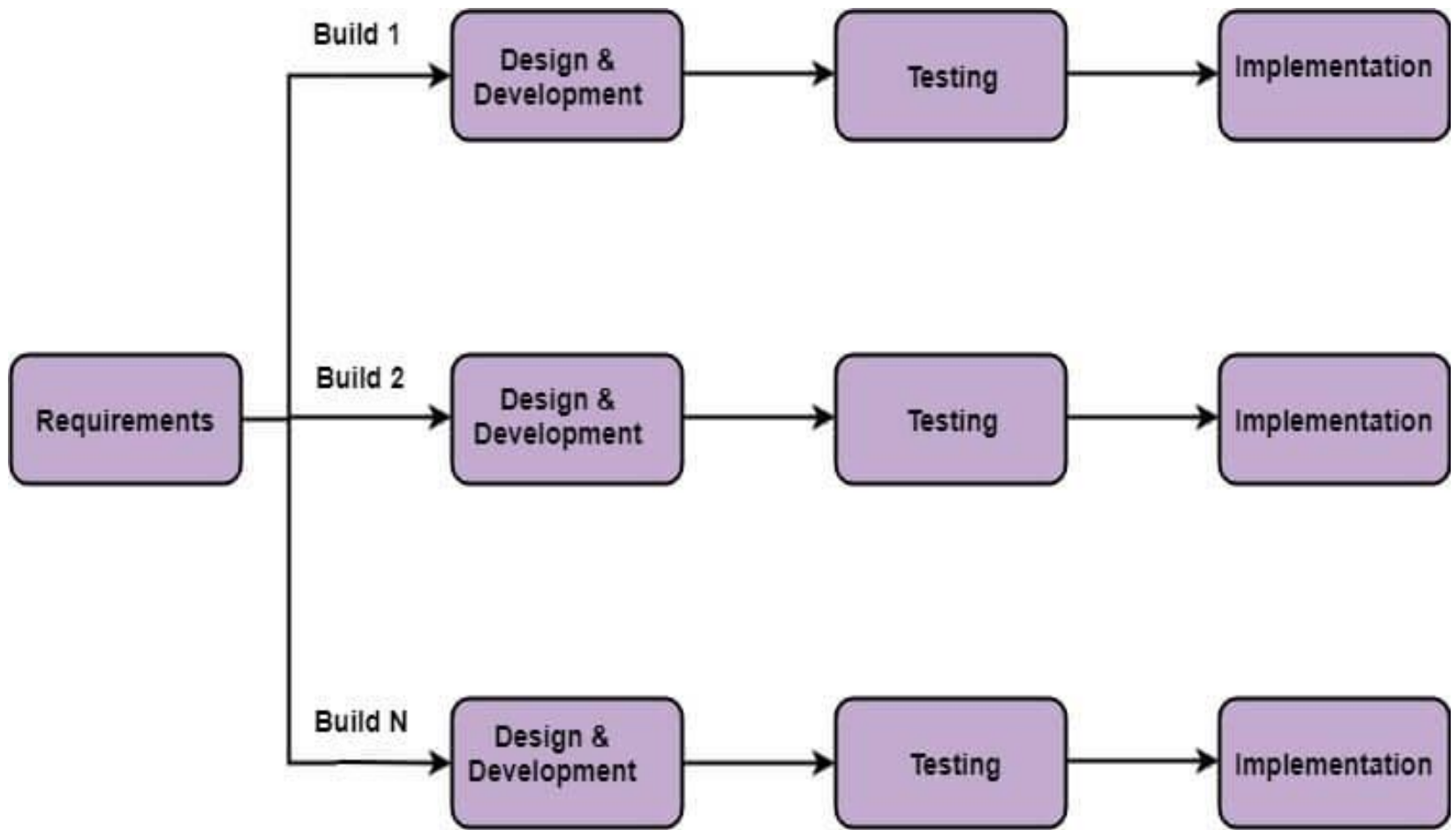


Fig: Incremental Model

THE PHASES OF INCREMENTAL MODEL ARE :

- 1. Requirement analysis:** In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.
- 2. Design & Development:** In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

CONT.

- **3. Testing:** In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.
- **4. Implementation:** Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

WHEN WE USE THE INCREMENTAL MODEL?

- A. A project has a lengthy development schedule
- B. When Software team are not very well skilled or trained.
- C. When the customer demands a quick release of the product.
- D. You can develop prioritized requirements first.
- E. Requirements of the system are clearly understood

THE ADVANTAGES OF AN INCREMENTAL MODEL.

