Contents

[Module 1 – Introduction to Software Architecture 2](#_Toc114843861)

[What is Software Architecture 2](#_Toc114843862)

[Who is Stakeholders 2](#_Toc114843863)

[Why Architecture needs to be described 2](#_Toc114843864)

[Definitions of Software Architecture 2](#_Toc114843865)

[Architecture Styles 2](#_Toc114843866)

[Architectural Views and Architectural Structure 3](#_Toc114843867)

[Reference Model and Reference Architecture 3](#_Toc114843868)

[Benefits of Software Architecture 4](#_Toc114843869)

[Three Structures 6](#_Toc114843870)

[Summary 6](#_Toc114843871)

[Module 3 - Capturing Architecturally Significant Requirements 7](#_Toc114843872)

[Module 4 - Documenting software Architecture 7](#_Toc114843873)

[4+1 architectural view model 7](#_Toc114843874)

[Kruchten’s 4+1 (arch view) description in detail 8](#_Toc114843875)

[Combining Views 12](#_Toc114843876)

[Module 5 12](#_Toc114843877)

[Layered Architecture 12](#_Toc114843878)

[Architecture evaluation (ATAM) 16](#_Toc114843879)

[Architecture conformance techniques during implementation 16](#_Toc114843880)

[Architecture & Testing 16](#_Toc114843881)

[Architecture reconstruction 17](#_Toc114843882)

# Module 1 – Introduction to Software Architecture

## [What is Software Architecture](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=what-is-software-architecture)

Software architecture is the blueprint of building software. Blueprint of a software system that helps stakeholders to understand how the system would be once it is implemented

Description in details, A description at a higher level of abstraction than objects and lines of codes So that Stakeholders understand and reason about without getting lost into a sea of details

## [Who is Stakeholders](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=who-is-stakeholders)

Software architecture is not necessary to say yes to all area of concern, but he should ready with answer for all to stakeholders must agree

## [Why Architecture needs to be described](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=why-architecture-needs-to-be-described)

1. Each stakeholder has his own interpretation of the systems Sometimes no understanding at all Architect is the middleman who co-ordinates with these stakeholders
2. How will everyone be convinced that his expectations from the system will be satisfied?
3. Even when the architect has created the solution blueprint, how does she handover the solution to the developers?
4. How do the developers build and ensure critical aspects of the system?
5. Misunderstanding leads to incorrect implementation Leads to 10 times more effort to fix at a later stage

## [Definitions of Software Architecture](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=definitions-of-software-architecture)

### [Architecture Styles](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=architecture-styles)

Architecture style first proposed by Shaw and Garlan– synonymous to “architecture pattern”

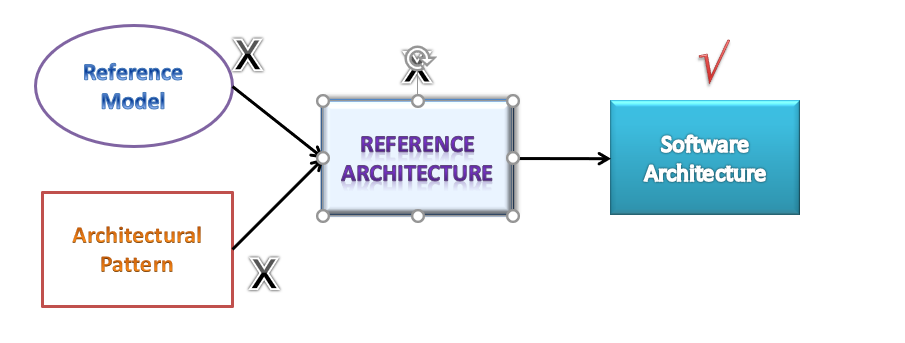
* A set of element types (what the element does- data store, compute linear regression function)  
  *Note:*Element A is storing data and Element B might do some business logic
* A set of interaction types (function call, publish-subscribe)  
  *Note:*Interaction types means how this elements are interacting with each other  
  *Note:*Interaction will be in lines, like data flow, publish-subscribe architecture, etc)
* Topology indicating interactions and interaction types  
  *Note:*topology means the way it is connected
* Constraints  
  *Note:*Constraints is like what is allowed and not allowed
* Also known as architectural pattern

### [Architectural Views and Architectural Structure](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=architectural-views-and-architectural-structure)

* Since architecture serves as a vehicle for communication among stakeholders
  + And each stakeholder is interested about different aspects of the system
  + It is too complex to describe, understand and analyze the architecture using one common vocabulary for all stakeholders
    - Essentially it needs to be described in a multi-dimensional manner  
      *Note:*Stackholder could be CTO, developer, DB, etc each is interested in there aspect like CTO - Standard and policy, cyber security, DB - storing data, so it is describe in multi-dimensional manner  
      *Note:*Software architecture it needs to described in a multi-dimensional manner so each dimension is essentially called as view
* View based approach
  + Each view represents certain architectural aspects of the system, created for a stakeholder
  + All the views combined together form the consistent whole  
    *Note:*The way we describe View is by Structure
* A Structure is the underlying part of a view- essentially the set of elements, and their properties
  + A view corresponding to a structure is created by using these elements and their inter-relationships  
    *Note:*Structure means boxes, lines, images and so on (which helps to create to block diagram). Views is those we create by this Structure

