Volume: 9 Issue: VI Month of publication: June 2021
DOI: https://doi.org/10.22214/ijraset.2021.35580

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue VI Jun 2021- Available at www.ijraset.com

How to Choose the Best Architecture Design Pattern

Kartik Gupta[1], Devesh Bhogre[2], Sanchita Biswas[3], Kshitish Deshpande[4], Bhavana Tiple[5]
[1, 2, 3, 4]B.Tech., [5]Assistant Professor, School of Computer Engineering & Technology,
Dr. Vishwanath Karad MIT World Peace University, Pune

Abstract - The utilization of engineering and configuration designs affect the quality credits of a framework, and the use of examples rely upon plan settings. There are unpredictable reliant connections between them. In this examination, we investigate how engineers use to design and configuration designs regarding quality credits concerns and plan settings. We extricated design-related posts from Stack Overflow and broke down the engineering conversations. Our examination uncovers what settings and quality ascribe engineers consider when utilizing design examples, and we have recognized new and already obscure connections between these plan components. These discoveries can improve engineers' information when they plan with design designs, quality ascribes, and plan settings.
Keywords—architecture design pattern, design, application, software design, software architecture

I. INTRODUCTION

A compositional example is a general, reusable answer for a usually happening issue in programming design inside a given context.[1] The building designs address different issues in computer programming, for example, PC equipment execution restrictions, high accessibility and minimization of a business hazard. Some building designs have been carried out inside programming systems.

The utilization of "design" in the product business was impacted by comparable ideas as communicated in conventional engineering, like Christopher Alexander's A Pattern Language (1977) which talked about the training as far as building up an example dictionary, inciting the professionals of software engineering to consider their plan vocabulary.

Use of this representation inside the programming calling got ordinary after the distribution of Design Patterns (1994) by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides—presently normally known as the "Pack of Four"— incidental with the early long stretches of the public Internet, denoting the beginning of complex programming frameworks "eating the world"[2] and the comparing need to classify the quickly rambling universe of programming advancement at the most profound conceivable level, while staying adaptable and versatile.

Even though a design passes on a picture of a framework, it's anything but engineering. An engineering design is an idea that addresses and depicts some fundamental strong components of product engineering. Innumerable various designs may execute a similar example and offer connected qualities. Examples are frequently characterized as "stringently depicted and generally accessible".[3][4]

II. LITERATURE SURVEY

[5]Compositional examples are a vital idea in the field of programming engineering: they offer grounded answers for design issues, help to record the compositional plan choices, work with correspondence between partners through typical argon, and depict the quality ascribes of a product framework as powers. Deplorably, finding and applying the proper compositional examples practically speaking remains generally specially appointed and unsystematic. This is because of the absence of agreement locally concerning the "theory" and granularity of building designs, just as the need for a sound example language. In this paper, we endeavour to set up shared belief in the building designs local area by proposing an example language that goes about as a superset of the current engineering design assortments and arrangements. This language is especially centred around building up the connections between the examples and plays out an order dependent on the idea of "building sees".

[6]Programming design audits are successful in distinguishing likely issues in structures, be that as it may, are costly, tedious, and for the most part, depend on broad engineering documentation. An engineering audit that obliges projects with extremely short advancement cycles, insignificant documentation, or regularly changing prerequisites could be valuable on the off chance that it recognizes significant design issues. We fostered a helpful, reasonable design survey technique that uses the engineering designs in a framework to distinguish significant issues in the accomplishment of value credits.

[7]Programming design has become a focal point subject for programmers, the two specialists and experts the same. At the core of each product framework is its product engineering, i.e., "the arrangement of chief plan choices about the framework".

Engineering penetrates all significant features of a product framework, for chief plan choices, maybe made whenever during a framework's lifetime, and conceivably by any partner. Such choices incorporate primary concerns, for example, the framework's significant level structure blocks - segments, connectors, and arrangements; the framework's sending; the framework's non-utilitarian properties; and the framework's development designs, including runtime variation. Programming designs found especially valuable for groups of frameworks - product offerings - are frequently systematized into compositional examples, building styles, and reusable, defined reference structures. This instructional exercise manages the cost of the member a broad treatment of the field of programming design, its establishment, standards, and components, including those referenced previously. Furthermore, the instructional exercise acquaints the members with the cutting edge just as the condition of the practice in programming design, and takes a gander at arising and likely future patterns in this field. The conversation is outlined with various true models. One model given unmistakable treatment is the design of the World Wide Web and its hidden engineering style, Representational State Transfer (REST).

