**Birla Institute of Technology and Science, Pilani**



**SOFTWARE ARCHITECTURE**

**Assignment**

**Section: C**

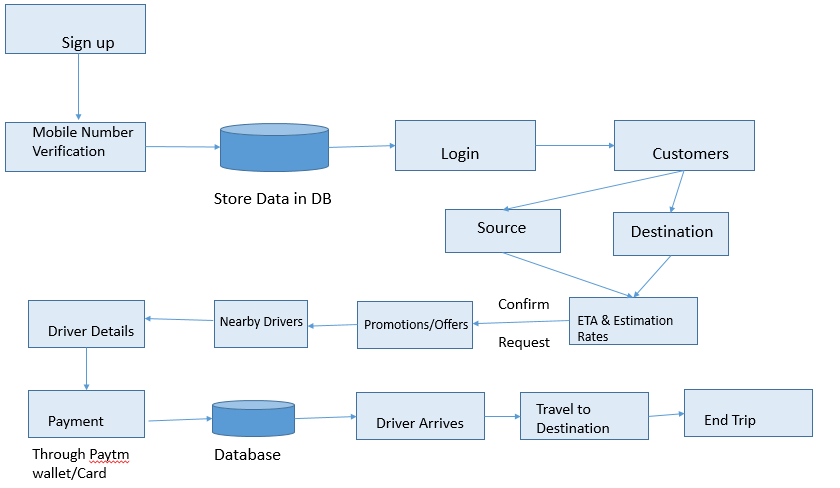
**Team No: 5**

**Topic: Online Cab Booking System**

|  |  |  |
| --- | --- | --- |
| SL.No | Team Members | Bits-email |
| 1 | Leenu Thomas | 2015HW70166@wilp.bits-pilani.ac.in |
| 2 | Ajeemsha Vn | 2015HW69392@wilp.bits-pilani.ac.in |
| 3 | Ajith R | 2015HW69395@wilp.bits-pilani.ac.in |
| 4 | Filna Rodrigues | 2015HW69727@wilp.bits-pilani.ac.in |
| 5 | Amal Baby | 2015HW70009@wilp.bits-pilani.ac.in |
| 6 | Archana Das | [2015HW70040@wilp.bits-pilani.ac.in](mailto:2015HW70040@wilp.bits-pilani.ac.in) |

**Online Cab Booking System**

**Architectural Pattern for Online Cab Booking system:**



We have chosen the above Architectural pattern, which will help others to understand the working flow of the online cab booking system. This pattern clearly showed how the User will book the cab, how to sign up, how to provide the pickup and drop locations, etc. New User will register his/her number in the application. Then, login to application. Make sure the user provided both pick-up and drop location. The system will estimate the arriving time, destination arrival time and the rate to pay for the cab. If the User is eligible for any offers or promotions, it will apply. Nearby cabs will be searched by the system and if any cabs are available, the system will displays the Cab/drivers details to the User. Payment can be through Paytm or card or cash. The cab will drop you in the destination location and End of the Trip. This is the brief understanding will get from the architectural pattern given above.

**Quality Attribute for Online cab booking system:**

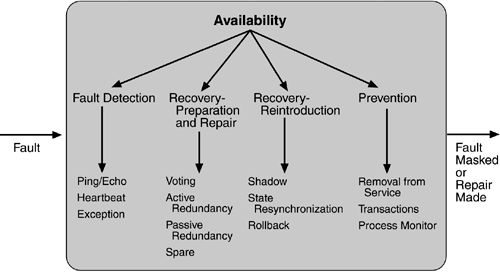
The main quality attribute required in online cab booking system is;

1. Availability
2. Modifiability
3. Performance
4. **Availability:**

A failure occurs when the system no longer delivers a service that is consistent with its specification; this failure is observable by the system's users. A fault (or combination of faults) has the potential to cause a failure. Recall also that recovery or repair is an important aspect of availability. Fundamentally, availability is about minimizing service outage time by mitigating faults.

In Online cab booking system, the availability required are Riders/cabs, hardware and software. If these are not available, this system can lead to a fault/failure.