### [Reference Model and Reference Architecture](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=reference-model-and-reference-architecture)

* A reference model
  + Decomposes the functionality into a set of smaller units
  + How they interact and share data
  + These units co-operatively implement the total functionality
* A reference architecture
  + Derived from the reference model
  + Concrete software elements, mapped to the units of the reference model, that implement the functionality  
    *Note:*a reference architecture is an example solution using that set of reference models

  
*Note:*

* Reference model - is abstract and belongs to particular domain
* Combination of reference model and Architecture pattern is Reference Architecture
* eg: Reference Architecture - Mobile commerence platform i.e., set of elements interactions and those elements basically would describe what are the typical functionalities but will really not address all the quality of service concerns but it will tell you that what are the essential components that are required
* software architecture - is essentially created from a reference architecture that's a specific instance of reference architecture there you instant hit each element with real modules that are going to be implemented and you address a lot of other concerns, you add lot more details and that makes it that makes it a software architecture, so it is basically an actual software system that is derived from the blue from the requirement so you know this one is essentially created from the requirement
* Architecture Pattern - is dealing with more of non-functional
* reference model and reference architecture deals with the functional aspects
* software architecture - if Airtel phone or any other company they basically uses this reference architecture and builds their own platform then it becomes a software architecture
* Software architecture - blueprint that is derived from the requirement is the software architecture it contains design decisions
* reference architecture - really does not have any design decisions taken

### [Benefits of Software Architecture](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=benefits-of-software-architecture)

1. Every stakeholder should understand “unambiguously” what the blueprint is

* Standard approach, vocabulary, output
* Common language for communication

*Note:*this particular benefit is something that makes a huge difference in our large corporation every stakeholder understands unambiguously what the blueprint is so that means there is a complete standardization of the approach vocabulary and the output that means everybody understands what the other group is doing and there is a common language for communication across different stakeholders that definitely helps in streamlining the work

1. Streamlining work assignments for multiple teams

* Avoiding information loss, enforcing traceability

*Note:*who is doing what what what time they're finishing is a misunderstanding of what is expected so so all these things are a huge overhead in terms of with respect to implementing a software and oftentimes the project teams really do not estimate such overheads they they don't even consider that they assume that everything is going will go on fine and they just compute the estimates based on the complexity of the software the function points and so on but this streamlining part is a real challenge and that never are hardly considered so architecture actually helps in streamlining things and it can avoid information loss it can avoid such ambiguity and it can enforce the traceability that what are the requirements that were considered whatare not considered if they're considered where it has been considered

1. Early analysis of meeting quality requirements and compromise between different QoS requirements

* Early prototyping of important aspects quickly
* Design decisions are made early
* Quicker to evaluate these decisions and correct it rather than discovering it later (10 – 100 times more costly)
* Early analysis of QoS and evaluation of architecture
* More accurate cost and schedule estimation

*Note:*If decisions are not made early or not evaluated or corrected later on oftentimes there are problems and there are rework so it is. At the architecture level because it is at a high level it is quicker to evaluate these these decisions and actually correct it rather than discovering it late. as it takes 10 to 100 times more effort if it is discovered during testing or during deployment, it can be a disaster all the analysis of quality of service and evaluation of architecture is possible. If software architecture creates properly all the analysis of meeting the quality requirements and different compromises are also possible to analyze, because not every quality of aspects can be really met sometimes they become the contradict each other, so one needs to really assess that what are the contradictory requirements and how one has to choose. The best place to analyze and decide such kind of contradictory requirement is at the level of software architecture it really shouldn't be done they at a later stage it should be done only at the architecture stage.

*Note:*streamlining of work assignments becomes a major problem that can actually make all this cost unskilled estimates to go here where things become wrong and in fact it becomes a major major overrun

1. Improve speed of development

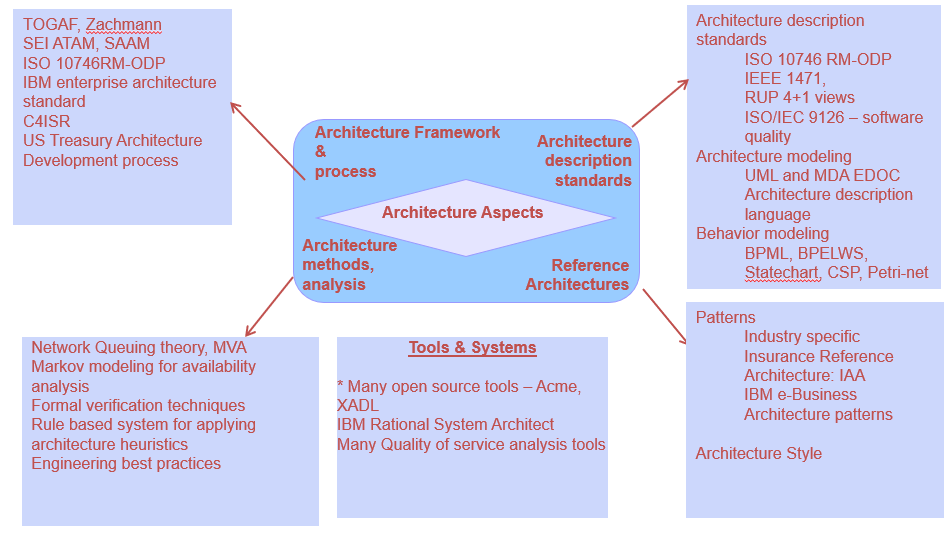
* Reuse
  + Helps in building a large product line faster by sharing common architecture
  + From one implementation to another similar implementation
* Based on the architecture, one can quickly decide build-vs –use external components
* Tool that can automate part of development, testing