- A. Customer feedback is received after the delivery of each component.
- B. Risk of requirement changes is reduced
- C. More flexible
- D. Easy to test and debug
- E. Give quick results

THE DISADVANTAGES OF AN INCREMENTAL MODEL?

- A. Needs a proper plan to integrate the components
- B. Needs a proper design to integrate the components
- C. More expansive as compared to the waterfall model.
- D. Problems might cause due to system architecture
- E. Total Cost is high.

2.5.3.B. THE SPIRAL MODEL

- The spiral model is a systems development lifecycle (SDLC) method used for risk management that combines the iterative development process model with elements of the Waterfall model. The spiral model is used by software engineers and is favored for large, expensive and complicated projects.
- the spiral model looks like a coil with many loops. The number of loops varies based on each project and is often designated by the project manager. Each loop of the spiral is a phase in the software development process.

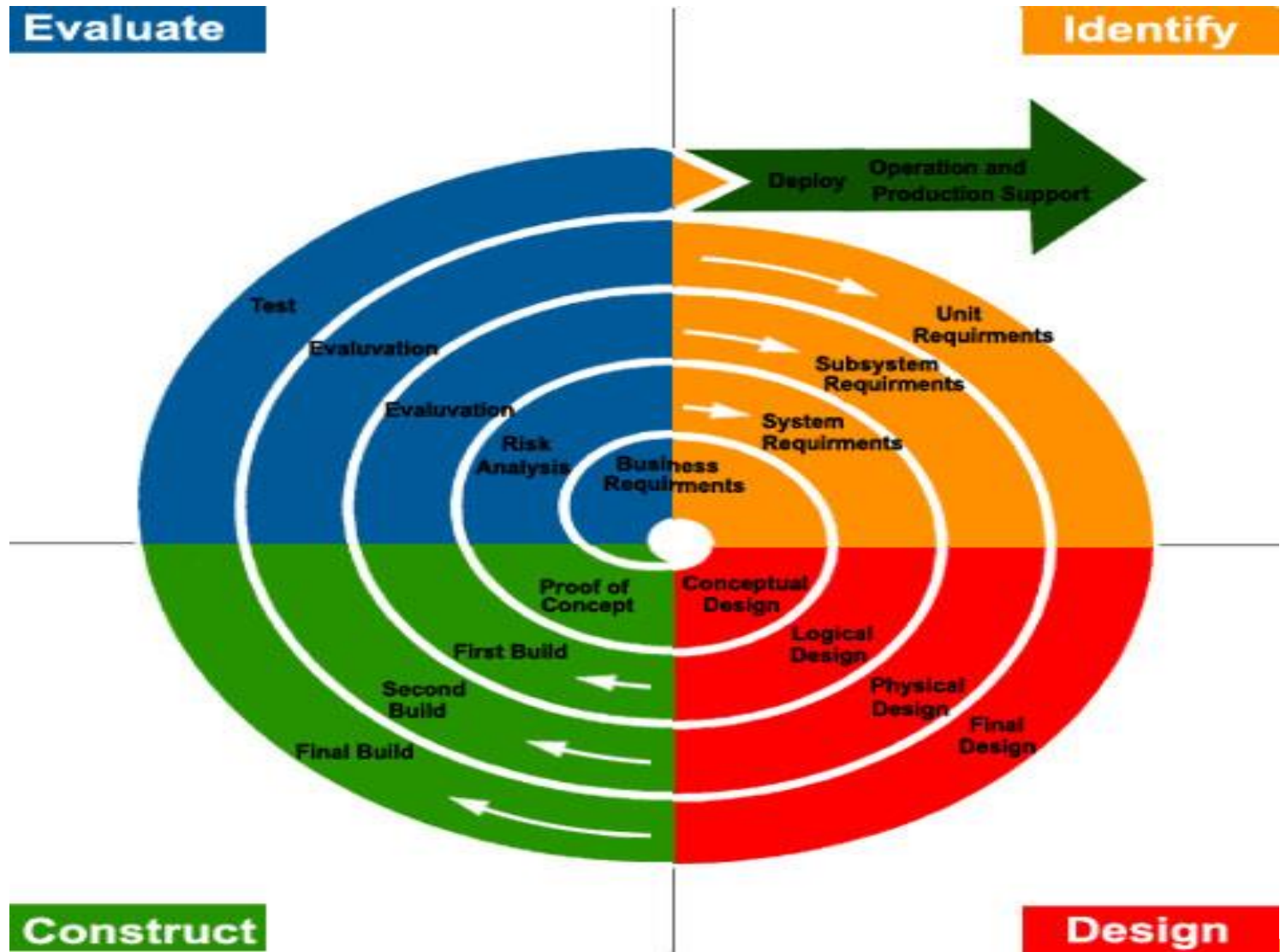


Figure (2.5) The Spiral model

- **A spiral model is divided into a number of framework activities, also called task regions. Figure 2.5 depicts a spiral model that contains six task regions:**
 1. **Customer communication-tasks** required to establish effective communication between developer and customer.
 2. **Planning-tasks** required to define resources, timelines, and other project-related information.
 3. **Risk analysis-tasks** required to assess both technical and management risks.

CONT.

- 4. **Engineering-tasks** required to build one or more representations of the application.
- 5. **Construction and release-tasks** required to construct, test, 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.

ADVANTAGE OF SPIRAL MODEL

- 1- Changing requirements can be accommodated.
- 2- Allows for extensive use of prototypes
- 3- Requirements can be captured more accurately.
- 4- Users see the system early.
- 5- Development can be divided into smaller parts and more risky parts can be
- 6- developed earlier which helps better risk management.

DISADVANTAGE OF SPIRAL MODEL

- 1-Management is more complex.
- 2-End of project may not be known early.
- 3-Not suitable for small or low risk projects and could be expensive for small projects.
- 4-Process is complex
- 5-requires excessive documentation.