**Tactics for Availability:**

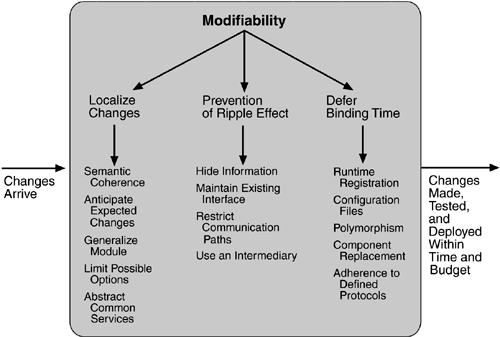


1. **Modifiability:**

Modifiability is the degree of ease at which changes can be made to a system, and the flexibility with which the system adapts to such changes. It supports the communication between software architects and software engineers since it captures the earliest and most important design decisions. Modification will happens as per the requirement change and deployments.

In Online cab booking system, the modifiability required are Application developers, testers and admins. If the requirements changed and the code changes affects the existing functionality, and the testers couldn’t find it. Then, the application is a fault/failure.

**Tactics for Modifiability**:



1. **Performance:**

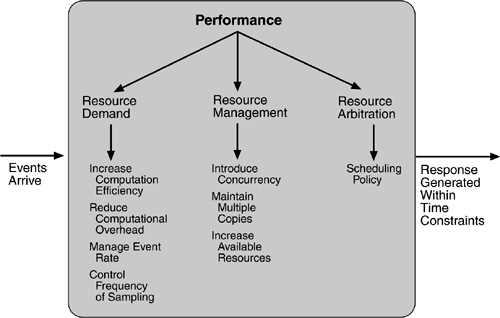
Performance is about time and the software system’s ability to meet timing requirements. When events occur – interrupts, messages, requests from users or other systems, or clock events marking the passage of time – the system, or some element of the system, must respond to them in time. Characterizing the events that can occur (and when they can occur)

and the system or element’s time-based response to those events is the essence is discussing performance.

The goal of performance tactics is to generate a response to an event arriving at the system within some time constraint. The event can be single or a stream and is the trigger for a request to perform computation. It can be the arrival of a message, the expiration of a time interval, the detection of a significant change of state in the system's environment, and so forth. The system processes the events and generates a response. Performance tactics control the time within which a response is generated.

In Online cab booking system, the modifiability required are Customers, network connections and other external systems. When the Customer is available and try to connect to cab booking system, if the network is not available at that time, it can be failure.

**Tactics for Performance:**



**Introduction on Online cab booking system:**

Online cab booking system is a software which is most trusted and reliable for the travel business, which help both cab drivers and the passengers. The passengers can book the cab to a particular destination from his/her current location in this system. The passenger can view how many cabs are available nearby them and if the cab is booked, he/she can view how much time the cab will take to reach at their current locations. The passengers can schedule his/her cab with a specific route and for a specific day. The passengers can view the cab driver details and contact numbers, after booking the trip. After the trip, bill is generated and send to both cab driver and the passenger.

There are mainly six modules in this software:

1. Admin
2. HR manager
3. Maintenance manager
4. Movement manager
5. Finance manager
6. Quality Assurance manager

In this online cab booking system, Admin is responsible for the creation and maintenance of the accounts to the system, also responsible for the creation of different kind of managers and looks after the maintenance of these accounts. HR manager is responsible for registering the employees for the cab facilities and after registering, he/she will assign the shift and batches to the employee’s/cab drivers. Also maintenance of cab drivers profile, shifts and batches are taken care by HR manager in this system. Maintenance manager is responsible for maintaining and adding vehicles, drivers, vendors of the vehicles, spare part details of the cab and its billing.

Movement manager is responsible for the maintenance of the driver shift details, route details, trip sheets and Vehicle allocated details. Finance manager is responsible for the cost estimation of the vendor, vehicle billing and viewing the feedbacks posted by the employees on the events as accidents and the driving nature of a particular vehicle/cab drivers. Quality Assurance Manager is responsible for inserting and maintaining the Accident details, by this details he requests for the finance manager to provide the Amount to the damage.