*Note:*Improves the speed of development in fact if in a large organization if reuse is is to be promoted and reuse is a buzzword or reuse is a mantra that many corporations use they really reuse the artifacts generated by one project team in another, so essentially that's possible only if one does software architecture properly  
*Note:*then tools that can automate part of development and testing that also can be used in fact there are tools

### [Three Structures](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=three-structures)

1. Module Structure How is the system to be structured as a set of code units (modules)?
2. Component-and-connector structures How is the system to be structured as a set of elements that have runtime behavior (components) and interactions (connectors) What are major executing components and how do they interact  
   *Note:*Components might be shared library, executable files, or webservices
3. Allocation structures How is the system to relate to non-software structures in it’s environment (CPU or cluster of CPUs, File Systems, Networks, Development Teams …)  
   *Note:*Software system is connected to environments like CPU system is going to deploy it on what machines how many machines how the data is going to be stored in different file system how the communication is actually happening through the networks

## [Summary](https://amitpnk.github.io/BITS-Software-engineering/#/./SoftwareArchitecture/Module1?id=summary)



# Module 3 - Capturing Architecturally Significant Requirements

## Topics

* Challenges in identifying ASRs
* Quality attribute Workshop
  + Understanding business goals from Sponsors
  + Identifying architectural drivers
  + Understanding Scenarios for each architectural driver via brainstorming with stakeholders
  + Prioritizing scenarios
  + Building a Utility tree
* Architecture design
  + Design strategy
  + Steps of Attribute-Driven design
  + Architecting in Agile projects



# Module 4 - Documenting software Architecture

## 4+1 architectural view model

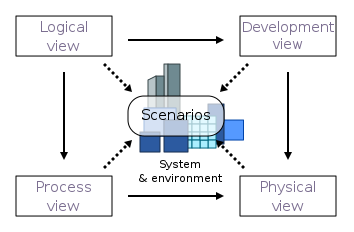
**4+1** is a [view model](https://en.wikipedia.org/wiki/View_model) used for "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views".[[1]](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-Kru95-1) The views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers, system engineers, and project managers. The four views of the model are logical, development, process and physical view. In addition, selected [use cases](https://en.wikipedia.org/wiki/Use_case) or scenarios are used to illustrate the architecture serving as the 'plus one' view. Hence, the model contains 4+1 views:[[1]](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-Kru95-1)

* ***Logical view***: The logical view is concerned with the functionality that the system provides to end-users. [UML diagrams](https://en.wikipedia.org/wiki/Unified_Modeling_Language) are used to represent the logical view, and include [class diagrams](https://en.wikipedia.org/wiki/Class_diagram), and [state diagrams](https://en.wikipedia.org/wiki/State_diagram).[[2]](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-Kontio05-2)
* ***Process view***: The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system. The process view addresses concurrency, distribution, integrator, performance, and scalability, etc. UML diagrams to represent process view include the [sequence diagram](https://en.wikipedia.org/wiki/Sequence_diagram), [communication diagram](https://en.wikipedia.org/wiki/Communication_diagram), [activity diagram](https://en.wikipedia.org/wiki/Activity_diagram).[[3]](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-3)[[2]](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-Kontio05-2)
* ***Development view***: The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. UML Diagrams used to represent the development view include the [Package diagram](https://en.wikipedia.org/wiki/Package_diagram) and the [Component diagram](https://en.wikipedia.org/wiki/Component_diagram).[[2]](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-Kontio05-2)
* ***Physical view***: The physical view (aka the *deployment view*) depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. UML diagrams used to represent the physical view include the [deployment diagram](https://en.wikipedia.org/wiki/Deployment_diagram).[[2]](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-Kontio05-2)
* ***Scenarios***: The description of an architecture is illustrated using a small set of [use cases](https://en.wikipedia.org/wiki/Use_case), or scenarios, which become a fifth view. The scenarios describe sequences of interactions between objects and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as the **use case view**.

The 4+1 view model is generic and is not restricted to any notation, tool or design method. Quoting Kruchten,

The “4+1” view model is rather “generic”: other notations and tools can be used, other design methods can be used, especially for the logical and process decompositions, but we have indicated the ones we have used with success.

