Assignment

Question #01: What are fault trees? Create a fault tree for any scenario of your choice and explain in detail the fault tree. Analyze the fault tree and explain how you would modify your system to accommodate these faults.

Ans:

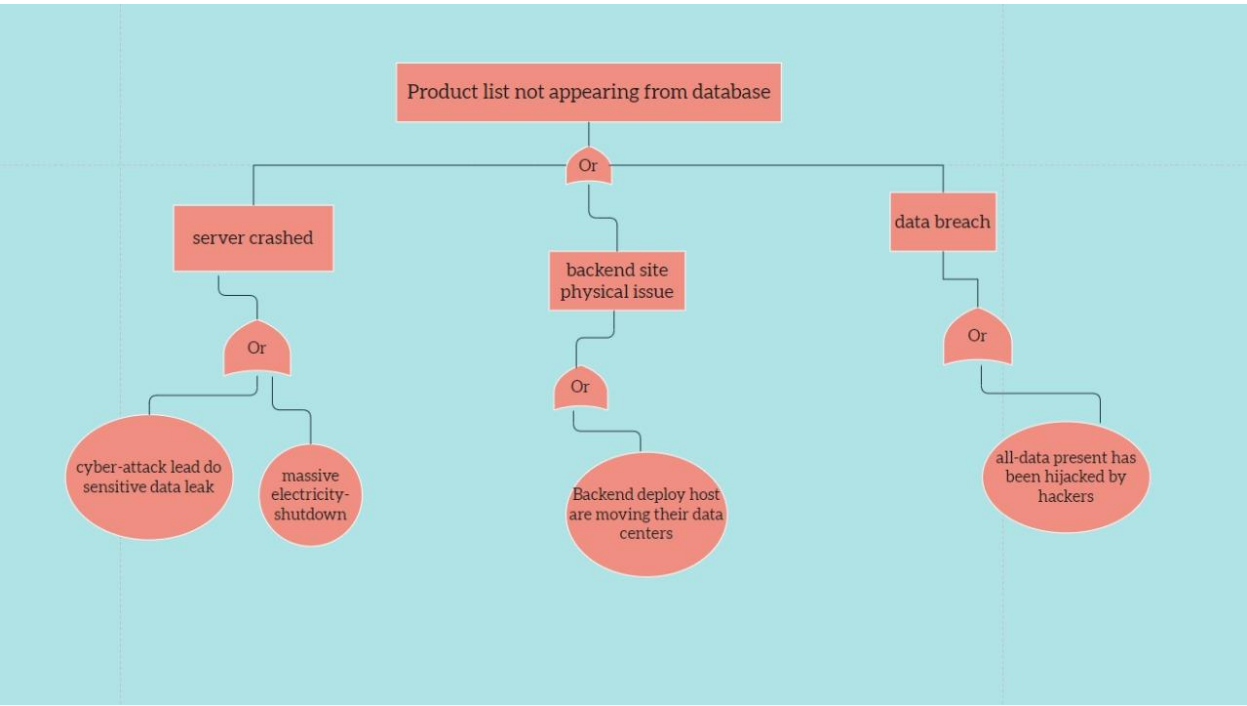
Fault Tree

Fault tree analysis is an analytical technique that specifies a state of the system that negatively impacts safety or reliability, and then analyzes the system's context and operation to find all the ways that the undesired state could occur. The technique uses a graphic construct (the fault tree) that helps identify all sequential and parallel sequences of contributing faults that will result in the occurrence of the undesired state, which is listed at the top of the tree (the "top event"). The contributing faults might be hardware failures, human errors, software errors, or any other pertinent events that can lead to the undesired state.

Scenario

The Infinity world (E-commerce Game store) was our team's semester project, which was hosted on the internet and very well going but after something user complains that the products was not shown on the website then our team analyze that the product list was not appeared from database. The reason behind this problem includes Saver crashed, backend site physical issue and Data breach.

