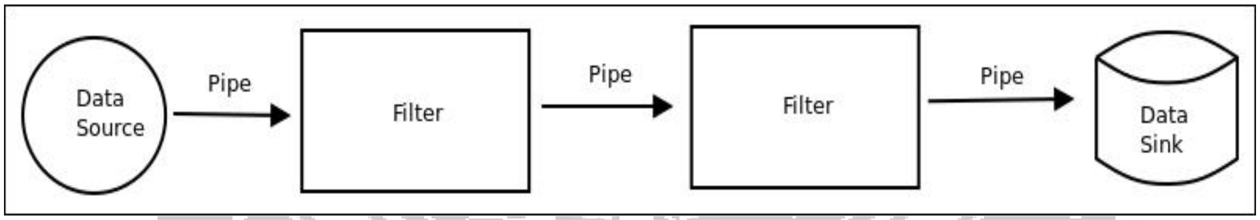
PIPE AND FILTER ARCHITECTURES

Pipe and Filter is a simple architectural style that connects a number of components that process a stream of data, each connected to the next component in the processing pipeline via a **Pipe**.

The Pipe and Filter architecture is inspired by the Unix technique of connecting the output of an application to the input of another via pipes on the shell.

The pipe and filter architecture consists of one or more data sources. The data source is connected to data filters via pipes. Filters process the data they receive, passing them to other filters in the pipeline. The final data is received at a **Data Sink**:



Definition

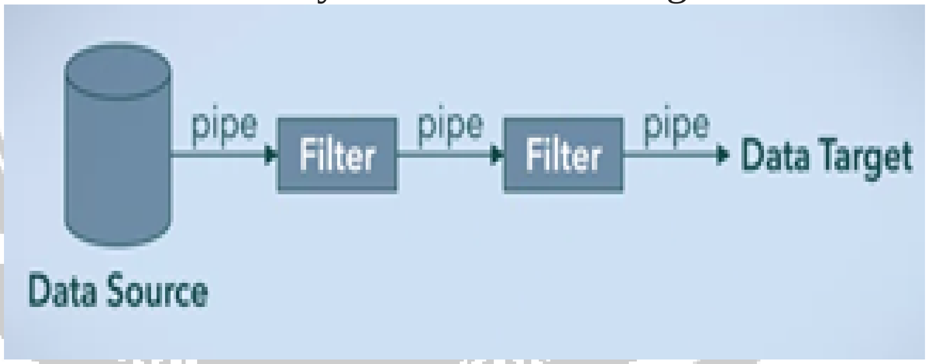
Pipe and Filter is another architectural pattern, which has independent entities called **filters** (components) which perform transformations on data and process the input they receive, and **pipes**, which serve as connectors for the stream of data being transformed, each connected to the next component in the pipeline.

Many systems are required to transform streams of discrete data items, from input to output. Many types of transformations occur repeatedly in practice, and so it is desirable to create these as independent, reusable parts, Filters. (Len Bass, 2012)

Description of the Pattern

The pattern of interaction in the pipe-and-filter pattern is characterized by successive

transformations of streams of data. As you can see in the diagram, the data flows in one direction. It starts at a data source, arrives at a filter's input port(s) where processing is done at the component, and then, is passed via its output port(s) through a pipe to the next filter, and then eventually ends at the data target.



A single filter can consume data from, or produce data to, one or more ports. They can also run concurrently and are not dependent. The output of one filter is the input of another, hence, the order is very important.

A pipe has a single source for its input and a single target for its output. It preserves the sequence of data items, and it does not alter the data passing through.

Advantages of selecting the pipe and filter architecture are as follows:

· Ensures loose and flexible coupling of components, filters.

· Loose coupling allows filters to be changed without modifications to other filters.

· Conductive to parallel processing.

· Filters can be treated as black boxes. Users of the system don't need to know the logic behind the working of each filter.

· Re-usability. Each filter can be called and used over and over again.

However, there are a few drawbacks to this architecture and are discussed below:

· Addition of a large number of independent filters may reduce performance due to excessive computational overheads.

· Not a good choice for an interactive system.

· Pipe-and-fitter systems may not be appropriate for long-running computations.

Applications of the Pattern

In software engineering, a **pipeline** consists of a chain of processing elements (processes, threads, functions, etc.), arranged so that the output of each element is the input of the next. (Wiki, n.d.).