— *Philippe Kruchten, Architectural Blueprints—The “4+1” View Model of Software Architecture*[[1]](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model#cite_note-Kru95-1)

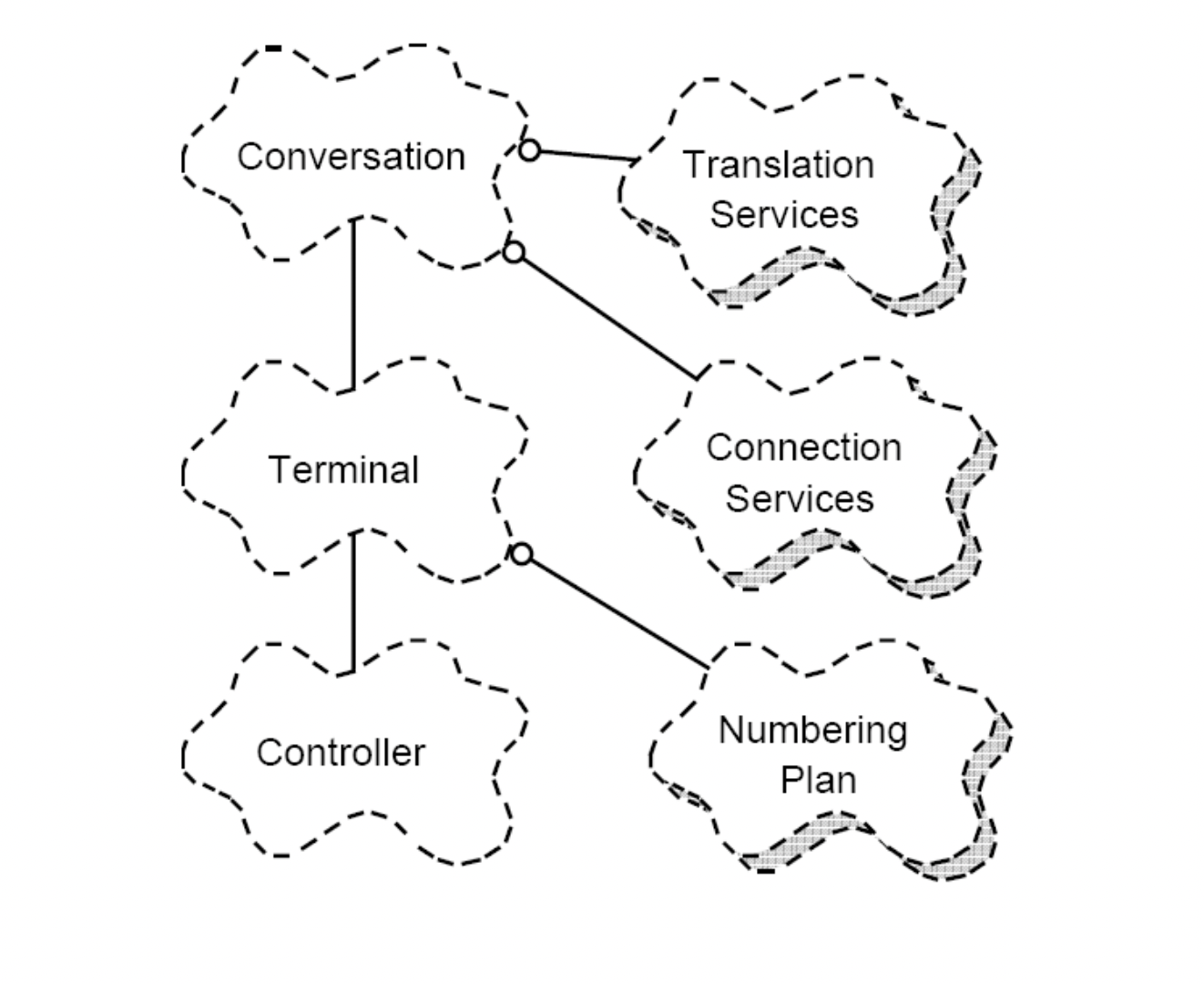


### Kruchten’s 4+1 (arch view) description in detail

In the 4+1 model the system is described using different viewpoints. The model has four views: logical, development, process, and physical. In addition, selected use cases or scenarios are utilized as the ‘plus one’ view to show the design. As a result, the model has 4+1 views. Hence the model is called The 4+1 Architectural View. Now let’s take a look at them individually.

