Software Engineering

University of Technology

Computer Science Department

Software Engineering 1st Class

Lecturer Farah Tawfiq

Lecture (1)
An Introduction To Software Engineering

1.1 The Computer Software

It is the product that software engineers design and build. It encompasses programs that execute within a computer of any size and architecture, documents that encompass hard-copy and virtual forms, and data that combine numbers and text but also includes representations of pictorial, video, and audio information.

The software might take the following forms:

1. Instructions: Computer programs, that when executed provide desired function and performance.

2. Data structured: That enable the programs to adequately manipulate information.

3. Documents: That describes the operation and use of programs.

An Introduction To Software Engineering

1.2 Software Engineering

As software engineers, we use our knowledge of computer-and computing to help solve problems. Often the problem with which we are dealing is related to a computer or an existing computer system, but sometimes the difficulties underlying the problem have nothing to do with computers. Therefore, it is essential that we first understand the nature of the problem.

We must solve the problem first. Then, if need be, we can use technology as a tool to implement our solution.

An Introduction To Software Engineering

Solving problems:

Most problems are large and sometimes tricky to handle, especially if they represent something new that has never been solved before. So we must begin investigating it by :

a) **Analyzing:** Breaking the problem into pieces that we can understand and try to deal with. We can thus describe the larger problem as a collection of small problems and their interrelationships.

- **b) Synthesis:** construct our solution from components that address the problem's various aspects, (putting together of a large structure from small building blocks).

An Introduction To Software Engineering

1.3 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.

An Introduction To Software Engineering

1.4 The Evolving Role of Software

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product.

1- As a product: it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is information transformer (producing, managing, acquiring, modifying, displaying, or transmitting) information that can be as simple as a single bit or as complex as a multimedia presentation.

2- As the vehicle used to deliver the product: software acts as the basis for the :

a. control of the computer (operating systems).

b. The communication of information (networks).

c. The creation and control of other programs (software tools and environments)

An Introduction To Software Engineering

. The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage

capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems.

University of Technology
Computer science Department

Software Engineering

Lecturer : Farah Tawfiq

Class: First

Branch :Software & Information System

2nd course 2020

Software Characteristics

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. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

Software Characteristics

2. Software doesn't "wear out."

The hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative affects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.

Software Characteristics

- Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected (ideally, without introducing other errors) However software doesn't wear out. But it does deteriorate! During its life, software will undergo change (maintenance). As changes are made, it is likely that some new defects will be introduced, causing the failure rate to spike when another change is requested the minimum failure rate level begins to risethe software is deteriorating due to change.

Software Characteristics

- 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.

Software Characteristics

- **Although the industry is moving toward component-based assembly, most 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. 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. The data structure and processing detail required to build the interface are contained with a library of reusable components for interface construction.

Software 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) process complex, but determinate, information structures. Other systems applications (e.g., operating system components, drivers, telecommunications processors) process largely indeterminate data.

2. Real-time software: Software that monitors/analyzes/controls real world events as they occur is called real time. Real-time differs from “interactive” or “time sharing“. A real-time system must respond within strict time constraints. The response time of an interactive (or time sharing) system can normally be exceeded without results.

Software Applications

3. Business software: Business information processing is the largest single software application area. Discrete "systems" (e.g., payroll, accounts receivable/payable, inventory).

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 and even system software characteristics.

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 nonnumerical 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.

The Attributes of Good Software

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

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.

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 userinterface and adequate documentation.