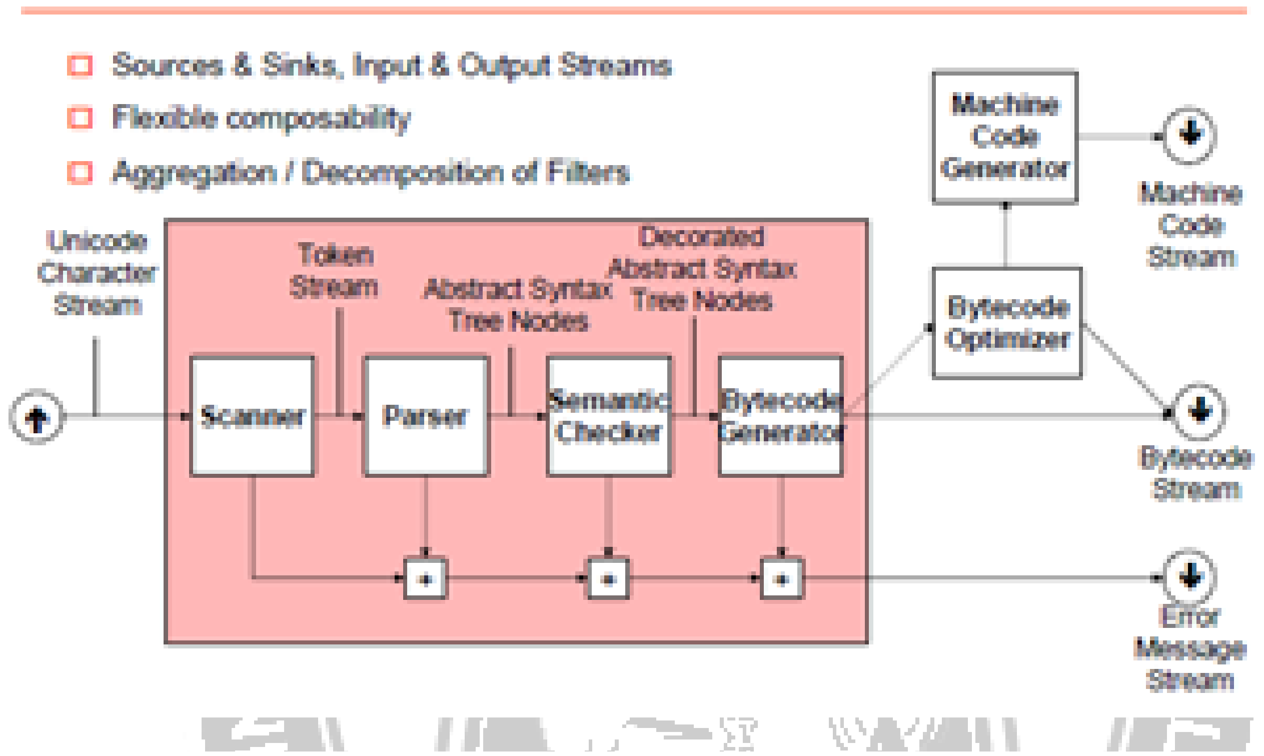
The architectural pattern is very popular and used in many systems, such as the text-based utilities in the UNIX operating system. Whenever different data sets need to be manipulated in different ways, you should consider using the pipe and filter architecture. More specific implementations are discussed below:

1. Compilers:

A compiler performs language transformation: Input is in language A and output is in language B. In order to do that the input goes through various stages inside the compiler — these stages form the pipeline. The most commonly used division consists of 3 stages: front-end, middle-end, and back-end.

The front-end is responsible for parsing the input language and performing syntax and semantic and then transforms it into an intermediate language. The middle-end takes the intermediate representation and usually performs several optimization steps on it, the resulting transformed program in is passed to the back-end which transforms it into language B.

Each level consists of several steps as well, and everything together forms the pipeline of the compiler.