Engineering designs contain a bunch of rules of how to structure a framework into parts and connectors. They likewise address regular manners by which choices manage certain parts of an engineering plan. One of the significant goals of utilizing design designs is to structure frameworks with unsurprising Non-Functional Requirements (NFRs). NFRs are significant in enormous scope programming frameworks and they can be indicated as quality credits (QAs). Planners and fashioners need to guarantee that both practical and non-prerequisites are met by the plan. There are different engineering and configuration designs accessible for use. While choosing a design, quality credits and their compromises should be thought about however practically speaking they are frequently discarded.

Plan settings impact a framework plan from numerous points of view, for example, formative, mechanical, business, operational, social, and different impacts. An arrangement of comparable functionalities can work diversely in various settings. While plan settings are critical to settling on plan choices, the settings of a framework are regularly disregarded and a portion of the plan set may not be expressly caught in the prerequisites records. There is a restricted exploration of plan settings, and there is no standard meaning of this idea in the ISO 42010 norm.

Programming improvement questions and replies (Q&A) destinations (e.g., Stack Overflow, R people group, and GitHub) assemble information that covers a wide scope of points. These destinations permit designers to share insight, offer assistance, and learn new procedures. Stack Overflow (SO) is quite possibly the most well-known and famous online Q&A discussions. It contains a huge number of posts by a huge number of engineers. SO gives capacities, for example, restoring and altering posts that can be dormant for significant stretches. It upholds casting a ballot contending answers and clients can procure notoriety focuses by posting fascinating inquiries and answers. Late investigations show that designers and engineers utilize web-based media to talk about engineering significant data (e.g., highlights and space ideas). In this work, we exploited the engineering and plan conversations accessible in SO. We directed an examination of example presents in SO on acquiring experiences on how engineers consider engineering designs as for quality ascribes and plan settings. We are propelled to research this point because there are barely any works that give down to earth bits of knowledge to help engineers use plan and design designs with QA contemplations specifically settings. Regularly issue areas direct whether an example can be utilized basically. For example, monetary frameworks can exploit modifiability in MVC design however such a framework additionally needs to think often about execution which MVC would settle.

There are two primary commitments of this work. In the first place, engineers who search for data on the most proficient method to apply design designs as far as the QAs are additionally worried about explicit plan settings. For instance, the most habitually posed kind of plan inquiries is "would it be advisable for me to utilize this design in this application?". Second, engineers frequently need to know certain data when planning with design designs, for example, the connections among QAs and engineering examples, qualities and expected issues of utilizing an example. We distinguished and recorded these contemplations (in TABLE VIII) through dissecting the SO posts. It assists engineers with perceiving a portion of the QAs and the settings that should be viewed when utilizing engineering and configuration designs.

There has been a spray in the presentation of new shopper hardware items in the previous few years. A great deal of these items has covering/regular functionalities. There is additionally plenty of items that, when consolidated into a solitary item, offer improved client encounters (E.g. Customarily a convenient music player joined with a cell phone gives an upgraded client experience of stopping the music playback while getting an approaching call, a TV with an inbuilt DVD/Blu-beam Disk player tries not to have a fledgling client mess with the associations, a shrewd TV with web perusing, improved voice location and face acknowledgement abilities permit a more associated insight). On the framework engineering front, a proficient framework design should have negligible unwarranted intricacy. The decision of programming engineering and equipment design assumes a vital part in deciding the life span and re-ease of use of these items while decreasing the unwarranted intricacy. The test from the framework fashioner's point of view is to wed at least two autonomous items into a solitary item with joined and conceivably more use-cases, diminished opportunity to advertise, better quality and dependability, simpler support, without settling on the key partner's needs and existing client impression of the two free items. This paper means to give a bunch of CE-explicit framework engineering designs (like plan examples and programming engineering designs) to help the framework fashioner.

