LITERATURE REVIEW

Selection of the right architecture style is very important to determine the quality of the software to be produced. Number of architectural styles and frameworks has been proposed in the literature. And these frameworks and architectural styles can be used depending on the prioritized non functional requirements and functional requirements of the software architecture. This chapter reviews the software architectural styles and frameworks, classifies them and discusses the limitations of them. And also this chapter identifies the gap in the existing architectural styles and frameworks.

2.1. SOFTWARE ARCHITECTURE EVOLUTION

Architecture of any system helps us to deal with the complexity of that system. As systems become more complex, they generally require more planning. Architecture is both the process and product of planning, designing and constructing space that reflects functional, social and aesthetic considerations [27]. It also encompasses project planning, cost estimating and constructing administration. In civil engineering, Architecture deals with the relationship between complexity and planning for buildings and cities. It needs various views from the customer side and from the builder’s side.

Similarly, the same concept can be used for software which is called software architecture. Building a very complexive, critical and highly distributed system requires the interconnected components that are basic building blocks and the views of end user, designer, developer and tester. The research work of Dijkstra in 1968 and David Parnas in the early 1970’s identified the concept of Software Architecture at first [28] [29]. Software architecture deals with the design and implementation of the high-level structure of the software. It is the result of assembling a certain number of
architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements of the system, as well as some other, non-functional requirements such as reliability, scalability, portability, and availability [30]. Software frameworks define the places in the architecture where application programmers may make the adaptations for specific functionality. A software framework is an abstraction in which common code providing generic functionality can be selectively overridden or specialized by user code [31]. The designers and programmers can spend their time to meet software requirements rather than dealing with the more standard low level details of providing a working system thereby reducing overall development time.

Even though ‘software architecture’ is relatively new to the field, the basic principles of this field have been applied since the mid 1980’s. There are several definitions of software architecture in the literature, all of them agree that it describes the organization of the overall system. The most commonly used definitions concentrate on the topological view [32] [33]. According to Perry & Wolf, Software Architecture is a set of architectural elements that have a particular form, and an underlying rationale [32].

Garlan & Shaw define software architecture as a collection of components, connectors and configurations where components define the locus of computation, connectors define the interactions between components and configurations define the topology of the components and connectors [33]. Francis Anderson (Chief Object Architect, Clear Systems) defines Software architecture as sets of solution frameworks to which all aspects of the problem domain are mapped [34].

Brahim Djoudi (Groupement des Industries de l'Armée de Terre (GIAT) Industries) states that the software architecture is the set of all the concepts (software components, frameworks, Data Base, paradigm,
programming language, etc.) used to "view" an entire software before it's designed [35]. According to Software Engineering Institute (SEI), software architecture is an abstract view of a software system distinct from the details of implementation, algorithms, and data representation [36]. It was defined in the Unified Modeling Language (UML) 1.3 that Architecture is the organizational structure of a system. Architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts [37]. Parts that interact through interfaces include classes, components and subsystems.

Architectural patterns are solutions to recurring architectural problems. It describes a coarse-grained solution at the level of subsystems or modules and their relationships. When the granularity gets increased and deals about the overall system, then the solutions are called as Architectural styles. Shortly, architectural patterns are called as architectural styles when they are applied in the higher level. Hence, the MVC and PAC patterns are referred as architectural styles in this research. Architectural framework is a skeleton or abstraction of the implementation of the software which is reusable.

The evolution of software architecture, architectural styles and frameworks discussed below is represented in fig 2.1. In 1928, an Algorithm that has the finite sequence of instructions to solve the problem was partially formulated. To have a visual representation of the instruction flow, Von Neumann developed ‘Flow Chart’ to plan computer programs in 1947 by getting the idea from the flow process chart (1921) and multi flow chart (1944) which were used mostly in the area of electrical engineering. Later, a diagram to describe the control flow of a business process and program, ‘Control Flow Diagram’(CFD) was developed in the late 1950’s.

The representation of flow of control was not enough to view the complex systems and it was not an easiest one for it. It doesn’t give a quick,
high level view of the work and instant access of interesting points. Hence ‘Block Diagram’ was developed in late 1950’s to understand the complete system by breaking it down into smaller sections or blocks. Each block performs a particular function and the diagram shows how they are connected together.

In the late 1960’s, the introduction of Abstract data types paves a way to group data structures that have similar behavior and to group data types and modules of one or more programming languages that have similar semantics.

Fig 2.1 Evolution of software architecture, architectural styles and frameworks

In 1968, the notion of abstract data types lead to a software design technique called ‘Modular Programming’ that increases the extent to which software is composed of separate parts called modules. Modules represent the separation of concerns and improve maintainability by enforcing logical
boundaries between components. And also it paved a way for the Main subroutine and Master slave architectural styles to implement the system. Master slave is the variation of the Main subroutine with concurrency. During 1970’s the development of ARPANET became the basis for the email and introduced the distributed system concept [38].

McIlroy, one among the author of early command shells found that processing the output of one program as the input to another took more time. In 1973, Ken Thompson implemented McIlroy’s ideas by adding a pipe system call to the shell. [39] This was acquired by other operating systems and appeared as pipe and filter architectural style.

