Process activities

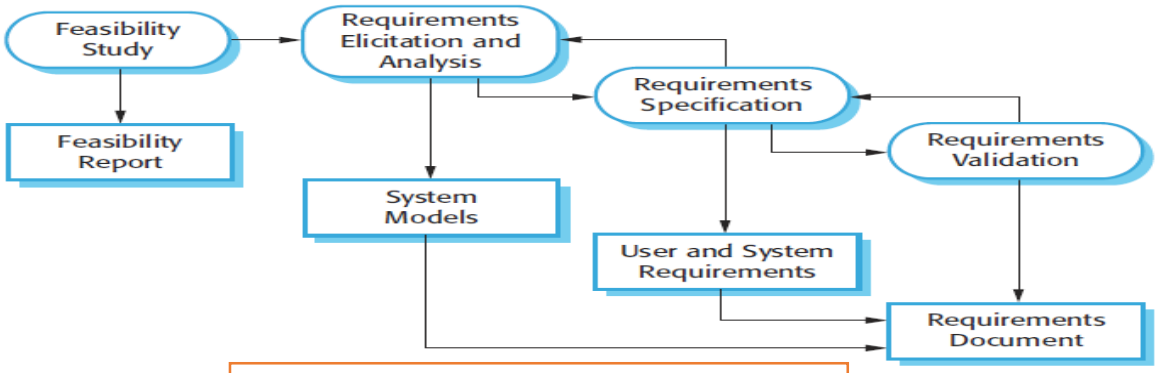
The four basic process activities of specification, development, validation, and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved. How these activities are carried out depends on the type of software, people, and organizational structures involved. There are different ways to organize these activities.

This section is simply to provide for software engineer interest with an introduction to how the activities can be organized.

1. Software specification

Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development. Requirements engineering is a particularly critical stage of the software process as errors at this stage unavoidably lead to later problems in the system design and implementation.

The requirements engineering process aims to produce an agreed requirements document that specifies a system requirements.



The requirements engineering process

There are four main activities in the requirements engineering process:

1. **Feasibility study** دراسة الجدوى: An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more

detailed analysis (feasibility report).

2. **Requirements elicitation استخراجand analysis:** This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and buyer, task analysis. This may involve the development of one or more system models and prototypes. These help the system developer understand the system to be specified.
3. **Requirements specification:** Requirements specification is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; System requirements are a more detailed description of the functionality to be provided.
4. **Requirements validation:** This activity checks the requirements for realismواقعية, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

Of course, the activities in the requirements process are not simply carried out in a strict sequence. Requirements analysis continues during definition and specification and new requirements come to light throughout the process. Therefore, the activities of analysis, definition, and specification are interleaved. In agile methods, such as Extreme Programming, requirements are developed incrementally according to user priorities and the elicitation of requirements comes from users who are part of the development team.