III. RELATED WORK

A. *Importance of Software Architecture Pattern*[8]

Programming design designs hold a great deal of significance as they can be utilized to take care of different issues inside various areas. Here are a couple of guides to show how these design designs are utilized to tackle regular issues:

- Complex client solicitations can be effectively portioned into more modest pieces and dispersed across numerous workers to complete work rapidly, rather than relying upon a solitary worker.
- Rearrangements of the intricate testing conventions to make it simple for engineers to test different portions of the product as opposed to testing the entire thing without a moment's delay.

Here are a few reasons why programming design designs are so significant for any product application:

1) *Characterizing the Basic Characteristics of an Application:* It has been seen that engineering designs help in characterizing the essential attributes and practices of an application. For example, some design examples can be normally utilized for profoundly adaptable applications, while others can be utilized for exceptionally dexterous applications. For picking the design designs that can meet your business destinations and requirements, it is essential to know the qualities, qualities, and shortcomings of every engineering.
2) *Keeping up Quality and Effectiveness:* At the point when an association decides to assemble an application, they need to guarantee that the picked designing group has their inclinations at the top of the priority list – both as far as quality and effectiveness. Any application that you fabricate may confront quality issues. The choice of programming design designs, as indicated by your product type, can assist you with limiting the quality issues while at the same time looking after productivity.
3) *Provide Agility:* Programming engineering designs even give deftness to the application. When the application is delivered, it may need to add a few highlights later on. As it is simpler to change a well-architected design than the product with helpless engineering, utilizing well-architected programming confirms that the cycle will be pretty much as easy as could be expected.
4) *Problem Solving:* Programming Architecture Patterns help computer programmers catch a very much demonstrated involvement with programming improvement and advance plan rehearses that are acceptable from each point of view. Moreover, every example manages a particular, repeating issue either in the plan or the execution of a product framework.
5) *Enhancing Productivity:* At the point when an organization employs another engineer for a venture, regardless of how well they compose a program, it takes them months to comprehend things before being useful. One of the arrangements is the normalization of programming dialects, which serves somewhat. Standard libraries, then again, can be another approach to handle this test. Likewise, programming designs appear to be significant when managing this issue. On the off chance that the specialists think about the engineering designs already, it turns out to be straightforward and handle what is happening in the task.

B. *Types of Software Architecture Pattern*

1) *Layered Architecture Pattern*

a) *Description:* You've likely effectively known about complex, also known as layered engineering, or n-level design. This engineering design has acquired prominence among planners and programming modelers the same, for it has a few shared traits with the ordinary courses of action of IT interchanges in numerous new companies and set up undertakings. Generally, layered engineering is characterized into four particular layers: show, business, steadiness, and information base; in any case, the example isn't bound to the predetermined layers. Suppose you're fabricating an enormous application. For this situation, you'd end up utilizing every one of the four layers or more to your product engineering design. On the other side, private ventures may consolidate the business and the determination layers into a solitary unit, particularly when the last is locked in as a fundamental piece of the business layer segments. What makes this example stand apart is that each layer assumes an unmistakable part inside your application and is set apart as shut, which implies a solicitation should go through the layer directly underneath it to go to the following layer.

Another of its ideas – layers of seclusion – empowers you to change parts inside one layer without truly influencing different layers.

To work on this interaction, how about we take an illustration of an eCommerce web application. The business rationale needed to deal with a shopping basket movement, for example, computation of the truck is straightforwardly brought from the application level to the show level. Here the application level goes about as a mixed layer to set up a consistent correspondence between the information and show layers. Also, the last level is the information level used to keep up information freely without the mediation of the application worker and the business rationale.

b) Usage

- Applications that are should have been fabricated rapidly.
- Venture applications that need to receive customary IT offices and interaction.
- Those groups have unpracticed engineers who are yet not ready to comprehend different models.
- Those applications which require exacting principles of practicality and testability

c) Shortcomings

- If a source code is chaotic and modules don't have clear jobs, it can transform into a major debacle
- At times the software engineers avoid past layers to make tight coupling and that prompts the creation of sensible wreck loaded with complex interdependencies
- At times the software engineers avoid past layers to make tight coupling and that prompts the creation of sensible wreck loaded with complex interdependencies