In 1977, ‘Three Schema Approach’ was developed by adopting layered architecture based on the modular programming. It is used to build information systems using three different views in systems development. Applications are broken into tiers and developers have to modify a specific layer not to rewrite the entire application over. Flexible and reusable applications can be developed using this scheme [40].

Howard Sturgis, James Mitchell, and Jay Israel worked at Xerox PARC, used the word client in their paper "Separating Data from Function in a Distributed File System" [41]. The client server model was introduced in the distributed environment. The blackboard model arose from abstracting features of the HEARSAY-II speech understanding system developed between 1971 and 1976 by taking advantage of Rule based programming and distributed computing [42].

In 1991, the combination of distributed computing and object oriented approach led to the broker style [43]. In order to provide an interface for the manipulation of multiple views of data, the Model view controller (MVC) style was introduced by Trygve Reenskaug during 1978 to 1979 and the modified version was implemented in Xerox PARC Smalltalk-80 [44]. The
Presentation Abstraction Control style was introduced to establish a model to develop interactive applications by Joëlle Coutaz in 1987. This style incorporates the cognitive organization of human knowledge [45].

Later, based on the three tier approach, a layer of six perspectives was introduced in 1987 by John Zachman. That is called as ‘The Zachman Framework’ which still plays an important role in the era of ‘Enterprise Architecture’ and influenced frameworks Department of Defence Architectural Framework (DODAF), The Open Group Architecture Framework (TOGAF), Treasury Enterprise Architecture Framework (TEAF) and Federal enterprise Architecture Framework (FEAF). Modified version of Zachman Framework with more number of views was released in 1993. In 1995, 4+1 view model was developed by Kruchten.

The purpose of Views used in these models is to enable human engineers to comprehend very complex systems, and to organize the elements of the problem and the solution around domains of expertise. A view of a system is a representation of the system from the perspective of a viewpoint. This viewpoint on a system involves a perspective focusing on specific concerns regarding the system, which suppresses details to provide a simplified model having only those elements related to the concerns of the viewpoint [46] [47].

U.S Government encouraged the researchers to develop the frameworks for defense side applications and it leads to the C4ISR Architecture Framework in 1996. ‘The Department of Defense Architecture Framework (DODAF)’ was released in 2003, which restructured the C4ISR framework ver2.0. The restructured C4ISR framework ver2.0 was released as, ‘The Department of Defense Architecture Framework (DODAF)’ in 2003 [48] [49].
‘The Open Group Architecture Framework (TOGAF)’ was developed by the members of open architecture forums in 1995. Recently in 2009, TOGAF Version 9 was released [50]. To integrate its myriad agencies and functions under single common and enterprise architecture, the ‘Federal enterprise Architecture Framework (FEAF)’ was developed in 1999 by the Federal Government [51]. ‘Treasury Enterprise Architecture Framework (TEAF)’ was developed to support the Treasury’s business processes in terms of products of the US Department of Treasury and published in July 2000 [52].

A reference model RM-ODP was developed by Andrew Herbert in 1984. It combined the concepts of abstraction, composition and emergence on the distributed processing developments. By including the set of UML profiles in the ODP and UML4ODP was released in 2004 [53]. In 2001, Aspect oriented programming boom out by inheriting the principles of Object oriented programming. And, it led to the Aspect oriented software development in late 2002.

The International Business Machines Corporation (IBM) announced ‘Service Oriented Modeling Architecture (SOMA)’ in 2004 that oppose the distributed processing and Modular programming. It is the first publicly announced SOA related methodology. In addition to this, to provide tactical and strategic solutions to enterprise problems, the Service Oriented Modeling Framework (SOMF) ver 1.1 was released by Michael Bell [54] [55].

### 2.2. SOFTWARE ARCHITECTURAL STYLES

Taylor et al. [56] defined an architectural style as a “named collection of architectural design decisions that (i) are applicable in a given development context, (ii) constrain architectural design decisions that are specific to a particular system within that context, and (iii) elicit beneficial qualities in each
resulting system.” An architectural pattern is defined as a “named collection of architectural design decisions that are applicable to a recurring design problem parameterized to account for different software development contexts where that problem appears.” Albin [57] listed six categories of architectural styles, namely data-centered architectural style, data flow architectural style, virtual machine architecture, call and return architecture, independent component architecture, and heterogeneous architecture style. The following subsections explain a few styles in detail.

2.2.1 Data-Centered Architectural Style

This style deals with systems with a centralized data store, where data access and manipulation is done by several clients. It has two styles based on the mode of communication. They are repository and blackboard styles. Repository systems, as shown in fig 2.2, may have multiple clients or subsystems.

![Fig 2.2 Repository system](image)

Each client may be a thin or thick client. If shared data is needed, it will be maintained in the centralized data center called the repository. The manipulation of data can be done by the clients in the repository. In addition to
the repository system’s functionality, a blackboard system notifies changes to subscribers. For this, the centralized data store will maintain a list of subscribers. These styles aim for the integrability of data. Clients and the data store are independent of each other. These styles are scalable and modifiable.

2.2.2 Data Flow Architecture Style

This style deals with the data flow inside the system. Here data enters into the system, flows through the subsystems or components once, and reaches the final output or (central) data store. The data will be transformed as it passes through the components or subsystems. This style can also be categorized into two types. They are batch sequential and pipe–filter styles.

In a batch sequential style, after getting processed or transformed in a component or subsystem, a batch of data will be released for the next component or subsystem. It means that only one component or subsystem will be functioning with a particular batch data at a time.