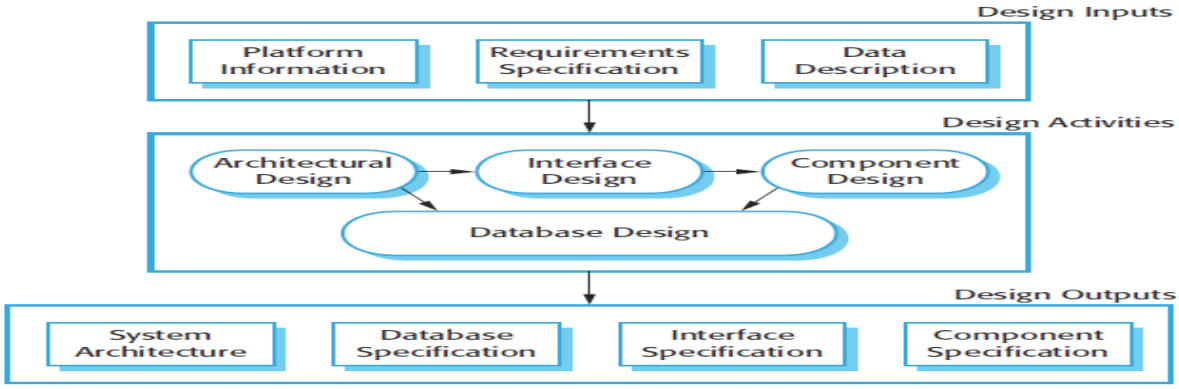
2. Software design and implementation

The implementation stage of software development is the process of converting a system specification into an executable system. It always involves processes of software design and programming.

A ***software design*** is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, the algorithms used.

Designers do not arrive at a finished design immediately but develop the design iteratively. They add formality اجراءات شكليةand detail as they develop their design with constant backtracking to correct earlier designs.

The next Figure is an abstract model of this process showing the inputs to the design process, process activities, and the documents produced as outputs from this process.



A general model of the design process

The diagram suggests that the stages of the design process are sequential. In fact, design process activities are interleaved. This lead to feedback from one stage to another in all design processes, and consequent design rework is inevitable in all design processes.

Most software interacts with other software systems. These include the operating system, database, middleware, and other application systems. These construct the 'software platform', the environment in which the software will execute. Information about this platform is an essential input to the design process, as designers must decide how best to integrate it with the software's environment. The requirements specification is a description of the functionality the software must provide and its performance and dependability requirements. If the system is to process existing data, then the description of that data may be included in the platform specification; otherwise, the data description must be an input to the design process.

The activities in the design process vary, depending on the type of system being developed. For example, real-time systems require timing design but may not include a database so there is no database design involved. **There are four activities that may be part of the design process for information systems as shown in the previous figure:**

1. **Architectural design**, where the software engineer identifies the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships, and how they are distributed.
2. **Interface design**, where the software engineer defines the interfaces between system components. This interface specification must be unambiguous. Once interface specifications are agreed, the components can be designed and developed concurrently.
3. **Component design**, where the software engineer takes each system component and design how it will operate. This may be a simple statement of the expected functionality to be implemented, with the specific design left to the programmer. Alternatively, it may be a list of changes to be made to a reusable component or a detailed design model
4. **Database design**, where the software engineer designs the system data structures and how

these are to be represented in a database. The work here depends on whether an existing database is to be reused or a new database is to be created.

These activities lead to a set of design outputs, which are also shown in previous figure. The detail and representation of these activities are varying considerably. For critical systems, detailed design documents arrange precise descriptions of the system must be produced.

If a model-driven approach is used, these outputs may mostly be diagrams. The structured method for design can be used their. A structured method includes a design process model, notations to represent the design, report formats, rules and design guidelines. The structured method considers a precursor to the UML and object-oriented design. They rely on producing graphical models of the system and, in many cases, automatically generating code from these models.

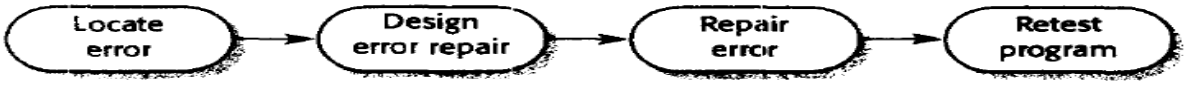
If agile methods of development are used, the outputs of the design process may not be separate specification documents but may be represented in the code of the program. After the system architecture has been designed, later stages of the design are incremental. Each increment is represented as program code rather than as a design model.

The development of a program to implement the system follows naturally from the system design processes. Software development tools may be used to generate a skeleton program from a design. This includes code to define and implement interfaces, and, in many cases, the developer need only add details of the operation of each program component.

Programming is a personal activity and there is no general process that is usually followed. Some programmers start with components that they understand, develop these, and then move on to less-understood components. Others take the opposite approach, leaving familiar components till last because they know how to develop them. Some developers like to define data early in the process then use this to drive the program development; others leave data unspecified for as long as possible.

