



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري محاضرة الاسبوع الأول والثاني

Introduction to Software Engineering

Topics covered



- ✧ Introduction to Software Engineering
- ✧ Software Crises
- ✧ Software Engineering Definitions
- ✧ Software Characteristics
- ✧ Well Engineered Software
- ✧ Software VS Program
- ✧ Software Applications

Introduction to Software Engineering



The term *Software Engineering* was first introduced in 1968 North Atlantic Treaty Organization (NATO) conference held in Germany. During the late 1960s the computing field as starting to face a software “crisis”.

Software Crisis



- ✧ Projects running over-budget.
- ✧ Projects running over-time.
- ✧ Projects were unmanageable and code difficult to maintain.
- ✧ Software was very insufficient.
- ✧ Software was of low quality.
- ✧ Software often did not meet requirements.
- ✧ Software was never delivered.

Software Engineering Definitions



- ✧ “***Software Engineering*** is concerned with software systems developed by teams rather than individual programmers uses engineering principles in the development of these systems and is made up of both non-technical aspects” (Sommerville).
- ✧ “***Software Engineering*** is a discipline that integrates methods, tools and procedures for the development of computer software” (Pressman).

Software Characteristics



✧ *Software is intangible*

Hard to understand development effort.

✧ *Software is easy to modify*

People make changes without fully understand it.

✧ *Untrained people can hack something together*

Quality problems are hard to notice.

Software Characteristics



- ✧ ***Software is easy to reproduce.***

Cost in its development , in other engineering products, manufacturing is the costly stage.

- ✧ ***The industry is labor-intensive***

Hard to automate.

- ✧ ***Most software is “custom build” rather than assembled from existing components.***

Software Characteristics



✧ *Software doesn't 'wear out'*

When a hardware component wears out, is replaced by a spare part. There are no software spare parts.

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

Well Engineered Software



1. *Maintainability*

The software can be easily understood and changed over time if problems occur.

2. *Reliability*

The software performs as expected. Continuity of correct service.

3. *Efficiency*

The software is produced in the expected time and within the limits of the available resources. The software-

Well Engineered Software



That is produced runs within the time expected for various computations to be completed.

4. *Usability*

The software can be used properly.

5. *Modifiability*

The software can be easily change.

6. *Portability*

The software system can be ported to other computers or systems without major rewriting of the software.

Well Engineered Software



7. *Testability*

The software can be easily tested.

8. *Reusability*

Some or all the software can be used again in other projects..

9. *Interoperability*

The software system can interact properly with other systems.

10. ***Correctness*** Software produces the correct output.

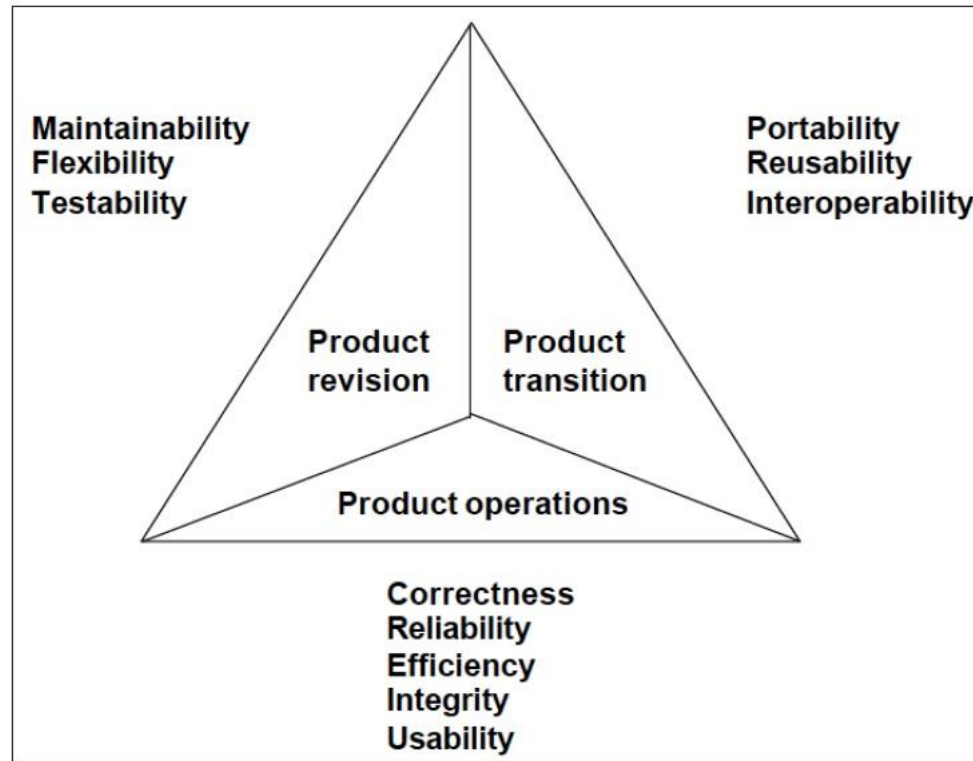


Figure 1: McCall's Software Quality Factors.

Software VS Program



Program	Software
Usually small in size	Usually large in size
Author himself is sole user	Large number of users
Single developer	Team of developers
Lacks proper user interface	Well- designed interface
Lacks proper documentation	Well- documented & user-manual prepared
Ad hoc developer	Systematic development

Software Applications



✧ Embedded Software

this type of software is placed in Read-Only-Memory (ROM) of the product and control the various functions of the product such as washing machine.

✧ Web Applications

The software related to web applications comes under this category such as HTML

Software Applications



✧ Artificial Intelligence Software

Artificial intelligence (AI) software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within area include robotics, expert systems, pattern recognition (image and voice).

✧ Business Software

This software designed to process business applications. It may also be a data warehousing tool which helps us to make decisions base on available data.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري محاضرة الاسبوع الثالث

Software Life Cycle Model / System Development Life Cycle (SDLC)

Topics covered



- ✧ What is Software Life Cycle Model?
- ✧ The need for Software Life Cycle Model
- ✧ Phases of Software Life Cycle Model

What is Software Life Cycle Model?



The software goes through a series of development phases. Typically, the software is specified, designed and then implemented.

If the customer is satisfied the software installed and while it is operational, it is maintained. The series of steps through which the software progress is called “**Software Life Cycle Model**”.

The need for Software Life Cycle Model



The development team must identify a suitable life cycle model for the particular project and then ahead to do it.

Without using a particular life cycle model the development of a software would not be in a systematic manor.

When a software is being developed by a team there must be a clear understanding a mong team members about when and what to do. Otherwise it would lead to project failure.

Phases of Software Life Cycle Model



1. *Requirements determination*

✧ *What is done During this Phase:*

Determine requirements to be met by software being contemplated.

✧ *The output of this Phase:*

Set of requirements and their priorities.

Phases of Software Life Cycle Model



2. Requirements specification

✧ What is done During this Phase:

Draw up understandable plan of **what the software will provide as outputs.**

Determine needs and propriety by consensus among end users.

✧ The output of this Phase:

Detailed specifications of information to be provided.
(revised user requirement).

Phases of Software Life Cycle Model



3. Feasibility analysis

✧ What is done During this Phase:

Taking into account available resources such as human, computer, time and money find whether specified requirements can be met.

✧ The output of this Phase:

Feasibility document specifying resource needs and availability. **Expected cost vs. benefits of software.**

Phases of Software Life Cycle Model



4. Software specification

✧ What is done During this Phase:

Obtain functional specification based on revised user requirements and feasibility study.

✧ The output of this Phase:

Functional specification. Budget, time schedule.
Physical requirements such as storage and processor.

Phases of Software Life Cycle Model



5. Hardware study

✧ What is done During this Phase:

Determine hardware requirements for software.

✧ The output of this Phase:

Hardware configuration- disk space, CPU power, computer network design.

Phases of Software Life Cycle Model



6. Software design

✧ What is done During this Phase:

Logical design of programs, design of databases, test and implementation plan.

✧ The output of this Phase:

Logical design of programs, data bases and test plan.

Phases of Software Life Cycle Model



7. Software Implementation

✧ What is done During this Phase:

Writing programs, creating databases, developing graphical user interfaces, documenting software. Training users.

Phases of Software Life Cycle Model



7. Software Implementation

✧ The output of this Phase:

Programs, databases, graphical user interfaces, output report formats, user manual and operational manual.

Phases of Software Life Cycle Model



8. *Software testing*

✧ *What is done During this Phase:*

programs are tested. Each program is called **a unit**. And unit testing is the verification that every unit meets its specification.

All units are combined and the whole software is tested. When the combined programs are successfully tested the software is finished.

✧ *The output of this Phase:*

Tested programs and overall software.

Phases of Software Life Cycle Model



9. Software evaluation

✧ What is done During this Phase:

Find out from users if system meets their needs. Such as E-commerce websites.

✧ The output of this Phase:

Evaluation report with suggestion for environment.

Phases of Software Life Cycle Model



10. Software modification / maintenance

✧ What is done During this Phase:

Change software, adding or deleting features to satisfy users (modified) needs.

✧ The output of this Phase:

Improved software containing modification and improvements.

Phases of Software Life Cycle Model

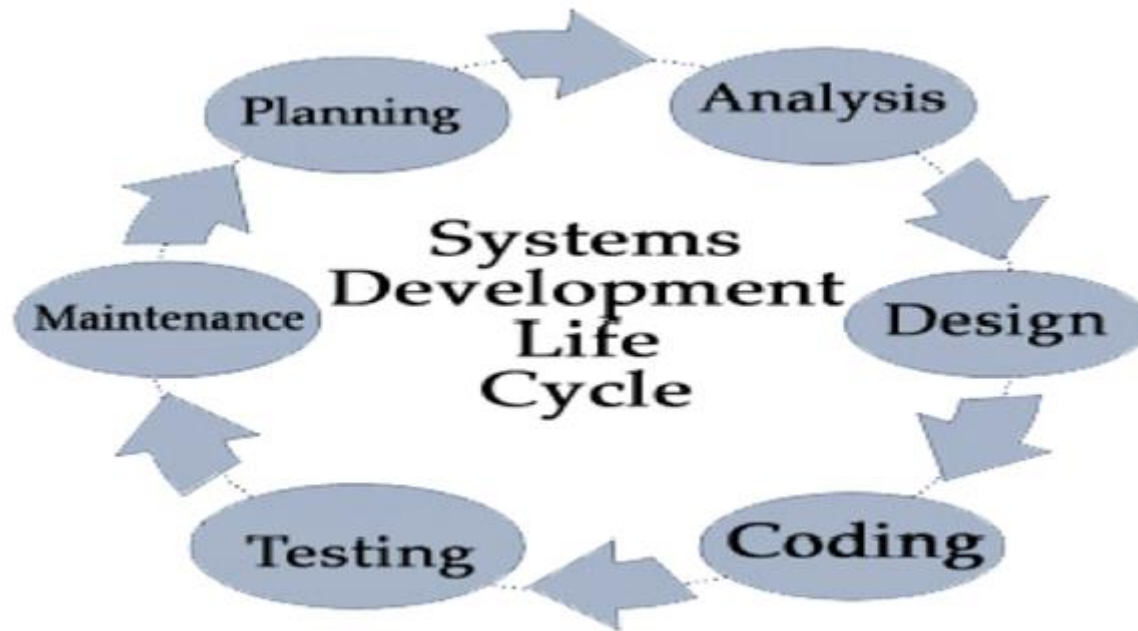


Figure 2: Software Life Cycle



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري

محاضرة الاسبوع الرابع

Waterfall Model

Topics covered



- ✧ What is Waterfall Model?
- ✧ Phases of Waterfall Model
- ✧ Strengths of Waterfall Model
- ✧ Weaknesses of Waterfall Model

What is Waterfall Model?



- ✧ The waterfall development process model is probably the oldest model. It is sometimes referred to as classical software life cycle model. Royce (1970) is one of the earliest people to write about this model.
- ✧ The waterfall model is considered the traditional approach to systems development. It describes a development approach that is linear and sequential objectives for each phase, and in which the output is the input for the next phase.

What is Waterfall Model?



- ✧ The waterfall model was so named because once goes over the falls, it can not turn back to or go back up. Similarly, in software development, once phases has been completed, you can not return to the prior phase.

Phases of Waterfall Model

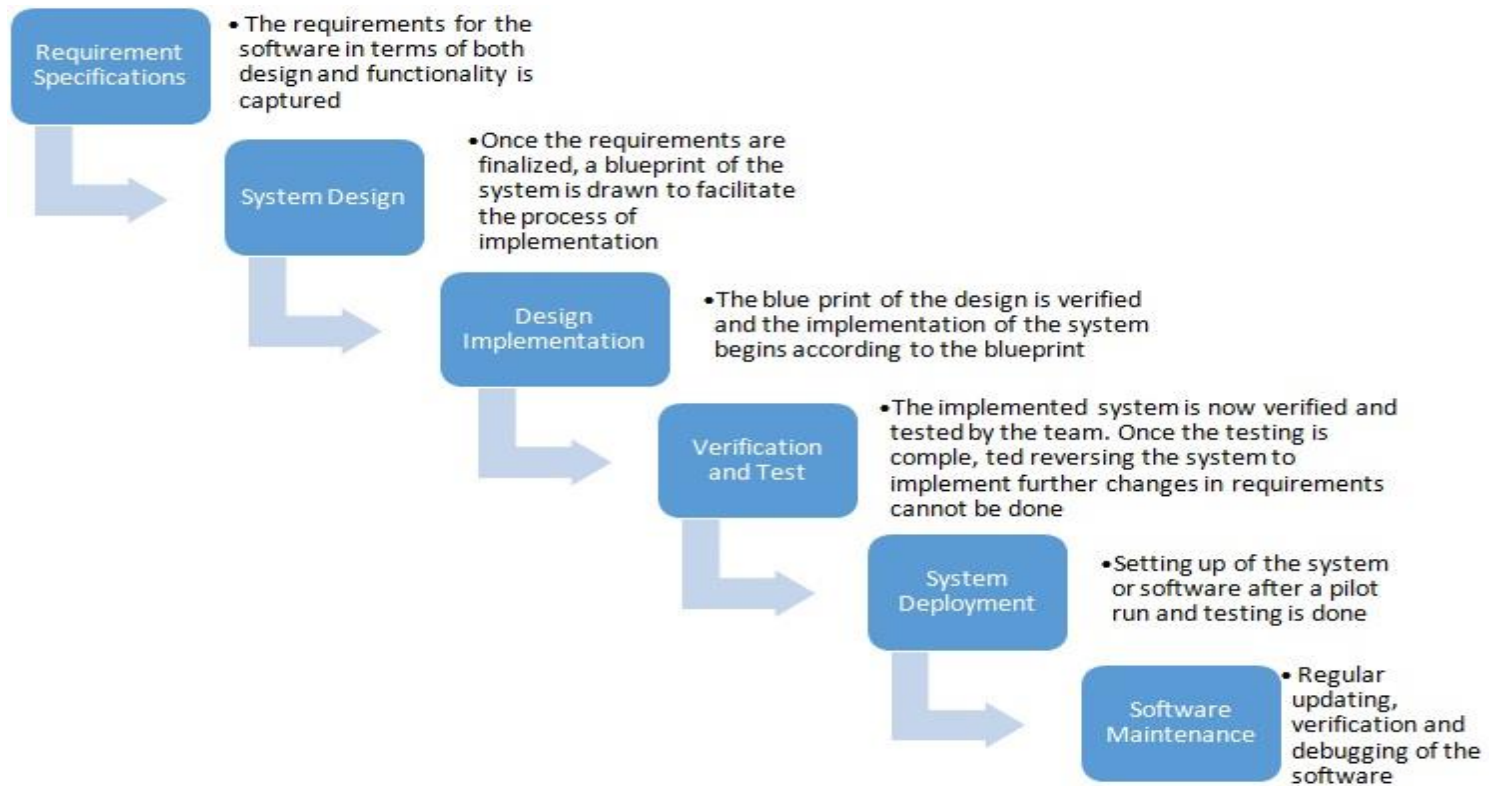


Figure 3: Waterfall Model

Strengths of Waterfall Model



- ✧ It is well understood.
- ✧ It is easy to manage.
- ✧ When teams are distributed geographically.
- ✧ Documentation produced can reduce maintenance costs.
- ✧ Verification is done after each phase.

Weaknesses of Waterfall Model



- ✧ It does not accommodate change to requirements very well.
- ✧ All requirements must be known and defined in the beginning.
- ✧ It does not allow the team to go back and repeat a phase.
- ✧ It can not be adapted to different project types very easily.

Weaknesses of Waterfall Model



- ✧ It is not works with large, complex applications.
- ✧ Documentation and verification after each phase slow down the process.
- ✧ The working version of the system is not seen until late in project' life.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري محاضرة الاسبوع الخامس

Prototyping Model

Topics covered



- ✧ What is Prototyping Model?
- ✧ Iterative approach to software development
- ✧ Phases of prototyping Model
- ✧ Strengths of prototyping Model
- ✧ Weaknesses of prototyping Model

What is Prototyping Model?



- ✧ Prototyping is development of a preliminary version of a software in order to allow certain aspects of what that software to be investigated.
- ✧ Often the primary purpose of a prototype is to obtain feedback from the intended users; the requirements specification for the software can then be updated to reflect this feedback, and so increase confidence in the final software.

What is Prototyping Model?



- ✧ Prototyping is a type of evolutionary development, the method of building a software where developers get the general idea of what is needed by the users, and then build a fast, high-level version of the software as the beginning of the project.
- ✧ The idea of prototyping is to quickly get a version of the software in the hands of the users and to jointly evolve the system through a series of iterative cycle of design.

Iterative approach to software development



Figure 4: Iterative approach to software development

Phases of prototyping Model (in each version)

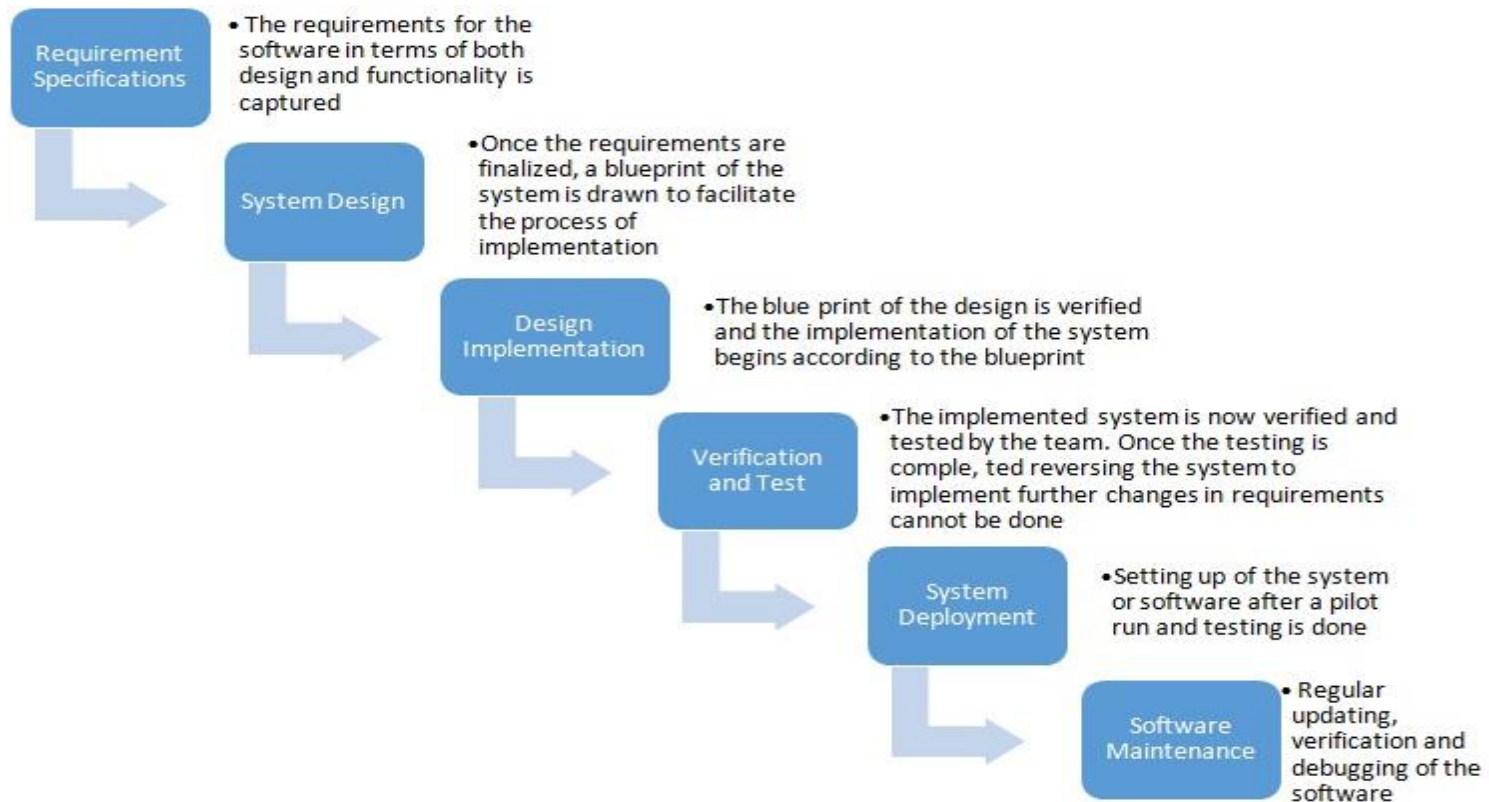


Figure 5: Phases of prototyping model (in each version)

Strengths of Prototyping Model



- ✧ Improved user communications.
- ✧ Users like it.
- ✧ Speeds up development process.
- ✧ Good for eliciting software requirements.
- ✧ Provides a tangible model to serve as basic for production version.
- ✧ This is useful when requirements are changing rapidly.