![Batch and Sequential style](image)

Fig 2.3 (a) Batch and Sequential style

In pipe–filter systems, filters are the components that do the computations. Pipes do not maintain the process states but move data between filters. In fig 2.3, one end of the pipe is connected to the output end of the previous filter and the other end is connected to the input end of the next filter. A series of filters can do incremental transformation of data. Filters can act standing alone. Hence, this style provides simplicity, reusability, and better maintenance. Moreover, it is easy to make the filters function in parallel and distributed modes. Both pipes and filters can be hierarchically arranged. A package of pipes and filters can act as a filter in this case.
2.2.3 Call and Return Architecture Style

For the past 50 years, this style was the most commonly used pattern in computer systems. Main programs and subroutines, remote procedure calls, object-oriented systems, and layered systems involve the mechanism of procedure call or function call. So, they all come under the call and return architecture style.

In the main program and subroutine architecture style, the program may be divided into subroutines and each subroutine will do a particular job. These subroutines may form a hierarchical decomposition as shown in fig 2.4. The main program will call a subroutine, the subroutine will do a computation or process, and control will be given back to the main program after the completion of the computation. This style aims for simplicity, modifiability, traceability, and reuse.
A remote procedure call system is a variation of a main program and subroutine system. The subroutines will reside in other systems connected in a network. Computations may be distributed, so that the load in a single system will be reduced. However, heavy computation slows down the communication.

In an object-oriented system, objects will bundle the data and the methods to manipulate or access the data. The data cannot be directly accessed from the outside world. The access can be done through the permitted methods available in the objects. This method is common and provides data abstraction, encapsulation, overriding, and inheritance. It aims for modularity, reusability, modifiability, and transparency of implementation.

The most important architecture which is dealt in this research is the layered architecture. The system comprises many layers and each layer provides a set of services with the help of the proper interconnection of components. Each layer can use certain services from the neighboring layers. Based on the services the layers provide, this style is categorized as closed architecture and open architecture. If the layer uses only the services of the
layer immediately below, then it will be called a closed architecture. If the layer uses the services of any of the lower-level layers, then it is known as an open architecture. This style mainly aims for modifiability and portability.

In a client–server model, the client machine will make a call to the routine available in the server and the server will do all the computations and return to the client. This model may have further layers with their own processing logic.

2.2.4 Independent Component Architecture Style

Independent component architecture deals with the communication of messages among the independent components, objects, and processes where the computations live. Each component sends data between them and do not control the operation of other components. This style is further classified into communicating process style and event style.

The client–server style can also be brought under the communicating process style, when the client sends a request to the server and the server returns the data or control to the client. To communicate, the client must know the name of the server. This is given in fig 2.5.

![Fig 2.5 Independent Component Architecture Style](image_url)
In the event style, the messages are passed to unnamed participants, which enables decoupling. This style is also known as publish/subscribe style. Here the components can publish their data to the subscribed components through a message manager as depicted in fig 2.6. At first, the components should register themselves with the manager to publish or receive data. When a component is ready to deliver data, it will publish data to the manager. The manager will then notify or forward the messages to the registered subscriber for the particular data.

![Publish/Subscribe Architecture style](image)

**Fig 2.6 Publish/Subscribe Architecture style**

### 2.2.5 Heterogeneous Architectural Style

To solve many design problems, different styles are combined. Garlan and Shaw [58] suggested three different ways of combining various architectural styles. According to Garlan and Shaw, the first way suggests that a system designed in one style may have components with different styles. Simply, it conveys that at one level of abstraction the system may have one style, but at another level of abstraction it may have a different style. In the second way, the architectural style of a system can be viewed or can be described in another style. An example of this is given in the previous subsection. The client and server pattern may be considered as call and return style, but when it comes with the data transfer communication model it may be
interpreted as independent component architecture style. The third way is the
system in one style may have certain components or subsystems with different
styles, and at the same time can be viewed as another style with another subset
of components or subsystems. Zhu [59] called these combined architectural
styles developed by the above mentioned ways as hierarchical heterogeneous
style, simultaneously heterogeneous style, and locationally heterogeneous
style respectively.

2.2.6 Broker Architectural style

The broker architectural pattern is similar to the pipe and filter pattern
because of the information processing at certain components and the
reusability of its components. The Common Object Request Broker
Architecture (CORBA) [60], standard defined by the Object Management
Group is “designed to facilitate the communication of systems that are
deployed on diverse platforms.” [61] This type of pattern would be optimal in
distributed systems [62].

The broker component acts as the middleware between the client and
the server as represented in fig 2.7.

![Broker style diagram](image)
It forwards the request from clients to servers and gets back the reply from the server. The broker uses unique identifiers to locate the requestor and the server. The Broker style consists of clients, servers, brokers, bridges, client-side proxies, and server-side proxies.

The brokers should register themselves with the server. Then only clients can use the brokers through method interfaces, to get services from the server. The broker will receive the service requests, identifies the server that provides the requested services and forwards the requests to the servers. If the requested server is in sleep mode, then the broker will activate the server before forwarding requests. The Broker Architecture style provides transparent process and distribution. It also ensures the location transparency. Testing and debugging will be easy in this style. The component’s behavior cannot be predicted when they are in dynamic systems.