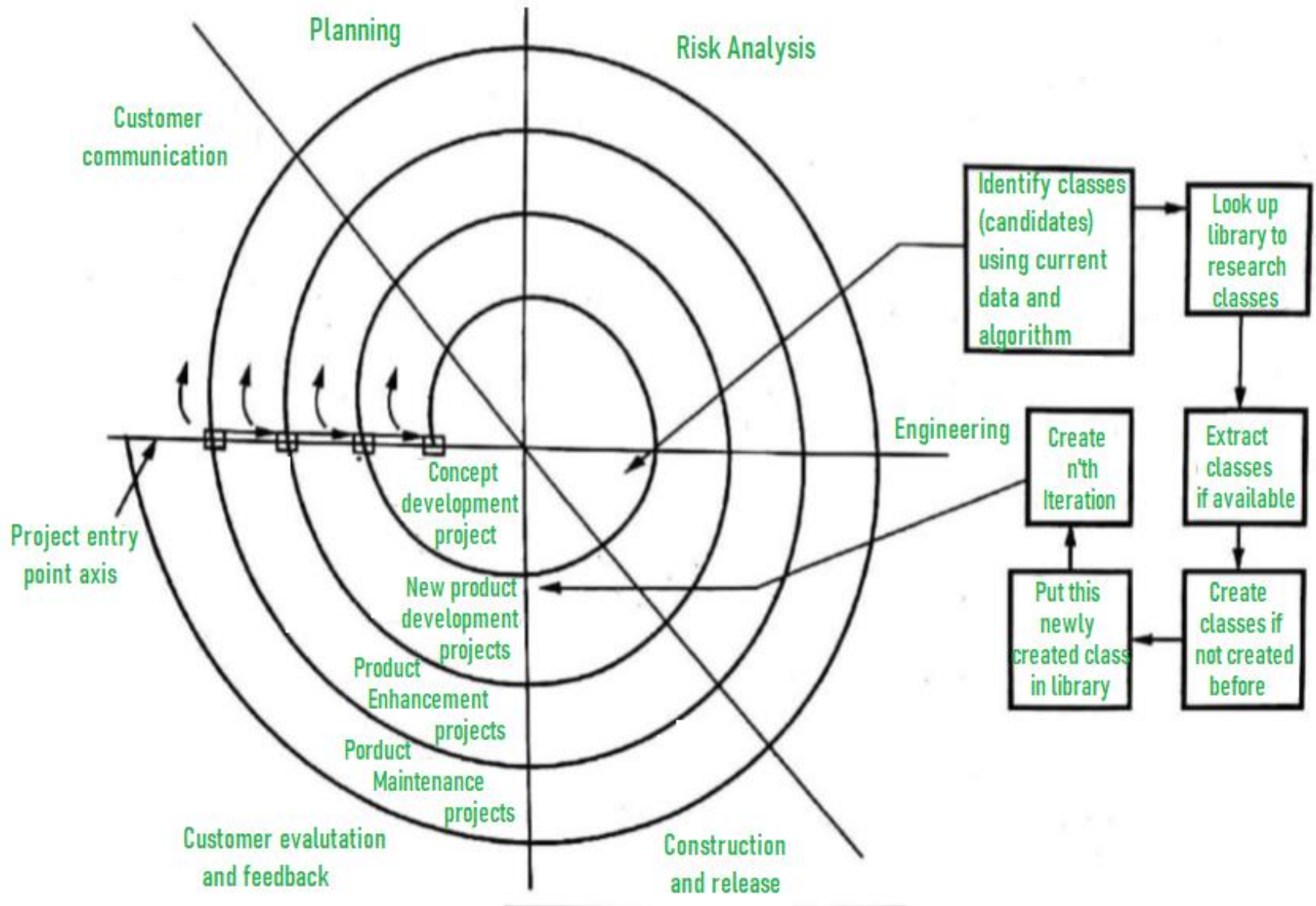


Figure (2.6) The Component – Based Development

GOOD LUCK

Software Engineering

chapter three

Software Requirements

3.1 Introduction

3.2 Requirements Analysis

3.3 Software Requirements Analysis Phases

3.4 Software requirements elicitation

3.4.1 Facilitated Action Specification Techniques (FAST)

3.4.2 Quality Function Deployment (QFD)

3.5 Analysis Principles

3.5.1 Information Domain

3.5.2 Modeling

3.5.3 Partitioning

3.5.4 Software Requirements Views

3.6 Software Prototyping

3.6.1 Prototyping Methods and Tools

3.7 Specification Principles

3.1 Introduction