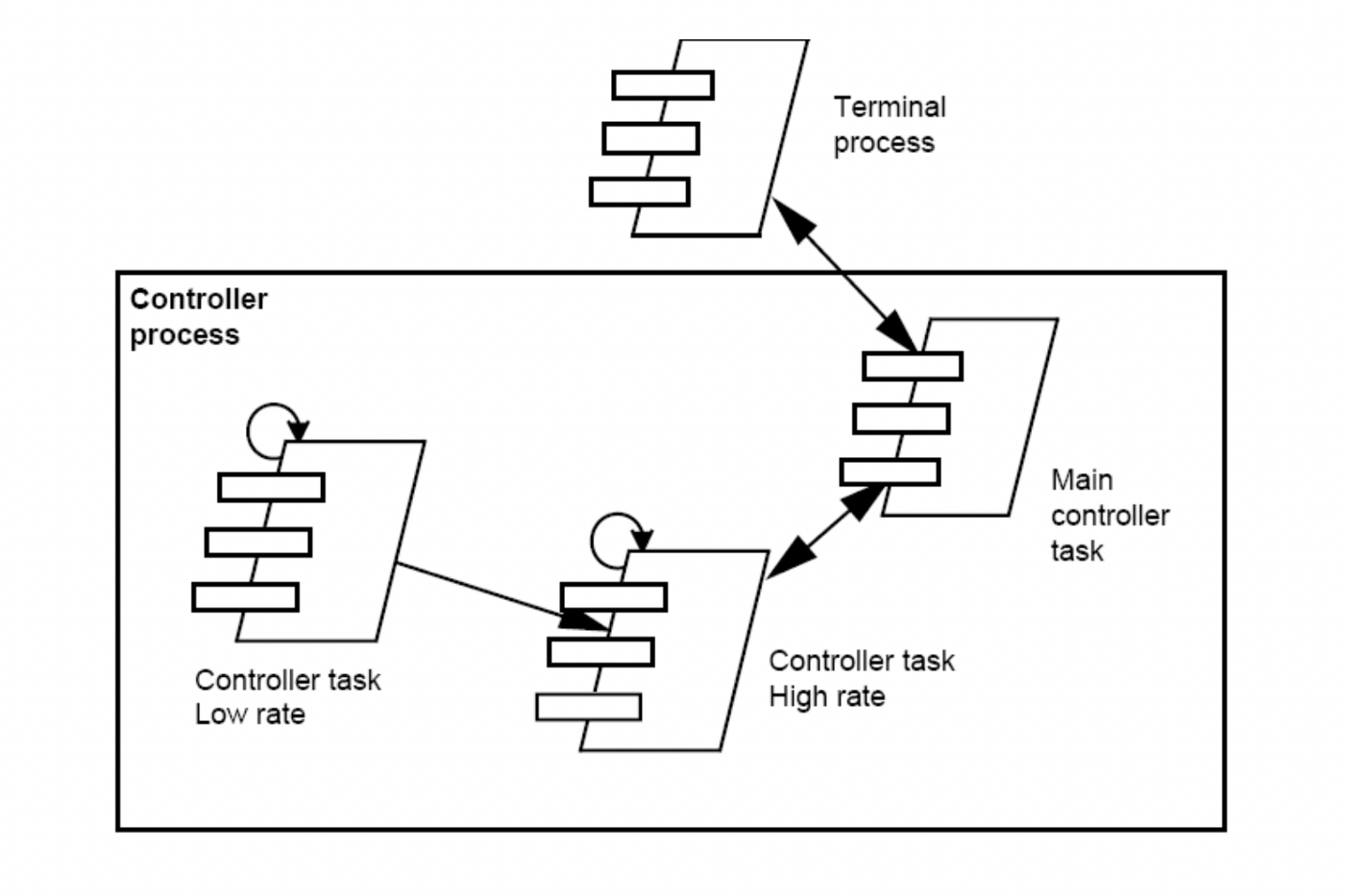
Model Description

**Logical View:**The logical view is concerned with the system’s functionality as it pertains to end-users. Class diagrams and state diagrams are examples of[UML diagrams](https://medium.com/javarevisited/5-best-uml-books-and-courses-for-java-programmers-13c551a9235d) that are used to depict the logical view.



Example for a Logical View (Image Source: <http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:mazeiar-kruchten-4_1.pdf>)