2.2.7 Service-Oriented Architectural Style

Service-oriented architecture (SOA) provides a set of services for the creation and development of applications. As these services use standards-based interfaces, they are loosely coupled. These interfaces can be invoked, published and discovered. Through these interfaces, the services provide message based interaction and schema to the application. These services should not be taken into account as a component-based service provider. The range of protocols and data formats is used to communicate the information for packing the business processes into interoperable services. The following fig 2.8 represents the service oriented architecture style. These services are autonomous, distributable on a network and share contracts and schemas during their communication. This style improves the business opportunities and reduces the cost by reusing the common services. In order to make other applications to find out the appropriate services and interfaces, services can be with their description.
45

Fig 2.8 Service Oriented Architecture Style

As the protocols and data formats adhere industry standards, the service provider and consumer may be deployed on different platforms.

2.2.8 Model View Controller Style

Model View Controller style separates the user interface and functionality of the system. So, it is the most well known for interactive systems. The functionality of the system will not be changed when interface design gets changed. This style allows keeping many different user interfaces under a single data and logic model. The data and processing logic of functionalities are available in the model component. The main job of the model component is to synchronize the components as shown in fig 2.9 [63]. It registers other components and propagates the changes to other components. The View component is to retrieve data from the model component. The data may be presented to the user in different formats. The Controller component processes all user input. This component will manage events and convert them into other component’s requests. Both the View component and the Controller component implement an update method for the change-propagation mechanism. Separation of concerns reduces the complexity and increases
maintainability. The changes in the Model component affect the entire system. This style is suitable for implementing web applications.

![Diagram of Model View controller Style](image)

**Fig 2.9 Model View controller Style**

### 2.2.9 Presentation-Abstraction-Control architectural style

The Presentation-Abstraction-Control architectural style (PAC) may be seen as a small variation of the MVC style. The abstraction and presentation component corresponds to the model and combination of the view and control components of MVC respectively. The control component acts as a mediator between agents. This style consists of a hierarchy of cooperating agents for interactive software systems as shown in fig 2.10. Specific functionality of application will be assigned to each agent. Every agent has three components that are presentation, abstraction, and control. An agent is a component that has event receivers and transmitters, data structures to maintain a state and a processor that receives events, update its state and may deliver new events.
The abstraction component of the top-level PAC deals with the global data model and functions that manipulate the global data model of the software. As the data within the abstraction component is media independent, this enables the agent to be adapted to different environments with minor changes. The control component of the top-level PAC agent allows lower-level agents to access and manipulate the global data model through the abstraction or presentation component. It manages the agent’s hierarchy and maintains top level agent to low level agent’s connection information. It maintains user interaction information with the system.
Bottom-Level Agents

Bottom-level PAC agents represent a specific semantic concept of the application domain as low-level as a simple that summarizes all the data in the system. The presentation component in this level provides a specific view of the concept and provides access to all the user’s functions on the application. The abstraction component does same job of the top-level PAC agent except that other PAC agents cannot rely on this data.

Intermediate-Level Agents

Intermediate-Level PAC agents group agents that have a simple representation, to form a composite representation. It maintains consistency between lower-level agents when coordinating multiple views of the same data. The abstraction component maintains the specific data of the intermediate-level PAC agent. The presentation component implements its user interface. The control component does the similar job like the component of bottom-level and top level agents.

2.3. SURVEY OF VARIOUS FRAMEWORKS

Classification is the problem of identifying which of a set of categories a new observation belongs to. As the frameworks were developed under the interests of different field masters, they were influenced by various perspectives. So, it is necessary to classify them as whether they are developed by standard bodies or individual interests or by private agencies.

The frameworks developed by standard bodies fall under the standard category and others fall under non standard category. And also they are subcategorized based on their usage in commercial or Government purpose as depicted in fig 2.11.
Frameworks developed and used for the Government departments and for Defense side applications are classified under the Government frameworks. Frameworks used for commercial purpose are classified under the commercial frameworks.

The Open Distributed model ISO RM-ODP falls under the standard and commercial frameworks. DODAF, FEAF and TEAF which were developed for the U.S Government agencies are coming under the standard and government frameworks. The well accepted and most widely used frameworks, TOGAF and Zachman frameworks are used by both the commercial and government agencies.

Even though TOGAF and Zachman frameworks are falling under non-standard category, mapping of these frameworks to DODAF, FEAF and other standard frameworks yielded good products in the industry. The classification described in this section will be very much useful for the customer to choose the suitable framework quickly for his organization based on the job nature.
also. The next subsection deals the comparison parameters that can be used by the customer to choose the appropriate tool. The following section analyses the well known frameworks and lists out their criteria.

In this research, we have taken the survey of few frameworks which are most widely used. The parameters used for comparison in existing surveys are not suitable for a customer to choose the tool. So, the methodologies, techniques and tools used in these frameworks are considered for the comparison.

**Zachman Framework**

The Zachman Framework describes the complex thing in different ways using different types of descriptions. It provides thirty-six categories to describe anything completely.

**Views / Viewpoints:** It has six different views to facilitate each player to view the system in their own particular way.