University of Technology
Computer science Department

Software Engineering

Lecturer : Farah Tawfiq

Class: First

Branch :Software & Information System

2nd course 2020

Third lecture

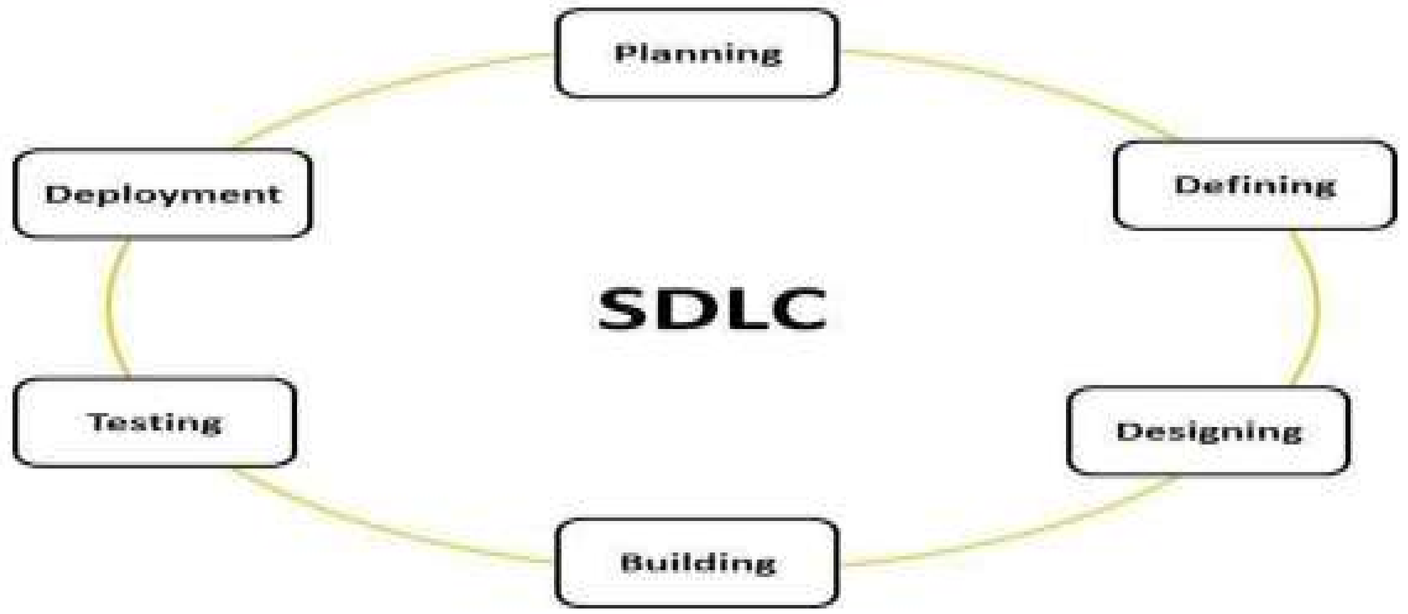
Software system development life cycle (SDLC)

Software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process. Any software development process is divided in several logical stages that allow a software company to organize its work efficiently in order to build a software product of the required functionality within a specific time frame and budget.

Phases involved in SDLC Model

- The phases that are generally present in each and every software development life cycle model are
- 1) Planning and Requirement Analysis
- 2) Defining Requirement
- 3) Designing the product architecture
- 4) Building or developing the product
- 5) Testing the product
- 6) Deployment in the market and maintenance.

A graphical representation of the various stages of a typical SDLC.



Explanation of the phases

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas. Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through .SRS. . Software Requirement Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the product architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification. This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity , budget and time constraints , the best design approach is selected for the product.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle. Developers have to follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers etc are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However this stage refers to the testing only stage of the product where products defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometime product deployment happens in stages as per the organizations. business strategy. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing). Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

Homework

What is the Website Development Life Cycle ?,explain .

University of Technology
Computer science Department

Software Engineering

Lecturer : Farah Tawfiq

Class: First

Branch :Software & Information System

2nd course 2020

Fourth lecture

SDLC Models

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also known as "Software Development Process Models".

Each process model follows a particular life cycle in order to ensure success in process of software development.