Weaknesses of Prototyping Model



- ✧ It is impossible to know at the outset of the project how long it will take.
- ✧ There is no way to know the number of iterations that will be required.
- ✧ It is difficult to build an accurate cost estimate.
- ✧ Documentation may be more difficult.
- ✧ This approach is not suitable for all software.

Weaknesses of Prototyping Model



- ✧ The user sees what appears to be a working version of software. Actually, it is only mock-ups of software.
- ✧ The developer is often makes implementation compromises in order to a get prototype work quickly.

The prototyping Model

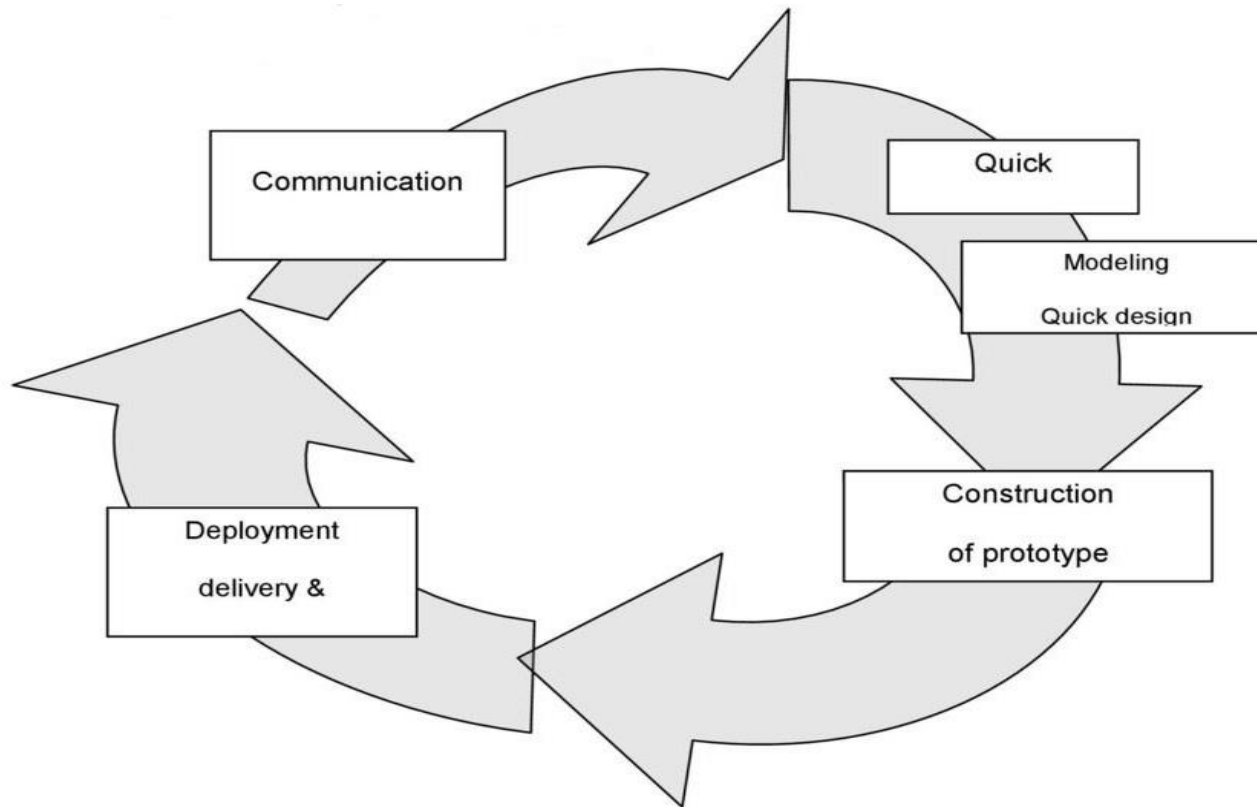


Figure 6: The prototyping model



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري محاضرة الاسبوع السادس

Incremental Model

Topics covered



- ✧ What is Incremental Model?
- ✧ Incremental process Model
- ✧ Phases of incremental Model (for each increment, component or in iteration)
- ✧ A multiple components incremental model
- ✧ Strengths of Incremental Model
- ✧ Weaknesses of Incremental Model

What is Incremental Model?



- ✧ The incremental model may be viewed as modification of waterfall model. In early days, each component followed a waterfall process model, passing through each step iteratively. In the incremental model the components were developed in an overlapping fashion.
- ✧ As software projects increased in size, it was recognized that it is much easier, and sometimes necessary, to develop the software if the large projects are subdivided into smaller components, which may thus be developed incrementally and iteratively.

What is Incremental Model?



- ✧ The incremental approach, attention is first focused on the essential features. Additional functionality of the software is produced and delivered to the customer in small increments.

- ✧ The components all had to be integrated and then tested as a whole final software.

Incremental process model

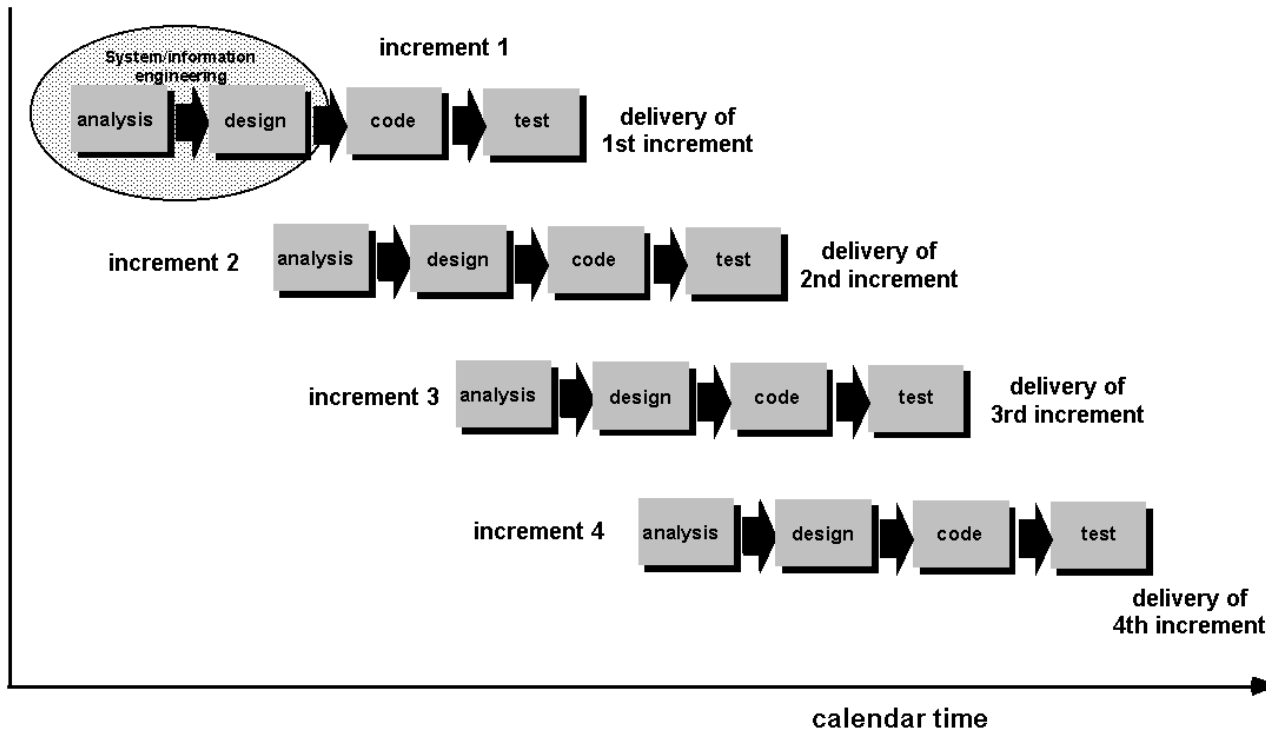


Figure 7: Incremental process model

Phases of incremental Model (for each increment, component or iteration)

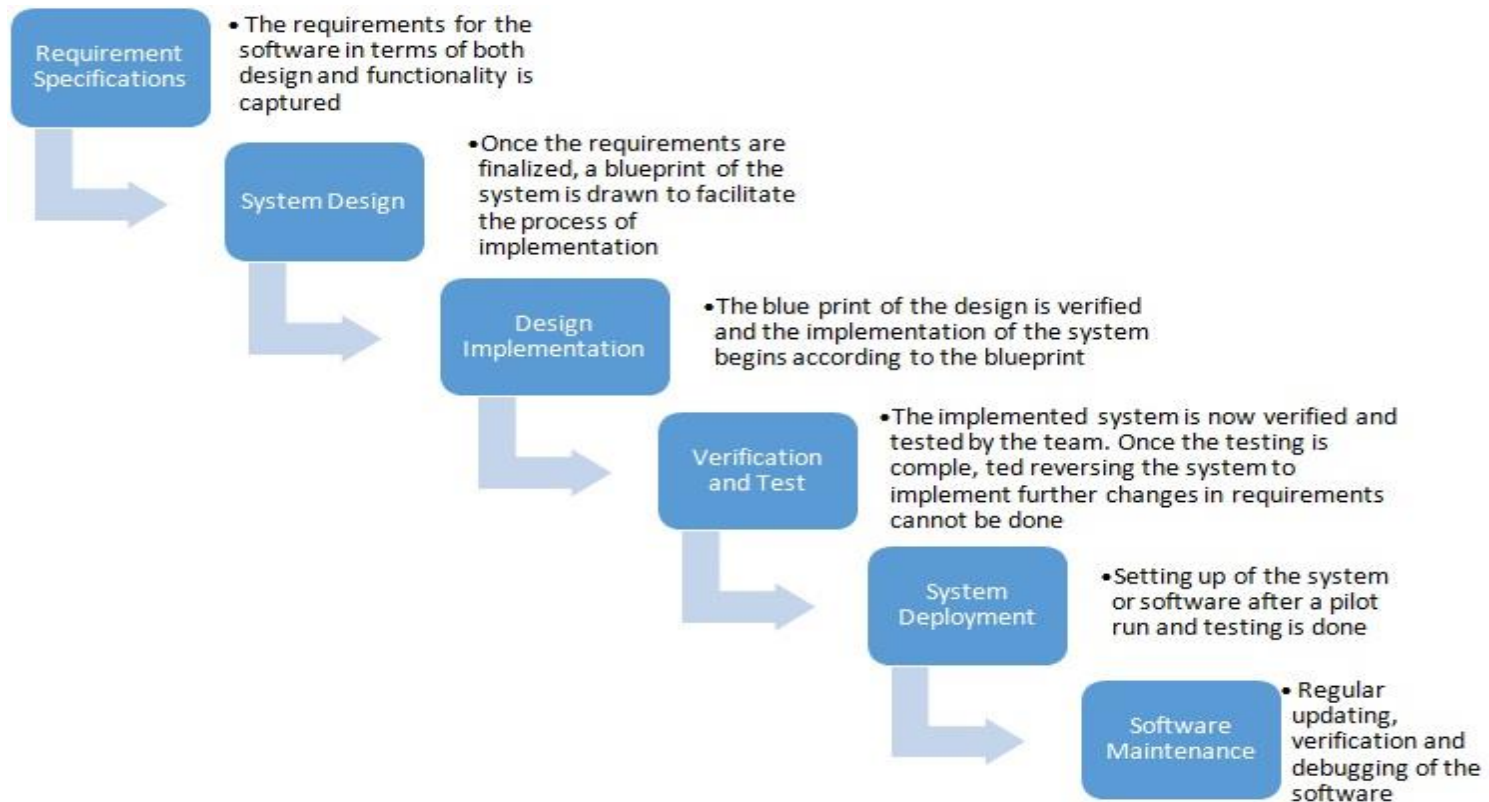


Figure 8: Phases of incremental model (for each increment or component in iteration)

A multiple components incremental model

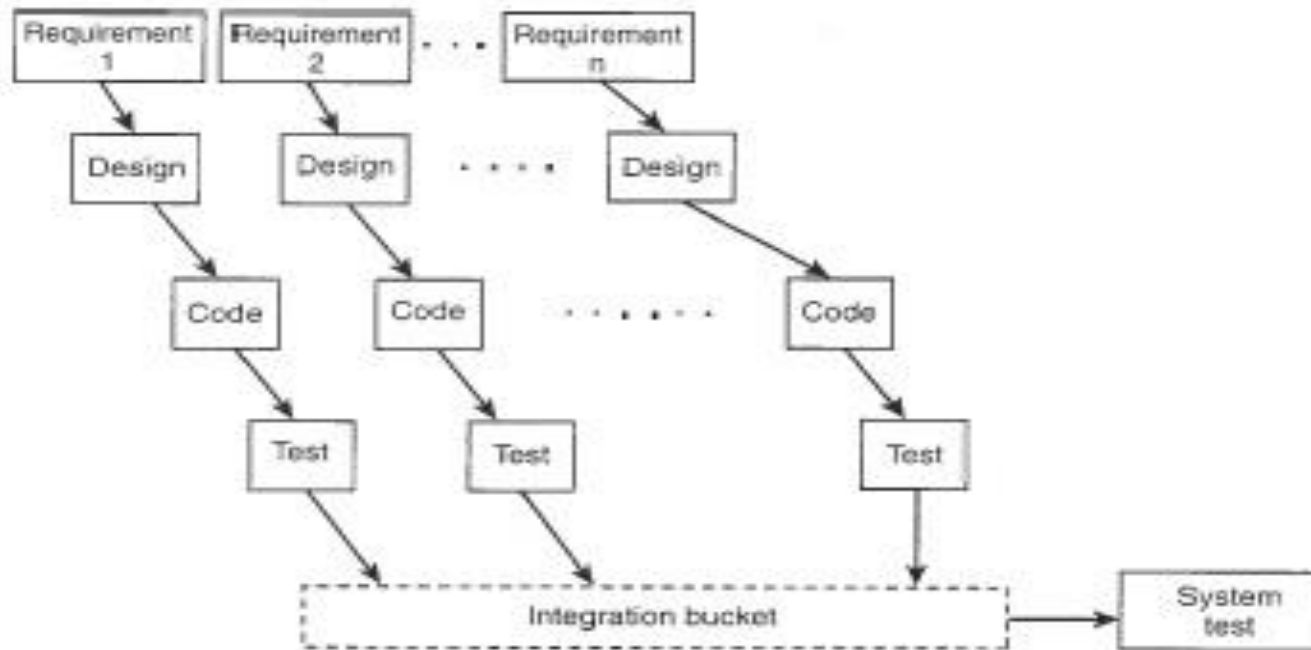


Figure 7: A multiple components Incremental model

Strengths of Incremental Model



- ✧ Working software is generated quickly and early during the software life cycle.
- ✧ It is more flexible than the other methods, and changing scope and requires is less costly with this model than with others..
- ✧ It is easier to test during a smaller iteration.
- ✧ It is easier to manage (each iteration is easily to manage).

Weaknesses of Incremental Model



- ✧ It is not easy to manage (whole software).
- ✧ The team must be able to estimate well to plan iterations.
- ✧ It is difficult to determine cost and time estimates early in the process.
- ✧ It requires experienced team members.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري محاضرة الاسبوع السابع

Rapid Applications Development Model (RAD)

Topics covered



- ✧ What is RAD Model?
- ✧ RAD process Model
- ✧ RAD Model Approach
- ✧ Phases of RAD Model
- ✧ Strengths of RAD Model
- ✧ Weaknesses of RAD Model

What is RAD Model?



- ✧ The model was proposed by IBM in the 1980s through the book of James Martin entitled “ Rapid Application Development”.
- ✧ In this model the user involvement is more important from the beginning to the end to ensure the requirement analysis and design of software.
- ✧ This model is similar to prototype, it is built and given to the user for evaluation and for the feedback.
- ✧ The prototype is refined as per the feedback obtained from the customer.

What is RAD Model?



✧ RAD model is similar to incremental model. It is proposed when requirements and solutions can be modularized as independent system or software components, each of which can be developed by different teams. After these smaller solutions, the modularization could be on a functional, technology or architectural basis, front-end-back-end, client side-server and so on.

RAD Model Approach

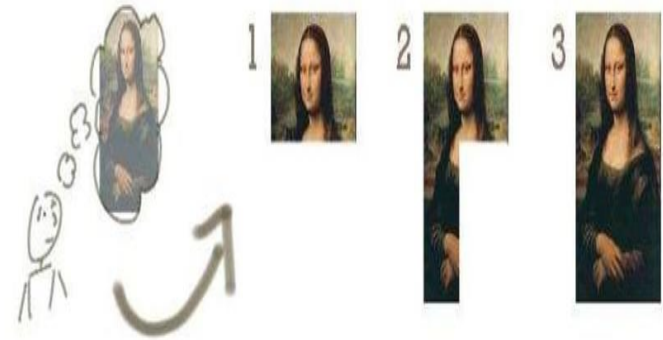
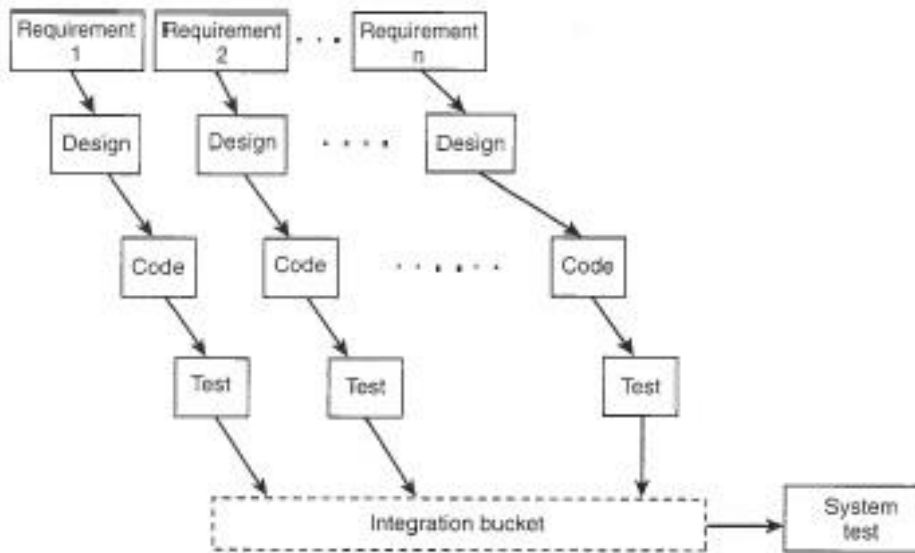


Figure 9: RAD model approach

RAD Model Approach



- ✧ RAD is also recommended when system components have already been developed by the organization in the context of other software systems, and these components can be used in the new software with minor or no modification.
- ✧ The model is called RAD due to the reusability of the system components and the possibility of splitting the requirement into smaller components that can then be assigned to different development team.

RAD Model Approach?



- ✧ The RAD recommended that developers use special techniques and computer tools to speed up the analysis, design, and implementation phases, such as GUI (Graphical User Interface), CASE (computer-aided software engineering) tools, JAD (joint application design), fourth-generation/visual programming languages that simplify and speed programming (e.g., Visual Basic.NET), and code generators that automatically produce programs from design specifications.

RAD process model



Rapid Application Development (RAD)

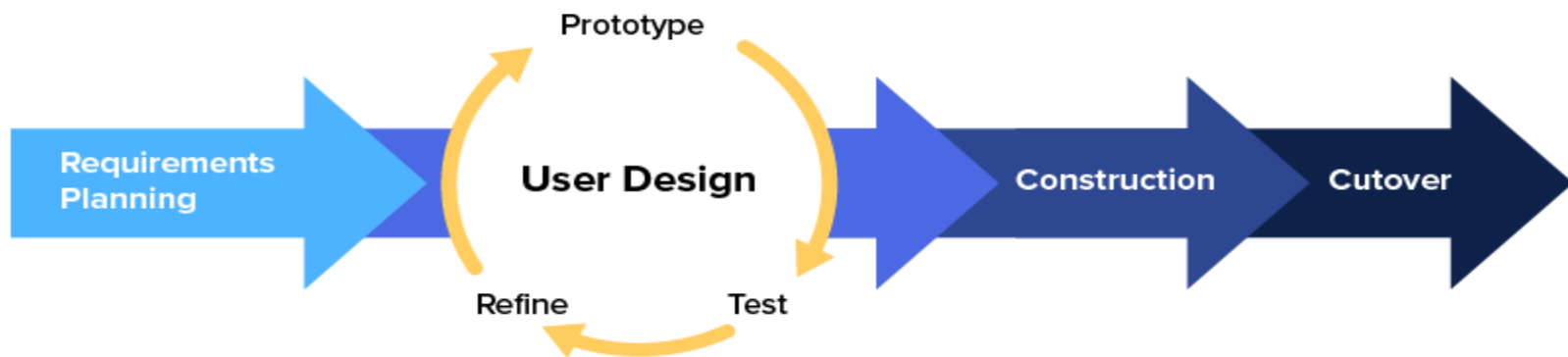


Figure 10: RAD process model

Phases of RAD Model



✧ *Requirements Planning Phase:*

The requirement is captured using any group elicitation technique. Only issue is the active involvement of users for understanding the project.