2) Event-Driven Architecture:

a) Description: On the off chance that you are searching for an engineering design that is dexterous and exceptionally performant, at that point you ought to pick an occasion driven design. It is comprised of profoundly decoupled, single-reason occasion handling parts that no concurrently get and measure occasions. This example organizes the conduct around the creation, identification, and utilization of the relative multitude of occasions alongside the reactions they summon. So, whether it's a little application or enormous, this profoundly versatile design is your go-to. Occasion driven example comprises of two geographies – arbiter and agent. A middle person is utilized when various advances are should have been arranged inside an occasion through a focal arbiter. Also, a Broker is utilized when the occasions are needed to be tied together without the utilization of a focal arbiter. Indeed, a genuine model that utilizes occasion driven engineering is an internet business website. The occasion driven design empowers the eCommerce site to respond to an assortment of sources at the hour of appeal. All the while, it dodges any accident of the application or any over-provisioning of assets.

b) Usage:

- For the applications wherein singular information blocks associated with a couple of modules
- Assists with UIs

c) Shortcomings:

- Testing singular modules must be done on the off chance that they are free, else they should be tried in a completely useful framework.
- At the point when a few modules are dealing with similar occasions, blunder taking care of gets hard to structure
- Advancement of framework wide information structure for occasions can turn out to be truly troublesome if the occasions have various necessities
- Since the modules are decoupled and free, keeping an exchange-based component for consistency gets troublesome.

3) Microkernel Architecture

a) Description: This engineering design comprises of two kinds of parts – center framework and module modules – where the center framework involves insignificant usefulness that is needed to make the framework operational. On the off chance that we take a business application's viewpoint, the center framework can be characterized as a broad business rationale without the custom code for unique cases, uncommon principles, or complex restrictive cycles. The other part, the module modules, is a bunch of free segments containing particular preparation, extra highlights, and custom code. These are intended to improve the center framework to create extra business capacities. For your arrangement, we can take an illustration of a common assignment scheduler. In this application, the microkernel could contain all the rationale for booking and setting off assignments, while the modules contain explicit undertakings. However long the modules hold fast to a predefined API, the microkernel can trigger them without having to realize the execution subtleties. Another illustration of a microkernel design is a work process. It contains ideas like the request for the various advances, assessing the consequences of steps, choosing what the subsequent stage is, and so on the particular execution of these means is less critical to the center code of the work process.

b) Usage

- For the applications that have a reasonable division between the fundamental schedules and higher-request rules

- The applications that have a fixed arrangement of center schedules and dynamic arrangement of deciding that necessities regular updates

c) Shortcomings

- The modules should have great handshaking code so that the microkernel knows about the module establishment and is prepared to work
- Changing a microkernel is troublesome or even unimaginable if there are various modules subject to it. The solitary arrangement is to make changes in the modules also.
- Even though it is hard to pick the correct granularity for the bit work ahead of time, it is considerably harder to transform it in the later stage.

4) Microservices Architecture Pattern

a) Description: You would be happy to find out about microservices design designs as it is a suitable option in contrast to solid applications and administration arranged models. The segments of the microservice engineering are sent as discrete units. This permits them to be conveyed effectively utilizing a successful and smoothed out conveyance pipeline, improved adaptability, and a serious level of use and segment decoupling inside the application. Since the parts are completely decoupled from each other, they are gotten to through some kind of far-off access convention. Likewise, these parts can be independently evolved, sent, and tried without interdependency on some other help segment. Allow us to give you knowledge by Adrian Cockcroft, the Director of Web Engineering and afterwards Cloud Architect of Netflix. "The video real-time application is one of the early adopters of the Microservice Architecture Pattern. Netflix went from a customary improvement model with 100 specialists creating a solid DVD-rental application to microservice engineering. This design permitted them to work in little groups answerable for the start to finish improvement of many microservices. These microservices cooperate to stream advanced amusement to a great many Netflix clients consistently."

b) Usage

- Organizations and web applications that require a quick turn of events
- Sites with little parts, server farms with clear cut limits, and where the groups are spread across the globe.

c) Shortcomings

- Planning the correct degree of granularity for an assistance segment is consistently a test.
- Every one of the applications does exclude undertakings that can be parted into autonomous units
- Errands that are spread between various microservices can adversely affect execution.

