Nowadays in the industry, there is a high demand of software systems with the increase in complexity and facing the challenges of quality requirements, which emphasizes on architecture and implementation mechanism.

The proposed approach helps the software architects to find an efficient way of dealing with requirements related to quality attributes of the system for implementing the suitable design pattern. The Architecture forms the basis of achieving quality attributes of the system.

Quality attributes are non-functional requirements of products which helps stakeholders to judge the quality of the system. Some of the quality attributes include usability, availability, configurability, interoperability, portability, performance, modifiability, security, reliability etc. Degree of a system to meet the quality attribute requirements is dependent on its design and architecture.

> Decision making has an important role in the creation of Software Architecture. Such decisions are required throughout the software development life cycle and to attain the quality attributes of a software product. Decisions which are made during architectural design have direct impact on software quality. Many design patterns are available in software architecture and these design patterns help achieve quality attributes in a software product but decision-making for the solution of these design patterns is still an issue. In this paper, a new perspective of selection of design pattern for a given problem is presented. PRIC (Problem, Requirement Identification, Compatible design pattern) model is proposed which helps decision-making of selection of design pattern to attain the required quality attributes.

PRIC Model: This model explains the proposed process of decision making in Software Architecture. PRIS Models has been shown in Fig. 1. Main phases of this process are as follows:

A. Problem

Understanding problem is the most important part of any software Architecture process. Layout of Software Architecture depends on identifying and understanding problem statement.

B. Requirement Gathering

After understanding problem, requirements are gathered. Requirement elicitation is one of the main things in Software Architecture and Development. Most of the software Projects fail only because of poor requirement elicitation. Gathering complete requirements using proper requirement elicitation techniques is one of the main success factor in Software Projects.

C. Identifying Functional & Non- Functional Requirements

After Requirement Elicitation one of the most important task is to identify and categorize functional and Non-functional requirements, so that decisions are taken accordingly.

D. Identifying Quality Attributes.

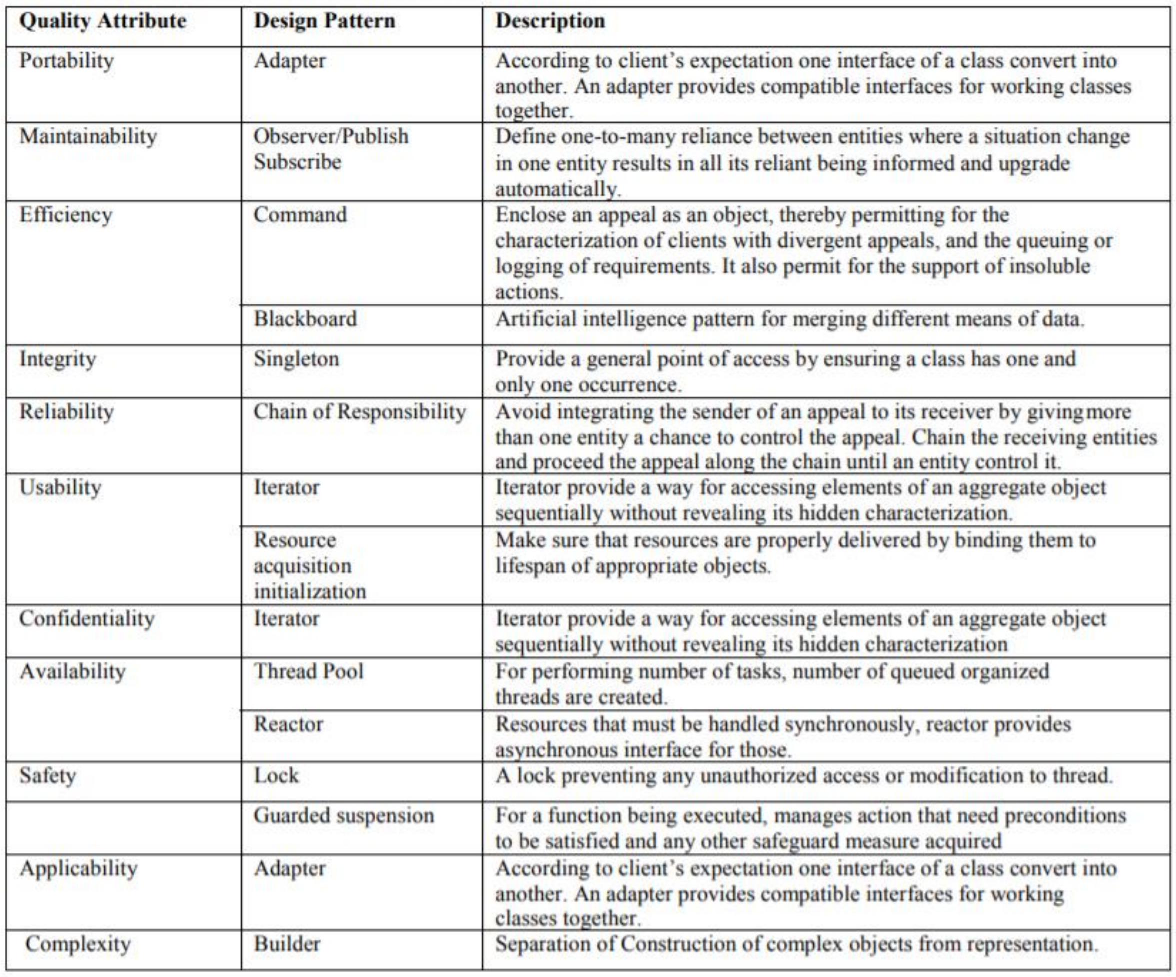
Quality attributes are identified after separately identifying functional & Non- Functional requirements. Quality attributed are identified and prioritized on the basis of requirements.