- ▶ After system engineering is completed, software engineers need to look at the role of software in the proposed system. Software requirements analysis is necessary to avoid creating software product that fails to meet the customer's needs. Data, functional and behavioral requirements are elicited from the customer and refined to create specification that can be used to design the system. Software requirements work products must be reviewed for clarity, completeness and consistency.

3.2 Requirements Analysis

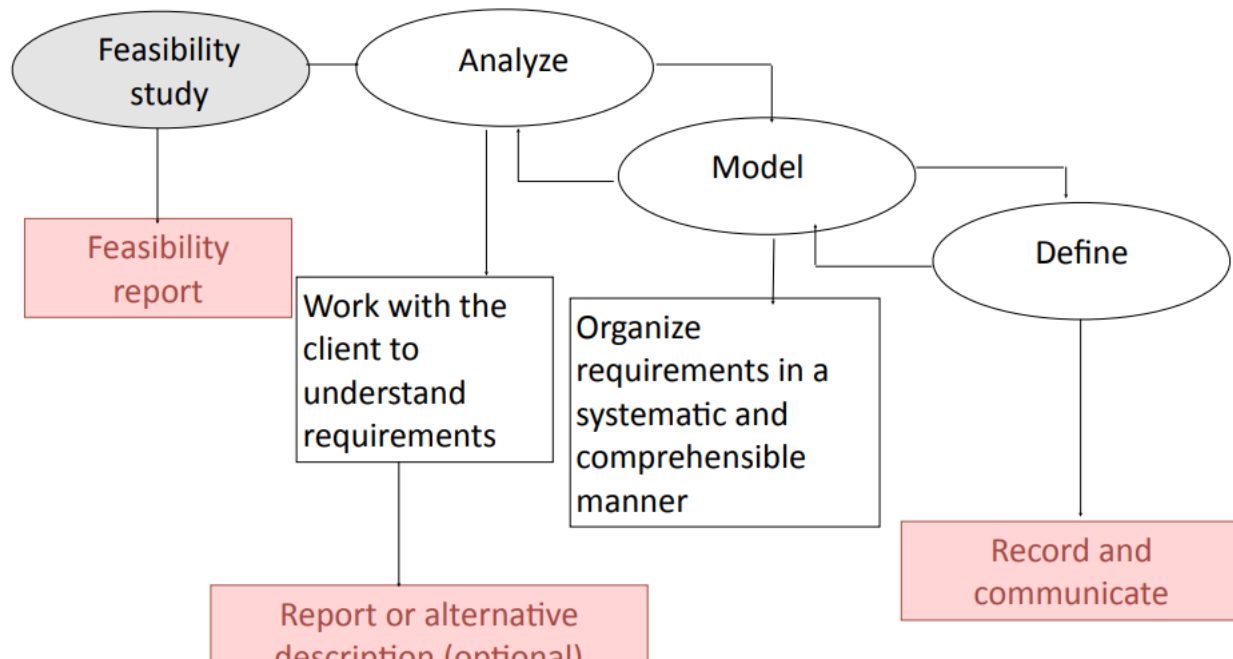
- ▶ Requirement Analysis, also known as Requirement Engineering, is the process of defining user expectations for a new
- ▶ software being built or modified.