Normally, programmers carry out some testing of the code they have developed. This often reveals program defects that must be removed from the program. This is called debugging. Defect testing and debugging are different processes. Testing establishes the existence of defects. Debugging is concerned with locating and correcting these defects.

The next figure illustrates the stages of debugging. Defects in the code must be located and the program modified to meet its requirements. Testing must then be repeated to ensure that the change has been made correctly. Thus the debugging process is part of both software development and software testing.

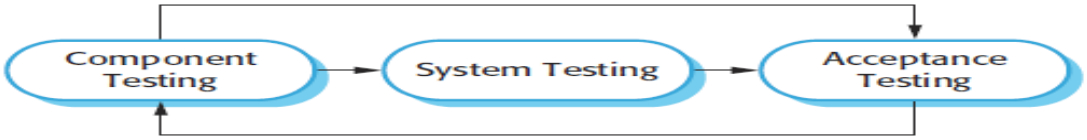


The debugging process

3. Software validation

Software validation or, more generally, verification and validation (V&V) is intended to show that a system both conforms to its specification and that it meets the expectations of the system customer. Program testing, where the system is executed using simulated test data, is the principal validation technique. Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development. The majority of validation costs are incurredتتحقق during and after implementation.

Systems should not be tested as a single, monolithicموحد, unit Except for small programs. The next figure shows a three-stage testing process in which system components are tested then the integrated system is tested and, finally, the system is tested with the customer's data. Ideally, component defectsخلل are discovered early in the process, and interface problems are found when the system is integrated. However, as defects are discovered, the program must be debugged and this may require other stages in the testing process to be repeated. Errors in program components are come to lightتكشف during system testing. The process is therefore an iterative one with information being fed back from later stages to earlier parts of the process.



Stages of testing

The stages in the testing process are:

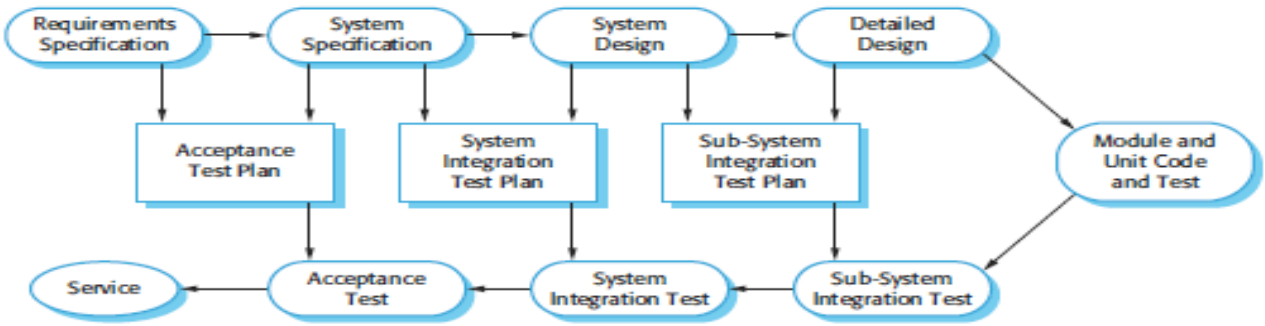
1. **Component (or unite) testing**: Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components. Components may be simple entities such as functions or object classes, or may be coherent groupings of these entities.
2. **System testing:** System components are integrated to create a complete system. This process is concerned with finding errors that result from not expected interactions between components and component interface problems. It is also concerned with showing that the system meets its functional and non-functional requirements, and testing the emergent system properties. For large systems, this may be a multi-stage process where components are integrated to form sub- systems that are individually tested before these sub-systems are themselves integrated to form the final

system.