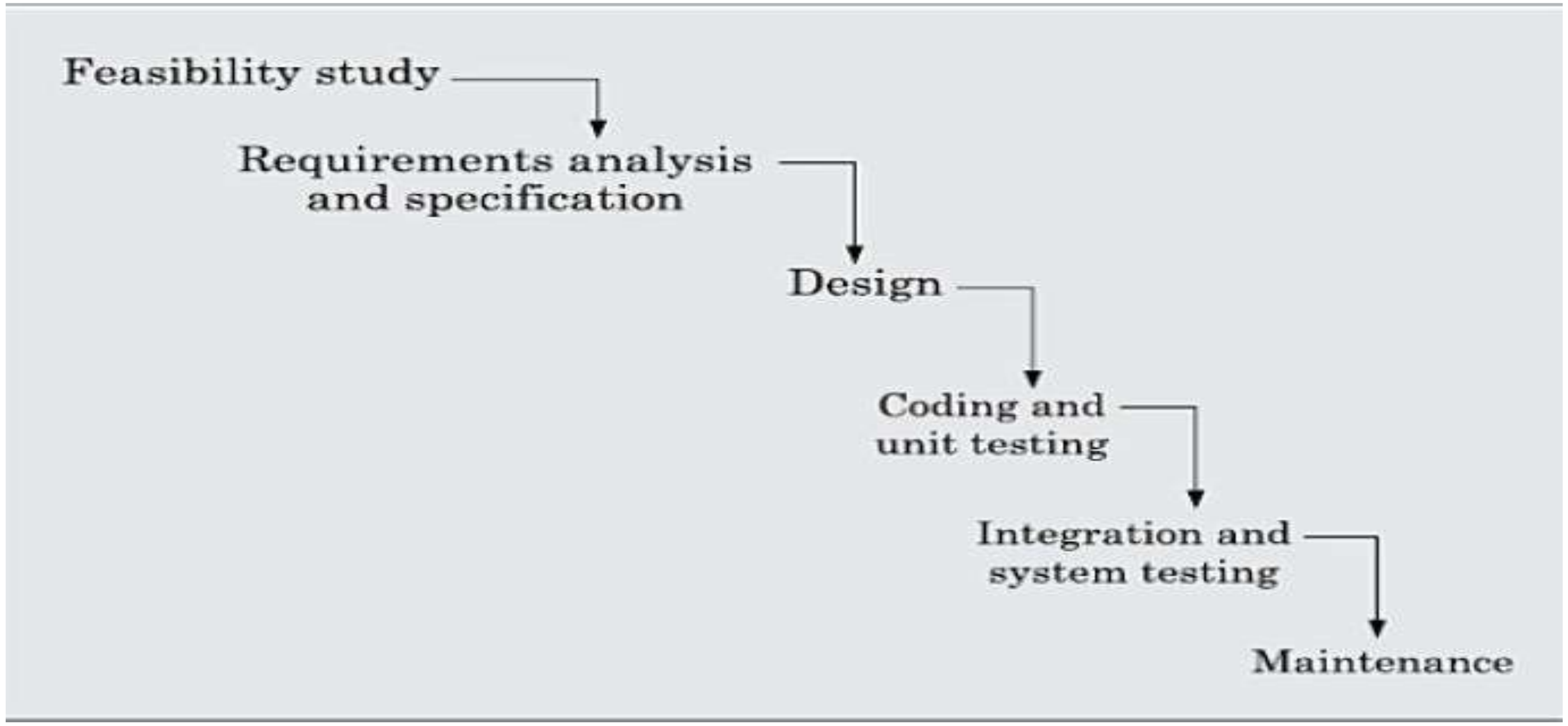
The Waterfall Model

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:

The Waterfall Model

The following figure represent the phases of Waterfall model


Feasibility study
Requirements analysis and specification
Design
Coding and unit testing
Integration and system testing
Maintenance

1: Feasibility study

The main focus of the feasibility study stage is to determine whether it would be *financially* and *technically feasible* to develop the software. The feasibility study involves carrying out several activities such as collection of basic information relating to the software such as the different data items that would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system, as well as various constraints on the development.

2: **Requirements analysis and specification**

The aim of the requirements analysis and specification phase is to Understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely requirements gathering and analysis, and requirements specification.

- **Requirements gathering and analysis:** The goal of the requirements gathering activity is to collect all relevant information regarding the software to be developed from the customer with a view to clearly understand the requirements.
- **Requirements specification:** After the requirement gathering and analysis activities are complete, the identified requirements are documented. This is called a *software requirements specification* (SRS) document.

3: **Design**

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the *software architecture* is derived from the SRS document.

4: **Coding and unit testing**

- The purpose of the coding and unit testing phase is to translate a software design into source code and to ensure that individually each function is working correctly. The coding phase is also sometimes called the *implementation phase*, since the design is implemented into workable solution in this phase.

5: **Integration and system testing**

Integration of different modules is undertaken soon after they have been coded and unit tested. During the integration and system testing phase, the different modules are integrated in a planned manner. Various modules making up a software are almost never integrated in one shot (can you guess

the reason for this?). Integration of various modules are normally carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and

the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained. System testing is carried out on this fully working system.

6: **Maintenance**

The total effort spent on maintenance of a typical software during its operation phase is much more than that required for developing the software itself. Many studies carried out in the past confirm this and indicate that the ratio of relative effort of developing a typical software product and the total effort spent on its maintenance is roughly 40:60. Maintenance is required in the following three types of situations:

- **Corrective maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
- **Perfective maintenance:** This type of maintenance is carried out to improve the performance of the system, or to enhance the functionalities of the system based on customer's requests.
- **Adaptive maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system.

The Prototype Model

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

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

2. The developer may be unsure of the efficiency of an algorithm, the

adaptability of an operating system, or the form that human/machine interaction should take. In these, and many other situations, a prototyping paradigm may offer the best approach.