5) Space-based Architecture

a) Description: The idea of tuple space – disseminated shared memory is the premise of the name of this engineering. It includes two essential segments – preparing unit and virtualized middleware. Allow us to investigate it exhaustively. The preparing unit part contains segments of utilization segments including electronic segments and backend business rationale. All things considered, more modest web applications could be conveyed in a solitary handling unit. Then again, bigger applications could part the application usefulness into numerous handling units. Besides, the virtualized-middleware segment contains components that control different parts of information synchronization and solicitation taking care of. They can be exclusively composed or can be bought as outsider items. One of the benefits of this example is that there is no focal data set. This eliminates that bottleneck, giving close limitless adaptability inside the application. For your insight, the applications that get demands from the program and play out a type of activity finds a way into this example class. Allow us to disclose to you with the assistance of a model – an offering closeout site. The webpage gets offers from the web clients through a program demand. On getting the solicitation the site records that bid with a timestamp, refreshes the data of the most recent bid and sends the data back to the program.

b) Usage

- For the applications and programming frameworks that work under substantial heap of clients
- For the applications that need to address and tackle adaptability and simultaneousness issues.

c) Shortcomings

- It is hard to foster the abilities to reserve the information for speed without upsetting different duplicates.

IV. CHOOSING PATTERN

For example, if your essential objective is versatility, you can take a gander at the beneath outline to figure out which sort of engineering is the most ideal decision. Likewise, you can become more acquainted with the danger zones related with a specific design.

A. Agility

1) *Layered:* Low agility, cumbersome & time-consuming to make changes in the architecture pattern.
2) *Event-Driven:* High agile, changes are isolated, can be made quickly without dependency on other components.
3) *Microkernel:* Highly agile, and quick modifications possible due to loosely coupled plugin-modules.
4) *Microservices:* Highly agile, applications using this pattern are loosely coupled allowing changes to be done easily.
5) *Space-Based:* Highly agile, small application size and dynamic nature of the pattern allows to respond well to coding changes.

B. Ease of Deployment

1) *Layered:* Not easy to deploy – one small change requires redeployment of entire application.
2) *Event-Driven:* Easy to deploy, reason – decoupled nature of event processor components.
3) *Microkernel:* Easy deployment, dynamic addition of plugin modules to the core system is possible.
4) *Microservices:* High rate of deployment, services deployed as separate units of software where changes done on one service doesn't impact overall operations.
5) *Space-Based:* Simple deployment, dynamic, sophisticated and cloud-based tools allow for applications to be easily pushed out to servers.

C. Testability

1) *Layered:* Easy to test, presentation component can be mocked to isolate testing.
2) *Event-Driven:* Testing is complicate, individual unit testing require specialized testing tools to generate events.
3) *Microkernel:* High rate of testability, plugin modules can be tested separately and can be easily mocked by the core system.
4) *Microservices:* High testability rate due to isolation of business functionality into independent applications.
5) *Space-Based:* Low testability rate, high user loads in test environment make it both time consuming and expensive.

D. Performance

1) *Layered:* Can't be used for high performance applications.
2) *Event-Driven:* This pattern can be used for high performance applications due to asynchronous capabilities.
3) *Microkernel:* This pattern allows application to perform well, easy customization is streamlining of applications.
4) *Microservices:* Not a highly performant due to the distributed nature of this pattern.
5) *Space-Based:* High performance rate due to in-memory data access and caching mechanics.

E. Scalability

1) *Layered:* Applications using this pattern are difficult to scale.
2) *Event-Driven:* Independent & decoupled event processors allows high scalability.
3) *Microkernel:* Can't help in producing scalable solutions
4) *Microservices:* Highly scalable solutions, each component can be highly scaled.
5) *Space-Based:* High scalability due to little or no dependency on a centralized database.

F. Ease of Development

1) *Layered:* Easy to develop and implement as it is well-known and not very complex
2) *Event-Driven:* Complicated development due to asynchronous nature of the patterns
3) *Microkernel:* Complex to implement as it requires thoughtful design and contract governance.
4) *Microservices:* High rate of development, easy due to smaller and isolated scope of service components.
5) *Space-Based:* Low rate of development, sophisticated caching and in-memory data grid make this pattern complex to develop.