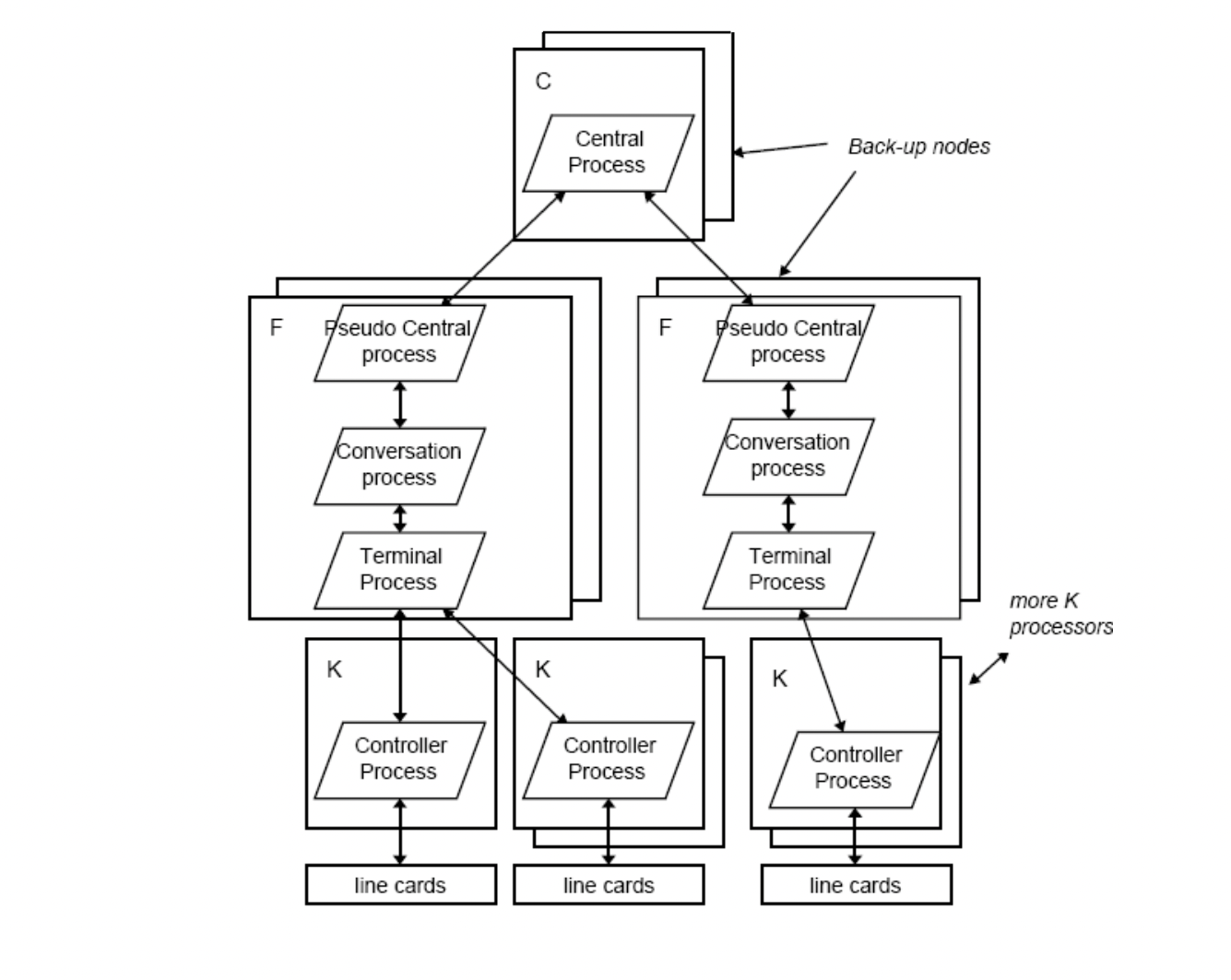
**2. process View:** The process view focuses on the system’s run-time behavior and deals with the system’s dynamic elements. It explains the system processes and how they communicate. [Concurrency](https://medium.com/javarevisited/8-best-multithreading-and-concurrency-courses-for-experienced-java-developers-8acfd3b25094), distribution, integrator, performance, and scalability are all addressed in the process view. The sequence diagram, communication diagram, and activity diagram are all UML diagrams that can be used to describe a process view.

[[](https://javarevisited.blogspot.com/2017/07/top-5-books-to-learn-uml-unified-modelling-language-java.html)](https://javarevisited.blogspot.com/2017/07/top-5-books-to-learn-uml-unified-modelling-language-java.html)

Example of a Process View (Image Source: <http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:mazeiar-kruchten-4_1.pdf>)

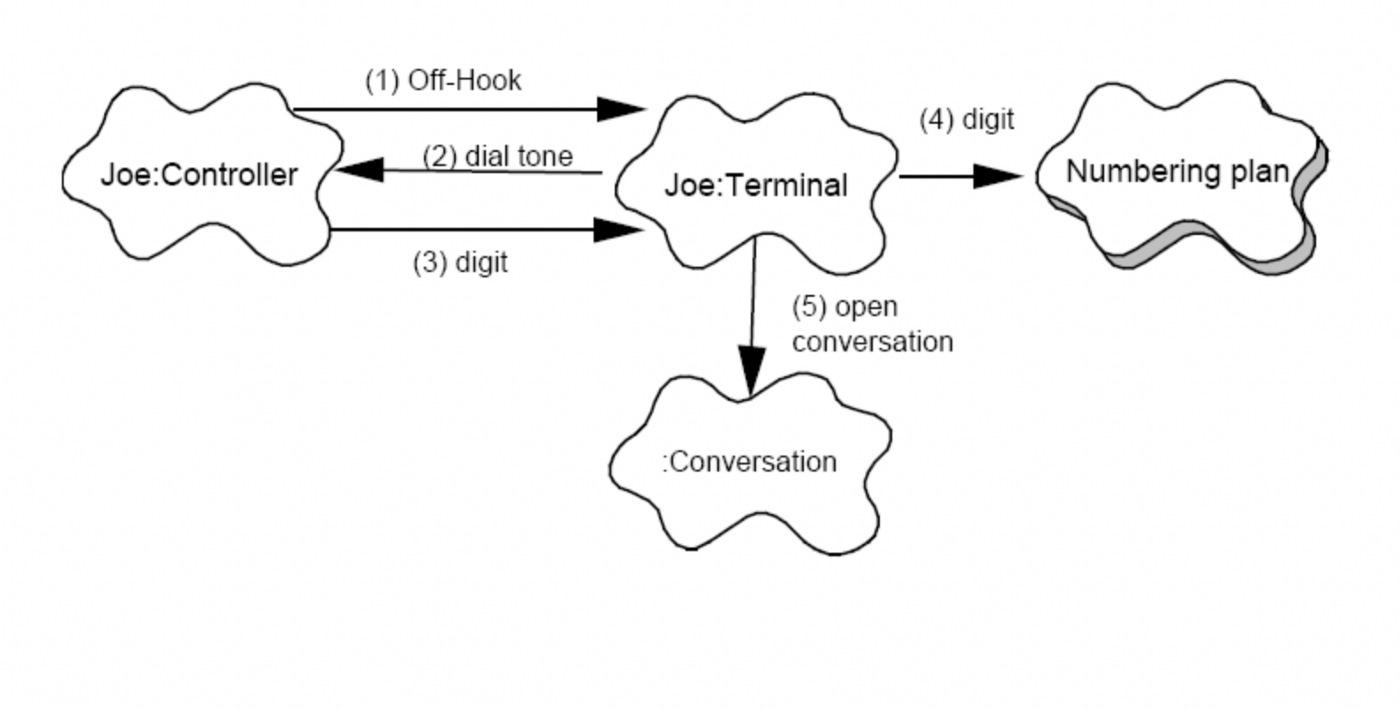
**3. Development View:** The development view depicts a system from the standpoint of a programmer and is concerned with software administration. The implementation view is another name for this view. It describes system components using the UML Component diagram. The Package diagram is one of the UML diagrams used to depict the development view.