Online cab booking system is most trusted and reliable for the travel business, which help both cab drivers and the passengers. The passengers can book/schedule the cab and can view the details of the cab drivers and also can view the arrival time of the cab to the passenger’s current location. The passengers can view the estimated price of the trip and can provide the feedback/rating of the particular trip. This system also helps the cab drivers/employees to view the shift details, batch details, passenger contact details, cab routes, billing details, working status, etc. This project intends to introduce more user-friendly approach in the various activities such as record updating, maintenance, and searching. This system will works between the passengers and the cab drivers and used for benefiting the passengers and to provide detailed information regarding Cab routes and cab driver details.

**Learning goals of the research topic:**

Online Cab Booking system is an online portal through which customers can view available cabs, register the cabs, view profile and book cabs. A computer based management system is designed to handle the entire primary Information required to manage the whole data. Separate database is maintained to handle all the details required for the correct statement calculation and generations. This project intends to introduce more user-friendly approach in the various activities such as record updating, maintenance, and searching.

Researching on this project, we have gone through Software Architecture, all the Quality attributes, all types of diagrams, tactics, etc. An architecture pattern allows us to define a guide for the ‘architecture’ of a software system, making it scalable, maintainable and testable. Differing from design patterns, these have a major abstraction level.

First, you need to understand what’s the meaning of every component of this architecture and this will helpful for getting concepts from the next patterns that are going to be mention on this article.

* **Model**: contains models of business logic, i.e. the classes that will model the solution for the problem and objective of the application. For example, if you are developing an app for a library, the ‘Book’, ‘Author’ and ‘PublishingHouse’ classes should be in this package. All of them should be independent from any Android component.
* **View**: is an interface that is going to be implemented by all the UI views, that’s it. Activities, Fragments, Adapters, Dialogs, etc. All of these are in this package and are responsible on displaying the data to the user.
* **Controller**: is responsible for ‘listening’ user inputs and update the model and the view. Normally, the view and controller are involved in the same java class, I mean, an activity accomplish the functions as a view and controller at the same time because it is listening (for example) user clicks and shows answers for them. The best idea is to have the controller in a separate class. However, this relationship makes the activity hard to test because the Testability best practice is not being used and we have user interactions and view responses at the same class. Moreover, the view and the controller can both access to the Model, which is not a good practice at the end because you have the model relationship with more than one class and this is not (too) scalable.

**Some of the quality attributes:**

Availability, Modifiability, Performance, Security, Testability, Usability.

**Availability:**

Availability is concerned with system failure and its associated consequences. A system failure occurs when the system no longer delivers a service consistent with its specification. Such a failure is observable by the system's users—either humans or other systems. A failure occurs when the system no longer delivers a service that is consistent with its specification; this failure is observable by the system's users. A fault (or combination of faults) has the potential to cause a failure. Recall also that recovery or repair is an important aspect of availability. Fundamentally, availability is about minimizing service outage time by mitigating faults.

**Modifiability:**

Modifiability is the degree of ease at which changes can be made to a system, and the flexibility with which the system adapts to such changes. It supports the communication between software architects and software engineers since it captures the earliest and most important design decisions. Modification will happens as per the requirement change and deployments.

**Performance:**

Performance is about timing. Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them. One of the things that make performance complicated is the number of event sources and arrival patterns. A performance scenario begins with a request for some service arriving at the system. Satisfying the request requires resources to be consumed. While this is happening the system may be simultaneously servicing other requests. An arrival pattern for events may be characterized as either periodic or stochastic. Performance is to generate a response to an event arriving at the system within some time constraint. The event can be single or a stream and is the trigger for a request to perform computation. It can be the arrival of a message, the expiration of a time interval, the detection of a significant change of state in the system's environment, and so forth. The system processes the events and generates a response. Performance tactics control the time within which a response is generated.

**Security:**

Security is a measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users. Security can be characterized as a system providing nonrepudiation, confidentiality, integrity, assurance, availability, and auditing.

1. Nonrepudiation is the property that a transaction (access to or modification of data or services) cannot be denied by any of the parties to it.
2. Confidentiality is the property that data or services are protected from unauthorized access.
3. Integrity is the property that data or services are being delivered as intended.
4. Assurance is the property that the parties to a transaction are who they purport to be.
5. Availability is the property that the system will be available for legitimate use.
6. Auditing is the property that the system tracks activities within it at levels sufficient to reconstruct them.