Diagram



Root Cause:

Server Crashed:

The server crashed and the possible reason for the unavailability of the past challenges is if the server where the website is hosted crashed for some reason. The server can crash due to the following reasons: the power outage and the cyber-attack that leads to leak sensitive data.

Backend site Physical issue:

The problem can be the site the database and backend is hosted is shifting their servers to a different location so a possible solution is to have a backup remote location in fact there is some mishap so that it can be used to overcome any fallback.

Data Breach:

It is needed to have a proper security control implemented so that no one should be able to penetrate to your server, so that best thing is to have a proper audit in your industry and there should be testing done so that no vulnerability can be come out as a weakness to let the hacker penetrate into your databases.

Question #02: Explain pipe-and-filter architecture.

Definition

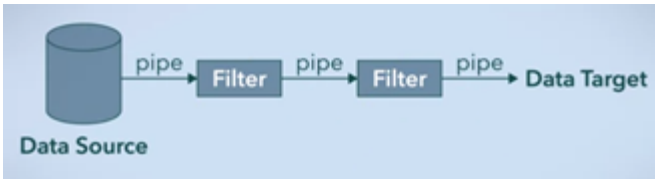
Pipe and Filter is another architectural pattern, which has independent entities called filters (components) which perform transformations on data and process the input they receive, and pipes, which serve as connectors for the stream of data being transformed, each connected to the next component in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



Data transformation in a pipe and filter architecture.

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.).

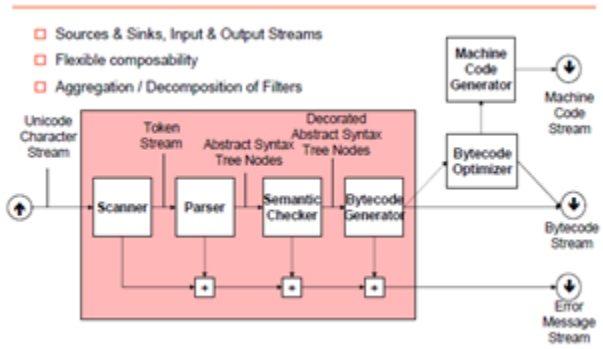
The architectural pattern is very popular and used in many systems, such as the text-based utilities in the UNIX operating system. Whenever different data sets need to be manipulated in different ways, you should consider using the pipe and filter architecture. More specific implementations are discussed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



Working of a compiler

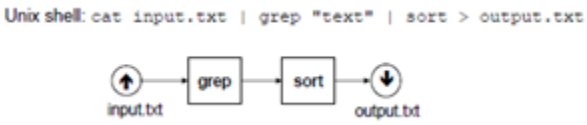
2. UNIX Shell:

The Pipeline is one of the defining features of the UNIX shell, and obviously, the same goes for Linux, MacOS, and any other Unix-based or inspired systems.

In a nutshell, it allows you to tie the output of one program to the input of another. The benefit it brings is that you don't have to save the results of one program before you can start processing it with another. The long-term and even more important benefit is that it encourages programs to be small and simple.

There is no need for every program to include a word-counter if they can all be piped into wc. Similarly, no program needs to offer its own built-in pattern matching facilities, as it can be piped into grep.

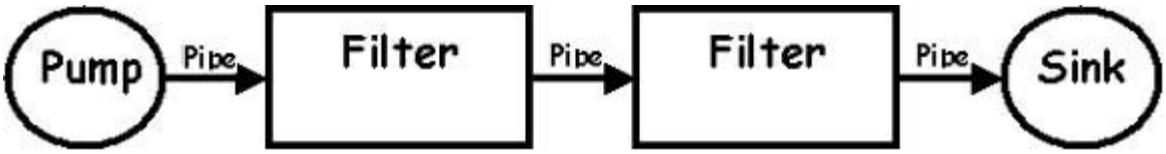
In the provided example, the input.txt is read and the output is then provided to grep as input which searches for the pattern "text" and then passes the results to sort, which sorts the results and outputs into the file, output.txt.