**4. Physical View:**The physical view portrays the system from the perspective of a system engineer. The physical layer, it is concerned with the topology of software components as well as the physical connections between these components. The deployment view is another name for this view. The deployment diagram is one of the UML diagrams used to depict the physical perspective.



Example of a Physical View (Image Source: <http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:mazeiar-kruchten-4_1.pdf>)

**5. Scenarios:**A small number of use cases, or scenarios, that become the fifth view, are used to illustrate the description of architecture. Sequences of interactions between [objects](https://javarevisited.blogspot.com/2020/05/object-oriented-programming-questions-answers.html#axzz6vwZEctyQ)and processes are described in the scenarios. They are used to identify architectural aspects as well as to demonstrate and assess the design of the architecture. They can also be used as a starting point for architecture prototype testing. The use case view is another name for this view.



Example of a Scenario (Image Source: <http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:mazeiar-kruchten-4_1.pdf>)

That brings me to the end of today’s article. Thank you very much for taking the time to read this, and I’d love to hear your thoughts if you have any. Goodbye and stay safe and I’ll see you again with another article.

## Combining Views

A combined view can be produced from different views of the same overarching viewtype. For example, the modules defined in a module uses view can be mapped to layers described in a layered diagram because both belong to the module viewtype. A combined view also can be produced from views of different viewtypes. For example, layers of a layered style, which belongs to the module viewtype, can be mapped to processes in the communicating-processes style, which belong to the C&C viewtype.

# Module 5

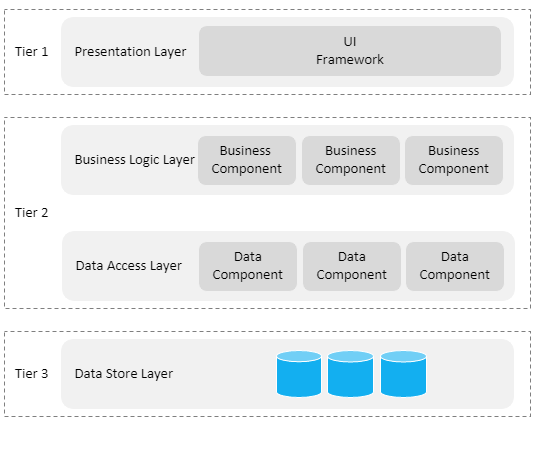
## Layered Architecture

The layered software architecture pattern is the most commonly used architecture pattern in software engineering. This architectural pattern is also known as the n-tier architecture style or the multi-layered architecture style. The purpose of a layered architecture is to organize the components of an application into horizontal logical layers and physical tiers.

A layer is a logical unit that separates a specific role and responsibility within an application. Each layer manages its own software dependencies. A higher layer can use services in a lower layer, but not the other way around. A tier is a physical unit where the code runs; for example, a web server or a database.

Each layer can be hosted in its own tier, however it is not required. Several layers can be hosted on the same tier. Scalability, maintainability and resiliency are increased when physically separating the tiers, however latency increases due to the additional network communication.

A traditional layered software architecture has three tiers and four layers. Consider the below architectural diagram.



The four standard layers are described as follows.

*Presentation Layer*  
This layer contains all user interfaces exposed to a user. It may provide different types of user interfaces, namely web, desktop and native mobile applications.

*Business Logic Layer*  
This layer handles all business logic, validations and business processes.

*Data Access Layer*  
This layer is responsible for interacting with a database. It is also known as the persistence layer.

Data Store Layer  
This layer is the actual data store for the application.

By introducing different layers for the software components, separation of concerns (SoC) is increased drastically, thus improving simplicity, maintainability and test-ability of the components.

The tiers are described as follows.