**Testability:**

Software testability refers to the ease with which software can be made to demonstrate its faults through testing. Testability refers to the probability that it will fail on its next test execution. For a system to be properly testable, it must be possible to control each component's internal state and inputs and then to observe its outputs.

**Usability:**

Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides. It can be broken down into the following areas:

1. Learning system features. If the user is unfamiliar with a particular system or a particular aspect of it, what can the system do to make the task of learning easier?
2. Using a system efficiently. What can the system do to make the user more efficient in its operation?
3. Minimizing the impact of errors. What can the system do so that a user error has minimal impact?
4. Adapting the system to user needs. How can the user (or the system itself) adapt to make the user's task easier?
5. Increasing confidence and satisfaction. What does the system do to give the user confidence that the correct action is being taken?

**Summary:**

**Software Architecture**

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

* It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
* Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product.

These decisions comprise of −

* + Selection of structural elements and their interfaces by which the system is composed.
  + Behaviour as specified in collaborations among those elements.
  + Composition of these structural and behavioral elements into large subsystem.
  + Architectural decisions align with business objectives.
  + Architectural styles guide the organization.

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows −

* Expose the structure of the system, but hide its implementation details.
* Realize all the use-cases and scenarios.
* Try to address the requirements of various stakeholders.
* Handle both functional and quality requirements.
* Reduce the goal of ownership and improve the organization’s market position.
* Improve quality and functionality offered by the system.
* Improve external confidence in either the organization or system.

Quality attribute requirements such as those for performance, security, modifiability, reliability, and usability have a significant influence on the software architecture of a system. Architects need to understand their designs in terms of quality attributes. For example, they need to understand whether they will achieve deadlines in real time systems, what kind of modifications are supported by their design and how the system will respond in the event of a failure. There are large and thriving attribute communities that study various quality attributes but they each have their own language and sets of concepts. However, architects tend to think in terms of architectural patterns. What the architect needs is a characterization of architectural patterns in terms of factors that affect the various quality attributes so that a software design can be understood in terms of those quality attributes.

Quality is a measure of excellence or the state of being free from deficiencies or defects. Quality attributes are the system properties that are separate from the functionality of the system. Implementing quality attributes makes it easier to differentiate a good system from a bad one. Attributes are overall factors that affect runtime behavior, system design, and user experience.

They can be classified as –

**Static Quality Attributes**

Reflect the structure of a system and organization, directly related to architecture, design, and source code. They are invisible to end-user, but affect the development and maintenance cost, e.g.: modularity, testability, maintainability, etc.

**Dynamic Quality Attributes**

Reflect the behaviour of the system during its execution. They are directly related to system’s architecture, design, source code, configuration, deployment parameters, environment, and platform.

They are visible to the end-user and exist at runtime, e.g. throughput, robustness, scalability, etc.

# Quality attributes types (ISO/IEC FCD 25010 diagram)

To consider the types of quality attributes, we can use a [diagram from ISO 25010](http://iso25000.com/index.php/en/iso-25000-standards/iso-25010):



This standard describes the quality attributes of a software product.

### **Common Quality Attributes**

The following table lists the common quality attributes a software architecture must have –

|  |  |  |
| --- | --- | --- |
| **Category** | **Quality Attribute** | **Description** |
| Design Qualities | Conceptual Integrity | Defines the consistency and coherence of the overall design. This includes the way components or modules are designed. |
| Maintainability | Ability of the system to undergo changes with a degree of ease. |
| Reusability | Defines the capability for components and subsystems to be suitable for use in other applications. |
| Run-time Qualities | Interoperability | Ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties. |
| Manageability | Defines how easy it is for system administrators to manage the application. |
| Reliability | Ability of a system to remain operational over time. |
| Scalability | Ability of a system to either handle the load increase without impacting the performance of the system or the ability to be readily enlarged. |
| Security | Capability of a system to prevent malicious or accidental actions outside of the designed usages. |
| Performance | Indication of the responsiveness of a system to execute any action within a given time interval. |
| Availability | Defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. |
| System Qualities | Supportability | Ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly. |
| Testability | Measure of how easy it is to create test criteria for the system and its components. |
| User Qualities | Usability | Defines how well the application meets the requirements of the user and consumer by being intuitive. |
| Architecture Quality | Correctness | Accountability for satisfying all the requirements of the system. |
| Non-runtime Quality | Portability | Ability of the system to run under different computing environment. |
| Integrality | Ability to make separately developed components of the system work correctly together. |
| Modifiability | Ease with which each software system can accommodate changes to its software. |
| Business quality attributes | Cost and schedule | Cost of the system with respect to time to market, expected project lifetime & utilization of legacy. |
| Marketability | Use of system with respect to market competition. |