✧ *User Design Phase:*

Joint teams of developers and user are constituted to prepare understand and review the requirements. The team may use automated tools to capture information from users.

Phases of RAD Model



✧ ***Construction Phase:***

This phase combined the detailed design encoding, testing phase of waterfall model here we release the product to customer. It is expected to use code generators, screen generators and other type's productivity tools.

✧ ***Cutover Phase:***

This phase incorporates acceptance by users, installation of the systems and user training.

Strengths of RAD Model



- ✧ Speed of development.
- ✧ Use of GUI (graphical user interface) and other development tools.
- ✧ Heavy user participation.

Weaknesses of RAD Model



- ✧ The process may be speedy that requirements frozen too early.
- ✧ Basic principles of software development (e.g. programming standards, documentation, data-naming standards, backup and recovery) are overlooked in the race to finish the project.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري محاضرة الاسبوع الثامن

Spiral Model

Topics covered



- ✧ What is Spiral Model?
- ✧ Spiral process Model
- ✧ What is Risk in Software Engineering?
- ✧ Phases of Spiral Model
- ✧ Strengths & Weakness of Spiral Model
- ✧ When to Use Spiral Model?

What is Spiral Model?



- ✧ Another evolutionary approach to software development is the Spiral model, proposed by Barry Boehm in his paper in 1986, "A Spiral Model of Software Development and Enhancement".
- ✧ The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral model is not fixed. Each loop of the spiral model represents a phase of the software process.

What is Spiral Model?



✧ Example:

The innermost loop might be concerned with requirements specification. The next loop with design, the next loop with implementation, and so on.

Each phase in this model is split into four sectors (or quadrants). The following activities are carried out. During each phase of the spiral model.

Spiral process model

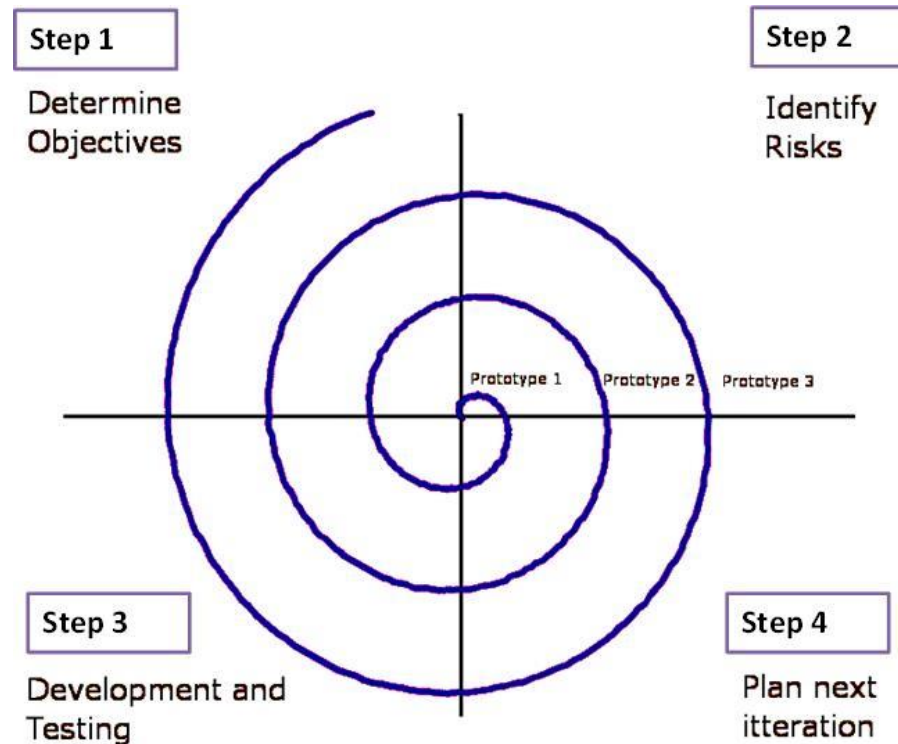


Figure 11: Spiral model

What is Risk in Software Engineering?



- ✧ Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time. A possibility of suffering from loss in software development process is called a software risk.
- ✧ Loss can be anything, increase in production cost, development of poor quality software, not being able to complete the project on time.
- ✧ Software risk exists because the future is uncertain and there are many known and unknown things that cannot be incorporated in the project plan.

Phases of Spiral Model



First Quadrant (Determine objectives):

- During the first quadrant, it is needed to identify the objectives of the phase.
- Examine the risk associated with these objectives.

Second Quadrant (Identify risks):

- A detailed analysis is carried out for each identification project risk.
- Steps are taken to reduce the risks. For example, if there is a risk that the requirements are not suitable, a prototype system may be developed.

Phases of Spiral Model



Third Quadrant (Development and testing):

- Develop and validate the next level of the product after resolving the identified risks.

Fourth Quadrant (Plan next iteration):

- Review the results achieved so far with customer and plan the next iteration around the spiral.

Strengths & Weakness of Spiral Model



Strength:

- Addresses risks associated with software development.
- Suitable to develop medium and large projects.

Weakness:

- Unsuitable for small projects.
- Requires considerable risk assessment expertise. If a major risk is not discovered, problems will undoubtedly.

When to Use Spiral Model?



- ✧ For medium to high-risk projects (medium and large projects).
- ✧ When significant changes are expected.
- ✧ When users are not exactly sure what their needs.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري

محاضرة الاسبوع العاشر

Software Requirements Specifications (SRS)

Topics covered



- ✧ What is Software Requirements Specifications?
- ✧ Types of Software Requirements
- ✧ Functional Requirements
- ✧ Non-Functional Requirements
- ✧ User Requirements
- ✧ System Requirements
- ✧ Some Videos for more explanation

What is Software Requirements Specifications?



- ✧ **Requirements** form a set of statements that describe the user's needs and desires. In developing a software system, these requirements must be clearly and fully understood by software engineers who develop the software system.

Types of Software Requirements



- ✧ Functional Requirements
- ✧ Non-Functional Requirements
- ✧ User Requirements
- ✧ System Requirements

Functional Requirements



- ✧ Describe functionality or system services.
- ✧ Functional user requirements may be high-level statements of **what the system should do** and functional system requirements should describe the system services in details e.g., the user shall be able to search either all of the initial set of database or select a subset from them.

Non-Functional Requirements



✧ Define system properties and constraints.

✧ **Non-Functional Classification:**

Product Requirements: Requirements, which specify that the delivered product must behave in particular way e.g., reliability, efficiency, integrity, usability.

Organizational Requirements: Requirements, which are consequence of organizational policies sand procedures e.g., process standard used (IEEE STANDARD-95 FORMAT).

Non-Functional Requirements



External Requirements: Requirements, which arise from factors which are external to the system and its development process e.g., the system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

McCall's model of software quality

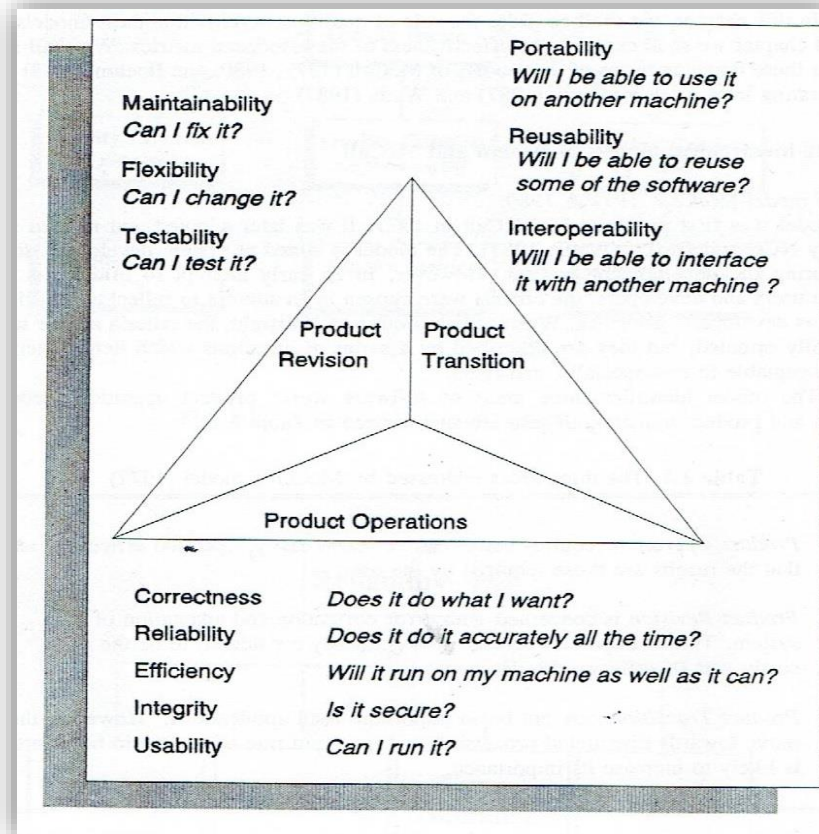


Figure 12: McCall's model of software quality

The criteria included in McCall's model

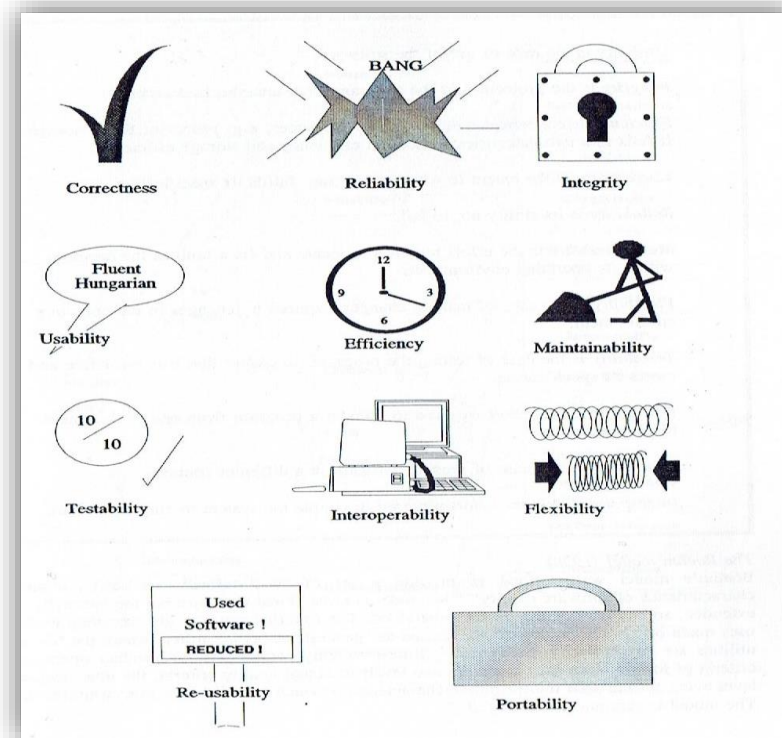


Figure 13: The criteria included in McCall's model

Functional & Non-Functional (Video1)



<https://www.youtube.com/watch?v=zCX-N1H8Vps&t=6s>

User Requirements



- ✧ Written for **users or customers**.
- ✧ Statement in natural language plus diagrams of the services the system provides and its operational constraints.

System Requirements



- ✧ Written for **developers**.
- ✧ detailed functional and non-functional requirements.
- ✧ Contains specified and clearly details more than user requirements.

User Requirements & System Requirements (Video2)



https://www.youtube.com/watch?v=vpNnZDwC_vs

Software Development Life Cycle SDLC (video3)



<https://www.youtube.com/watch?v=i-QyW8D3ei0&t=208s>



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري

محاضرة الاسبوع الحادي عشر

Requirements Elicitation and Analysis

Topics covered



- ✧ Identification of need
- ✧ Feasibility study
- ✧ Need for feasibility study
- ✧ Types of feasibility study

Identification of need



- ✧ The analyst (system or software engineer) meets with the customer and the end with intent of understanding the **product's objectives** and to **define goals** required meeting the objectives.
- ✧ The analyst then gathers the additional information likes **technology availability** , **resources required**, **cost**, **schedule**, etc.

Feasibility study



- ✧ In **Software Engineering** is a study to evaluate feasibility of proposed project or system.
- ✧ Feasibility study is one of stage among important four stages of SDLC. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of **how much beneficial product development** will be for the organization in a practical point of view.
- ✧ Feasibility study is carried out based on many purposes to analyze whether software product **will be right** in terms of development, implantation, contribution of project to the organization etc.

Need for feasibility study



Feasibility study is so important in any project. As after completion of feasibility study it gives a conclusion of whether to **go ahead** with proposed project as it is practically feasible **or to stop** proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.

Types of feasibility study



The feasibility study mainly concentrates on below five mentioned areas:

➤ Economic feasibility

In Economic Feasibility study **cost and benefit** of the project is analyzed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on.

After that it is analyzed whether project will be beneficial in terms of finance for organization or not.

Feasibility study



➤ Technical feasibility

In Technical Feasibility current resources both **hardware**, **software** along with **required technology** are analyzed/assessed to develop project.

This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development.

Along with this, feasibility study also analyzes **technical skills and capabilities of technical team**, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.

Feasibility study



➤ Legal feasibility

In Legal Feasibility study project is analyzed in **legality point of view**. This includes analyzing barriers of legal implementation of project, **data protection** acts or **social media laws**, project certificate, **license**, copyright etc. Overall, it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.

Feasibility study



➤ Schedule feasibility

In Schedule Feasibility Study mainly **timelines/deadlines** is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can't be completed on time.

Feasibility study



➤ Operational feasibility

In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product **will be to operate and maintenance after deployment**. Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري

محاضرة الاسبوع الثاني عشر

Structured Analysis

(Data Flow Diagram +Data Dictionary)

Topics covered



- ✧ Structured Analysis
- ✧ Structured Analysis Models (Tools):
 - i. Data Flow Diagram
 - ii. Data Dictionary
 - iii. Decision Trees
 - iv. Decision Tables
 - v. Structured English
 - vi. Pseudocode

Structured Analysis



- ✧ **Structured Analysis** is a development method that allows the **analyst** to understand the system and its activities in a logical way.
- ✧ It is a systematic approach, which **uses graphical tools** that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable **by user**.
- ✧ It has following attributes :
 - It is **graphic** which specifies the presentation of application.
 - It divides the processes so that it **gives a clear picture of system flow**.
 - It is an approach that works from high-level overviews to lower-level details.

Structured Analysis

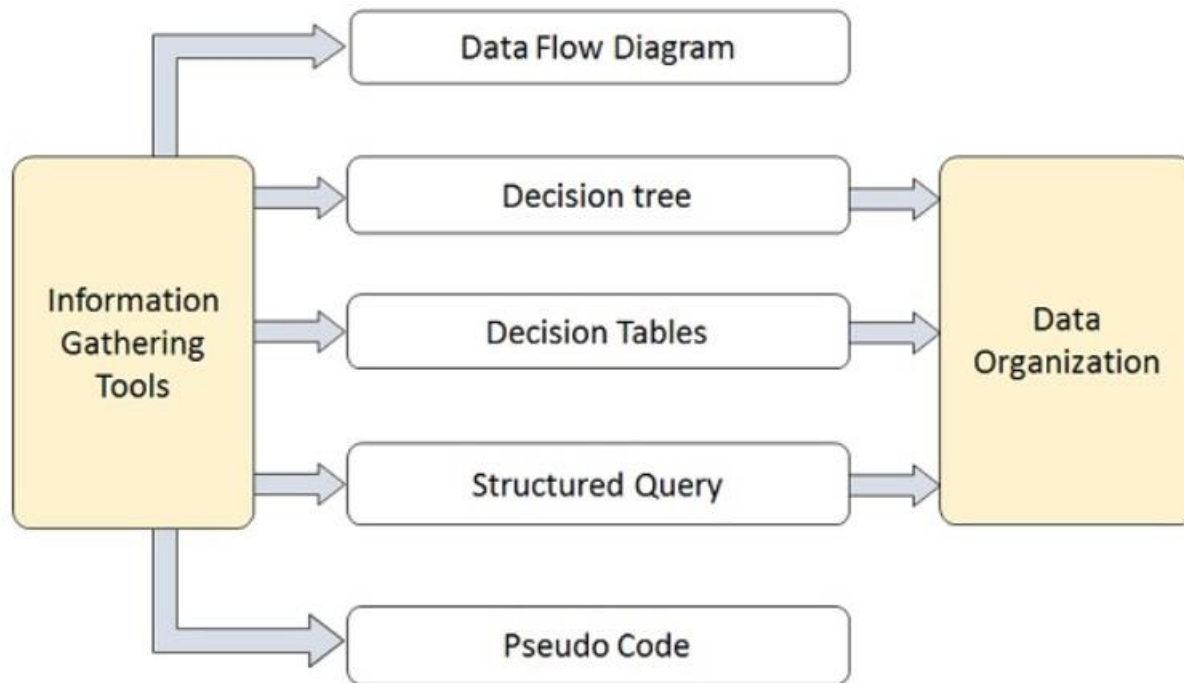


Figure 14: Structured Analysis

Data Flow Diagram (DFD)



- ✧ It is a technique developed by **Larry Constantine** to express the requirements of system in **a graphical form**.
- ✧ A primary tool in structured analysis that graphically illustrates a system's components process and the flow of data between them.
- ✧ It shows the flow of data between various functions of system and specifies how the current system is implemented.
- ✧ It is an **initial stage of design phase** that functionally divides the requirement specifications down to the lowest level of detail.

Data Flow Diagram (DFD)



- ✧ Its graphical nature makes it a good **communication tool** between **user and analyst** or analyst and system designer.
- ✧ It gives an overview of **what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.**

Basic Elements of DFD



- ✧ DFD is **easy to understand** and quite effective when the required design is not clear, and the user wants a notational language for communication.
- ✧ However, it requires a large number of iterations for obtaining the most accurate and complete solution.

Basic Elements of DFD







Symbol Name	Symbol	Meaning
Square		Source or Destination of Data
Arrow		Data flow
Circle		Process transforming data flow
Open Rectangle		Data Store

Figure 15: Basic Elements of DFD

DFD Example: Airline Reservation System

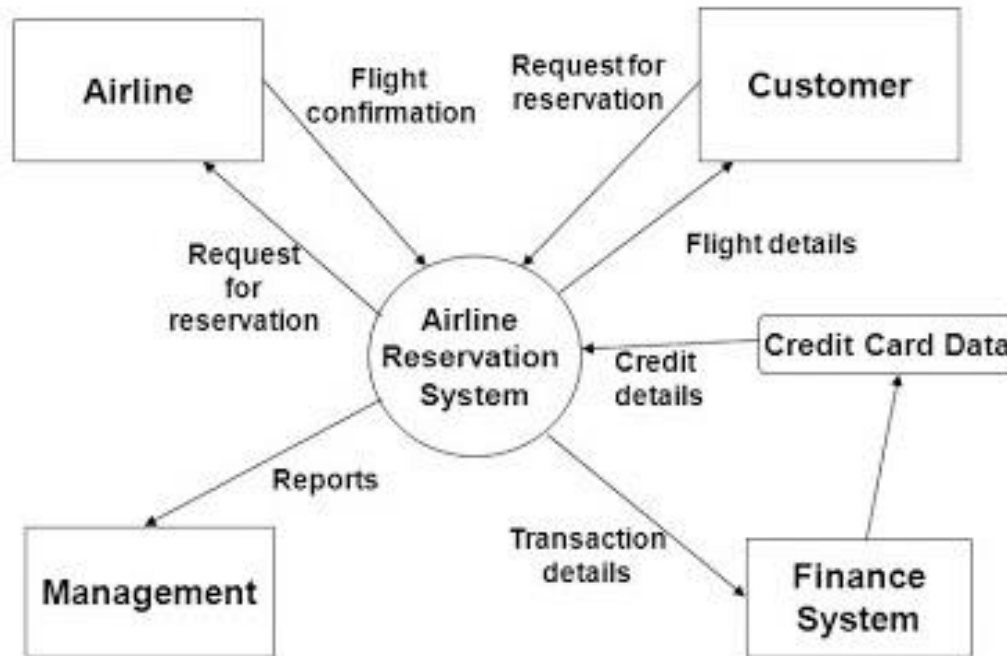


Figure 16: Basic Elements of DFD for Airline Reservation System

Data Dictionary



- ✧ A structured **description** of database.
- ✧ A data dictionary is a structured stores of the data elements in the system. **It stores the descriptions of all DFD** data elements.
- ✧ A data dictionary **improves the communication between the analyst and the user**. It plays an important role in building a database.
- ✧ The information you see in data dictionary is called **metadata**, which, quite simply, is data about data.

