Software Engineering: Section 2

SFW Processes

A software process is a set of related activities that leads to the production of a software product.

There are many different software processes but all must include four activities that are fundamental to software engineering:

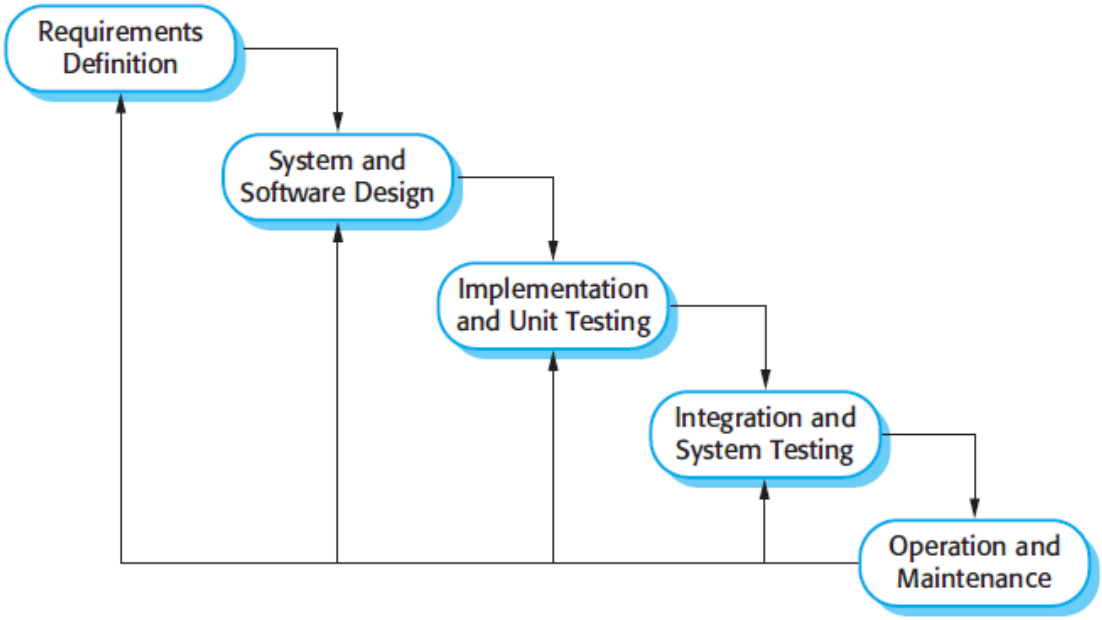
1. **Software specification** The functionality of the software and constraints on its operation must be defined.
2. **Software design and implementation** The software to meet the specification must be produced.
3. **Software validation** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution** The software must evolve to meet changing customer needs.

Software processes are complex and, like all intellectual and creative processes, rely on people making decisions and judgments. There is no ideal process.

Software process model is a simplified representation of a software process. Each process model represents a process from a particular perspective, and thus provides only partial information about that process.These generic models are not definitive descriptions of software processes. Rather, they are abstractions of the process that can be used to explain different approaches to software development.

The waterfall model

This takes the fundamental process activities of specification, development, validation, and evolution and represents them as **separate process phases** such as requirements specification, software design, implementation, testing, and so on.



The waterfall model is an example of a plan-driven process—in principle, you must plan and schedule all of the process activities before starting work on them.