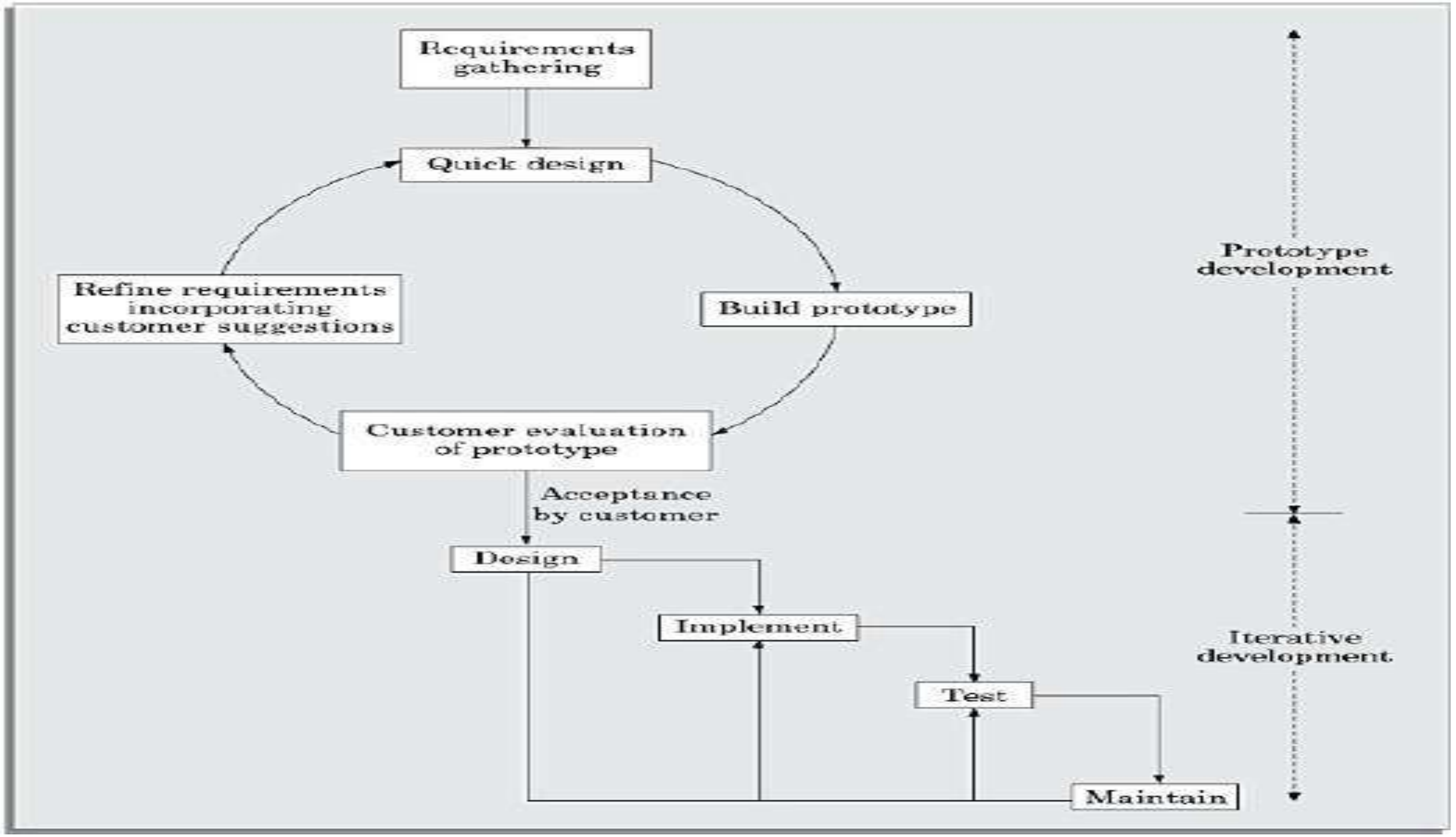
The stages of the prototyping model

1) Requirements gathering (listen to customer): developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory(الزامي) . A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g., input approaches and output formats).

2) Construction of a prototype (build): writing the software code depending on the quick design information.

3) Evaluation (customer test): the prototype is evaluated by the customer/user and used to refine (صقل)requirements for the software to be developed.

Prototyping model of software development



University of Technology
Computer science Department

Software Engineering

Lecturer : Farah Tawfiq

Class: First

Branch :Software & Information System

2nd course 2020

Fifth lecture

Incremental Development Model

This life cycle model is sometimes referred to as the *successive versions* model and sometimes as the incremental model. In this life cycle model, first a simple working system implementing only a few basic features is built and delivered to the customer. Over many successive iterations successive versions are implemented and delivered to the customer until the desired system is realized. The incremental development model has been shown in Figure (1)