V. CONCLUSION

Through the literature study, we have assessed several architectures and come to the conclusion that every architectural design pattern has usecases it is best suited for. For example, the Layered Architecture Pattern is the best option for small utility applications since the advantages that any other design pattern bring are minimal and eclipsed by the ease of development. On the flipside, the Microservices Architecture Pattern is the best option for building multi-module goliath applications since the decoupled nature allows multiple teams to asynchrounously develop the modules, and integration is easily done via networking. Hence, based on our findings, we have laid out the characteristics of each design pattern on the basis of 6 parameters, which can be used to assess the suitability of a design pattern to an application.

VI. FUTURE WORK

In the future, we would like to implement each of the design patterns discussed to collect more data about how each design pattern performs with respect to the parameters, and hence gain more insights into the suitability of design patterns for different situations and applications.

REFERENCES

[1] N. Medvidovic and R. N. Taylor, "Software architecture: foundations, theory, and practice," 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010, pp. 471-472, doi: 10.1145/1810295.1810435.
[2] Andreessen, Marc (20 August 2011). "Why Software Is Eating The World". The Wall Street Journal. Retrieved 25 April 2020.
[3] Chang, Chih-Hung; Lu, Chih-Wei; Lin, Chih-Hao; Yang, Ming-Feng; Tsai, Ching-Fu (June 2008). "An Experience of Applying Pattern-based Software Framework to Improve the Quality of Software Development: 4. The Design and Implementation of OS2F". Journal of Software Engineering Studies, Vol. 2, No. 6. the Third Taiwan Conference on Software Engineering (TCSE07). pp. 185–194. Archived from the original on 2011-09-22. Retrieved 2012-05-16.
[4] "Architectural Patterns: Definition". AAHN INFOTECH (INDIA) PVT. LTD. Archived from the original on 2012-06-23. Retrieved 2012-05-16.
[5] Avgeriou, Paris & Zdun, Uwe. (2005). Architectural Patterns Revisited - A Pattern Language.. 81. 431-470.
[6] N.Harrison and P. Avgeriou, "Pattern-Based Architecture Reviews," in IEEE Software, vol. 28, no. 6, pp. 66-71, Nov.-Dec. 2011, doi: 10.1109/MS.2010.156.
[7] N. Medvidovic and R. N. Taylor, "Software architecture: foundations, theory, and practice," 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010, pp. 471-472, doi: 10.1145/1810295.1810435.
[8] Chethana. S, Dr. G. N. Srinivasan, 2016, A Study of Architectural Design Patterns for Software Architecture, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) ICRET – 2016 (Volume 4 – Issue 21)
[9] T. Bi, P. Liang and A. Tang, "Architecture Patterns, Quality Attributes, and Design Contexts: How Developers Design with Them," 2018 25th Asia-Pacific Software Engineering Conference (APSEC), 2018, pp. 49-58, doi: 10.1109/APSEC.2018.00019.
[10] J. S. Fant, "Building domain specific software architectures from software architectural design patterns," 2011 33rd International Conference on Software Engineering (ICSE), 2011, pp. 1152-1154, doi: 10.1145/1985793.1986026.
[11] Jing Wang, Yeong-Tae Song and L. Chung, "From software architecture to design patterns: a case study of an NFR approach," Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Network, 2005, pp. 170-177, doi: 10.1109/SNPD-SAWN.2005.38.
[12] K. Sartipi, "Software architecture recovery based on pattern matching," International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings., 2003, pp. 293-296, doi: 10.1109/ICSM.2003.1235434.
[13] N. Harrison and P. Avgeriou, "Pattern-Based Architecture Reviews," in IEEE Software, vol. 28, no. 6, pp. 66-71, Nov.-Dec. 2011, doi: 10.1109/MS.2010.156.
[14] L. Xiaoli, W. Guoqing, J. Min, Y. Min and W. Weiming, "Software architecture for a pattern based Question Answering system," 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007), 2007, pp. 331-336, doi: 10.1109/SERA.2007.120.
[15] S. U. Gardazi, H. Khan, S. F. Gardazi and A. A. Shahid, "Motivation in software architecture and software project management," 2009 International Conference on Emerging Technologies, 2009, pp. 403-409, doi: 10.1109/ICET.2009.5353138.