(a)

An architectural pattern is a proven structural organization schema for software systems.

Before major development starts, we need to choose an architecture that will provide us with the desired quality attributes. Therefore we need a way to discuss architectural options and their quality consequences in advance, before they can be applied to a design. Our decisions at this stage can only be based on experience with architectural choices in previous systems. Such architectural experience, reduced to its essence and no longer cluttered with the details of the systems that produced it, is recorded in architectural patterns.

An architectural pattern is a description of a set of predefined subsystems and their responsibilities.An architectural pattern also describes rules and guidelines for organizing the relationships among the subsystems.

(b)

**Pipe-filter pattern**:The Pipe-filter architectural pattern provides a structure for systems that produce a stream of data. Each processing step is encapsulated in a filter component. Data is passed through pipes. The pipes may be used for buffering or for synchronization.

Example: Example of the Pipe-filter pattern are Unix shell commands, such as:

cat file | grep xyz | sort | uniq > out

This pattern divides the task of a system into several sequential processing steps. The steps are connected by the data flow through the system: the output of a step is the input for the next step. In the example, the cat filter reads the file and passes the contents of the file to the grep filter. The grep filter selects lines containing xyz, and passes these lines to the sort filter. The sort filter sorts the lines, and passes the sorted lines to the uniq filter. The uniq filter removes duplicate lines, and passes the result to out.

The Pipe-filter pattern has several advantages.

**1) It is easy to add new filters:** a system built according to the pipe-filter pattern is easy to extend.

**2) Filters are reusable:** it is possible to build different pipelines by recombining a given set of filters. Because of the standard interface, filters can easily be developed separately, which is also an advantage.

**3) No files required:**Filters do not need to store intermediate results in files, and need not share state. Input and output can come from, and go to different places.

**4) Concurrent processing:** It shows natural concurrent processing, when input and output consist of streams, and filters start computing when they receive data. Analysis of the behavior of a pipe-filter-based system is easy, because it is a simple composition of the behaviors of the filters involved.

(c)

It shows natural concurrent processing, when input and output consist of streams, and filters start computing when they receive data. Analysis of the behavior of a pipe-filter-based system is easy, because it is a simple composition of the behaviors of the filters involved. When the input is called x, the behavior of the first filter is described by function g, and the behavior of the second filter is described by function f, the result of the pipeline can be described as: f(g(x)) Because of this composition property, it is possible to analyze throughput as well (throughput is determined by the slowest filter), and the possibility of deadlocks. Deadlocks may occur when at least one of the filters needs all data before producing output. In such a case, the size of the buffer may be too small, and the system may deadlock. The Unix sort filter is an example of such a filter