- Planner's View (Scope)
- Owner's View (Enterprise or Business Model)
- Designer's View (Information Systems Model)
- Builder's View (Technology Model)
- Subcontractor View (Detailed Specifications)
- Actual System View

**Domain:** It mainly focuses on Categorizing Deliverables [31].

**Origin:** This framework is well suited for Manufacturing Industries [31].

**Focus:** It focuses mainly on Business process.
Phase of SDLC: In the Design stage or planning stage, it can be used [31].

System development methodology: Organization’s own methodology can be followed.

System modeling Technique: OMG-Model driven Architecture, Organization’s own technique

Business Modeling Technique: BPML is used for this framework.

Advantages:

- Provides improved professional communications within community [64].
- Understanding the reasons for and risks of not developing any one architectural representation [64].
- Provides variety of tools and/or methodologies [64].
- Developing improved approaches [64].

Weakness:

- It may lead to more documentation depending on the cases [65]
- It may guide to a process-heavy approach to development [65].
- It isn’t well accepted by all the developers [65].
- It seems in its first appearance as a top-down approach to developers. [65].
- It is to be biased towards traditional and data-centric techniques. [65].

NATO Architecture Framework / C4ISR / DoDAF

The Department of Defense Architecture Framework (DoDAF) provides the organization of an enterprise architecture (EA) into consistent
views. It is well suited for large complicated systems and interoperability challenges. "Operational views" used here are to deal with the external customer's operating domain.

**Views / Viewpoints:** DoDAF provides multiple views, each of which describes various aspects of the architecture. DoDAF defines the following views:

- Overarching All View (AV).
- Operational View (OV).
- Systems View (SV).
- Technical Standards View (TV).

**Domain:** It mainly focuses on operating domain [31].

**Origin:** This framework is developed for Defence [31].

**Focus:** It focuses mainly on Architecture data and Business process.

**Phase of SDLC:** It is used in a Process or Planning stage [31].

**System development methodology:** The Framework does not advice the use of any one methodology. It depends on the organization’s decision.

**System modeling Technique:** If the system to be developed is larger, then UML tools are likely the best choice.

**Business Modeling Technique:** IDEF Family

**Advantages:**

- Defines a common approach for describing, presenting, and comparing DoD enterprise architectures [48].
- Common principles, assumptions and terminology are used [48].
- Across the organizational boundaries architecture descriptions can be compared [48].

- Deployment costs and reinvention of same system can be reduced. [48].

**Weakness:**

- No common ontology of architecture elements [66].

- Baseline (current) and objective (target) architectures are not addressed [66].

- How the architectures can be used to measure effectiveness is not dealt [66].

- Business-financial plans are not addressed. [66].

**TOGAF**

The Open Group Architecture Framework (TOGAF) provides a comprehensive approach to the design, planning, implementation, and governance of enterprise information architecture.

**Views / viewpoints** : TOGAF identifies many views to be modeled in an architecture development process. The architecture views, and corresponding viewpoints come under the following categories:

- Business Architecture Views

- Information Systems Architecture views

- Technology Architecture views

- Composite views

**Domain:** It mainly focuses on Business, data and applications [31].
Origin: This framework is developed due to the motivation in Defence side framework.

Focus: It focuses mainly on Business process, Data, applications and Technology.

Phase of SDLC: It is used in a Process or Planning stage [31].

System development methodology: Rational Unified process (RUP) is used as a system development Methodology.

System modeling Technique: UML, BPMN are widely used in TOGAF system modeling.

Business Modeling Technique: IDEF is used for business modeling in TOGAF

Advantages:

- Increased transparency of accountability [67].
- Controlled risk [67].
- Protection of assets [67].
- Proactive control [67].
- Value creation [67].

Weakness:

- Lots of Detail [68].
- Planning methods and governance framework [68].
- Weak on Information Architecture [68].
- Can lead startup efforts into too much too soon [68].


**TEAF**

Treasury Enterprise Architecture Framework (TEAF) was developed by the US Department of the Treasury and published in July 2000. It is based on the Zachman Framework. The Treasury Enterprise Architecture Framework (TEAF) supports Treasury’s business processes in terms of products. This framework guides the development and redesign of the business processes for various bureaus.

*Views / Viewpoints: It provides four different views.*

- Functional Views
- Information View
- Organizational View
- Infrastructure View

*Domain:* It has a domain on Business processes [52][31].

*Origin:* This framework is developed for Treasury department [52].

*Focus:* It focuses mainly on Business process.

*Phase of SDLC:* It is used in a communication or Planning stage [31].

*System development methodology:* It does not refer any specific methodology. It depends on the organization’s decision.

*System modeling Technique:* Flow chart, UML can be used in TEAF.

*Business Modeling Technique:* IDEF, ERD can be used as business modeling techniques.
Advantages:

- Provides the guidance to the treasury bureaus and offices in satisfying OMB and other federal requirements [52].
- Support Treasury bureaus and offices based on their individual priorities and strategic plans [52].
- Leads to Treasury-wide interoperability and reusability [52].

Weakness:

The TEAF does not contain a detailed description of how to generate the specification documents (work products) that are suggested for each cell of the TEAF Matrix [69].

FEAF

Federal Enterprise Architecture (FEA) was developed for the Federal Government to provide a common methodology for information technology (IT) acquisition, use, and disposal in that Federal government. It was built to develop a common taxonomy and ontology to describe IT resources. The FEAF provides documenting architecture descriptions of high-priority areas. It guides to describe architectures for functional segments in multi-organization manner of the Federal Government.