Example — Pipelining in the UNIX shell

Pipe-And-Filter

A very simple, yet powerful architecture, that is also very robust. It consists of any number of components (filters) that transform or filter data, before passing it on via connectors (pipes) to other components. The filters are all working at the same time. The architecture is often used as a simple sequence, but it may also be used for very complex structures.



The filter transforms or filters the data it receives via the pipes with which it is connected. A filter can have any number of input pipes and any number of output pipes.

The pipe is the connector that passes data from one filter to the next. It is a directional stream of data, that is usually implemented by a data buffer to store all data, until the next filter has time to process it.

The pump or producer is the data source. It can be a static text file, or a keyboard input device, continuously creating new data.

The sink or consumer is the data target. It can be another file, a database, or a computer screen.

Examples

- Unix programs. The output of one program can be linked to the input of another program.
- Compilers. The consecutive filters perform lexical analysis, parsing, semantic analysis, and code generation.

Where does it come from?

The popularity of the architecture is mainly due to the Unix operating system. It has become popular because Ken Thomson (who created Unix, together with Dennis Ritchie) decided to limit the architecture to a linear pipeline. Using the architecture at all was an idea of Doug McIlroy, their manager at Bell Labs at the time (1972). Both filters (coroutines) and pipes (streams) were not new, but it is not clear to me who designed the architecture of linking the coroutines by streams. As far as I can see, the design was made by Doug McIlroy.

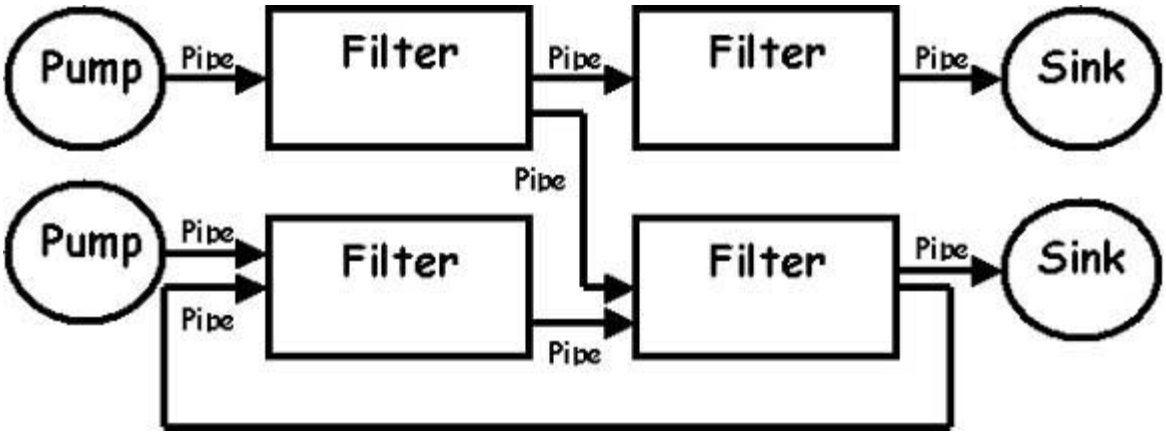
When should you use it?

This architecture is great if you have a lot of transformations to perform and you need to be very flexible in using them, yet you want them to be robust.

How does it work?