Data Dictionary



Data Dictionary

Data Dictionary outlining a Database on Driver Details in NSW

Field Name	Data Type	Data Format	Field Size	Description	Example
License ID	Integer	NNNNNN	6	Unique number ID for all drivers	12345
Surname	Text		20	Surname for Driver	Jones
First Name	Text		20	First Name for Driver	Arnold
Address	Text		50	First Name for Driver	11 Rocky st Como 2233
Phone No.	Text		10	License holders contact number	0400111222
D.O.B	Date / Time	DD/MM/YYYY	10	Drivers Date of Birth	08/05/1956

ScreenCast-O-Matic.com

Figure 17: Data Dictionary

Data Dictionary (Video)



✧ <https://www.youtube.com/watch?v=kH0bcw9P2Lc>

Note:

- ✧ Data model (tool) improves the communication between the **analyst** and the **user**.
- ✧ During the analysis (Requirement) phase, the data model (tool) presents the logical organization of data without indicating how the data are stored, created, or manipulated so that analysts can focus on the business without being distracted by technical details.
- ✧ Later, during the **design phase**, the data model (tool) is changed or reflect exactly how the data will be stored in databases and files.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري

محاضرة الاسبوع الثالث عشر

Structured Analysis

(Decision Trees +Decision Tables)

Topics covered



- ✧ Structured Analysis
- ✧ Structured Analysis Models (Tools):
 - i. Data Flow Diagram
 - ii. Data Dictionary
 - iii. Decision Trees
 - iv. Decision Tables
 - v. Structured English
 - vi. Pseudocode

Decision Trees



- ✧ **Decision trees** are a method for defining complex relationships by describing decisions and avoiding the problems in communication.
- ✧ A decision tree is a diagram that shows **actions and conditions** within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision Trees



- ✧ Decision trees depict the relationship of each **condition** and their permissible **actions**.
- ✧ A **square node** indicates an **action**, and a **circle** indicates a **condition**. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.

Decision Trees

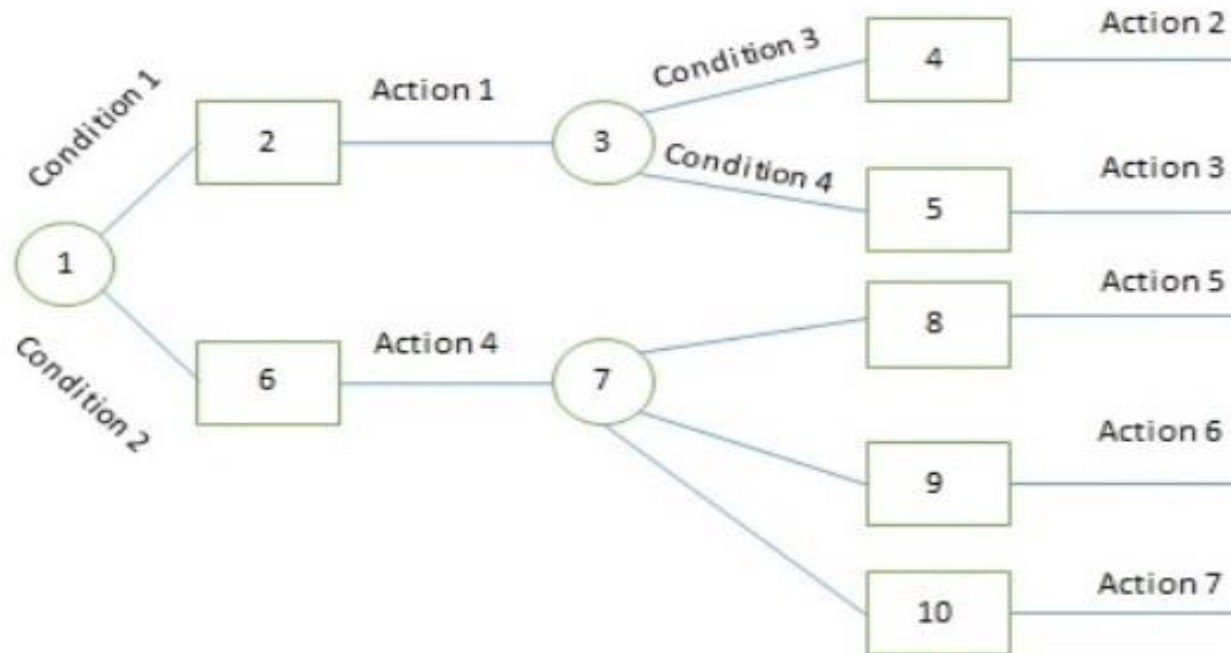


Figure 17: Decision Trees 1

Decision Trees (For example, refer the following decision tree)



✧ Conditions:

C1: Advance payment made

C2: Purchase amount = Rs 10,000/-

C3: Regular Customer

✧ Actions:

A1: Give 5% discount

A2: Give no discount

Decision Trees (For example, refer the following decision tree)



✧ Rules:

R1: Yes Advance payment made = **Give 5% discount**

R2: No Advanced payment made + **Yes** Purchase amount = Rs 10,000/- + **Yes** Regular Customer = **Give 5% discount**

R3: No Advanced payment made + **Yes** Purchase amount = Rs 10,000/- + **No** Regular Customer = **Give no discount**

R4: No Advanced payment made + **No** Purchase amount = Rs 10,000/- = **Give no discount**

Ru1 +R2 = Give 5% discount **R3+R4 = Give no discount**

Decision Trees (For example, refer the following decision tree)

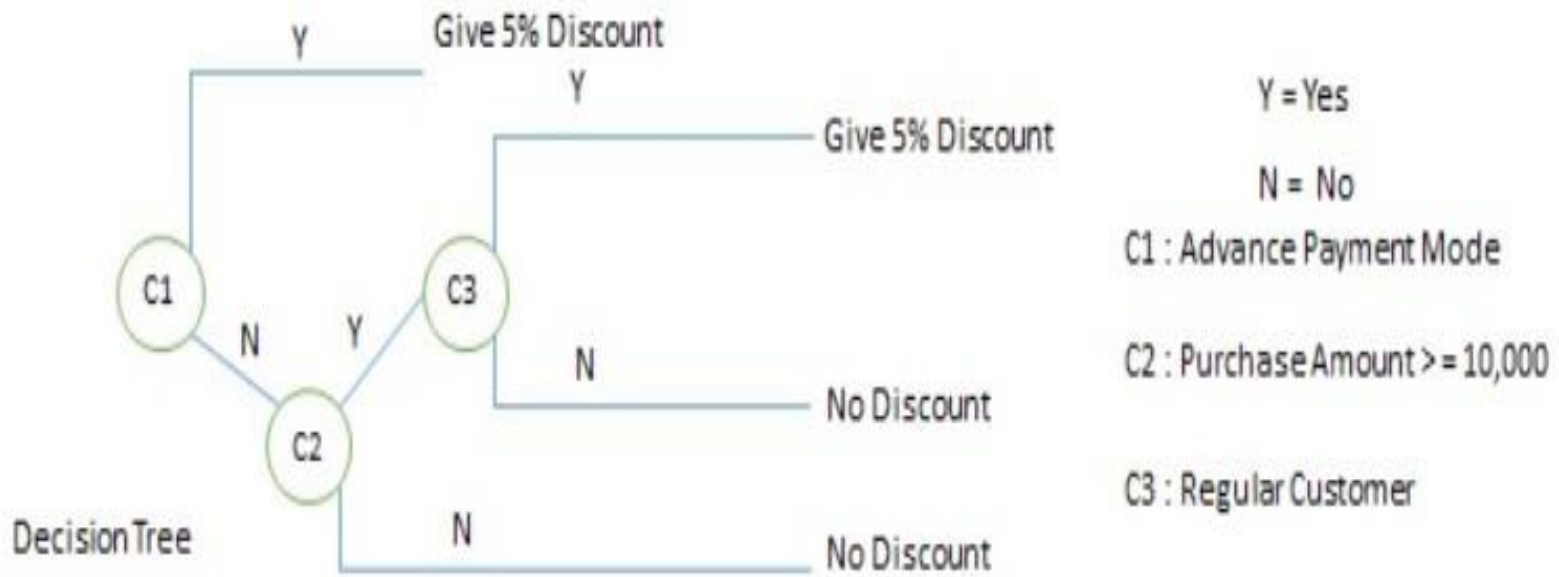


Figure 18: Decision Trees 2

Decision Tables



✧ **Decision tables** are a method of describing the complex logical relationship in a precise manner which is easily understandable.

It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.

It is a matrix containing row or columns for defining a problem and the actions.

Components of a Decision Table



- **Condition Stub** – It is in the upper left quadrant which lists all the condition to be checked.
- **Action Stub** – It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
- **Condition Entry** – It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
- **Action Entry** – It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

Decision Tables



- ✧ The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action.
- ✧ In rules section,
 - Y** shows the existence of a condition.
 - N** represents the condition, which is not satisfied.
 - A** blank - against action states it is to be ignored.
 - X** (or a check mark will do) against action states it is to be carried out.

Decision Table (For example, refer the following decision table)



✧ Conditions:

C1: Advance payment made

C2: Purchase amount = Rs 10,000/-

C3: Regular Customer

✧ Actions:

A1: Give 5% discount

A2: Give no discount

Decision Trees (For example, refer the following decision tree)



✧ Rules:

R1: Yes Advance payment made = **Give 5% discount**

R2: No Advanced payment made + **Yes** Purchase amount = Rs 10,000/- + **Yes** Regular Customer = **Give 5% discount**

R3: No Advanced payment made + **Yes** Purchase amount = Rs 10,000/- + **No** Regular Customer = **Give no discount**

R4: No Advanced payment made + **No** Purchase amount = Rs 10,000/- = **Give no discount**

Ru1 +R2 = Give 5% discount **R3+R4 = Give no discount**

Decision Tables (For example, refer the following decision table)



CONDITIONS	Rule 1	Rule 2	Rule 3	Rule 4
Advance payment made	Y	N	N	N
Purchase amount = Rs 10,000/-	-	Y	Y	N
Regular Customer	-	Y	N	-
ACTIONS				
Give 5% discount	X	X	-	-
Give no discount	-	-	X	X

Table 2: Decision Table



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري

محاضرة الاسبوع الرابع عشر

Structured Analysis

(Pseudocode + Flow Chart)

Topics covered



- ✧ Structured Analysis
- ✧ Structured Analysis Models (Tools):
 - i. Data Flow Diagram
 - ii. Data Dictionary
 - iii. Decision Trees
 - iv. Decision Tables
 - v. Pseudocode
 - vi. Flow Chart

Pseudocode



- ✧ Pseudocode is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations.
- ✧ It is used for creating an outline or a rough draft of a program.
- ✧ Pseudocode summarizes a program's flow, but excludes underlying details.
- ✧ System analyst and designers write pseudocode to ensure that programmers understand a software project's requirements and align code accordingly.

Example of Pseudocode



✧ Conditions:

C1: Advance payment made

C2: Purchase amount = Rs 10,000/-

C3: Regular Customer

✧ Actions:

A1: Give 5% discount

A2: Give no discount



Example of Pseudocode

✧ Rules:

R1: Yes Advance payment made = **Give 5% discount**

R2: No Advanced payment made + **Yes** Purchase amount = Rs 10,000/- + **Yes** Regular Customer = **Give 5% discount**

R3: No Advanced payment made + **Yes** Purchase amount = Rs 10,000/- + **No** Regular Customer = **Give no discount**

R4: No Advanced payment made + **No** Purchase amount = Rs 10,000/- = **Give no discount**

Ru1 +R2 = Give 5% discount **R3+R4 = Give no discount**

Example of Pseudocode



```
if customer pays advance
then
    Give 5% Discount
else
    if purchase amount >=10,000
    then
        if the customer is a regular customer
        then Give 5% Discount
        else No Discount
        end if
    else No Discount
    end if
end if
```

Figure 19: Pseudocode

Flow Chart



- ✧ A flowchart is a graphical representations of steps. It was originated from computer science as a **tool for representing algorithms and programming logic** but had extended to use in all other kinds of processes.
- ✧ Nowadays, flowcharts play an extremely important role in displaying information and assisting reasoning.
- ✧ They help us visualize complex processes, or make explicit the structure of problems and tasks.

Flow Chart Symbols

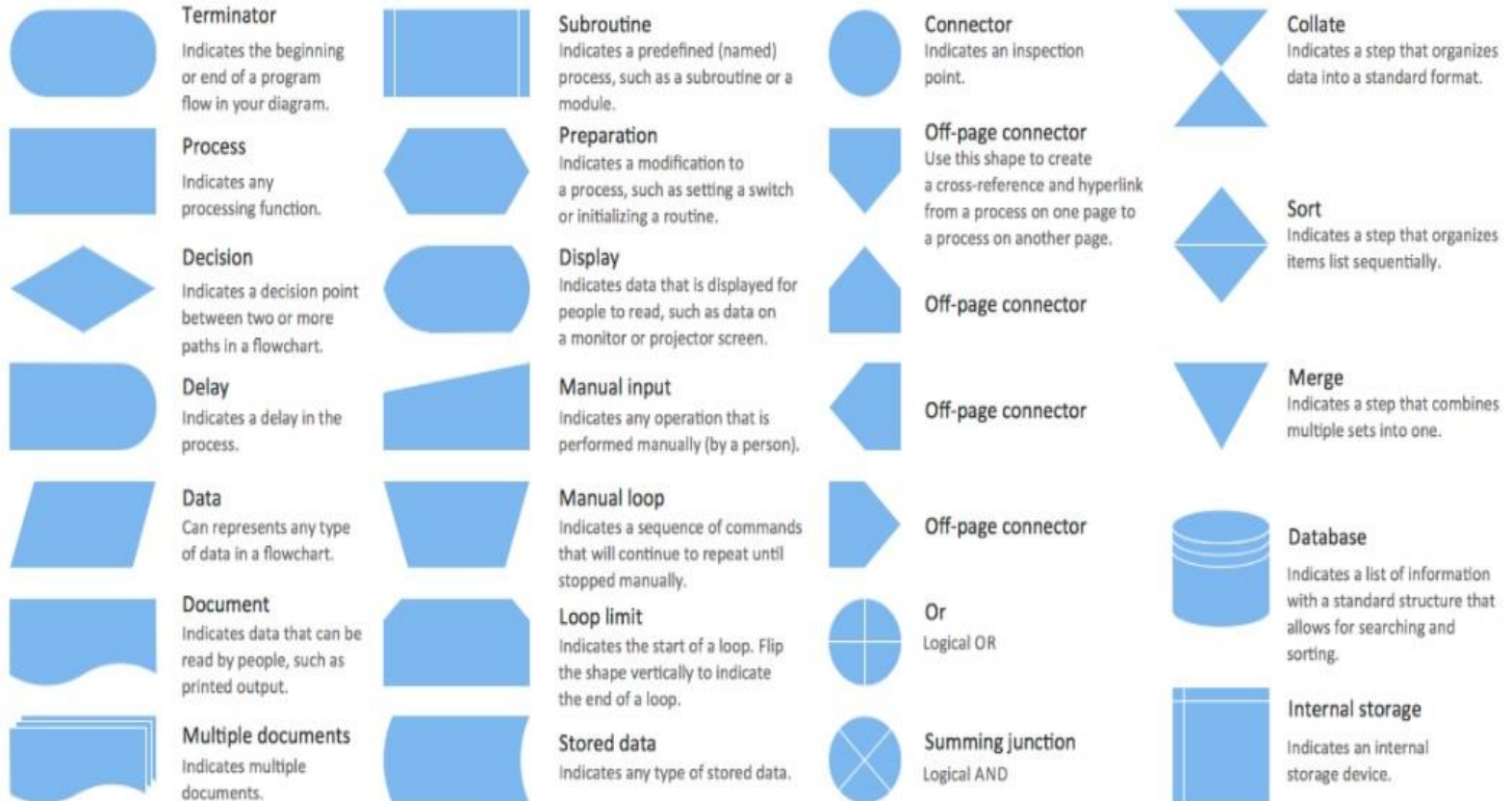


Figure 20: Flow Chart Symbols
Dr. Ali Yahya Ghem

Pseudocode & Flow Chart










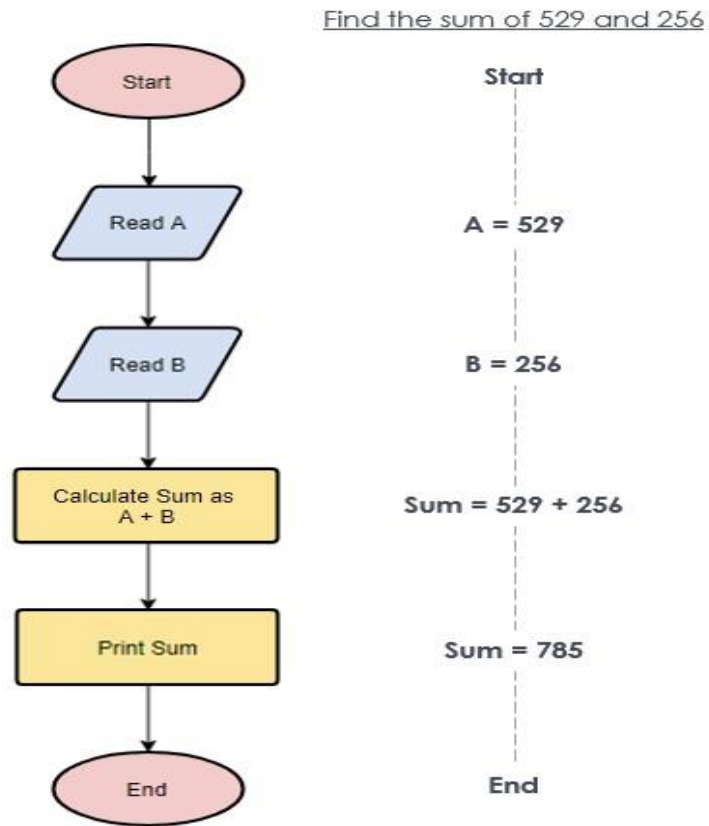
	1) Flowchart	2) Pseudocode
Start/End		Begin/End (not necessary)
Input/Output		Read/Print : read x,y
Process		[An equation] : $x=y+6$ Comput: compute x as $y+6$
Decision		If/ if-else / While
Predefined process		[function_name()]: average(x,y)
Line		Not Applicable
Connector		Not Applicable

Table 3: Pseudocode & Flow Chart

Example 1: : The flowchart example below shows a simple summation process



Figure 21: Flow Chart (1)



Example 2: The flowchart example below shows how profit and loss can be calculated

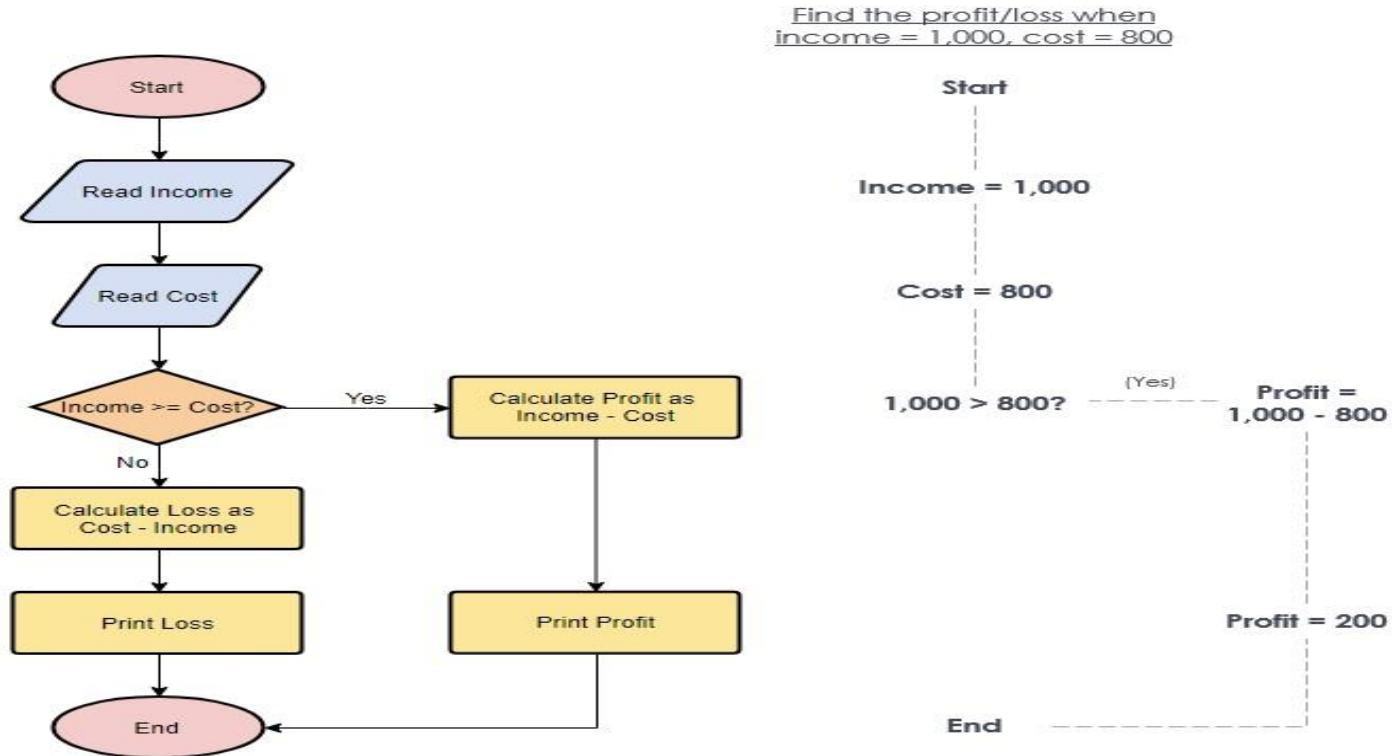


Figure 22: Flow Chart (2)