*Tier 1: The Presentation Tier*  
This tier hosts the front-end code base, that is, the presentation layer. This is the topmost level of the application and it is essentially a layer users can access directly.

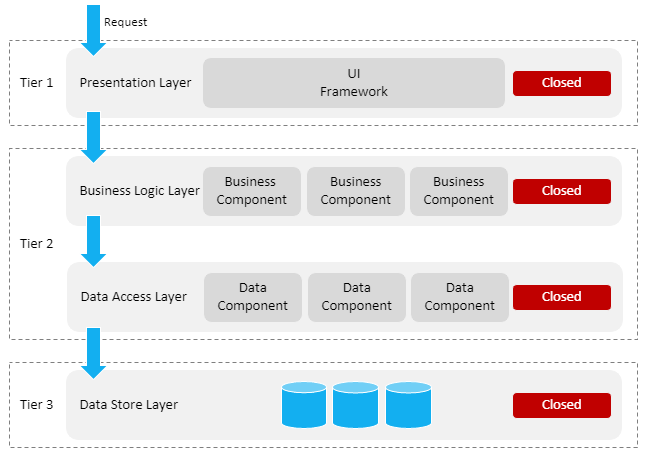
*Tier 2: The Application Tier*  
This tier hosts the back-end code base, that is, the business logic layer and data access layer. It is also known as the middle tier.

*Tier 3: The Data Tier*  
This tier hosts the data store, that is, the data store layer. Databases, file system, blob storage, document database, etc are examples of resources found in a data store.

There are two ways to implement a layered software architecture, namely closed layered architecture and opened layered architecture.

*Closed Layered Architecture*

In a closed layered architecture, a layer can only call the next layer immediately below it. Consider the below closed layered architecture diagram.



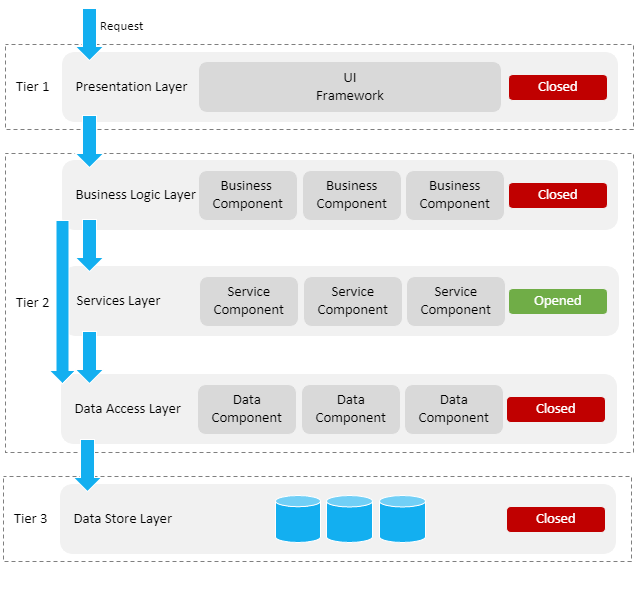
In the above architecture diagram, each layer is marked as closed. This means that in order for a request to progress to the bottom most layer, the request will need to flow through each layer. For example, when a request is initiated via the presentation layer, the request will first go through the business logic layer, next to the data access layer and finally to the data store layer.

The purpose of the closed layered architecture is to ensure that layers are completely isolated from one another, meaning that changes made in one layer of the architecture does not impact other layers. In other words, layers are loosely coupled, where each layer has little to no knowledge of other layers.

Note that when implementing a closed layered architecture, this limits the dependencies between layers, thus introducing unnecessary network traffic, since one layer simply passes requests along to the next layer.

*Opened Layered Architecture*

In an opened layered architecture, a layer can call any layer below it, provided the layer below it is marked as opened. Consider the below opened layered architecture diagram.



In the above diagram, a new services layer was introduced. This layer provides a number of service components. Not all service components in the services layer interact with a database. For this reason, the layer is marked as opened. This means that the business logic layer has access to both the services layer and data access layer, that is, the business logic layer does not need to go through the services layer in order to use the data access layer.

Architecture sinkhole anti-pattern

A most common problem with the layered software architecture is when requests flow through multiple layers of the architecture as simple pass-through requests with no logic performed within each layer. This is known as the architecture sinkhole anti-pattern. Mark Richards suggests to utilize the 80-20 rule to determine if this anti-pattern is a problem.

Every layered architecture will have at least some scenarios that fall into the architecture sinkhole anti-pattern. The key, however, is to analyze the percentage of requests that fall into this category. The 80-20 rule is usually a good practice to follow to determine whether or not you are experiencing the architecture sinkhole anti-pattern.

[Mark Richards](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html)

Summary

The layered architecture pattern is a very simple, structured and solid pattern that is well known in enterprise software. This pattern is also portable between cloud and on-premise platforms.

There are a number of architectural patterns available, each with its own pros and cons. Choosing an architectural pattern depends on the scenario or project requirements.

Further information on the layered architecture pattern in software engineering can be found at the following links:

## Architecture evaluation (ATAM)



## Architecture conformance techniques during implementation



## Architecture & Testing



## Architecture reconstruction