**Performance** shows the response of the system to performing certain actions for a certain period.

There **are two ways how to measure performance**:

* **Latency:** Time spent on responding to an event
* **Channel capacity**. The number of events that occur at a certain point in time.

In practice, **the possible performance indicators** include, for example:

* Average/maximum number of system users per time unit.
* Average page load time.
* Average method execution time.

Performance issues very often grow into problems that can affect everything, from the server’s capacity or how you develop your front-end to the efficiency of database queries or the capacity of communication channels.

Performance is almost always included in the list of critical quality attributes that need to be considered by the architect since it affects the entire system and can affect many parts of the architectural solution

**Interoperability** is an attribute of the system or part of the system that is responsible for its operation and the transmission of data and its exchange with other external systems. A well-designed system facilitates integration with third-party systems. To improve the interoperability, you can use well-designed external interfaces, standardization systems, etc.

Naturally, there are a lot of problems for interaction:

* Outdated external systems.
* Different formats of data in similar external systems.
* Different versions of the API in external systems.
* Backward compatibility of the API for integration.
* Poor quality and lack of standards of external systems.

Interoperability cannot be ignored. In the best case, you will have to create additional layers for the interaction API. At worst, it will be necessary to rebuild the entire system.

**Usability** is one of the most essential attributes, because, unlike in cases with other attributes, users can see directly how well this attribute of the system is worked out. One of the critical problems of usability is too much interaction or too many actions necessary to accomplish a task. Incorrect sequences of steps in multistage interfaces are also a problem of usability. Data elements and controls may be designed not according to the accepted patterns of user experience, which also complicates the interaction.

Examples of important indicators for this attribute are:

* List of supported devices, OS versions, screen resolutions, and browsers and their versions.
* Elements that accelerate user interaction, such as “hotkeys,” “lists of suggestions,” and so on.
* The average time a user needs to perform individual actions.
* Support of accessibility for people with disabilities.

**Reliability** is an attribute of the system responsible for the ability to continue to operate under predefined conditions. Most often, the system fails due to the inaccessibility of external elements, such as databases, systems, and [network](https://hackernoon.com/tagged/network) connections.

**Availability** is part of reliability and is expressed as the ratio of the available system time to the total working time. Important indicators for this attribute are:

* Availability.
* Planned downtime.
* The time needed to update the software, and so on.

Availability is often expressed in the number of nines after the comma that is nines of availability (hours / minutes / seconds):

* 2 9’s (99%) = up to 87.6h / 5256.0m / 315360.0 seconds of downtime per year.
* 3 9’s (99.9%) = up to 8.76h / 525.6m / 31536.0 seconds of downtime per year.
* 4 9’s (99.99%) = up to 0.876h / 52.559999999999995m / 3153.6 seconds of downtime per year.
* 5 9’s (99.999%) = up to 0.0876h / 5.256m / 315.36 seconds of downtime per year.
* 6 9’s (99.9999%) = up to 0.00876h / 0.5256000000000001m / 31.536 seconds of downtime per year.
* 7 9’s (99.99999%) = up to 8.76E-4h / 0.05256m / 3.1536 seconds of downtime per year.

**Security** is responsible for the ability of the system to reduce the likelihood of malicious or accidental actions as well as the possibility of theft or loss of information. Several measures are used to protect systems: authentication, encryption, audit, and others.

Examples of this attribute in the work of the system are:

* The ability of the system to detect DDoS attacks and respond to them.
* Restrictions of user access by authentication/authorization.
* Prevention of SQL injection.
* Encryption of passwords and content.
* Secure connection.