Software Requirements Analysis

- Identify the "customer" and work Together to negotiate "product –level"
- Build an analysis model
 - Focus on data
 - define function
 - represent behavior
- Prototype areas of uncertainty
- Develop a specification that will guide design
- Conduct formal technical reviews.

3.3 Software Requirements Analysis Phases

- Problem recognition
- Evaluation and synthesis (focus is on what not how)
- Modeling
- Specification
- Review



3.4 Software requirements elicitation

- ▶ Customer meetings are the most commonly used technique.
- ▶ Use context free question to find out customers goal and benefits, identify stakeholders, gain understanding of problem, determine, customer reactions to proposed solution, and assess meeting effectiveness.
- ▶ If many users are involved, be certain that a representative cross section of users is interviewed.

3.4.1 Facilitated Action Specification Techniques (FAST)

1. Meeting held at neutral site, attended by both software engineers and customers.
2. Rules established for preparation and participation.
3. Agenda suggested to cover important points and to allow for brainstorming.
4. Meeting controlled by facilitator (customer, developer or outsider).
5. Definition mechanism (flip charts, stickers, electronic device, etc.) is used.
6. Goal is to identify problem, propose elements of solution, negotiate different approaches, and specify a preliminary set of solution requirements.

3.4.2 Quality Function Deployment (QFD)

1. Translates customer needs into technical software requirements.
2. Uses customer interviews, observation, surveys, and historical data for requirements gathering.
3. Customer voice table (contains summary of requirements).
4. Normal requirements (must be present in product for customer to be satisfied).
5. Expected requirement (things like ease of use or reliability of operation, that customer assumes will be present in a professionally developed product without having to request them explicitly).
6. Exciting requirements (features that go beyond the customers expectations and prove to very satisfying when they are present).

7. Function deployment (used during customer meeting to determine the value of each function required for system).
8. Information deployment (identifies data objects and events produced and consumed by the system).
9. Task deployment (examines the behavior of product within in environment).
10. Value analysis (used to determine the relative priority of requirements during function , information, and task deployment).

3.5 Analysis Principles

1. The information domain of the problem must be represented and understood.
2. The functions that the software is to perform must be defined.
3. Software behavior must be represented.
4. Models depicting information, function and behavior must be partitioned in a hierarchical manner detail.
5. The analysis process should move from the essential information toward implementation details.