Views / Viewpoints: Like zachman framework, FEAF is also having five different views in its framework.

- Planner’s View (Scope) and Owner’s View (Business Model)
- Designer's View (Information Systems Model)
- Builder's View (Technology Model)
- Subcontractor’s View (Detailed Specifications)
Domain: It has a domain on provision of services [31].

Origin: This framework is well suited for Enterprise Architecture planning.

Focus: It focuses mainly on Business process, Data, Application and Technology.

Phase of SDLC: It is used in a Communication or Planning stage [31].

System development methodology: RUP (Rational Unified process) is followed in FEAF.

System modeling Technique: UML is used as a system modeling tool in FEAF.

Business Modeling Technique: BPML is the technique used in FEAF.

Advantages:

- Serve customer needs better, faster, and cost effectively [51].
- Promote Federal interoperability [51].
- Promote Agency resource sharing [51].
- Reduced costs for Federal and Agency [51].
- Improve ability to share information [51].
- Supports capital IT investment planning in Federal and Agency [51].

Weakness:

- The Federal Government could risk allocating too much time and resources to an enterprise architecture description effort yielding potentially little return at significant cost [51].
• The Federal Enterprise Architecture program requires technical and acquisition expertise [51].

• The Federal IT community must keep its eyes on the basic principles rather than near-term objectives and achievements [51].

• The Federal Government has to pay up-front for the right to exercise options in the future [51].

• Concern over territoriality and loss of autonomy may impede the Federal Enterprise Architecture effort due to long-term, realignment of Agency functions and responsibilities [51].

• It is hard to have common, cross-Agency models and standards to ensure interoperability [51].

**ISO RM-ODP**

The ISO Reference Model for Open Distributed Processing provides a framework standard to support the distributed processing in heterogeneous platforms. Object modeling approach is used to describe the systems in distributed environment.

**Views / Viewpoint:** The five viewpoints described by RM-ODP are:

• The enterprise viewpoint

• The information viewpoint.

• The computational viewpoint

• The engineering viewpoint

• The technology viewpoint

**Domain:** It has a domain on information sharing in distributed environment.
**Origin:** This framework is well suited for major computing and telecommunication companies.

**Focus:** It focuses mainly on Business process, Technical Functionality and Solution.

**Phase of SDLC:** It is used in a Processing and communication stage.

**System development methodology:** Object oriented method and IAD can be followed here [53].

**System modeling Technique:** UML, OMG (Model driven Architecture) are used as system modeling techniques [53].

**Business Modeling Technique:** BPMN is used as business modeling technique in RM-ODP.

**Advantages:**

- It provides lot of details for the analysis phases of the development of applications [53].
- It provides the platform to integrate the requirements from different languages consistently. [53].
- It provides a set of established reasoning patterns to identify the fundamental entities of the system and the relations among them. It provides the appropriate degrees of abstraction and precision for building useful system specifications [53].
- It provides a set of mechanisms and common services to build robust, efficient and competitive applications, interoperable with other systems [53].
Weakness:

RM-ODP has the problem of inter-view and inter-view consistency. A number of cross-view checks to be done to maintain the consistency. These checks don’t guarantee the consistency [70].

Comparative chart of frameworks

The following table 2.1(a) and (b) describe the comparison between all the frameworks discussed in the preceding pages.

<table>
<thead>
<tr>
<th>S/N</th>
<th>Framework</th>
<th>ZACHMAN FRAMEWORK</th>
<th>DoDAF</th>
<th>TOGAF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No of Views</td>
<td>Six</td>
<td>Four</td>
<td>Four</td>
</tr>
<tr>
<td>2</td>
<td>Domain</td>
<td>Categorizing deliverables</td>
<td>Operating domain</td>
<td>Business, Data and Applications</td>
</tr>
<tr>
<td>3</td>
<td>Origin</td>
<td>In- Manufacturing</td>
<td>Defence</td>
<td>Defence</td>
</tr>
<tr>
<td>4</td>
<td>Focus</td>
<td>Business process</td>
<td>Architecture Data, Business process</td>
<td>Business process, Data, Applications, Technology</td>
</tr>
<tr>
<td>5</td>
<td>Phase of SDLC</td>
<td>Planning (Design)</td>
<td>Process/Planning</td>
<td>Process/Planning</td>
</tr>
<tr>
<td>6</td>
<td>System development methodology</td>
<td>Organization’ own methodology</td>
<td>Organization’ own methodology</td>
<td>RUP</td>
</tr>
<tr>
<td>7</td>
<td>System modeling technique</td>
<td>DMG-Model driven Architecture, Organization’s own technique</td>
<td>UML</td>
<td>UML, BPMN</td>
</tr>
<tr>
<td>8</td>
<td>Business model technique</td>
<td>IDEF</td>
<td>IDEF Family</td>
<td>IDEF Family</td>
</tr>
</tbody>
</table>
| 9   | Advantages | ● Improving professional Communications  
● wide variety of tools  
● improved approaches for Architectural representations | ● common Approach  
● common principles, assumptions and terminology  
● comparable architecture descriptions across organizational boundaries  
● reduction of deployment costs | ● increased transparency of accountability  
● controlled risk  
● protection of Assets  
● proactive Control  
● value creation |
| 10  | Weakness   | ● documentation heavy approach  
● process heavy approach to development  
● seems like Top down Approach  
● biased towards traditional, data centric techniques | ● No common ontology of architecture elements  
● not addressing baseline and objective architectures  
● not addressing capabilities and measures of effectiveness  
● lack of business financial artifacts | ● lots of detail planning methods and governance framework  
● weak on information  
● Architecture can lead startup efforts into too much too soon |
| 11  | Tools      | ● Adaptive EA Manager  
● Mega V6.1  
● SystemArchitect V10  
● Simon Tool | ● EA Webmodeler  
● Corporate Modeler  
Enterprise Edition V10  
● SystemArchitect V10  
● Metis product family | ● System Architect I0  
● Metastorm ProVision  
EA V6.0  
● IDS Scheer  
● EA Webmodeler |
Table 2.1(b) Architectural Frameworks comparative chart