Microsoft ViSiO

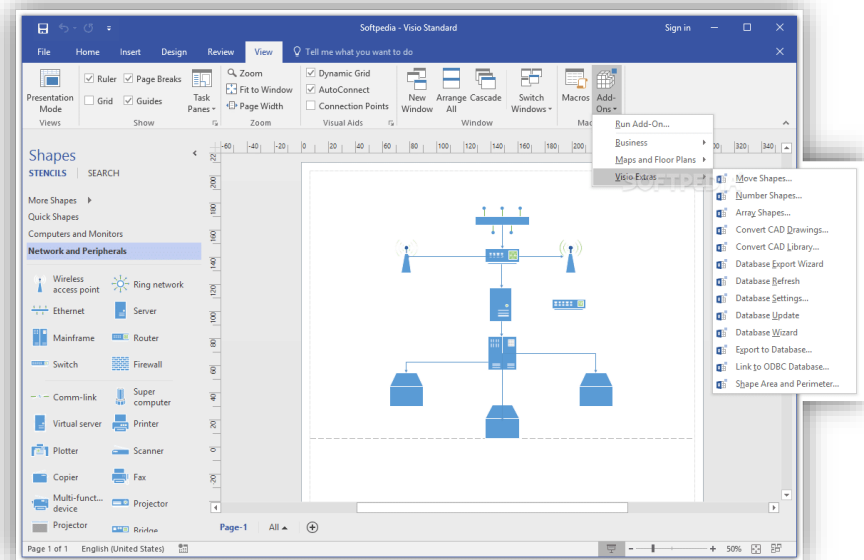


Figure 23: Microsoft Visio



Q: Draw a Flow Chart for the following :

✧ **Conditions:**

C1: Advance payment made

C2: Purchase amount = Rs 10,000/-

C3: Regular Customer

✧ **Actions:**

A1: Give 5% discount

A2: Give no discount



Q: Draw a Flow Chart for the following :

✧ Rules:

R1: Yes Advance payment made = **Give 5% discount**

R2: No Advanced payment made + **Yes** Purchase amount = Rs 10,000/- + **Yes** Regular Customer = **Give 5% discount**

R3: No Advanced payment made + **Yes** Purchase amount = Rs 10,000/- + **No** Regular Customer = **Give no discount**

R4: No Advanced payment made + **No** Purchase amount = Rs 10,000/- = **Give no discount**

Ru1 +R2 = Give 5% discount **R3+R4 = Give no discount**

Assignment 2 😊



#95482756

Guide for selecting appropriate tools



- ✧ Use **DFD** at high or low level analysis for providing good system documentations.
- ✧ Use **data dictionary** to simplify the structure for meeting the data requirement of the system.
- ✧ Use **pseudocode** English if there are many loops and actions are complex.
- ✧ Use **decision tables** when there are a large number of conditions to check and logic is complex.
- ✧ Use **decision trees** when sequencing of conditions is important and if there are few conditions to be tested.
- ✧ Use **flow chart** to clarify complex processes.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري

محاضرة الاسبوع الخامس عشر

Structured Analysis

Entity-Relationship Model/Diagram (ER)

Topics covered



✧ What is Entity-Relationship Model (ER)?

✧ Constraints of relationship

Modality & Cardinality

✧ Forms of Cardinality

One-to-one

One-to-many

Many-to-many

Entity-Relationship Diagram (ER)



- ✧ **Entity-Relationship Model** is a popular notation used to show the relationship between entities. It was first introduced as part of entity-relationship model by Peter Chen in 1976.
- ✧ All of these modeling notations may be used for requirements definition and analysis as well as for design.
- ✧ In fact, the preferred approach is to use same notation starting at requirements analysis and continuing through the design of the software.
- ✧ ER Model creates a set of **entities with their attributes**, a set of **constraints and relation** among them.

Entity-Relationship Diagram (ER)



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

Entity-Relationship Model (ER)

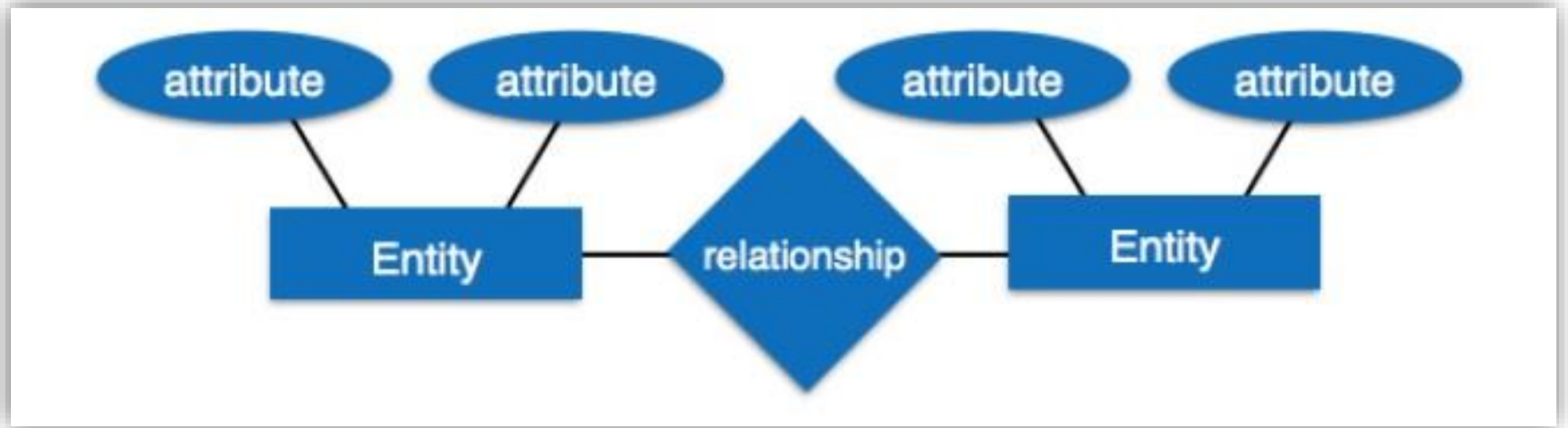


Figure 24: Entity-Relationship Model (ER)

The constraints of relationship



❖ Modality (Zero)

❖ Cardinality

- One-to-one
- One-to-many
- Many-to-many

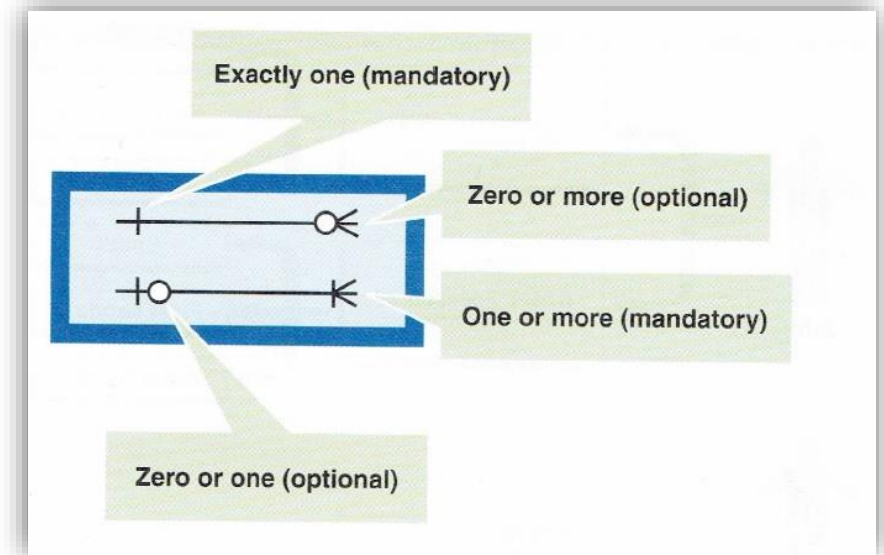


Figure 25: Constraints of relationship

Modality (zero or one, zero or many)



Information Engineering Style

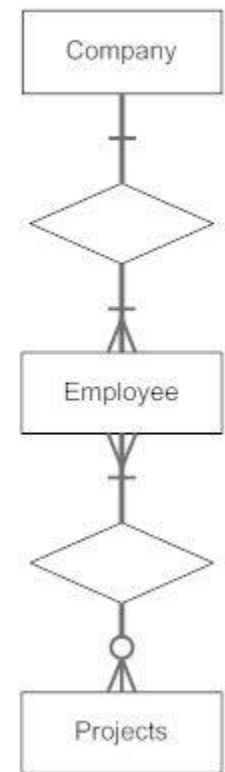
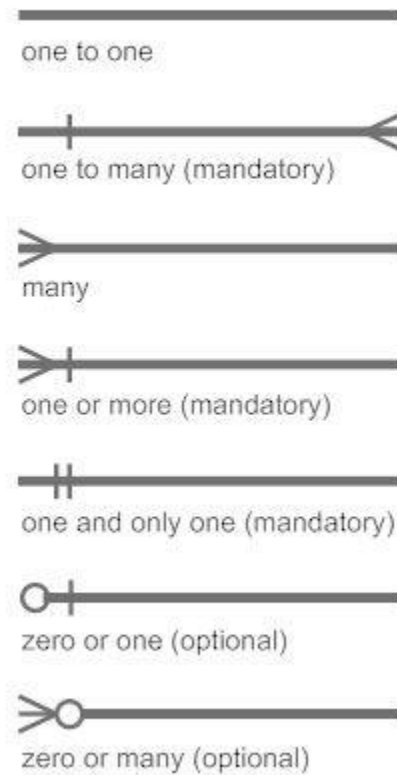


Figure 26: zero or one, zero or many

Cardinality (One-to-one)



Information Engineering Style

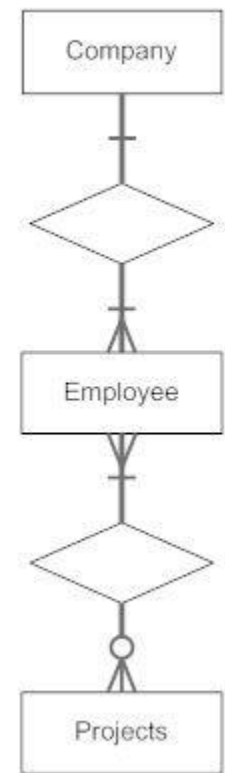
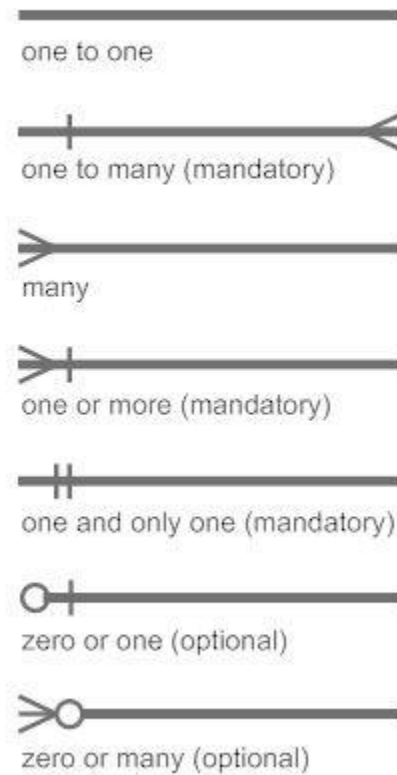


Figure 27: One-to-one

Cardinality (One-to-many)



Information Engineering Style

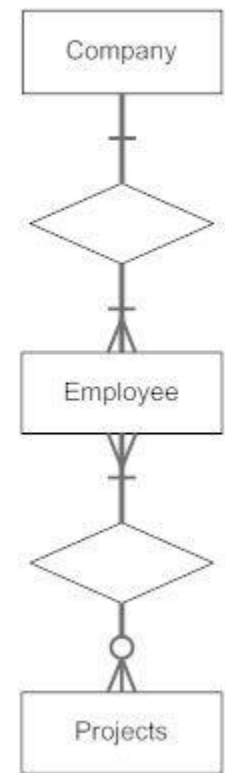
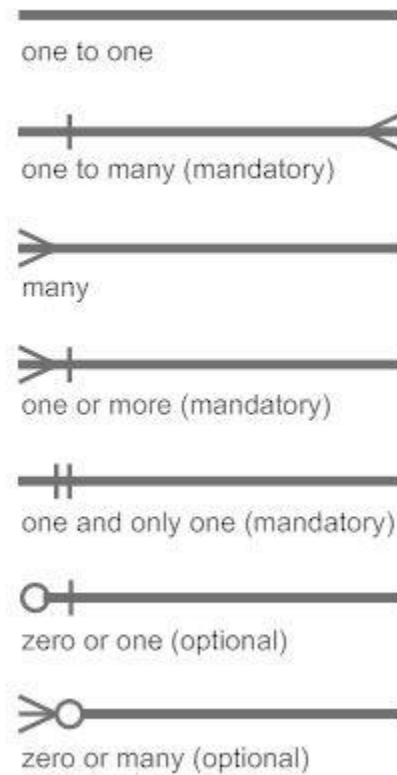


Figure 27: One-to-many

Cardinality (Many-to-many)



Information Engineering Style

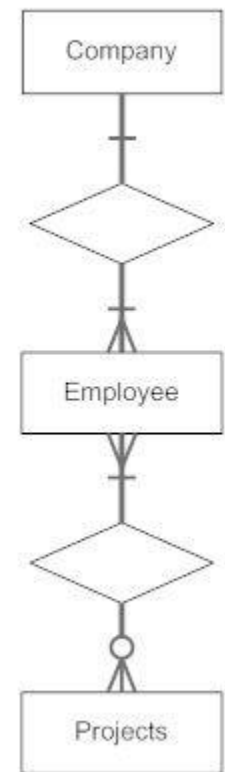
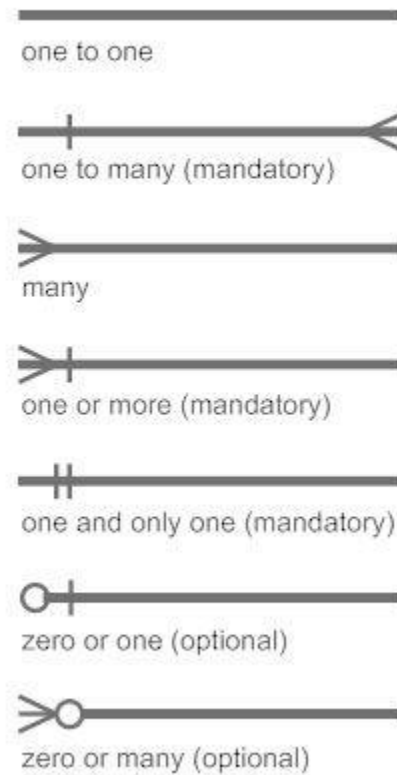


Figure 28: Many-to-many



Quiz?

Explain in ER model the relationship between programmers and modules where a programmer may write several modules and each module may also be written by several programmers.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري

محاضرة الاسبوع التاسع عشر

Software Design

Topics covered



- ✧ What is Software Design?
- ✧ Characteristics of a good software design.
- ✧ What is Modularity or Modularization?
- ✧ Software Design and its activities (Architectural design and Detailed design).
- ✧ Coupling and Cohesion.

Software Design



- ❖ **Software design** is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- ❖ For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. **The output of this process can directly be used into implementation in programming languages.**

Characteristic of a good software design



Most researchers and software engineers agree on a few desirable characteristics that every good software design for general application must process. The characteristics are listed below:

✧ **Correctness:** Implement all functionalities identified in the SRS document.

✧ **Understandability:** A good design is easily understandable.

Characteristic of a good software design



✧ **Efficiency:** It should be efficient.

✧ **Maintainability:** It should be easily amenable to change.

Possibly the most important goodness criterion is **design correctness**. A design has to be **correct to be acceptable**. Given that a design solution is correct, **Understandability** of design is possibly the most important issue to be considered while judging the goodness of a design.

Characteristic of a good software design



- ✧ A design that is **easy to understand** is also **easy to develop, maintain and changes**. Thus, unless a design is easily understandable, it would require tremendous effort to implement and maintain it.

Characteristic of a good software design



According to Frank Tsui and Orlando Karam (2007), a good software design is:

- ✧ Easy to understand.
- ✧ Easy to change.
- ✧ Easy to reuse.
- ✧ Easy to test.
- ✧ Easy to integrate.
- ✧ Easy to code.

What is Modularity or Modularization?



- ✧ **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.
- ✧ Modular design follows the rule of '**divide and conquer**' problem-solving strategy, this is because there are many other benefits attached with the modular design of a software.

What is Modularity or Modularization?



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
- ✧ Concurrent execution can be made possible
- ✧ Desired from security aspect

Software Design and its activities



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

Coupling and Cohesion



- **Coupling** often refers to how the modules belong together. Modules should be as independent as possible from other modules. (**Low Coupling**).
- **Cohesion** often refers to how the functions of a module belong together. Related code should be close to each other to make it highly cohesive (**High Cohesion**).

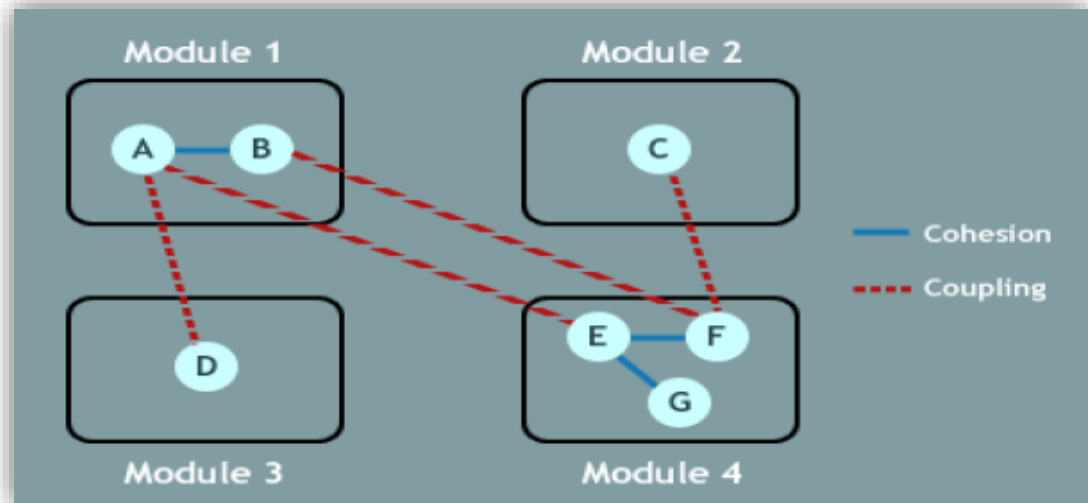


Figure 29: Coupling and Cohesion

**“Good Software Design has
Low Coupling and High
Cohesion”**

Coupling and Cohesion

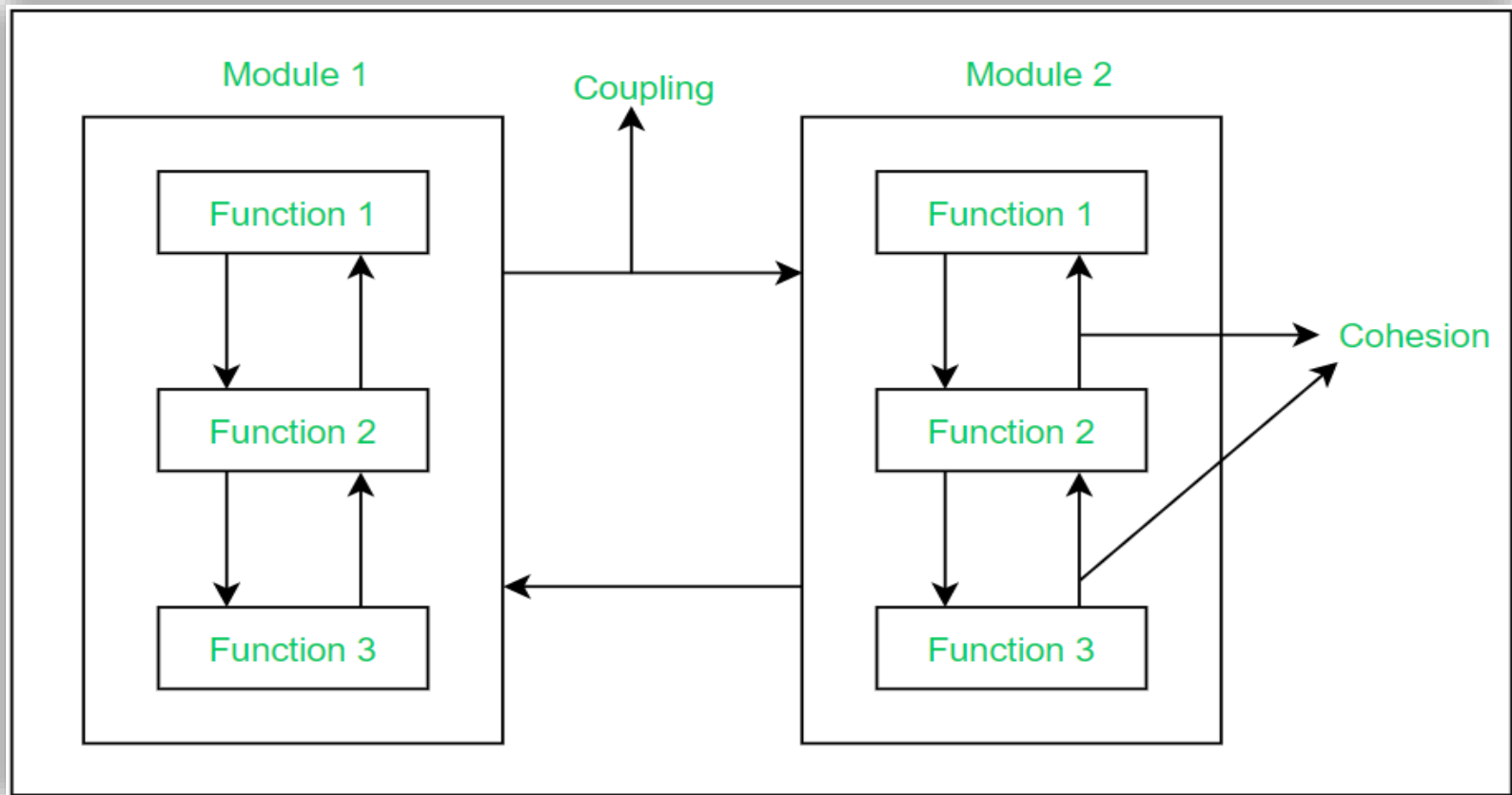


Figure 30: Coupling and Cohesion

Cohesion



- The **person** class has **low cohesion**, simply because Person's responsibilities is relevant to save information about people. It do not relate to functionalities about read/write to file. So, to reduce low cohesion, we should separate the implementation about read/write file into other class such as File, ...

```
public class Person {  
    private int    age;  
    private String name;  
  
    // getter, setter properties.  
  
    // method  
    public void readInfor();  
  
    public void writeInfor();  
}
```