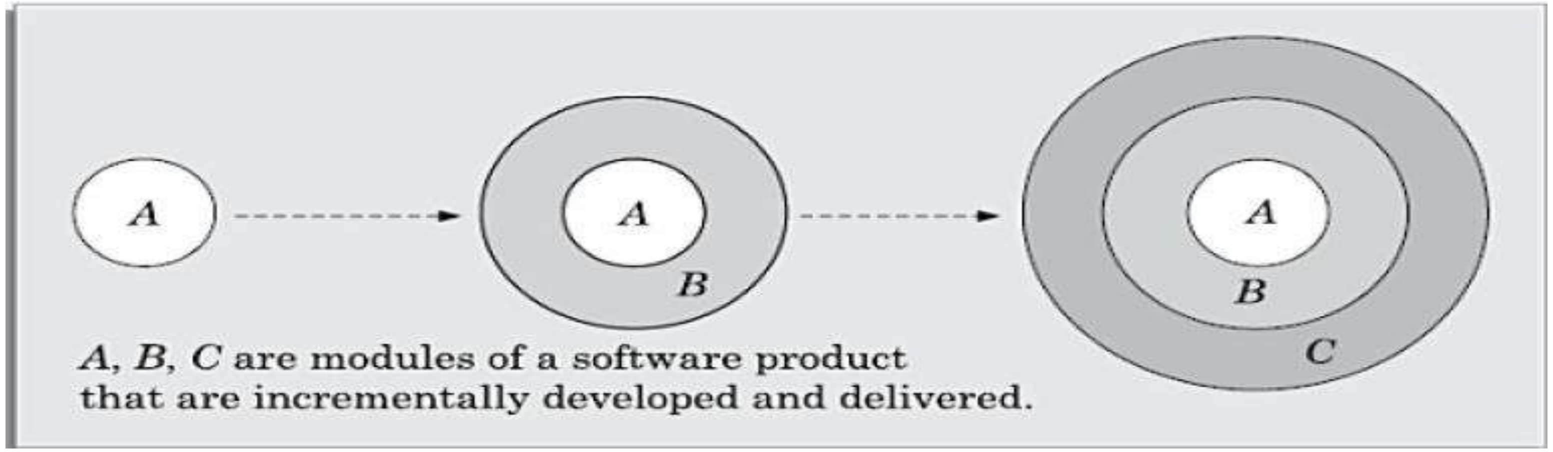

A
A
B
A
B
C
A, B, C are modules of a software product
that are incrementally developed and delivered.

Fig.(1) incremental development model

Life cycle activities of incremental development model

- In this model, the requirements of the software are first broken down into several modules or features that can be incrementally constructed and delivered At any time, plan is made only for the next increment and no long-term plans a r e made. Therefore, it becomes easier to accommodate change requests from the customers.
- Once the initial core features are developed, these are refined into increasing levels of capability by adding new functionalities in successive versions.
- As each successive version of the software is constructed and delivered to the customer, the customer feedback is obtained on the delivered version and these feedbacks are incorporated in the next version.
- Each delivered version of the software incorporates additional features over the previous version and also refines the features that were already delivered to the customer.

Incremental model of software development.

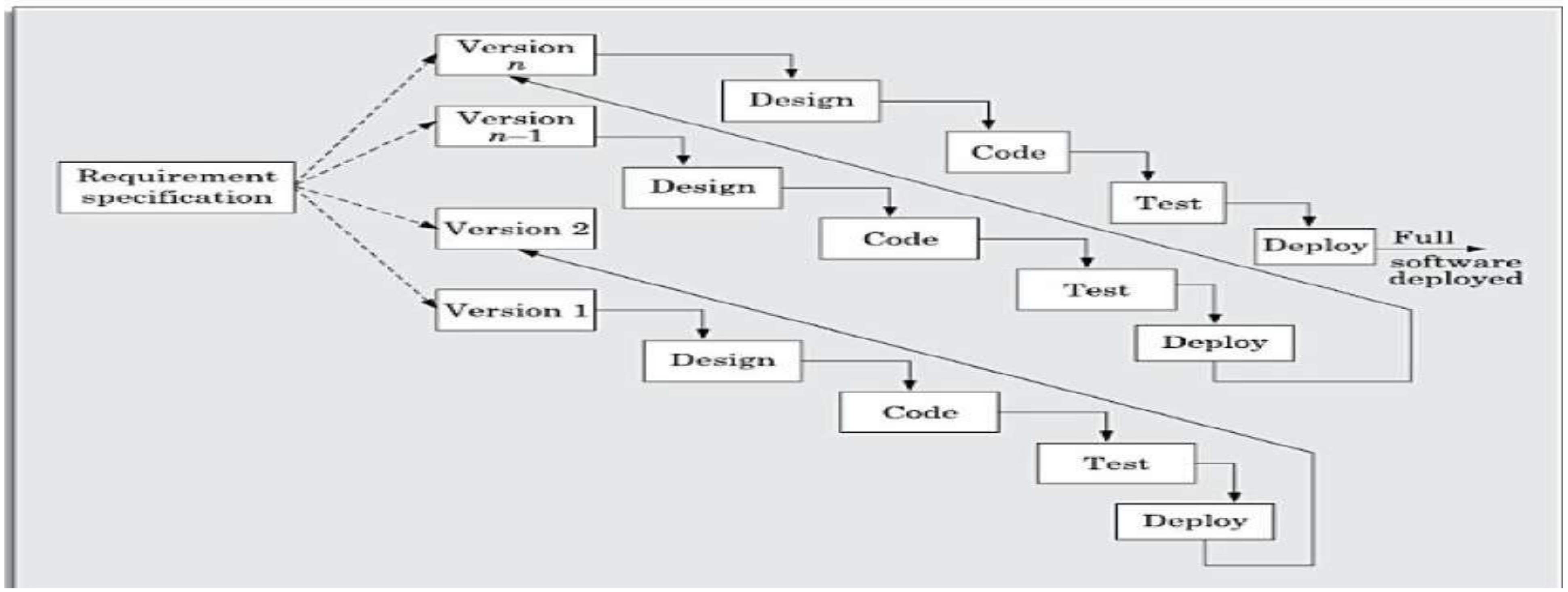


Fig.(2)

Homework

- What are the advantage and disadvantages of Incremental model?