E. Compatible Design Patterns for Quality Attributes

Selecting Compatible Design Pattern according to identified quality attribute is the main decision. Choosing wrong design pattern can cause Software failure, but if chosen rationally it can help out and ease the task of Software Architect.

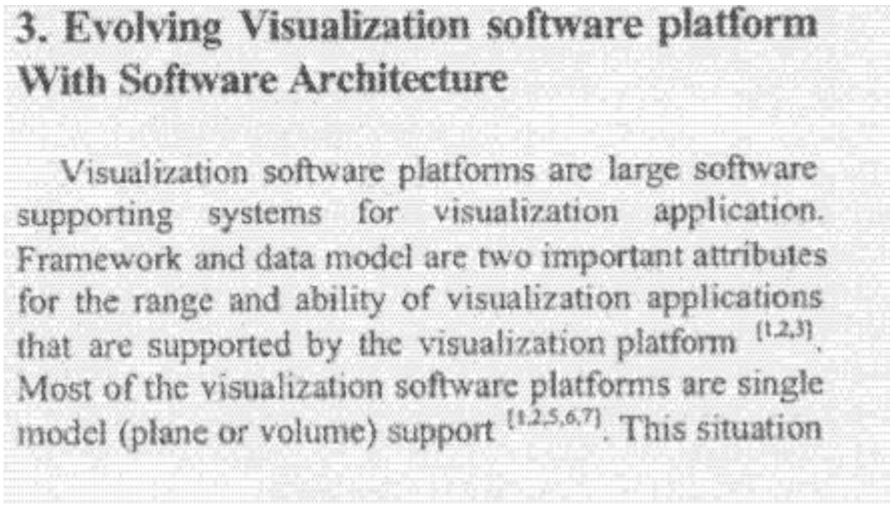
To explain the further applicability and importance of PRIC Model related to some quality attributes is being discussed along with design pattern suggested for each attribute in Table 1. which will make the process of decision making easy in Software Architecture phase

Table 1: Suggested Design Patterns for Quality Attributes



Quality Attribute	Design Pattern	Description
Portability	Adapter	According to client's expectation one interface of a class convert into another. An adapter provides compatible interfaces for working classes together.
Maintainability	Observer/Publish Subscribe	Define one-to-many reliance between entities where a situation change in one entity results in all its reliant being informed and upgrade automatically.
Efficiency	Command	Enclose an appeal as an object, thereby permitting for the characterization of clients with divergent appeals, and the queuing or logging of requirements. It also permit for the support of insoluble actions.
	Blackboard	Artificial intelligence pattern for merging different means of data.
Integrity	Singleton	Provide a general point of access by ensuring a class has one and only one occurrence.
Reliability	Chain of Responsibility	Avoid integrating the sender of an appeal to its receiver by giving more than one entity a chance to control the appeal. Chain the receiving entities and proceed the appeal along the chain until an entity control it.
Usability	Iterator	Iterator provide a way for accessing elements of an aggregate object sequentially without revealing its hidden characterization.
	Resource acquisition initialization	Make sure that resources are properly delivered by binding them to lifespan of appropriate objects.
Confidentiality	Iterator	Iterator provide a way for accessing elements of an aggregate object sequentially without revealing its hidden characterization
Availability	Thread Pool	For performing number of tasks, number of queued organized threads are created.
	Reactor	Resources that must be handled synchronously, reactor provides asynchronous interface for those.
Safety	Lock	A lock preventing any unauthorized access or modification to thread.
	Guarded suspension	For a function being executed, manages action that need preconditions to be satisfied and any other safeguard measure acquired
Applicability	Adapter	According to client's expectation one interface of a class convert into another. An adapter provides compatible interfaces for working classes together.
Complexity	Builder	Separation of Construction of complex objects from representation.