Figure 31: Low Cohesion

Cohesion

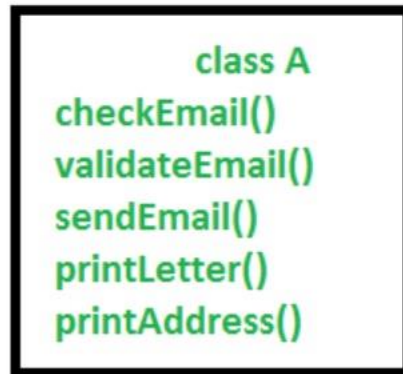


Fig: Low cohesion

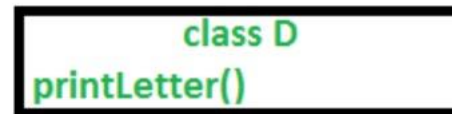
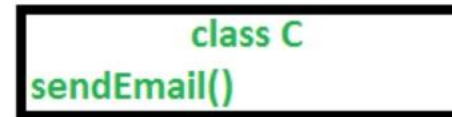
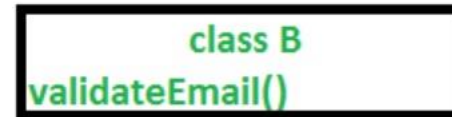
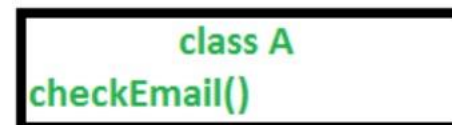


Fig: High cohesion

Figure 32: Cohesion

Coupling



- Low coupling can be achieved by having less classes linking to one another. The best way to reduce coupling is by providing an API (interface). OOP languages like C#, Java, C++ offer quite a few mechanisms to offer low coupling like public classes, public methods, interfaces and other interaction points between different classes and modules.

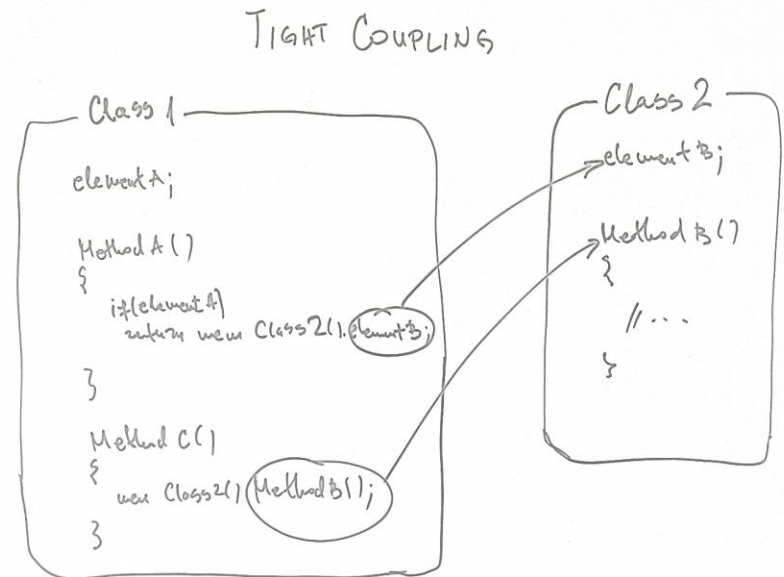


Figure 33: High or Tight Coupling



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري محاضرة الاسبوع العشرون

Software Design Strategies

Topics covered



- ✧ Structured Design
- ✧ Function Oriented Design
- ✧ Object Oriented Design
- ✧ Software Interface Design

Structured Design



- ❖ **Structured design** is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. **Benefit of structured design** is, it gives **better understanding** of how the problem is being solved. Structured design also makes it **simpler** for designer to concentrate on the problem more accurately.
- ❖ Structured design is mostly based on '**divide and conquer(Control)**' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

Structured Design



- ✧ The small pieces of problem are solved by means of solution **modules**. Structure design emphasis that these modules be well organized in order to achieve precise solution.

- ✧ These modules communicate with each other. **A good structured design** always follows some rules for communication among multiple modules, namely:
 - **Cohesion** - grouping of all functionally related elements.
 - **Coupling** - communication between different modules.
 - ✧ A good structured design has high cohesion and low coupling arrangements.

Structured Design



✧ **Structured programming:** is a style of programming designed to make programs more comprehensible and programming errors less frequent. Structured programming should usually include:

1. **Block structure:** The statements in the program must be organized into functional groups.
2. **Avoidance of jumps:** (“GO-TO-less programming”)
3. **Modularity:** Programs should be broken up into subroutines, even if some of the subroutines are called only one.

Structured Design



✧ Note:

In different programming languages, a **subroutine** may be called a **routine**, **subprogram**, **function**, **method**, or **procedure**. Technically, these terms all have different definitions.

Structured Design



Structured Pascal program

```
IF x<=y THEN
  BEGIN
    z := y - x;
    q := SQRT (Z);
  END
ELSE
  BEGIN
    z:= x-y;
    q:= -sqrt (z)
  END;
WRITELEN (z , q);
```

Unstructured Pascal program

```
IF x>y THEN GOTO 2;
Z := Y - X;
q := sqrt (z);
GOTO 1;
2: Z:= X - Y;
q:= -sqrt (z);
1:= writeln (z , q);
```

Function Oriented Design



- ❖ In **function oriented design**, the system comprises of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.
- ❖ This design mechanism divides the whole system into **smaller functions**, which provides means of abstraction by concealing the information and their operation. These functional modules can share information among themselves by means of information passing and using information available globally.

Function Oriented Design (Vending Machine)



Figure 32: Vending Machine

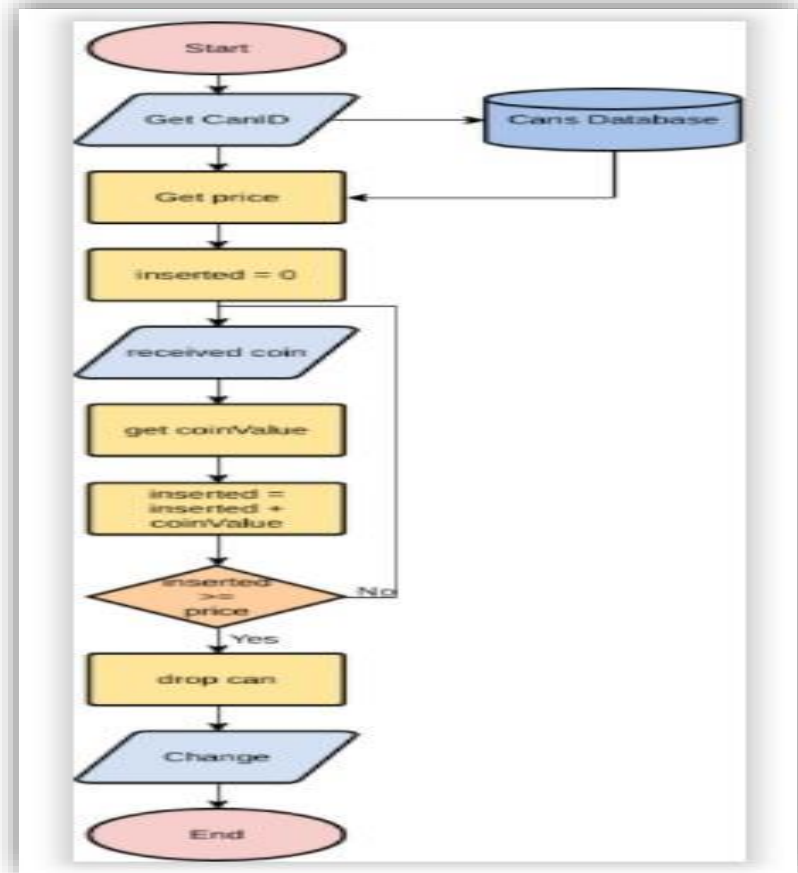


Figure 33: DFD of Vending Machine

Function Oriented Design (ATM)



ATM machine Functions

- Cash Withdraw
- Deposit
- Transfer
- Balance inquiry
- Mobile Top-Up

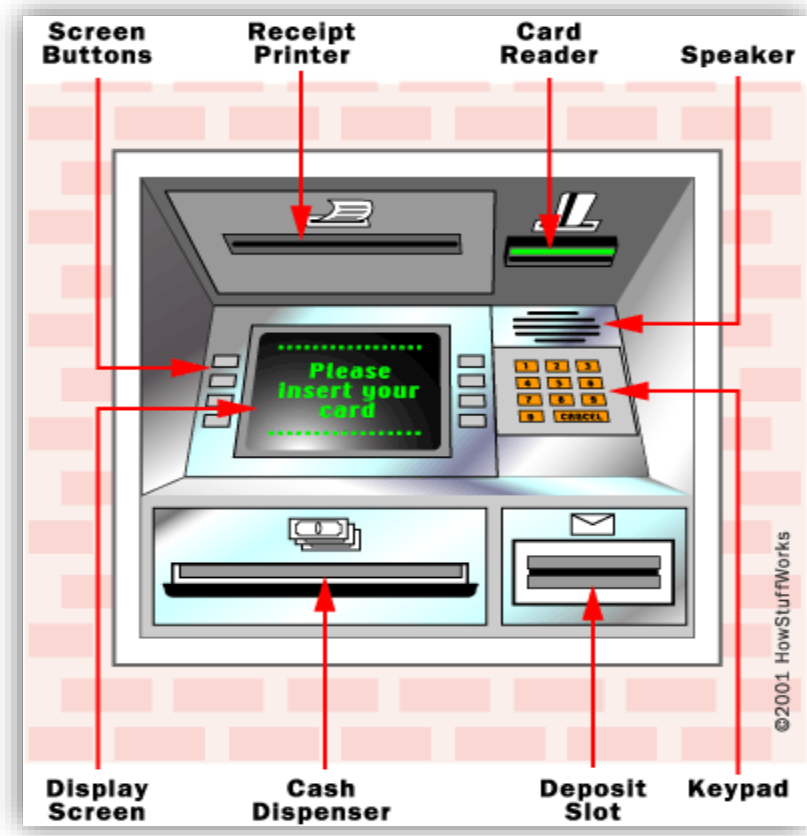


Figure 35: ATM Machine

Report (2)



- As a software designer, What are the main functions and sub functions need to consider to design ATM Machine (Function Oriented Design) for a new bank? You can use some design tools such as:
 - DFD



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري

محاضرة الاسبوع الواحد والعشرون

Graphical User Interface (GUI)

Topics covered



- ✧ Graphical User Interface (GUI)
- ✧ GUI Elements
- ✧ Application Specific GUI Components

Software Interface Design



- ❖ **Software Interface Design (UI):** User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc.
- ❖ **User interface** is part of software and is designed in such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.
- ❖ UI can be **graphical, text-based, audio-video based**, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

More Popular Software Interface



The software becomes more popular if its user interface is:

- ✧ Attractive
- ✧ Simple to use
- ✧ Responsive in short time
- ✧ Clear to understand
- ✧ Consistent on all interfacing screens

Graphical User Interface



- ✧ **Graphical User Interface (GUI)** provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software.
- ✧ Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI designs that work with more **efficiency, accuracy, and speed.**

GUI Elements



- ✧ **GUI** provides a set of components to interact with software or hardware.
- ✧ **Every graphical component** provides a way to work with the system. A GUI system has following elements such as windows

GUI Elements



- **Window** - An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen. A window may contain another window of the same application, called child window.



Figure 36: GUI Elements

GUI Elements



- **Tabs** - If an application allows executing multiple instances of itself, they appear on the screen as separate windows. **Tabbed Document Interface** has come up to open multiple documents in the same window. This interface also helps in viewing preference panel in application. All modern web-browsers use this feature.

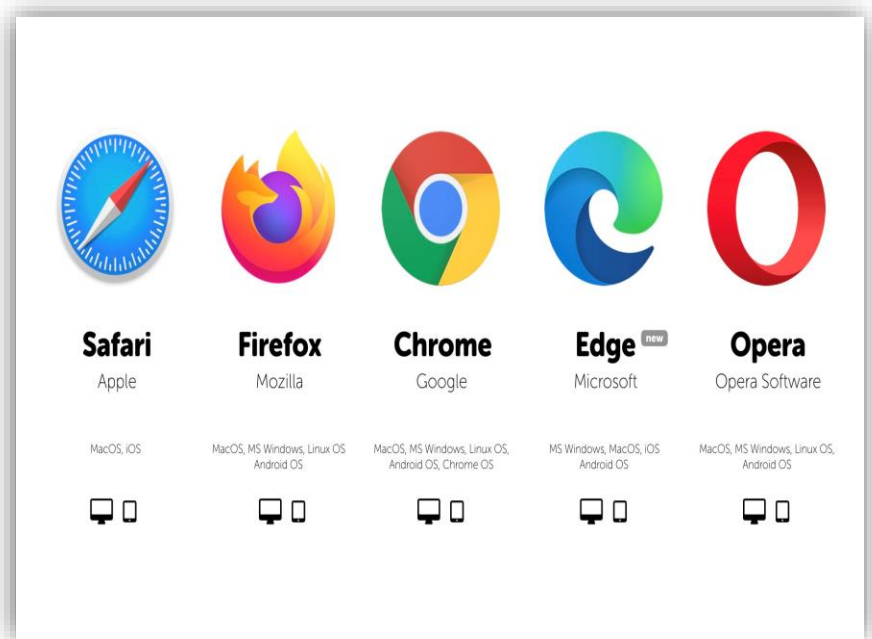


Figure 37: GUI Elements

GUI Elements



- **Menu** - Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.

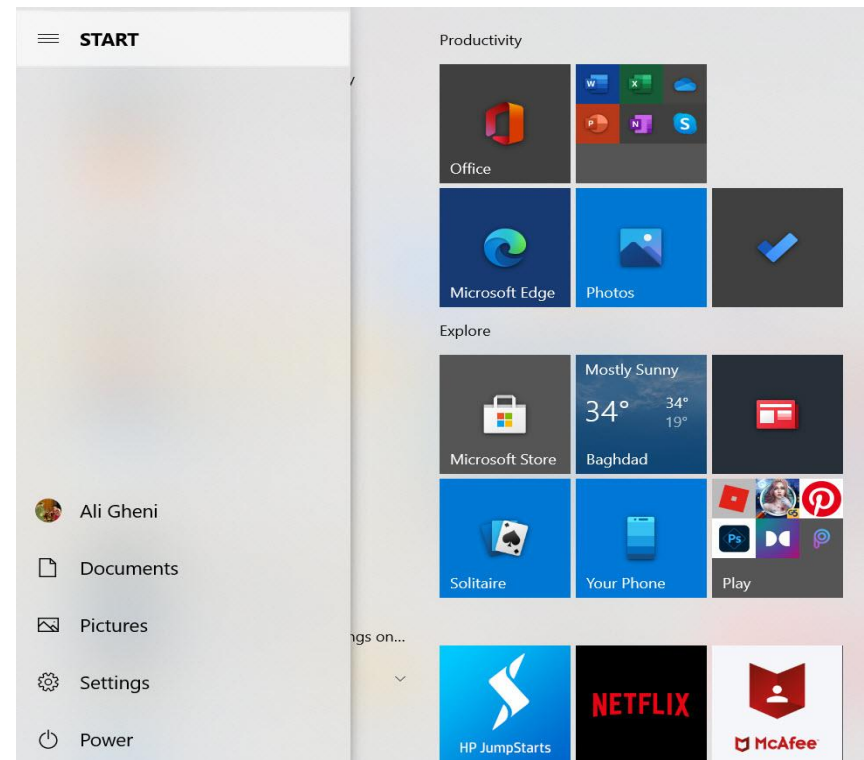


Figure 38: GUI Elements

GUI Elements



- **Icon** - An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.

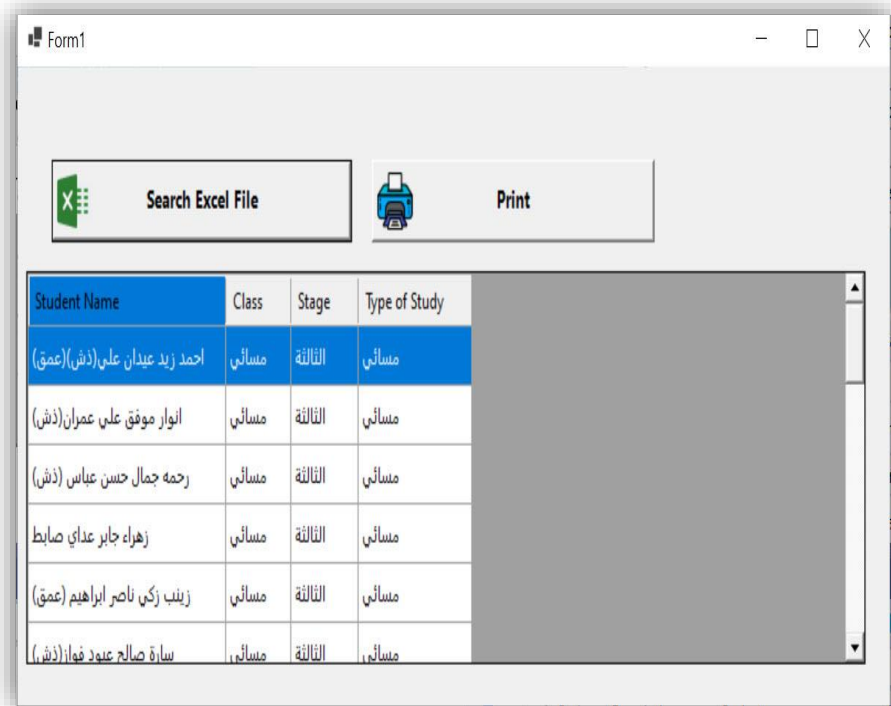


Figure 39: GUI Elements

GUI Elements



- **Cursor** - Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. Cursors are also named pointers in GUI systems. They are used to select menus, windows and other application features.



Figure 40: GUI Elements

Question???

As a software Designer, find out the main reasons of why Nokia failed?





Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري

محاضرة الاسبوع الثاني والعشرون

Software Testing

Topics covered



- ✧ What is Software Testing?
- ✧ Software Validation
- ✧ Software Verification
- ✧ Testing Approaches

Functionality testing (Black-box testing)

Implementation testing (White-box testing)

What is Software Testing?



- ❖ **Software Testing** is evaluation of the software against 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 comprises of Validation and Verification.

Software Validation



- ❖ **Software 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 emphasizes on user requirements.

Software Verification



- ❖ **Software Verification** is the process of confirming if the software is meeting the business requirements, and is developed adhering 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 concentrates on the design and system specifications.

Testing Approaches



- ❖ Functionality testing (Black-box testing)
- ❖ Implementation testing (White-box 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.

Exhaustive tests are the best-desired method for a perfect testing. Every single possible value in the range of the input and output values is tested. It is not possible to test each and every value in real world scenario if the range of values is large.

Black-box testing



- It is carried out to test functionality of the program and also called 'Behavioural' 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.