<table>
<thead>
<tr>
<th>S/N</th>
<th>Framework TERMS</th>
<th>TEAF</th>
<th>FEAF</th>
<th>ISO RM-ODP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No of Views</td>
<td>Fou</td>
<td>Five</td>
<td>Five</td>
</tr>
<tr>
<td>2</td>
<td>Domain</td>
<td>Business processes</td>
<td>Provision of services</td>
<td>information sharing in distributed environment</td>
</tr>
<tr>
<td>3</td>
<td>Origin</td>
<td>Treasury Department</td>
<td>Enterprise Architecture planning</td>
<td>major computing and telecommunication companies</td>
</tr>
<tr>
<td>4</td>
<td>Focus</td>
<td>Business processes</td>
<td>Business process, Data, Applications &amp; Technology</td>
<td>Business process, Technical functionality &amp; Solution</td>
</tr>
<tr>
<td>5</td>
<td>Phase of SDLC</td>
<td>planning / communication</td>
<td>Planning &amp; communication</td>
<td>Processing &amp; communication</td>
</tr>
<tr>
<td>6</td>
<td>System development methodology</td>
<td>Organization &amp; own methodology</td>
<td>RUP</td>
<td>Object oriented method, IAD</td>
</tr>
<tr>
<td>7</td>
<td>System modeling technique</td>
<td>Flow chart, UML</td>
<td>UML</td>
<td>UML, OMG(Model driven Architecture)</td>
</tr>
<tr>
<td>8</td>
<td>Business model technique</td>
<td>IDEF, ERD</td>
<td>BPML</td>
<td>BPMN</td>
</tr>
<tr>
<td>9</td>
<td>Advantages</td>
<td>• satisfying OMB support individual Priorities and strategic Plans interoperability and reusability</td>
<td>• serve customer needs better, faster and cost effectively • promote federal Interoperability • provide agency resource sharing • reduced costs • improve ability to share information • support Federal and agency capital IT investment planning</td>
<td>• improved requirement collection and analysis phases • consistently integrated requirements expressed in separate languages • set of already established reasoning patterns • used for building robust, efficient and competitive applications • backed by industrial products with enough acceptance</td>
</tr>
<tr>
<td>10</td>
<td>Weakness</td>
<td>• No detailed description of Specification document for each cell • Missing the techniques for creating specification document</td>
<td>• little return at significant cost • need technical and acquisition expertise • need a watch on future • less future Maneuverability • loss of autonomy may impede effort • difficult to ensure interoperability</td>
<td>• problem of inter-view Consistency • Not a truly guaranteed cross-view checks • No precise notion of Consistency</td>
</tr>
<tr>
<td>11</td>
<td>Tools</td>
<td>• EA Webmodeler • Corporate Modeler Enterprise Edition V10 • FEAMS V0.2 • Metis product family</td>
<td>• Adaptive EA Manager • Flashline4 • FEAMS V0.2 • SystemArchitect V10</td>
<td>• ConsVISor • TINA • Simon Tool • MagicDraw</td>
</tr>
</tbody>
</table>
These tables summarize the frameworks based on the industry side criteria and it discusses the benefits and drawbacks of each framework. Additional information of available supportive tools is also presented.

The comparative chart can be used as the reference guide during architecture design process. It will invoke the user to choose the right framework based on business needs. Users can easily identify the supporting tools available for their frameworks.

The frameworks analyzed here are not addressing the impact of non functional requirements anywhere. So, the next section reviews the literature to see how the architecture analysis has been done on the building architecture sciences like vastu, feng chui, wabi sabi and wu-wei.

2.4. ANCIENT INDIAN ARCHITECTURAL SCIENCE - VASTU

Architectural Science is a discipline with an emphasis on applying methods from science and engineering to the investigation of architecture, human environments and design processes. There are many well known architectural sciences like Feng Shui, vastu, wabi sabi, wu wei, etc. All these sciences focus on getting the optimal beneficial results in life from the universal life energy within a space.

Feng Shui has five elements that are Earth, Metal, Wood, Fire and Water, to represent a particular direction [71]. In Feng Shui, South facing Plots are considered auspicious and much attention is given to south, as warm air is coming from south and south east in china. To experience the positive results and to prevent the freezing air from Northern zone, high hills and mountains along northern region is preferred by Fen chui. Hence it is popular in china and the states of many freezing regions. Fen chui has an Octagonal shape universal Cosmo grid called Ba Gua.
However in Vastu, we have five elements termed as Water, Fire, Earth, Wind and Space, which are the only elements exist on earth. Without anyone of it, life cannot exist. In feng Shui, there is Metal and wood which is also the Earth element. In Vastu, more attention is given to as cosmic energy comes from North east due to the prana energy obtained from the 22.5 degree tilt of the earth.