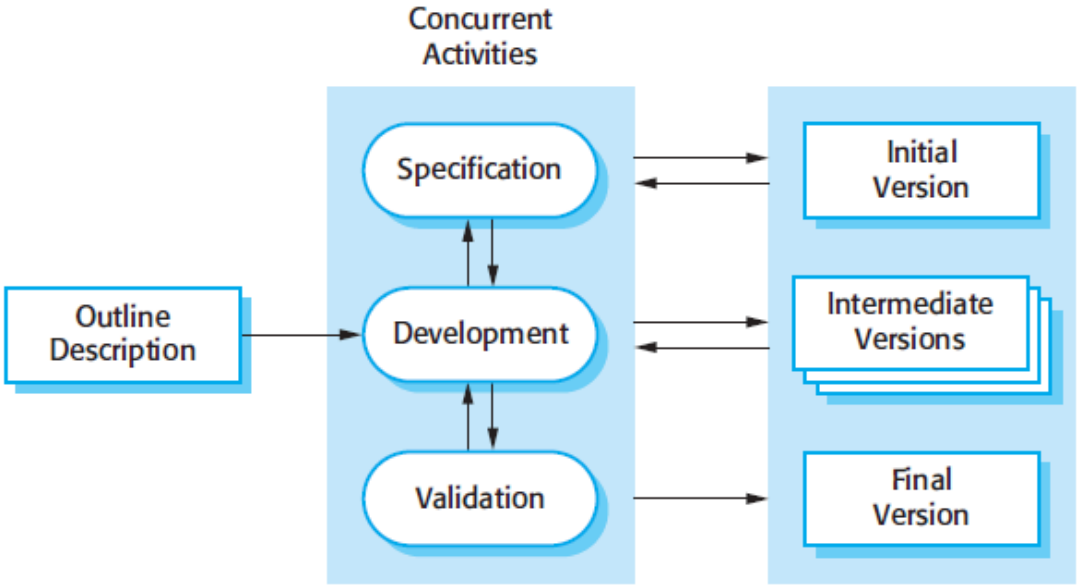
The principal stages of **the waterfall model** directly reflect the fundamental development activities:
1. **Requirements analysis and definition** The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a **system specification.**

2. **System and software design** The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.`
3. **Implementation and unit testing** During this stage, the software design is realized as a set of programs or program units. **Unit testing** involves verifying that each unit meets its specification.
4. **Integration and system testing** The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
5. **Operation and maintenance** Normally (although not necessarily), this is the longest life cycle phase. The system is installed and put into practical use. **Maintenance** involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

In principle, the result of each phase is one or more documents that are approved ('signed off'). The following phase should not start until the previous phase has finished.
In practice, these stages overlap and feed information to each other. During design, problems with requirements are identified. During coding, design problems are found and so on. The software process is not a simple linear model but involves feedback from one phase to another. Documents produced in each phase may then have to be modified to reflect the changes made.

Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an **early stage** in the process, which makes it difficult to respond to changing customer requirements.
In principle, the waterfall model should only be used when the requirements are **well understood and unlikely to change radically during system development.**

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



Incremental software development, which is a fundamental part of agile approaches, is better than a waterfall approach for most business, e-commerce, and personal systems.
Incremental development reflects the way that we solve problems.
We rarely work out a complete problem solution in advance but move toward a solution in a series of steps, backtracking when we realize that we have made a mistake. By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.
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 accommodating 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. **More 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.

Reuse-oriented

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 are similar to what is required. They look for these, modify them as needed, and incorporate them into their system.

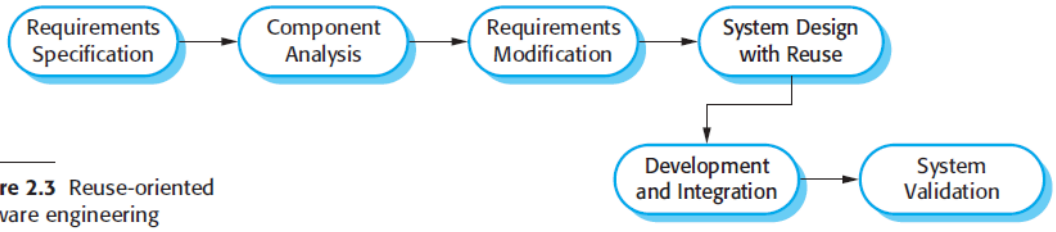


Figure 2.3 Reuse-oriented software engineering

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 cater 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 procured is developed, and the components are integrated to create the new system. System integration, in this model, may be part of the development process rather than a separate activity.

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 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.