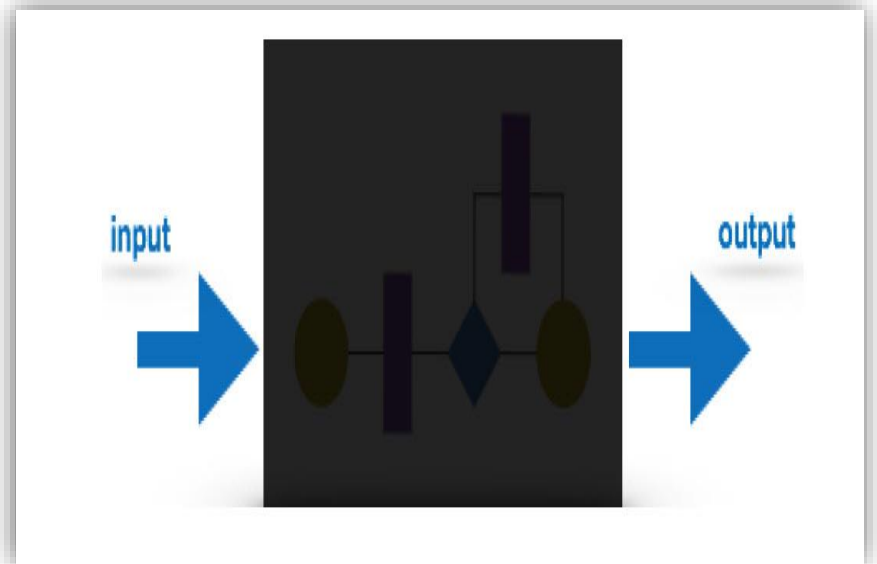


Figure 41: Black-box testing

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. In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

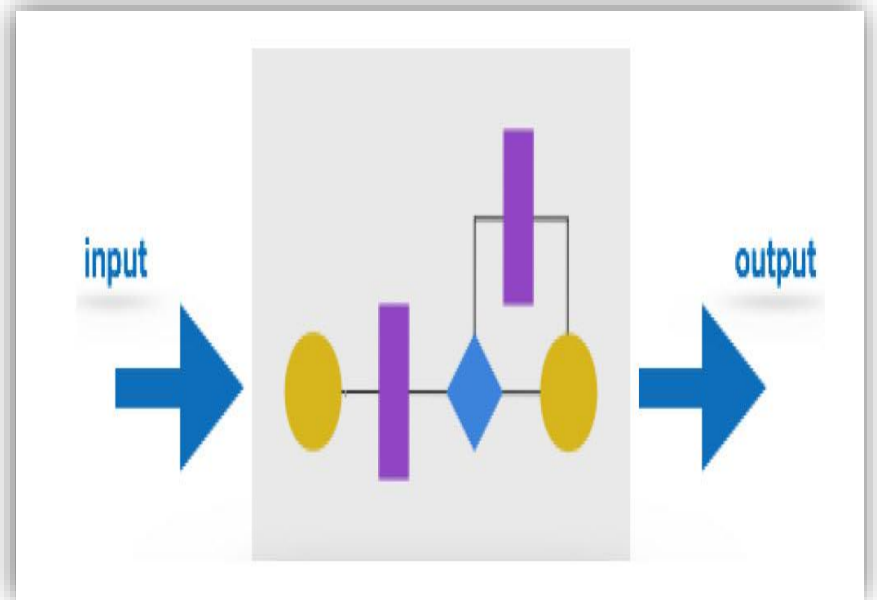


Figure 42: White-box testing



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري

محاضرة الاسبوع الثالث والعشرون

Testing Levels

Topics covered



✧ Testing Levels

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
 - Alpha Testing
 - Beta Testing
- Regression Testing

Testing Levels



- ✧ **Testing itself may be defined** at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified.
- ✧ Testing separately is done just to make sure that there are **no hidden bugs or issues** left in the software. Software is tested on various levels -

Unit Testing



- While coding, the programmer performs some tests on that 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.

The screenshot shows a software application window titled 'Form1'. The window contains a data entry form for a faculty member. The form has a header with a logo and the text 'كلية التربية للعلوم الصرفة (ابن الهيثم)'. Below the header, there are several input fields: 'اسم التدريسي' (Instructor Name), 'المادة' (Subject), 'المرحلة' (Level), 'الفرقة' (Section), 'وقت بدأ المحاضرة' (Lecture Start Time), 'وقت انتهاء' (Lecture End Time), and 'الرابط' (Link). At the bottom of the form, there is a table with columns for 'اسم التدريسي', 'المادة', 'المرحلة', and 'الرابط'. The table is currently empty. On the right side of the window, there is a vertical toolbar with buttons for 'إضافة اسم جديد' (Add New Name), 'حفظ' (Save), 'طباعة' (Print), 'حذف' (Delete), 'مسح' (Clear), and 'خروج' (Exit).

Figure 43: Unit testing

Integration testing

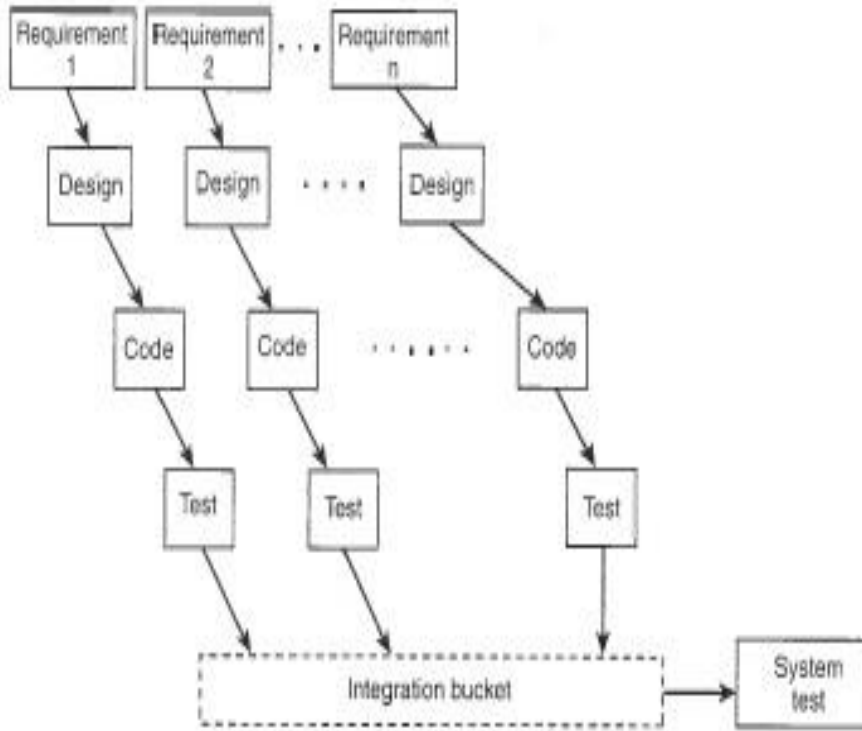


Figure 44: Integration testing

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updating etc.

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. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.
- ✧ **Security & Portability** - These tests are done when the software is meant to work on various platforms and accessed by number of persons.

Acceptance Testing



- ✧ When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.
- ✧ **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.

Alpha Testing VS Beta Testing



Figure 45 Alpha testing VS Beta testing

Alpha Testing VS Beta Testing



No.	Alpha Testing	Beta Testing
1)	It is always done by developers at the software development site.	It is always performed by customers at their own site.
2)	It is also performed by Independent testing team.	It is not be performed by Independent testing team.
3)	It is not open to the market and public.	It is open to the market and public.
4)	It is always performed in virtual environment.	It is always performed in real time environment.
5)	It is used for software applications and projects.	It is used for software products.
6)	It follows the category of both white box testing and Black Box Testing.	It is only the kind of Black Box Testing.
7)	It is not known by any other name.	It is also known as field testing.

Table 46 Alpha testing VS Beta testing

Regression Testing



- ✧ Whenever a software product is updated with **new code, feature or functionality**, it is tested thoroughly to detect if there is any **negative impact of the added code**. This is known as regression testing.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري

محاضرة الاسبوع الرابع والعشرون

Software Project Management

Topics covered



- ✧ What is a project?
- ✧ What are the triple constraint of the project?
- ✧ What is a software project?
- ✧ What is a Project Life Cycle (PLC)?
- ✧ What is System Development Life Cycle (SDLC)
- ✧ System Development Life Cycle (SDLC) and Project Life Cycle (PLC)

What is a project?



- ✧ **Project** : A set of activities intended to produce a specific output, which has a definite beginning and end.
- ✧ **A Project** is “ a temporary endeavor undertaken to create a unique product, service, or result.”

What are the triple constraint of the project?



- ❖ **Scope:** What a work will be done as part of the project? What unique project, service, or result does the customer or sponsor expect from the project? How the scope be verified?
- ❖ **Time:** How long should it take to complete the project? What is the project's schedule? How will the team track actual schedule performance? Who can approve changes to the schedule?
- ❖ **Cost:** What should it cost to complete the project? What is the project's budget? How will costs be tracked? Who can authorize changes to the budget?

What are the triple constraint of the project?

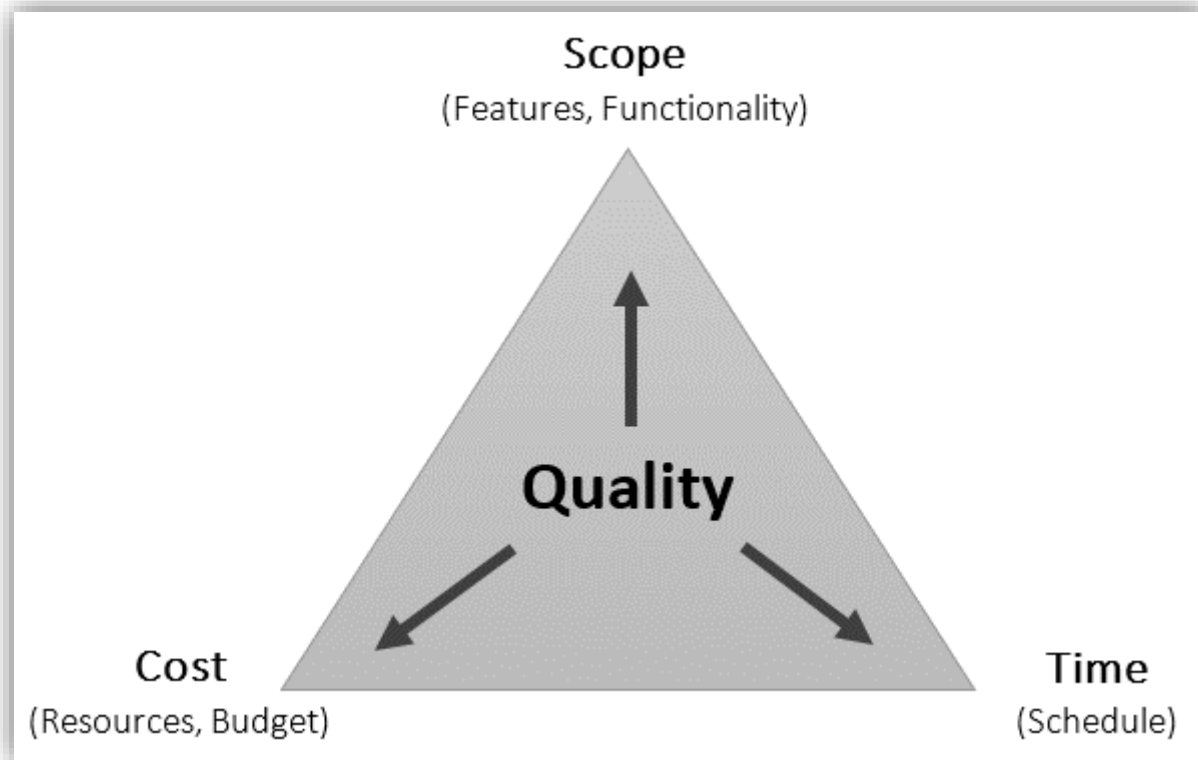


Figure 46: Triple constraints

What is IT and software project?



❖ **Information Technology Project (IT Project) and software project involve using :**

- **Hardware**
- **Software**
- **Networks**

To create a product, service, or result.

Examples of IT and software projects



- A large network of healthcare providers updates its information systems and procedures to reduce hospital acquired diseases.
- A team of students creates a smartphone application and sells it online.
- A company develops a driverless car.
- A college upgrades its technology infrastructure to provide wireless Internet access across the whole campus as well as online access to all academic and student service information.
- A company implements a new system to increase sales force productivity and customer relationship management that will work on various laptops, smartphones, and tablets.
- A television network implements a system to allow viewers to vote for contestants and provide other feedback on programs via social media sites.
- A government group develops a system to track child immunizations.
- A large group of volunteers from organizations throughout the world develops standards for environmentally friendly or green IT.
- A global bank acquires other financial institutions and needs to consolidate systems and procedures.
- Government regulations require monitoring of pollutants in air and water.
- A multinational firm decides to consolidate its information systems into an integrated enterprise resource management approach.

What is a Project Life Cycle (PLC)?



✧ **Project Life Cycle (PLC)** : is a collection of logical stages or phases that maps the life of the project from its beginning to its end in order to define, build, and deliver the product of a project-that is, the information system.

What is a Project Life Cycle (PLC)?



- ❖ **Define project goal:** The project is in the process of getting selected, sponsored, funded, and launched.
- ❖ **Plan project:** determines how the project work will get accomplished.
- ❖ **Execute Project:** The project team does the work.
- ❖ **Closing:** close out the project.
- ❖ **Evaluate the project:** Does the project meet its goal after it has been completed? (Electronic commerce site)

What is a Project Life Cycle (PLC)?

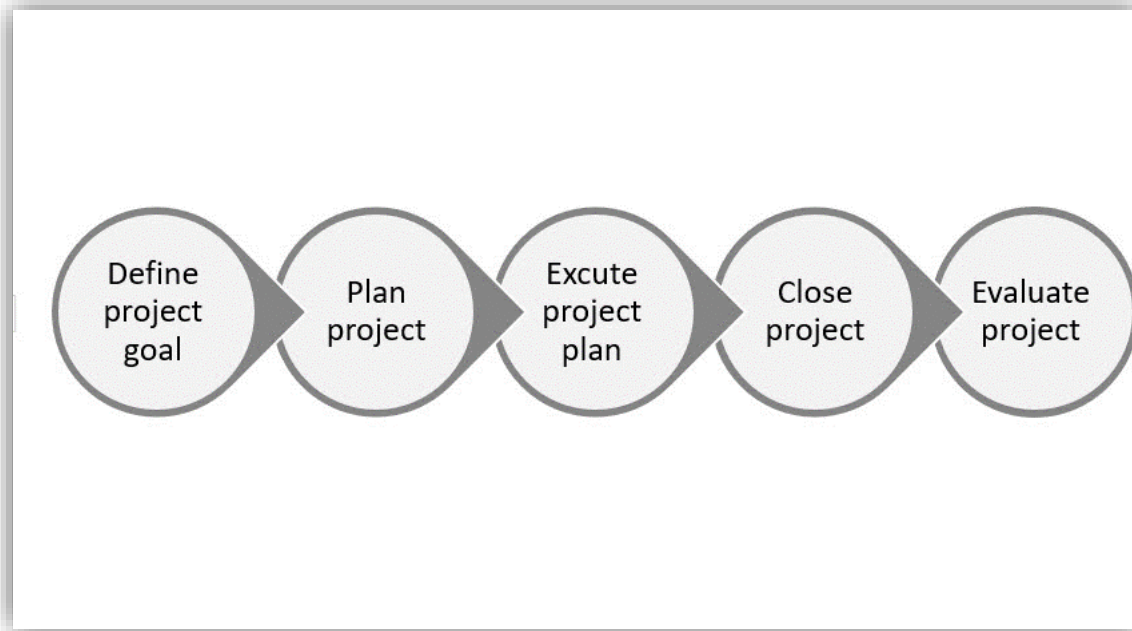


Figure 47: PLC

System Development Life Cycle (SDLC)?

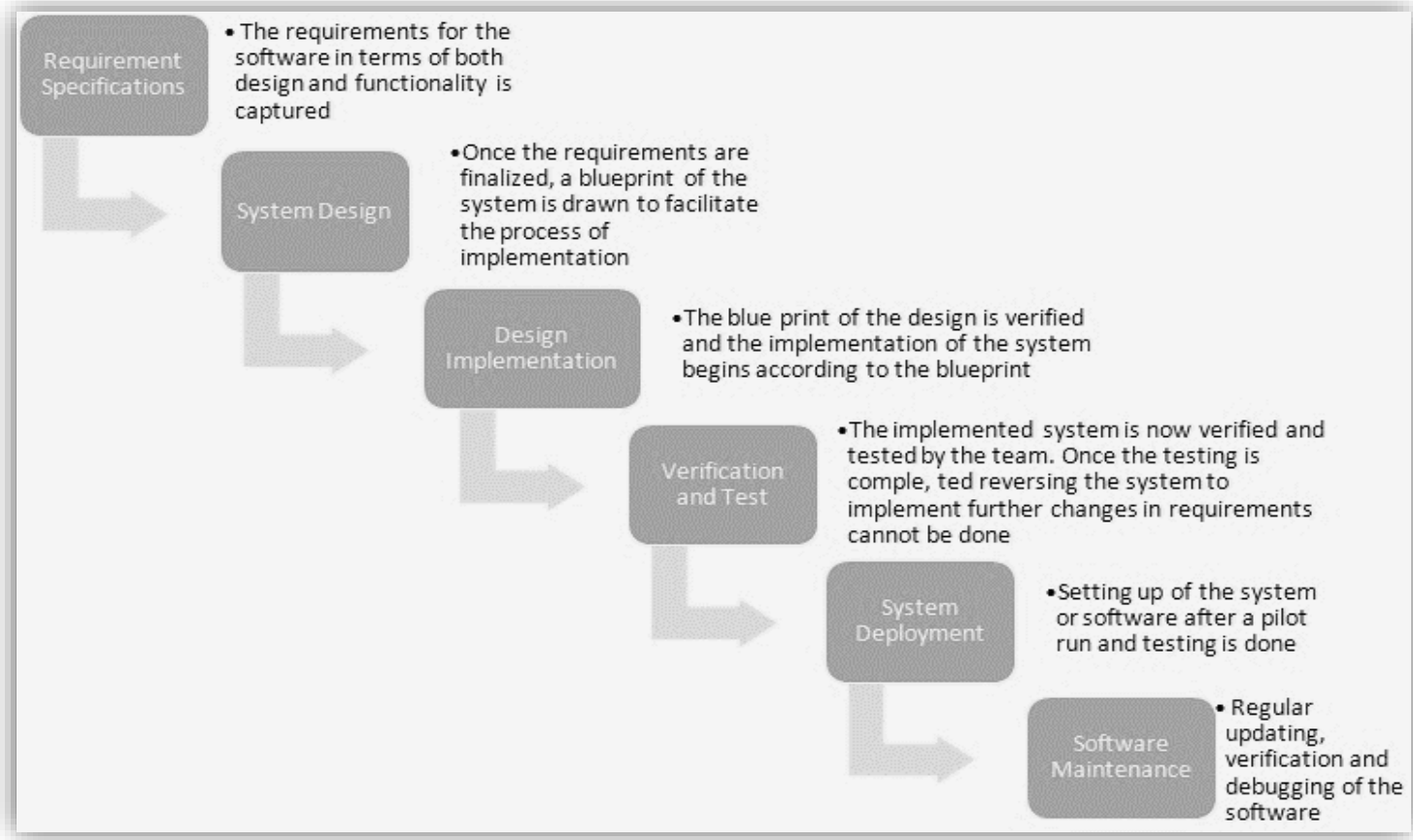


Figure 48 SDLC

System Development Life Cycle (SDLC) and Project Life Cycle (PLC)

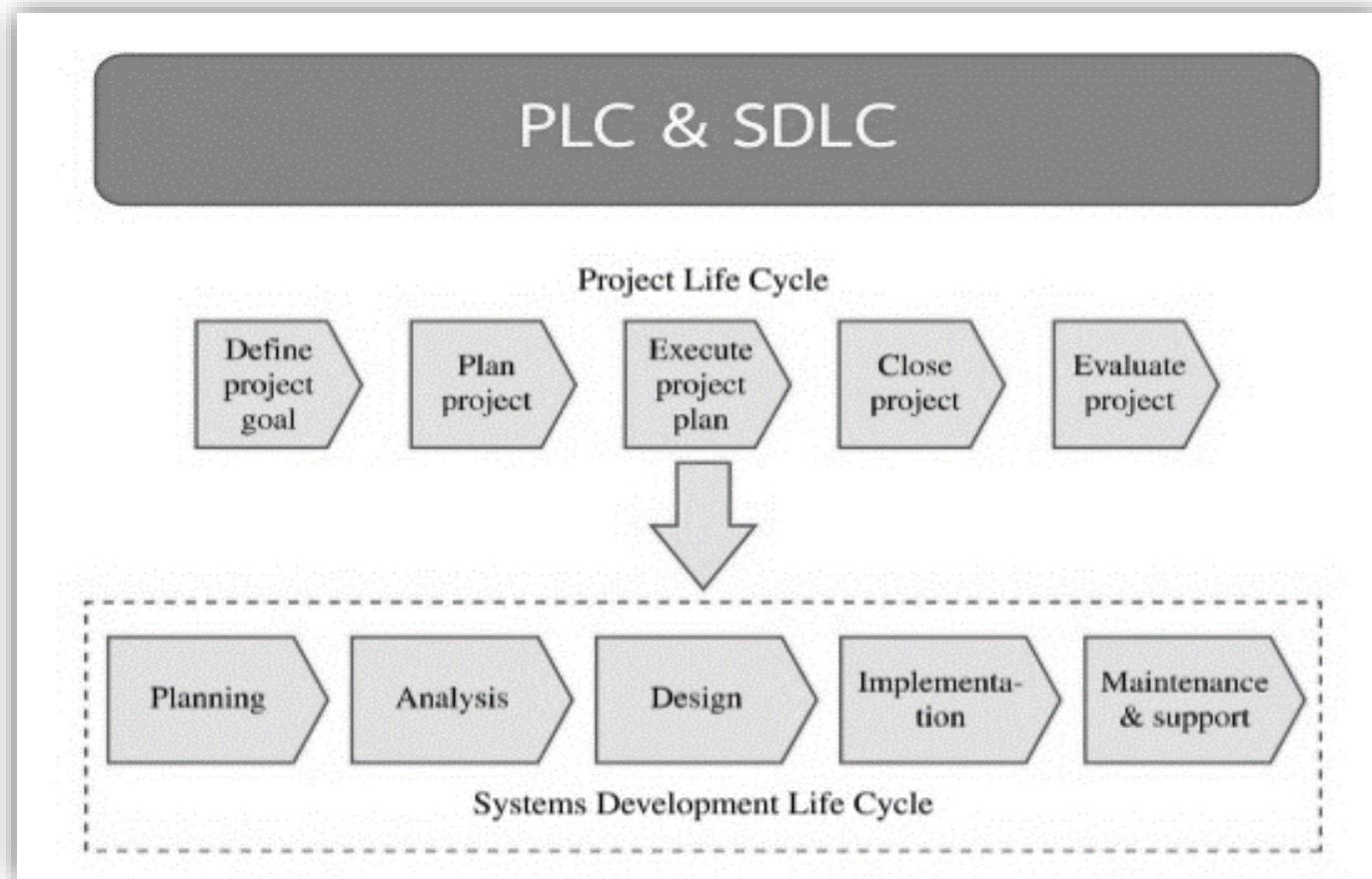


Figure 49 PLC and SDLC



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري

محاضرة الاسبوع الخامس والعشرون

Software Maintenance

Topics covered



- ✧ What is Software Maintenance?
- ✧ Software Maintenance Types.
- ✧ Software Maintenance process model.
- ✧ What are the problems in software maintenance? (Legacy software)?
- ✧ Factors of Software Maintenance.