SPIRAL MODEL

- This model gets its name from the appearance of its diagrammatic representation that looks like a spiral with many loops (see Figure 3). The exact number of loops of the spiral is not fixed and can vary from project to project.
- Each loop of the spiral is called a *phase* of the software process.
- A prominent feature (ميزة بارزة)of the spiral model is handling unforeseen risks that can show up much after the project has started. In this context, please recollect that the prototyping model can be used effectively only when the risks in a project can be identified upfront before the development work starts. this model achieves this by incorporating much more flexibility compared to SDLC other models.
- In the spiral model prototypes are built at the start of every phase. Each phase of the model is represented as a loop in its diagrammatic representation. Over each loop, one or more features of the product are elaborated and analysed and the risks at that point of time are identified and are resolved through prototyping. Based on this, the identified features are implemented.

SPIRAL MODEL

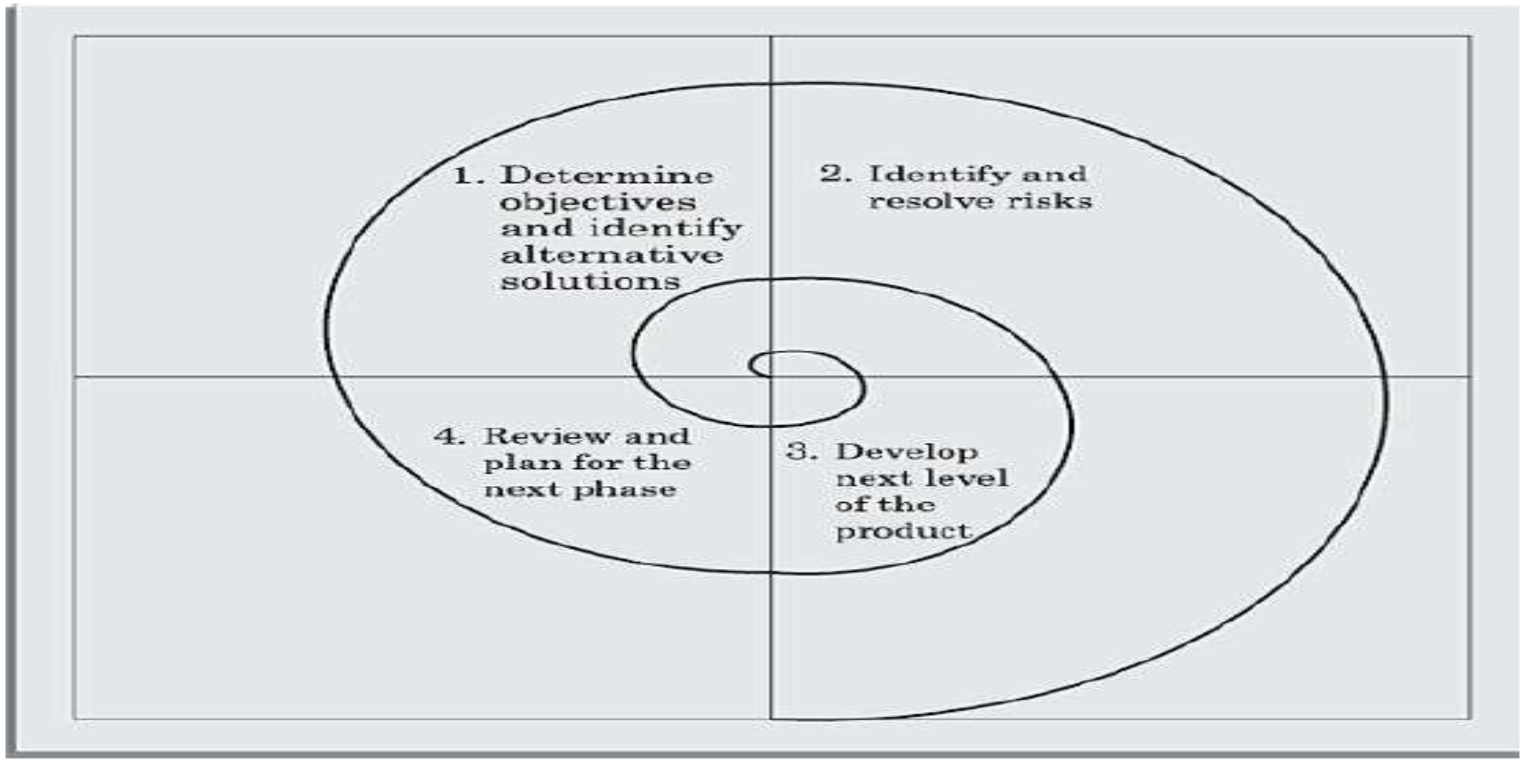

1. Determine objectives and identify alternative solutions
2. Identify and resolve risks
3. Develop next level of the product
4. Review and plan for the next phase

Fig.(3)