The “Area “on which they apply it differentiates these two sciences. However in Vastu, it is universally applicable regardless of where we live under any conditions. Cosmic energy comes from Northeast (More open space) and it should be stored in Southwest (Less space and windows). Magnetic energy flows from North to South. So, sleeping towards South gives good sleep and good health. Solar energy comes from the Sun from East to West. Cooking faced towards East will attain the Vitamin D of Morning sun. To conclude about these sciences, Vastu Shastra is the origin of Feng Shui. Vastu Shastra can be applied regardless of religion and beliefs as it is a science of cosmos. Vastu can be applied to any place as it is based on the ayadi calculation.

Wabi Sabi is the Japanese aesthetic philosophy that gives importance to elegance, beauty, serenity, simplicity, a meditative quality, and appreciation of and connection with nature. Vastu is compatible with the goals and intentions of Wabi Sabi. Both view beauty as an essential component. The sadness and melancholy associated with wabi-sabi is not supported by Vastu, but its true underlying meaning is realized even more through Vastu. Wabi Sabi involves with the spiritual and physical truth of the world [72]. Vastu acknowledges that buildings have a lifespan and it relates the building with the unchanging source of all manifestation. As the spiritual energy of Wabi Sabi suggestions, there is no true "perfection" in this life; but, buildings designed according to Vastu maintain the perfection derived from the laws of nature. This mathematical perfection allows us greater connection with nature and greater
inner freedom and spiritual experience, the underlying goal of wabi-sabi. Both Vastu and Wabi Sabi include beauty, honoring of nature, and use of natural materials as necessary ingredients.

Likewise, Vastu guidelines support the Taoist concept of wu wei, which is based on the ability to intuit placement that creates a good flow of energy and is in alignment with the Tao [73]. Since the Tao and pure consciousness, the source of the creation process and laws of nature, are different expressions of the same universal consciousness, Vastu is in agreement with Wu Wei.

Comparing with all these sciences, vastu is the one that tries to balance the natural elements and forces to provide the maximum benefits. The next subsection briefs about the origin of vastu and basic principles.

2.4.1 Vastu

The word ‘vastu’ is derived from the Sanskrit word ‘vas’ meaning a place to dwell [74]. Vastu is essentially an art of living in a setting from which one can derive maximum benefits. Vastu Shastra was introduced and used 2000 years ago. It is a proven architectural science, which is based on the effects of natural forces and elements. In-depth knowledge of natural sciences and their relationships helped the ancient sages to derive these concepts long before the advent of the modern science. The kings and sculptors used Vastu Shastra to plan cities and to build temples and palaces.

It was believed that the Mayan, an Indian sage, was the father of Vastu architecture. He was an architect and a town planner of ancient India. His books include Mayamata Vastu Shastra (architecture), Surya Siddanta (a treatise on Astronomy), and Aintiram (the grammar of astro-mechanics).

Vastu Vidya is the study of effective utilization of natural elements in the living place. Vastu Shastra deals with the dos and don’ts in Vastu Vidya. If a person has an in-depth knowledge of Vastu Vidya, he can change the Vastu according to his requirements.
Vastu Shastra is based on (i) the scientific principles relating to the sun, (ii) the five basic elements (air, water, earth, fire, and space), (iii) the magnetic property of the earth, (iv) rotation of the earth, and (v) the eight directions [75].

**Mandalas**

Vastu Shastra lists many mandalas. These mandalas are defined in a graph. Vastu stipulates that any human-built structures should follow the same patterns to avail the maximum benefits. Some mandalas have been prescribed as structures for specific purposes that humans can use. Nowadays, Vastu Purusha Mandala is most widely used, which is actually a Paramasayika Mandala. The term Vastu Purusha Mandala is named on the basis of the research work of the western scholar Kramrisch [76].

These thirty-two mandalas are based on one of the two patterns given in fig 2.11 and 2.12. A total of sixteen mandalas follow the first pattern, whereas the remaining sixteen follow the second pattern. The two design patterns are as follows:

(1) The central object with zero to fifteen levels may fall into this pattern. Mathematically this may be represented as \( \{x=True \text{ and } n=0, 1, 2\ldots 15\} \) where \( x \) denotes the existence of the central object and \( n \) denotes the number of levels it may possess.

Fig 2.12 pattern 1

(2) Levels that are concentric or overlap each other may fall into this pattern. Mathematically this may be represented as \( \{x=False \text{ and } n=1, 2\ldots 16\} \) where \( x \) denotes the existence of the central object and \( n \) denotes the number of levels it may possess.

Fig 2.13 pattern 2

This chapter dealt in detail software architecture in terms of patterns, styles and frameworks. Characteristics of frameworks are compared and tabulated. Frameworks focused only on functional requirements, and there is little focus on nonfunctional requirements, even though nonfunctional
requirements will have large impact on the system under development. This
gap set the tone for research towards quality attributes of software
architecture. Various architectural sciences have been investigated and
compared, and the principle behind vastu is presented. The next chapter deals
with the quality attributes which play major role in the architectural design
process.