2)



3. Evolving Visualization software platform With Software Architecture

Visualization software platforms are large software supporting systems for visualization application. Framework and data model are two important attributes for the range and ability of visualization applications that are supported by the visualization platform [1,2,3]. Most of the visualization software platforms are single model (plane or volume) support [1,2,5,6,7]. This situation makes many modern visualization platforms are unable to support those visualization applications that need both plane and volume processing ability. SGItm's Open Inventortm is a famous visualization platforms to separate scene data from application code [5,6,7], but it also has limitation for not be able to process volume data in application. To solve above problems, we need to use our software architecture model to extend Open Inventor from single data model application support platform to multi data model application support platform without change its character of separating scene data from application code.

3.1. System Analysis for Software Architecture

The first step of our work is to define Open Inventor's normal running environment and designing goals. By reading the software's application guide, we got that:

1) Open Inventor is an Object Oriented visualization support platform developed in C++ programming language;

2) Open Inventor is able to separate scene data from application code;

3) Open Inventor is supporting point and plane data model and is not able to process volume data set;

The second step of our work is to analyze Open Inventor's software architecture with our model in section 2(to make things simple and clearly, here we use indirect graph to describe our work. In our work, we use rectangle to denote component and indirect line denote connecter).

With data structure and codes of Open Inventor, we find out the following facts:

1) Open Inventor has a core composed of a group of objects (List, Action and Engine), the core schedules and controls the application that set up by the platform;

2) Besides programming codes, Open Inventor uses plain text files to save scene data. The scene data files' extended names are ".IV" and scene data in ".IV" file are organized as a scene tree with some special key words (such as: Separator, Group, Node, Field etc.);

3) Open Inventor uses a special input object to read "Scene data" from ".IV" file. The input object use file I/O protocol to read data from scene file;

4) With the data from Input object, Open Inventor's core uses two protocols (event and procedure call) to interact with other components to generate a scene tree for process. The scene tree generated here are composed of a group of scene objects which corresponding to the key words in .IV files;

5) Based on the scene, Open Inventor's core has also another mechanism for recording the state of the scene application (this mechanism is compose of State, Element and DB objects);

6) Also to the scene tree, Open Inventor's core use event trigger and procedure call to control a group objects to render the scene;

7) Except the Input object, the communication among Open Inventor objects are all realized by event message and procedure call;

From the above description, we get Open Inventor's software architecture in figure 1.

3.2. System Evolution Based on Software Architecture

After we had gotten Open Inventor's software architecture model, the third step of our work is to evolution the architecture with new data model (volume data) processing ability.

1) We firstly defined a new data component from Open Inventor's old data component. The component not only keeps all the old ".IV" file's structure and scene node's key word interfaces, but also adds a key word of new interface (volume). This makes the interface in new ".IV" file provide key information for real volume data process. Obviously, the new data component is inherited from Open Inventor's old data component.

2) Corresponding to the new data component, we have also generated a new "scene generate" component. An new component which is also inherited from the original comp onent in Open Inventor, is of course able to process both plane and volume data;

3) Similar to 2), A new "scene tree" is inherited from the original component in Open Inventor, the new component includes volume info inside it;

4) Because "scene tree" component is inherited from the original component in Open Inventor, "system state" component and "system control" component need not change their way to record and control the system's actions;

5) Because the algorithms for volume rendering are very different from those for plane data, a new rendering component is generate by an inheritance from the old rendering component. From this new component, more rendering components are generated for multi volume rendering algorithms.

From above description, we get an evolved software architecture in figure 2.

Conclusion:

- Software architecture plays an important role in the construction and maintenance of software.

- If the software architecture is properly developed it will help in the discovery of architectural bad smells, i.e., architectural choices that have detrimental effects on system lifecycle properties.

- Well developed architecture enables more reliable assessment of system quality attributes like performance, security, interoperability, reliability, availability.

Hence, in this paper, a model named PRIC is proposed. This model presents the process of problem identification, requirement gathering, extraction of function and non- functional requirement, and then identifying quality attributes along with the selection of suitable design pattern according to the required quality attributes. Since every software starts from a specific problem domain, it is dependent on the software developers to identify the problem, specify the requirements from the problem domain according to the customer's satisfaction.