Risk handling in spiral model

- A risk is essentially any adverse circumstance that might hamper the successful completion of a software project. As an example, consider a project for which a risk can be that data access from a remote database might be too slow to be acceptable by the customer. This risk can be resolved by building a prototype of the data access subsystem and experimenting with the exact access rate. If the data access rate is too slow, possibly a caching scheme (مخطط التخزين المؤقت)can be implemented or a faster communication scheme can be deployed to overcome the slow data access rate. Such risk resolutions are easier done by using a prototype
- The spiral model supports coping up (التأقلم)with risks by providing the scope to build a prototype at every phase of software development.

evolutionary model

The principal idea behind the evolutionary life cycle model is conveyed by its name. In the incremental development model, complete requirements are first developed and the SRS document prepared. In contrast, in the evolutionary model, the requirements, plan, estimates, and solution evolve over the iterations, rather than fully defined and frozen in a major up-front specification effort before the development iterations begin. Such evolution is consistent with the pattern of unpredictable feature discovery and feature changes that take place in new product development.

Advantages of evolutionary model

The evolutionary model of development has several advantages. Two important advantages of using this model are the following:

1 - Effective elicitation of actual customer requirements: In this model, the user gets a chance to experiment with a partially developed software much before the complete requirements are developed. Therefore, the evolutionary model helps to accurately elicit user requirements with the help of feedback obtained on the delivery of different versions of the software. As a result, the change requests after delivery of the complete software gets substantially reduced.

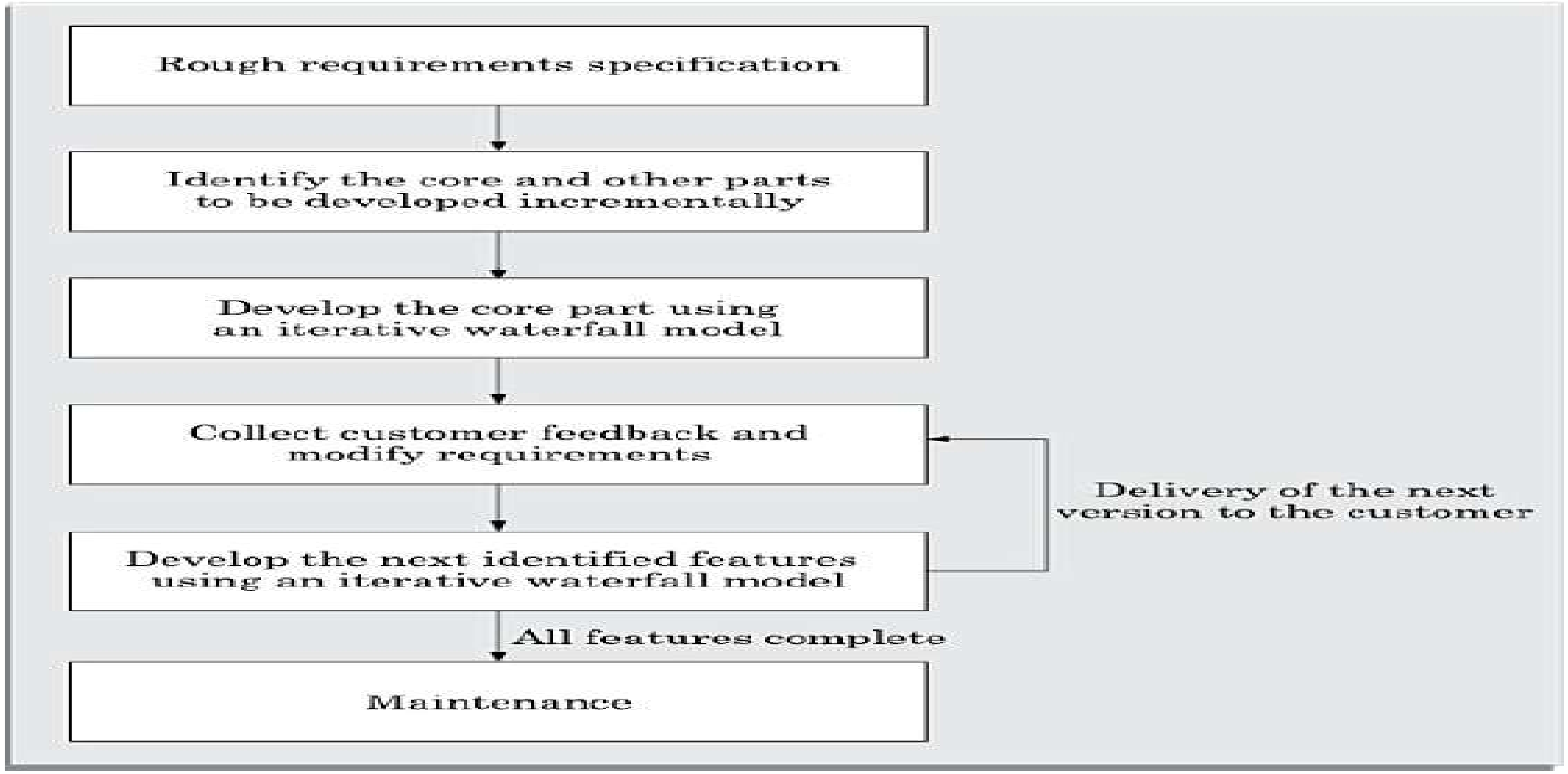
2 - Easy handling change requests: In this model, handling change requests is easier as no long term plans are made. Consequently, reworks required due to change requests are normally much smaller compared to the sequential models.