**Maintainability** is the ability of the system to support changes. Changes can be related to new business requirements or the correction of old errors and affect system components or separate methods. Also, maintainability affects the time needed to restore the system after a failure. Excessive dependencies between components have a very negative effect on maintainability. In programming, there is a notion of anti-pattern [spaghetti code](https://en.wikipedia.org/wiki/Spaghetti_code), which means excessive coherence in the code. In architecture, there is no such thing, but architecture is very close to programming in this sense. It is because of the maintainability attribute that such concepts as separation of responsibility, microservice architectures, and modularity have appeared. At the same time, this attribute affects not only development processes but also management processes (for example, splitting teams into product-related parts).

**Modifiability** determines how many common changes need to be made to the system to make changes to each item. The ideal is the case where each change affects only one element.

**Testability** shows how well the system allows performing tests, according to predefined criteria. In addition to testing performance, testability makes it possible to effectively divide the system into subsystems.

The main indicators for this attribute are:

* Percentage of coverage with modular, integration, or unit tests.
* The final list of required test environments as well as the final list of used approaches to testing (manual/automatic, regression, integration, etc.).

**Scalability** is the ability of the system to handle load increases without decreasing performance, or the possibility to rapidly increase the load.

There are two ways to improve scalability:

* **Vertical**: To increase, we add more resources, such as memory, disks, or processors into one system.
* **Horizontal**: We increase the number of computing units and divide the load.

The key indicators for measuring this attribute are:

* If the system allows for horizontal scaling.
* The time needed to increase scaling, in seconds.
* Scaling limitations: the number of servers or the network capacity.
* Possibility to scale: the increase in the number of transactions or the amount of content.

And this is only a small part of the indicators which you need to follow when designing. Scalability is one of the most essential attributes, no matter what is the project’s stage.

**Reusability** is a chance of using a component or system in other components/systems with small or no change. Segregation of responsibilities, modularization, decreasing of copy-paste are all about reusability. Copying code, or worse, using different components for the same result in different modules, is one of the biggest problems of reusability.

**Supportability** is the ability of the system to provide useful information for identifying and solving problems. The main issues in ensuring supportability can be addressed with the following means:

* **No diagnosis:** How the activity and performance of the system are controlled. It includes various types of logging.
* **No tools for troubleshooting:** This includes backups, various systems for creating snapshots of the system, and tools for auditing the system. When the system fails, it is always more pleasant to wait for an automatic restart than to solve the issue manually.
* **No health checking:** This includes a variety of systems for measuring compilation time, deployment time, database size, or mobile application size.

Most often these are not considered in start-ups or small projects initially. The cost of maintaining the supportability attribute is high, and the result is only visible on a large scale. However, with the growth of the team and the product, this attribute becomes one of the important ones.

**Methods used for learning:**

We have used below methods for our learning:

* Google
* Test books
* Reference books
* Youtube
* Recorded sessions
* Contact sessions notes

**List all the references you have used top learn:**

1. https://sites.google.com/site/misresearch000/home/software-architecture-quality-attributes
2. <https://www.slideshare.net/rickkazman/sap3-chapter-8>
3. <http://www.users.abo.fi/lpetre/SA11/SALecture2.pdf>
4. <http://www.ece.ubc.ca/~matei/EECE417/BASS/ch05lev1sec4.html>
5. <http://www.informit.com/articles/article.aspx?p=1959673&seqNum=5>
6. <http://www.se.rit.edu/~swen-440/slides/instructor-specific/Kuehl/Lecture%2017%20Quality%20Attributes%20and%20Tactics%20LV.pdf>
7. <https://elearn.bits-pilani.ac.in/user/>
8. <https://www.cs.unb.ca/~wdu/cs6075w10/sa2.htm>

**Conclusion**

The development of this project took efforts to understand how Cab Booking systems work in order to facilitate their daily businesses. Information Technology plays a vital role not only in a particular field, it provides various kinds of solutions and services to the various problems prevailing in many fields. Cabs exploits information technology at the maximum extent. It uses the information technology in an efficient way for providing better passenger services.

Moreover it helped us to understand more on software architecture and its various quality attributes and related tactics on it. Also it reminded us, how Software Architecture plays a vital role in day to day life.