What is Software Maintenance?



- Software maintenance is becoming an important activity of a large number of software organizations.
- **Software Maintenance:** The process of modifying a software system or component.



Software Maintenance Types



Software Maintenance Types



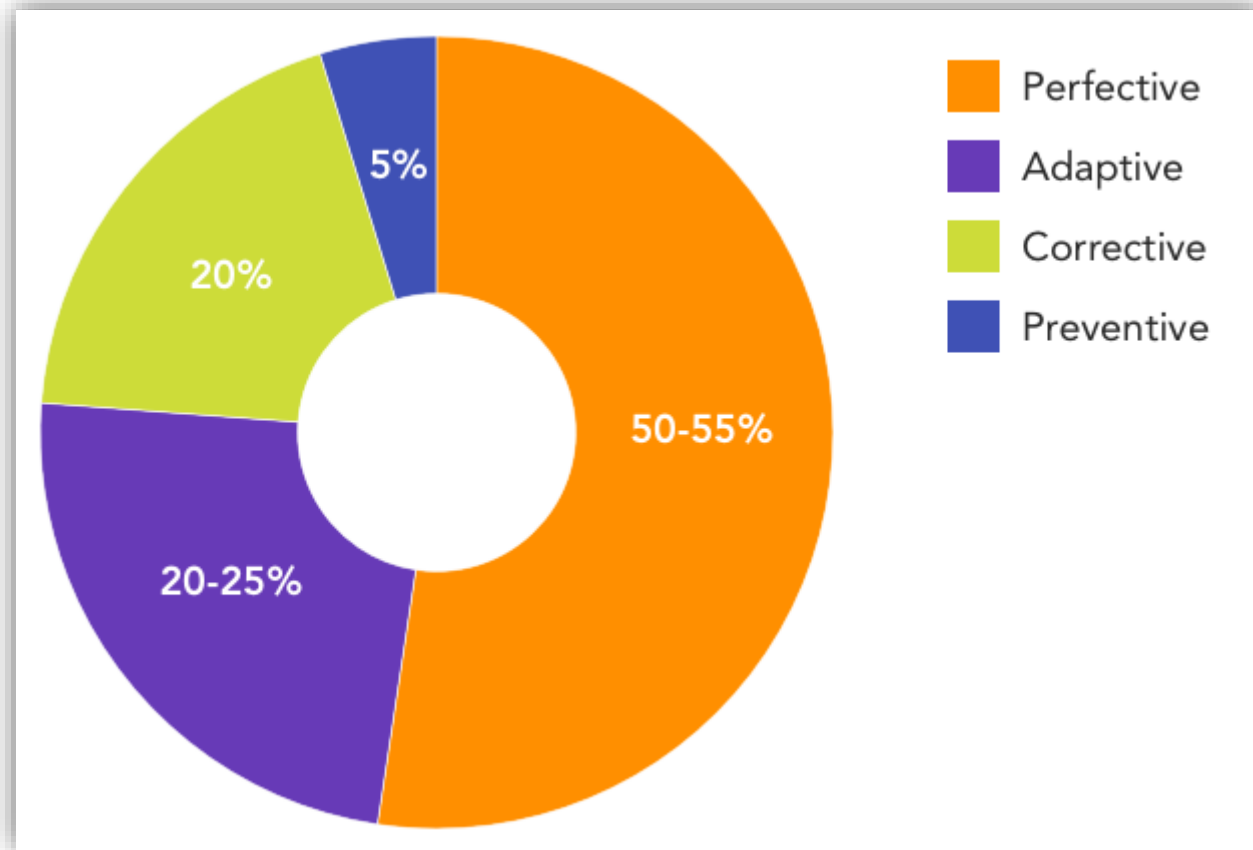
www.educba.com

Software Maintenance Types



- ❖ **Corrective:** is necessary to correct the bugs observed while the system is in use.
- ❖ **Adaptive:** When the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware or software.
- ❖ **Perfective:** A software product needs maintenance to support the new features that users want it to support, to change different functionalities of the system according to customers, or to enhance the performance of the system.
- ❖ **Preventive:** Data restructuring, code restructuring

Software Maintenance Types



Software Maintenance process model



- ✧ Gather Change Requirements
- ✧ Analyze Change Requirements
- ✧ Devise Code Change Strategies
- ✧ Apply Code Change Strategies to the Old Code

Update Documents

Integrate and Test

Software Maintenance process model

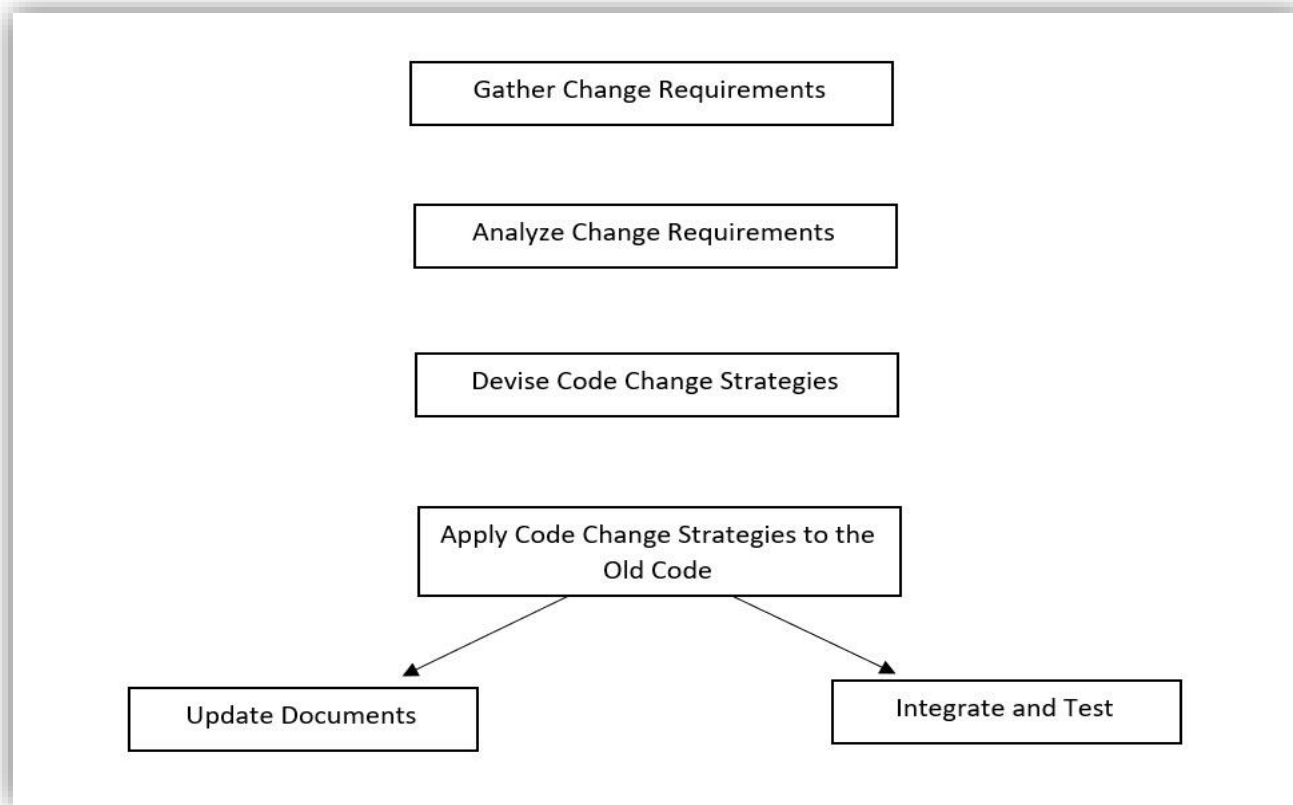


Figure 50 Maintenance process model

What are the problems in software maintenance? (Legacy software)



✧ **Legacy software:** Any software system that is hard to maintain. The typical problems associated with legacy systems are poor documentation, unstructured and lack of personnel knowledgeable in product. Many of the legacy systems were developed long time back. But, it is possible that a recently developed system having poor design and documentation can considered to be a legacy system.



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

م.د. علي يحيى غني



نظري

محاضرة الاسبوع السادس والعشرون

Software Quality Assurance (SQA)

Topics covered



- ✧ What is Quality?
- ✧ What is Software Quality?
- ✧ What is Software Quality Assurance (SQA)?
- ✧ McCall's factors model

Product operation factors

Product revision factors

Product transition factors

What is Quality?



- ✧ **Quality** is 'hard to define, impossible to measure, easy to recognize'(Kitchenham, 1989a).
- ✧ **Quality** is generally transparent when present, but easily recognized in its absence, e.g. when a new car falls to pieces, or a computer program fails to perform properly.
- ✧ **Quality = fitness for purpose, (Juran)**

What is Software Quality?



Kitchenham (1989b) refers to software quality ‘fitness of needs’ and claims quality involves matching expectations.

This definition specifically recognizes the **two features** of a piece of quality software:

- ✧ Is It a good solution?
- ✧ Does it address the right problem?

McCall's factors model



- ✧ The model was first proposed by McCall in 1977. The model is aimed at system developers, to be used during the development process. There are three areas addressed by McCall's model:
 - ✧ Product operation factors
 - ✧ Product revision factors
 - ✧ Product transition factors

McCall's factors model

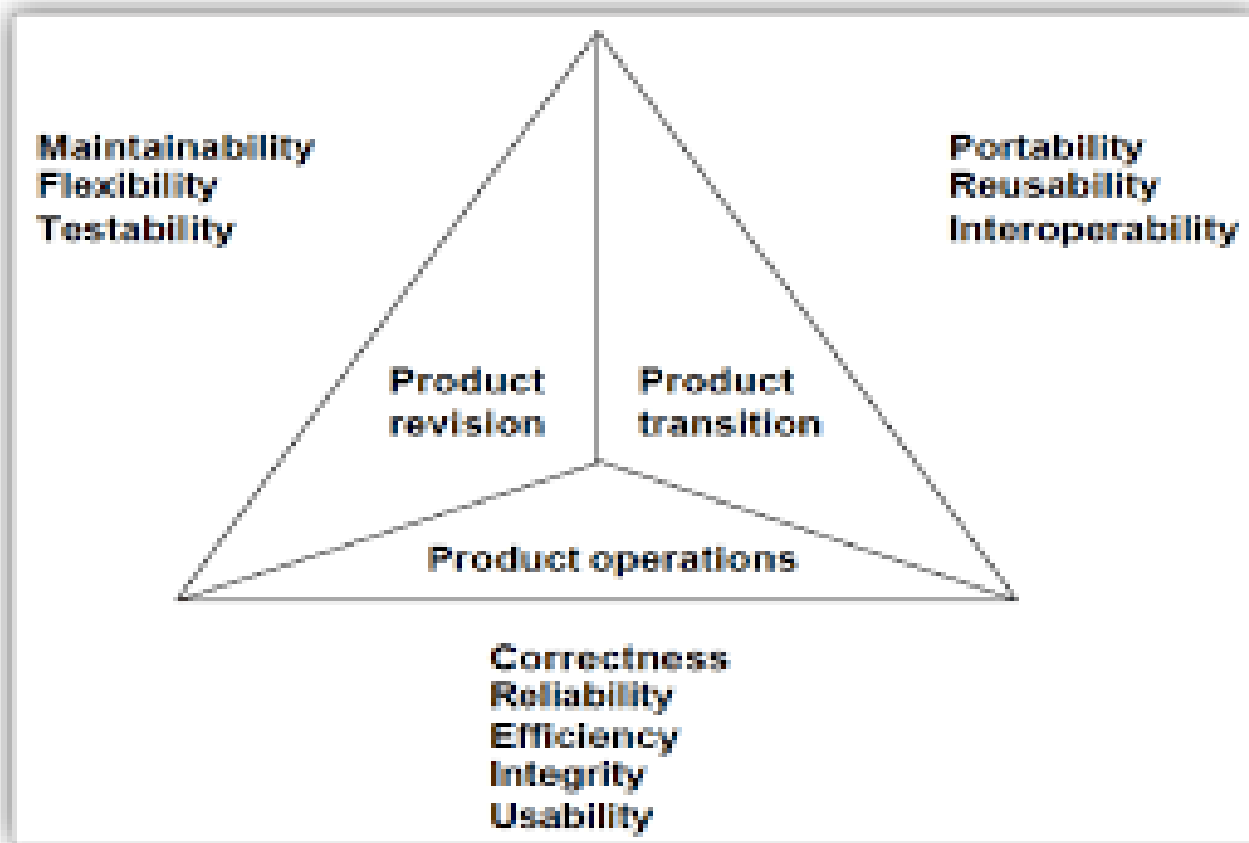


Figure 51 McCall's model

Product operation factors



1. **Correctness:** Does it do what I want?
2. **Reliability:** Does it do it accurately all the time.
3. **Efficiency:** Will it run on my machine as well as it can?
4. **Integrity:** Is it secure.
5. **Usability:** Can I run it?

Product revision factors



- 1. Maintainability:** Can I fix it?
- 2. Flexibility:** Can I change it?
- 3. Testability:** Can I test it?

Product transition factors



- 1. *Portability:*** Will I able to use it on another machine?
- 2. *Reusability:*** Will I be able to reuse some of the software?
- 3. *Interoperability:*** Will I able to interface it with another machine?

McCall's factors model

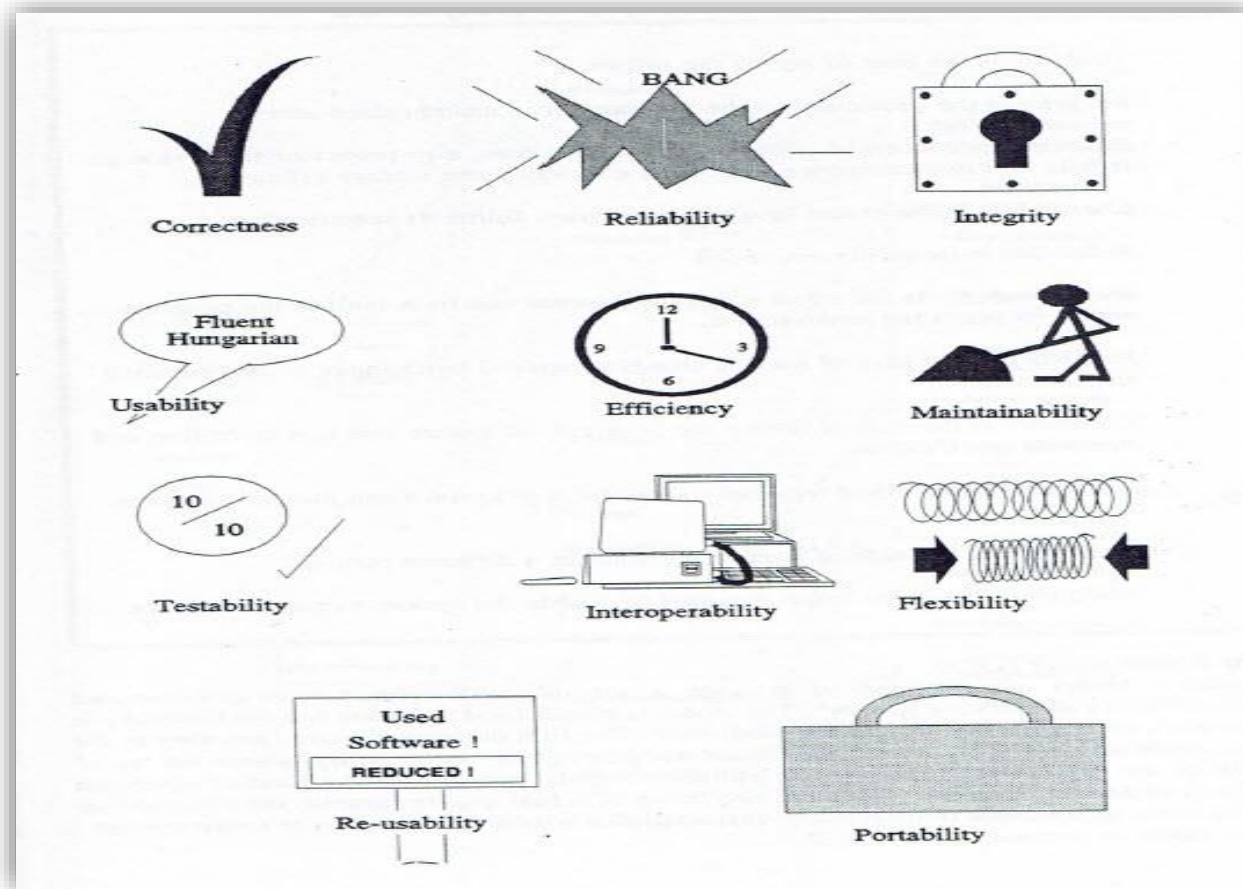


Figure 51 McCall's model



Thanks



Software Engineering

هندسة البرمجيات

جامعة بغداد

كلية التربية للعلوم الصرفة/ابن

الهيثم

قسم علوم الحاسبات

المرحلة الثالثة

أ.م.د. علي يحيى غني



نظري

محاضرة الاسبوع السابع والعشرون

Unified Modeling Language (UML)

Topics covered



- ✧ What is Unified Modelling Language (UML)?
- ✧ Why UML is an important?
- ✧ Review kinds of Diagrams (models)
 1. Use Case Diagrams
 2. Activity Diagrams
 3. Class Diagrams
 4. Sequence Diagrams
 5. Collaboration Diagrams
 6. Component Diagrams

What is Unified Modelling Language (UML)?



- ✧ **Unified Modelling Language (UML)**: is a language just as sure as, Pascal, C# (C sharp), German, English, and Latin are languages. And the UML is probably one of the newest languages invented by humankind, invented around 1997.
- ✧ **The UML** is an official of a pictural language where there are common symbols and relational that have one common meaning.

Why UML is an important?



- ✧ A relation person might ask: Why then, if **Bill Gates** is making billions writing software without a significant emphasis on formal modeling, should I care about UML? **The answer is that 80% of all software project fail.** These projects exceed their **budget, don't provide the features customers need or desire** or worse, are **never delivered.**
- ✧ So, in this case more time need needs to be spent on software analysis and design, and this means **diagrams (models).**

1. Use Case Diagrams



Use Case Diagrams are the equivalent of modern cave art. A use case's main symbols are the actor and the use case oval.

2. Activity Diagrams



An activity diagram is the UML version of a flowchart. Activity diagrams are used to analyze processes and, if necessary, perform process reengineering.

3. Class Diagrams



Class Diagrams are used to show the classes in a system and the relationship between those classes.

4. Sequence Diagrams



Sequence diagrams shows the classes along the top and messages sent between those classes, modeling a single flow through the objects in the system.

5. Collaboration Diagrams



Collaboration diagrams use the same classes and messages but are organized in a spatial display.

6. Component Diagrams



Component Diagrams an example of implementation model.

Diagram



<https://app.diagrams.net/>



Thanks