The analysis process

Build a prototype

The
Problem

Requirement
Elicitation

Develop
Specification

Review

Create
Analysis
Model

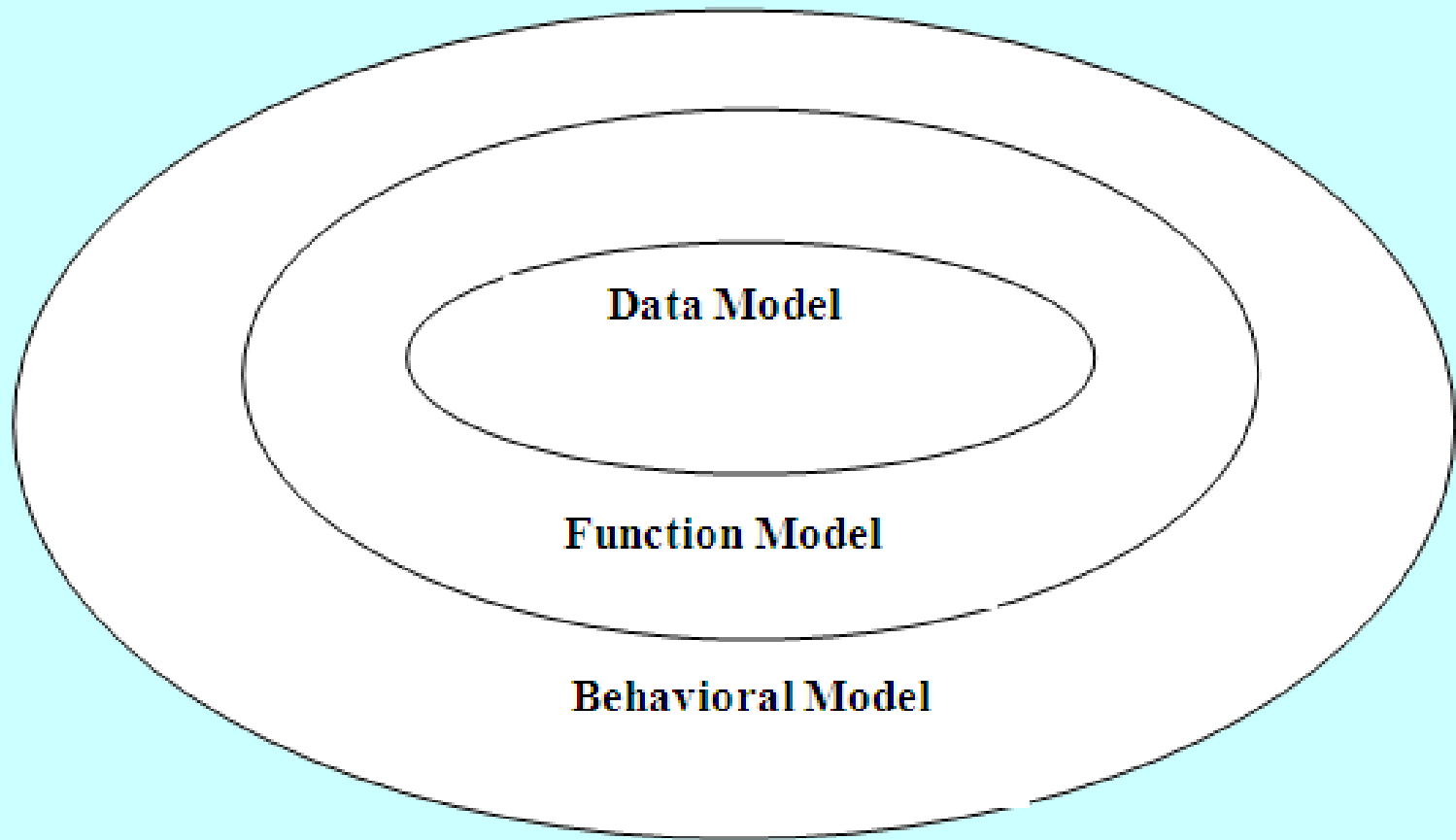
3.5.1 Information Domain

- ▶ Encompasses all data objects that contain numbers, text, images, audio or video.
- ▶ Information content or data model (shows the relationships among the data and control objects that make up the system).
- ▶ Information flow (represents the manner in which data control objects change as each moves through the system).
- ▶ Information structure (representations of the internal organizations of various data and control items).

3.5.2 Modeling

- ▶ **Data Model** (shows relationships among system objects).
- ▶ **Function Model** (description of the functions that enable the transformation of system objects).
- ▶ **Behavioral Model** (manner in which software responds to events from the outside world).

The Analysis Model



Analysis Principle | Model the Data Domain

- Define data object
- Describe data attributes
- Establish data relationships

Analysis Principle || Model Function

- Identify functions that transform data objects
- Indicate how data flow through the system
- Represent producers and consumers of data

Analysis Principle Model Behavior

- Identify different states of the system
- Specify events that cause the system to change state

3.5.4 Software Requirements Views

- ▶ Essential view- presents the functions to be accomplished and the information to be processed without regard to implementation.
- ▶ Implementation view- presents the real world manifestation of processing functions and information structures.
- ▶ Avoid the temptation to move directly to the implementation view, assuming that the essence of the problem is obvious.

3.6 Software Prototyping

- ▶ Throwaway prototyping (prototype only used as a demonstration of product requirements, finished software is engineered using another paradigm).
- ▶ Evolutionary prototyping (prototype is refined to build the finished system).
- ▶ Customer resources must be committed to evaluation and refinement of the prototype.
- ▶ Customer must be capable of making requirements decisions in a timely manner.

3.6.1 Prototyping Methods and Tools

- ▶ Fourth generation techniques (4 GT tools allow software engineer to generate executable code quickly).
- ▶ Reusable software components (assembling prototype from a set of existing software components).
- ▶ Formal specification and prototyping environments (can interactively create executable programs from software specification models).

3.7 Specification Principles

- ▶ Separate functionality from implementation.
- ▶ Develop a behavioral model that describes functional responses to all system stimuli.
- ▶ Define the environment in which the system operates and indicate how the collection of agents will interact with it.
- ▶ Create a cognitive model rather than an implementation model.
- ▶ Recognize that the specification must be extensible and tolerant of incompleteness.
- ▶ Establish the content and structure of a specification so that it can be changed easily.

THANKS