Disadvantages of evolutionary model

1 - Feature division into incremental parts can be non-trivial: For many development projects, especially for small-sized projects, it is difficult to divide the required features into several parts that can be incrementally implemented and delivered. Further, even for larger problems, often the features are so interwined and dependent on each other that even an expert would need considerable effort to plan the incremental deliveries.

***2 - Ad hoc* design:** Since at a time design for only the current increment is done, the design can become ad hoc without specific attention being paid to maintainability and optimality.

Evolutionary model of software development



Rough requirements specification
Identify the core and other parts to be developed incrementally
Develop the core part using an iterative waterfall model
Collect customer feedback and modify requirements
Delivery of the next version to the customer
Develop the next identified features using an iterative waterfall model
All features complete
Maintenance

Fig.(4)

University of Technology
Computer science Department

Software Engineering

Lecturer : Farah Tawfiq

Class: First

Branch :Software & Information System

2nd course 2020

Sixth lecture

Software Process And Project Metrics

- **Introduction**

Software process and project metrics are quantitative measures that enable software engineers to gain insight into the efficiency of the software process and the projects conducted using the process framework. In software project management, we are primarily concerned with productivity and quality metrics(معايير الانتاجية و الجودة). The four reasons for measuring software processes, products, and resources (to characterizeتوصيف, to evaluate, to predict, and to improve).

Measures, Measurement ,Metrics

- Measure - provides a quantitative indication of the size of some product or process attribute
- Measurement - is the act of obtaining a measure
- Metric - is a quantitative measure of the degree to which a system, component, or process possesses a given attribute

Process and Project Indicators

- Metrics should be collected so that process and product indicators can be ascertained تم التحقق منه
- Process indicators enable software project managers to: assess project status, track potential risks, detect problem area early, adjust workflow or tasks, and evaluate team ability to control product quality

Process Metrics

- Private process metrics are known only to the individual or team concerned.
- Public process metrics enable organizations to make strategic changes to improve the software process.
- Metrics should not be used to evaluate the performance of individuals.
- Statistical software process improvement helps an organization to discover where they are strong and where are weak.

University of Technology
Computer science Department

Software Engineering

Lecturer : Farah Tawfiq

Class: First

Branch :Software & Information System

2nd course 2020

Seventh lecture

Project Metrics

- Software project metrics are used by the software team to adapt project workflow and technical activities.
- Project metrics are used to avoid development schedule delays, to mitigate potential risks, and to assess product quality on an on-going basis.
- Every project should measure its inputs (resources), outputs (deliverables), and results (effectiveness of deliverables).

Product metrics

- Focus on the quality of the product
- Complexity of the design ,includes :

- internal algorithm complexity

-architectural complexity

- data flow complexity

- Code measure (e.g.halstead)
- Measure of process effectiveness

-

Software Measurement

- Direct measures of software engineering process include cost and effort.
- Direct measures of the product include lines of code (LOC), execution speed, memory size, defects per reporting time period.
- Indirect measures examine the quality of the software product itself (e.g. functionality, complexity, efficiency, reliability, maintainability).

Software Quality Metrics

- Factors assessing software quality come from three distinct points of view (product operation, product revision, product modification).
- Software quality factors requiring measures include correctness (defects), maintainability (mean time to change), integrity (threat and security), and usability (easy to learn, easy to use, productivity increase).
- Defect removal efficiency (DRE) is a measure of the filtering ability of the quality assurance and control activities as they are applied throughout the process framework.

Integrating Metrics with Software Process

- Many software developers do not collect measures.
- Without measurement it is impossible to determine whether a process is improving or not.
- Baseline metrics data should be collected from a large, representative sampling of past software projects.
- Getting this historic project data is very difficult, if the previous developers did not collect data in an on-going manner

Statistical Process Control

- It is important to determine whether the metrics collected are statistically valid and not the result of noise in the data.
- Control charts provide a means for determining whether changes in the metrics data are meaningful or not.

Establishing a Software Metrics Program

1. Identify business goal
2. Identify what you want to know
3. Identify subgoals
4. Identify subgoal entities and attributes
5. Formalize measurement goals
6. Identify quantifiable questions and indicators related to subgoals
7. Identify data elements needed to be collected to construct the indicators
8. Define measures to be used and create operational definitions for them
9. Identify actions needed to implement the measures
10. Prepare a plan to implement the measures