3. **Acceptance testing:** This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data. Acceptance testing may reveal errors and omissions حذفin the system requirements definition, because the real data exercise the system in different ways from the test data. Acceptance testing may also reveal requirements problems where the system's facilities do not really meet the user's needs or the system performance is unacceptable.

If an incremental approach to development is used, each increment should be tested as it is developed, with these tests based on the requirements for that increment.

When a plan-driven software process is used (e.g., for critical systems development), testing is driven by a set of test plans. An independent team of testers works from these pre-formulated test plans, which have been developed from the system specification and design. The next figure illustrates how test plans are the link between testing and development activities. This is sometimes called the V-model of development.



Testing phases in a plan-driven software process

Acceptance testing is sometimes called 'alpha testing'. Custom systems are developed for a single client. The alpha testing process continues until the system developer and the client agree that the delivered system is an acceptable implementation of the requirements.

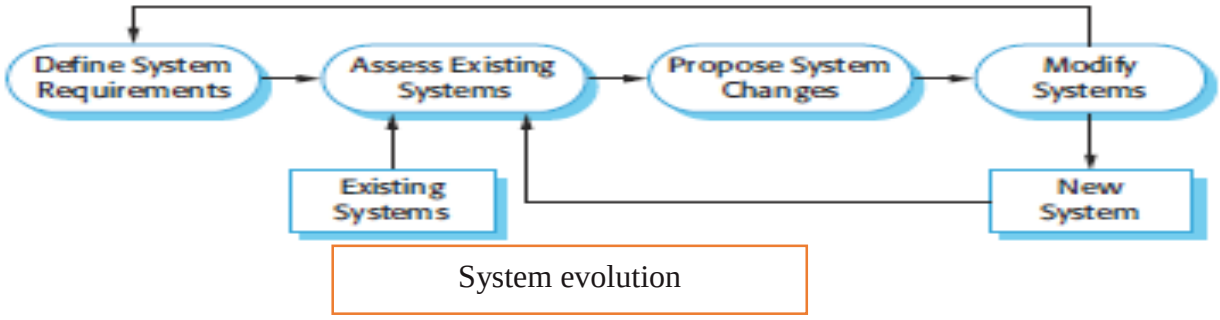
When a system is to be marketed as a software product, a testing process called "beta testing" is often used. Beta testing involves delivering a system to a number of potential customers who agree to use that system. They report problems to the system developers. This exposes the product to real use and detects errors that may not have been expected by the system builders. After this feedback, the system is modified and released either for further beta testing or for general sale.

4. Software evolution

The flexibility of software systems is one of the main reasons why more and more software is being incorporated in large, complex systems. Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design. However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware.

Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance). People think of software development as an activity in which a software system is developed from an initial concept through to a working system. However, they sometimes think of software maintenance as dull (not intelligent) and uninteresting processes. Although the costs of maintenance are often several times the initial development costs, maintenance processes are sometimes considered to be less challenging than original software development.

The development and maintenance are relevant. Few software systems are now completely new systems and it makes much more sense to see development and maintenance as a continuous sequence. Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.



System evolution

Coping with change

Coping with change or sometime called “Process iteration” means that, change is sure to happen in all large software projects. As new technologies become available, new design and implementation possibilities emerge. Therefore whatever software process model is used, it is essential that it can accommodate changes to the software being developed.

Change adds to the costs of software development because it usually means that work that has been completed has to be redone. This is called rework. For example, if the relationships between the requirements in a system have been analyzed and new requirements are then identified, some or all of the requirements analysis has to be repeated. It may then be necessary to redesign the system to deliver the new requirements, change any programs that have been developed, and re-test the system.

There are two related approaches that may be used to reduce the costs of rework:

1. **Change avoidance**, where the software process includes activities that can anticipate possible changes before significant rework is required. For example, a prototype system may be developed to show some key features of the system to customers. They can experiment with the prototype and refine their requirements before committing to high software production costs.
2. **Change tolerance**, where the process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed.