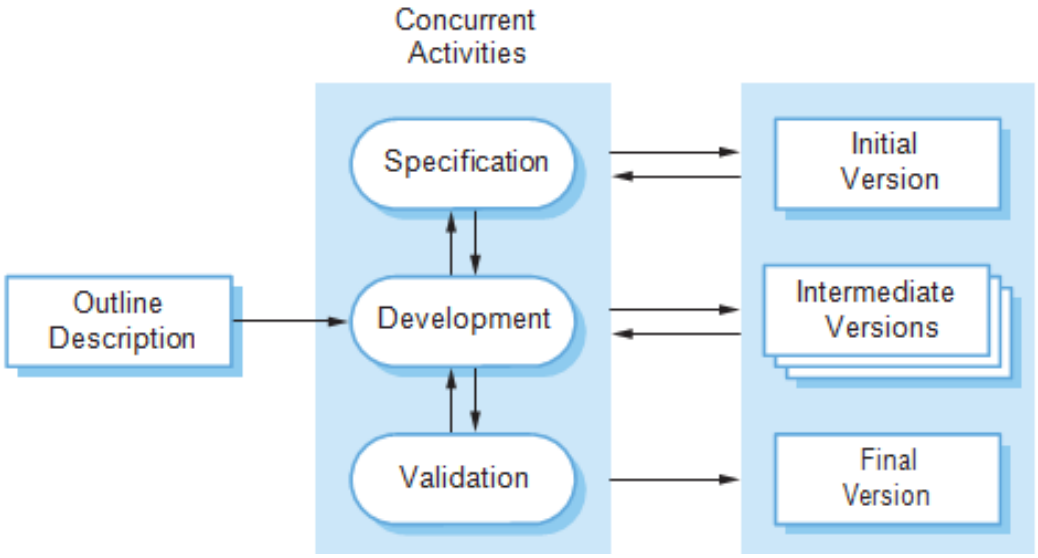
Second. Incremental development model

Incremental development is based on the idea of developing an initial implementation, exposing this عرضه/تقديمهto user comment and evolving it through several versions until an adequate (sufficient, suitable) system has been developed. Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities.



Incremental development model, which is a fundamental part of agile approaches, is better than a waterfall approach for most business, e-commerce, and personal systems. This model reflects the way that how the software engineer solves problems. To solve problem, the solution is divided into a series of steps, backtracking when we realize that we have made a mistake.

Each increment or version of the system incorporates some of the functionality that is needed by the customer. Generally, the early increments of the system include the most important or most urgently required functionality. This means that the customer can evaluate the system at a relatively early stage in the development to see if it delivers what is required. If not, then only the current increment has to be changed and, possibly, new functionality defined for later increments.

Incremental development has three important benefits, compared to the waterfall model:

1. The cost of changing customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
2. It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented. Customers find it difficult to judge progress from software design documents.
3. Rapid delivery and deployment of useful software to the customer is possible, even if all of the functionality has not been included. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development in some form is now the most common approach for the development of application systems. This approach can be either plan-driven, agile, or, more usually, a mixture of these approaches. In a plan-driven approach, the system increments are identified in advance; if an agile approach is adopted, the early increments are identified but the development of later increments depends on progress and customer priorities.

There are two fundamental types of incremental development model:

1. **Exploratory development** نموذج التطوير استطلاعي/استكشافيwhere the objective of this model is to work with the customer to explore their requirements and deliver a final system. The development starts with the parts of the system that are understood. The system evolves by adding new features proposed by the customer.

2. **Throwaway prototyping** نموذج اولي عديم الفائدة where the objective of the development process model is to understand the customer's requirements and hence develop a better requirements definition for the system. The prototype

concentrates on experimenting with the customer requirements that are poorly understood.

The advantage of a software process that is based on an incremental model is that the specification can be developed incrementally.

The incremental approach has two problems:

1. **The process is not visible**: Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
2. **Systems are often poorly structured**: Continual change tends to corrupt the software structure. Incorporating software changes becomes increasingly difficult and costly.

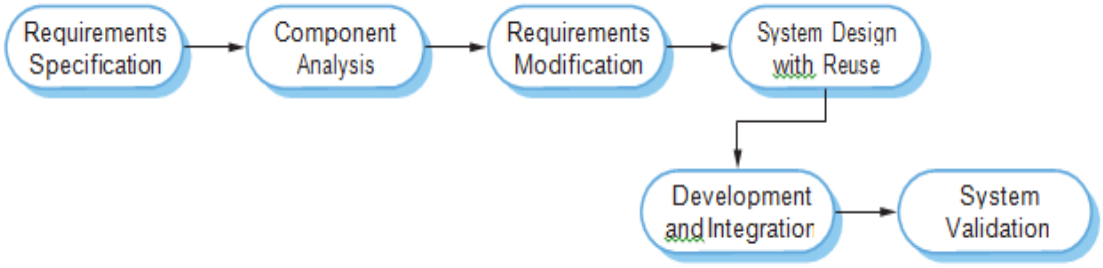
For small and medium-sized systems (up to 500,000 lines of code), the incremental approach is the best approach to development. The problems of this development model become particularly serious for large, complex, long- lifetime systems, where different teams develop different parts of the system. It is difficult to establish a stable system architecture using this approach, which makes it hard to integrate contributions from the teams.

For large systems, a mixed process that incorporates the best features of the waterfall and the incremental development models. This may involve developing a throwaway prototype using an incremental approach to resolve uncertainties/ambiguity in the system specification. The software engineer can then re-implement the system using a more structured approach. Parts of the system that are well understood can be specified and developed using a waterfall-based process. Other parts of the system, such as the user interface, which are difficult to specify in advance, should always be developed using an exploratory programming approach.

Third. Reuse-oriented software engineering

This model considered as the newest software process model. In the majority of software projects, there is some software reuse. This often happens informally when people working on the project know of designs or code that is similar to what is required. They look for these, modify them as needed, and incorporate them into their system.

Reuse-oriented approaches rely on a large base of reusable software components and an integrating framework for the composition of these components. Sometimes, these components are systems in their own right (COTS or commercial off-the-shelf systems) that may provide specific functionality such as word processing or a spreadsheet.



A general process model for reuse-based development is shown in previous Figure. Although the initial requirements specification stage and the validation stage are comparable with other software processes, the intermediate stages in a reuse-oriented process are different. **These stages are:**

1. **Component analysis**: Given the requirements specification, a search is made for components to implement that specification. Usually, there is no exact match and the components that may be used only provide some of the functionality required.
2. **Requirements modification**: During this stage, the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components. Where modifications are

impossible, the component analysis activity may be re-entered to search for alternative solutions.

3. **System design with reuse**: During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and organize the framework to supplies for this. Some new software may have to be designed if reusable components are not available.
4. **Development and integration**: Software that cannot be externally obtained is developed, and the components and COTS systems are integrated to create the new system. System integration, in this model, may be part of the development process rather than a separate activity.

There are three types of software component that may be used in a reuse-oriented process:

1. Web services that are developed according to service standards and which are available for remote request.
2. Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
3. Stand-alone software systems that are configured for use in a particular environment.

Reuse-oriented software engineering has the obvious advantage of reducing the amount of software to be developed and so reducing cost and risks. It usually also leads to faster delivery of the software. However, requirements compromises are inevitable محتوم(sure to happen) and this may lead to a system that does not meet the real needs of users. Furthermore, some control over the system evolution is lost as new versions of the reusable components are not under the control of the organization using them.