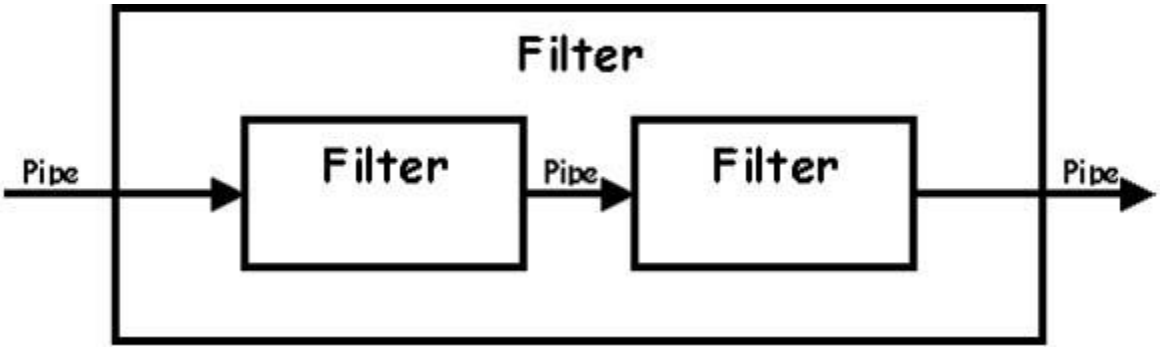
The application links together all inputs and outputs of the filters by pipes, then spawns separate threads for each filter to run in.

Here's an idea of the relationships that can be created between the different filter processes, through pipes.



All filters are processes that run (virtually) at the same time. That means, they can run as different threads, coroutines, or be located on different machines entirely. Every pipe connected to a filter has its own role in the function of the filter. So if you connect a pipe, you also need to specify the role it plays in the filter process. The filters should be made so robust that pipes can be added and removed at runtime. Every time the filter performs a step, it reads from its input pipes, performs its function on this data, and places the result on all output pipes. If there is insufficient data in the input pipes, the filter simply waits.

The architecture also allows for a recursive technique, whereby a filter itself consists of a pipe-filter sequence:



Problems

- If a filter needs to wait until it has received all data (e.g. a sort filter), its data buffer may overflow, or it may deadlock.
- If the pipes only allow for a single data type (a character or byte) the filters will need to do some parsing. This complicates things and slows them down. If you create different pipes for different datatypes, you cannot link any pipe to any filter.

Common implementation techniques

- Filters are commonly implemented by separate threads. These may be either hardware or software threads/coroutines.

Links

- Course on Software Architectures - Pipe-And-Filter
- The Evolution of the Unix Time-sharing System

Question #03: Describe four views of software architecture.

We design software, we try to describe its design and related information; The collective knowledge can be described in terms of words (document), visual (diagram), or demonstration.

In the internal description of an application, a single representation of an architectural view cannot meet the eyes of all associated various skilled stakeholders (Management, Infrastructure Operators, Product Managers, Developers, Business Executors).

Every stakeholder is only interested in some part of software design information.

Philippe Kruchten defined 4+1 Views Model to capture the description of Software Implementation or Architecture into multiple concurrent views.

A View Model is a subpart of the overall software implementation description knowledge and targets a subset of the audience.