USER INTERFACE DESIGN – SOFTWARE ENGINEERING

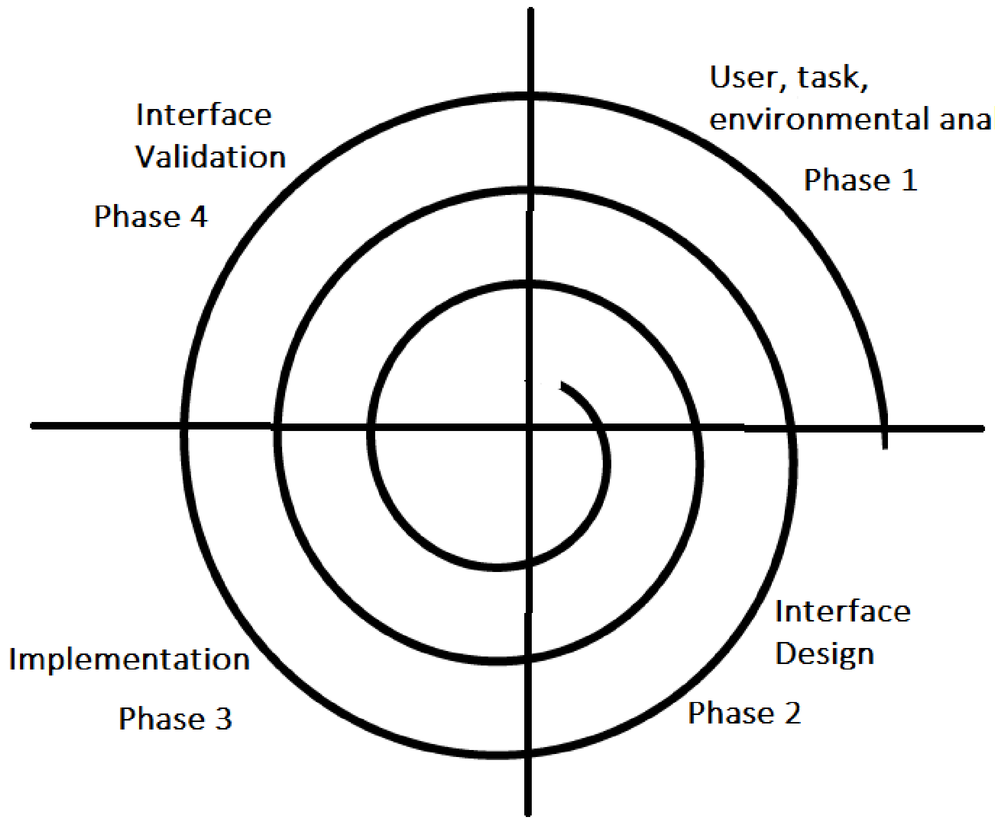
The user interface is the front-end application view to which the user interacts to use the software. The software becomes more popular if its user interface is:

1. **Attractive**
2. **Simple to use**
3. **Responsive in a short time**
4. **Clear to understand**
5. **Consistent on all interface screens**

Types of User Interface

1. **Command Line Interface:** The Command Line Interface provides a command prompt, where the user types the command and feeds it to the system. The user needs to remember the syntax of the command and its use.
2. **Graphical User Interface:** Graphical User Interface provides a simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, the user interprets the software.

User Interface Design Process



User Interface Design Process

The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities.

1. User, Task, Environmental Analysis, and Modeling

Initially, the focus is based on the profile of users who will interact with the system, i.e., understanding, skill and knowledge, type of user, etc., based on the user's profile users are made into categories. From each category requirements are gathered. Based on the requirement's developer understand how to develop the interface. Once all the requirements are gathered a detailed analysis is conducted. In the analysis part, the tasks that the user performs to establish the goals of the system are identified, described and elaborated. The analysis of the user environment focuses on the physical work environment. Among the questions to be asked are:

1. Where will the interface be located physically?

2. Will the user be sitting, standing, or performing other tasks unrelated to the interface?
3. Does the interface hardware accommodate space, light, or noise constraints?
4. Are there special human factors considerations driven by environmental factors?

2. Interface Design

The goal of this phase is to define the set of interface objects and actions i.e., control mechanisms that enable the user to perform desired tasks. Indicate how these control mechanisms affect the system. Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task. Always follow the three golden rules stated by Theo Mandel. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.

Interface Construction and Implementation

The implementation activity begins with the creation of a prototype (model) that enables usage scenarios to be evaluated. As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.

4. Interface Validation

This phase focuses on testing the interface. The interface should be in such a way that it should be able to perform tasks correctly, and it should be able to handle a variety of tasks. It should achieve all the user's requirements. It should be easy to use and easy to learn. Users should accept the interface as a useful one in their work.

User Interface Design Golden Rules

The following are the golden rules stated by Theo Mandel that must be followed during the design of the interface. **Place the user in control:**

1. **Define the interaction modes in such a way that does not force the user into unnecessary or undesired actions:** The user should be able to easily enter and exit the mode with little or no effort.
2. **Provide for flexible interaction:** Different people will use different interaction mechanisms, some might use keyboard commands, some might use mouse, some

might use touch screen, etc., Hence all interaction mechanisms should be provided.

3. **Allow user interaction to be interruptible and undoable:** When a user is doing a sequence of actions the user must be able to interrupt the sequence to do some other work without losing the work that had been done. The user should also be able to do undo operation.
4. **Streamline interaction as skill level advances and allow the interaction to be customized:** Advanced or highly skilled user should be provided a chance to customize the interface as user wants which allows different interaction mechanisms so that user doesn't feel bored while using the same interaction mechanism.
5. **Hide technical internals from casual users:** The user should not be aware of the internal technical details of the system. He should interact with the interface just to do his work.
6. **Design for direct interaction with objects that appear on-screen:** The user should be able to use the objects and manipulate the objects that are present on the screen to perform a necessary task. By this, the user feels easy to control over the screen.