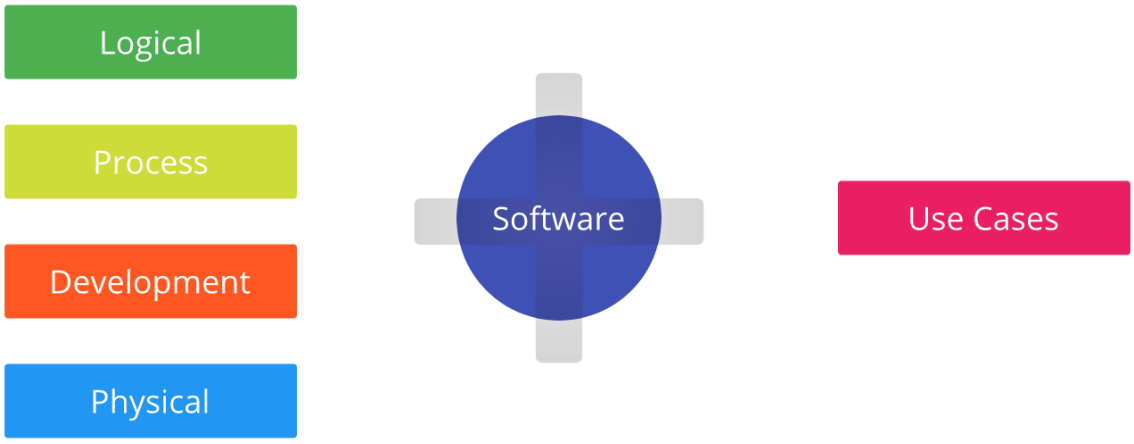
4+1 Views Model

4+1 views model is an information organization framework; it includes logical, process, development, physical layout and end-user perspective information of an application.

A view is an aspect (subpart) of information.

A notion is a way of representing information.

It is usual to find a single type of notation is used in multiple views. Views do not enforce any notion.



4+1 Model

1.Logical View captures the functional requirements of the application as decomposition of structural elements or abstractions.

For Developers, Engineer Managers

- Object Decomposition is capturing of application behavior into classes and packages. It is the base of functional analysis in case of Object-Oriented Paradigm.

UML Class Diagrams and UML Package Diagrams can be used to represent classes and packages, respectively.

- Data Modelling is the analysis of data gathering and organizing data into logical entities.

ER diagram is a well-known notation to represent business entities and relationships.

- System and subsystems view is breaking down of application into modules and arrangement of their responsibilities and relationships.

UML Component diagram can be used.

2.Process View captures the process, behavior, task concurrency, and flow of information and non-functional aspects.

- States Transition can be used to understand state and transition in case of a workflow-based system.

UML State Diagram is used to represent state and state transitions.

- Information Flow represents information routing from one entity to another entity.

Data Flow Diagram (DFD), Application Prototype (UX), UML Activity, and UML Sequence diagram represent the various flow information levels.

- Process Decomposition represents runtime partitioned application decomposition. It can be called Service Decomposition for network partitioned processes.

- Non-functional aspects like Throughput, Availability, and Concurrency. These are easier to put in words than in diagrams.

3.Development View focuses on the management of an application.

For Management & Developers

- Teams Organization — Roles and Responsibilities of team members

- Development Methodology is a way of development workflow implementation. Agile Methodologies are popular these days.

Tip: It is crucial to have agility than Agile Framework.

Popular Agile Methodologies Framework: Scrum, XP, Kanban.

- Development Standards — Set of guidelines & coding standards, automation tools.

e.g. VCS System and their branching system, CI, Deployment Automation, Code Linting, Code Style

- Test Planning of functionalities.

How do you test? Automation or Manual or hybrid?

What do you test? Scope of test?

How do you record test step sequences?

- Work logs and Tracker system to manage and track tasks.

JIRA, Asana, Harvest are popular commercial tools.

- Road-maps give ideas about deliverables.

4.Physical View represents the deployment layout or infrastructure of an application. In essence, it captures hardware mapping of application components or processes.

For DevOps

- Topology Architecture | Deployment Plan

The cluster of application instances and their places in the geography of physical or virtual machines available.

UML deployment diagram, Network diagram are standard options.

- System Capacity of the application

- Configuration Management

Tip: It is essential to keep configuration out of the application and build a workflow to change the configuration to have a higher degree of observability and flexibility.

Use Cases captures an End-User Perspective into a systematic, logical information structure. It complements all other views and is used to validate them.

For Every Stakeholder

Stories are used for elaboration; It may be a combination of textual documents with or without UML Sequences, Actor diagrams, Prototypes.

Closing Notes

4+1 Model Views covers much of the software if done right. It is not strict; You can skip any view or sub view, which does not make any sense in expressing your software design.

4+1 Views Model is too result-oriented. Views only represent the results of decisions that may not be evident over time or to the people who were not part of the original design discussion. This problem can be solved, including an additional view: Architecture Decision View.

Architecture Decision View captures contextual knowledge behind any architectural decision. It focuses on causation and circumstances that lead to decisions.