Bulletin of Electrical Engineering and Informatics
Vol. 12, No. 2, April 2023, pp. 1226~1236
ISSN: 2302-9285, DOI: 10.11591/eei.v12i2.4081

A systematic review of non-functional requirements mapping into architectural styles

Muhammad Nouman[1], Muhammad Azam[2], Ashraf Mousa Saleh[3], Abdullah Alsaeedi[4], Hayfa Yousef Abuaddous[3]
[1]Department of Computer Science, Faculty of Science, National Textile University Faisalabad, Punjab, Pakistan
[2]Department of Electrical Engineering and Computer Science, University of Missouri, Columbia, USA
[3]Department of Software Engineering, Faculty of Computer Sciences and Informatics, Amman Arab University, Amman, Jordan
[4]Department of Computer Science, College of Computer Science and Engineering, Taibah University, Madinah, Saudi Arabia

Article Info

Article history:

Received May 14, 2022
Revised Aug 5, 2022
Accepted Aug 18, 2022

Keywords:

Architectural patterns
Architecture styles
Components of software architecture
Non-functional requirements
SDLC
Software architecture

ABSTRACT

Fortunately, the software attracted enough businesses to the market, allowing them to earn money in less time with less work and more accurate results. Software development life cycle (SDLC) is used for software development as it is responsible for system functionality, efficiency, maintainability, and any other non-functional system requirements. Each stage of the SDLC process is critical. However, software requirements and software architecture are both fundamental activities that play a vital role in all other SDLC stages. Non-functional requirements are critical to the success of any software because they explain all system quality attributes such as complexity, reliability, security, and maintainability, among others. The architectural styles assist you in determining which architecture may be best for your project requirements. This paper discusses several of the most important architectural styles that are best suited for mapping desired non-functional requirements for software development, as well as their comparison based on various quality attributes (non-functional requirements).

This is an open access article under the CC BY-SA license.

Corresponding Author:

Ashraf Mousa Saleh
Department of Software Engineering, Faculty of Computer Sciences and Informatics
Amman Arab University
Amman, Jordan
Email: asaleh@aau.edu.jo

1. INTRODUCTION

Architectural styles are a critical idea in the field of programming architecture: they offer grounded answers to architectural issues, help to report the architectural plan choices, work with correspondence between partners through typical jargon, and portray the quality credits of a product framework as powerful. Deplorably, finding and applying suitable architectural examples practically speaking remains, to a great extent, impromptu and unsystematic [1]. From the point of view of the architectural style, the issue doesn't get a lot of consideration, nor does the reasoning behind picking a particular arrangement. A style is viewed as having parts, connectors, and issues identified with control and information flow. Mindlessness is attracted to architectural arrangements, the semantics of the styles, and the potential architectural examination that can be performed on the frameworks based on the styles [2].

The arranging of software architecture for any issue is started after the culmination of necessity, and not long before the planning stage. That is the reason the architecture stage is otherwise called the "initial design step". Arranging and fostering the architecture for a software framework implies addressing the

framework with a significant degree of reflection. Software architecture is a stage in the software development life cycle (SDLC) that characterizes how to take care of the issue of the client to such an extent that the arrangement contains every one of the useful and non-utilitarian prerequisites of the client [3].

Both software requirements and software architectures are fundamental activities that play an important role in all other SDLC stages [4]. Software requirements are the baseline requirements that guide the evolution of software under construction throughout the SDLC [5]. The system architecture is the overall structure of the system, and it indicates the fulfillment of the main system features and quality attributes. Software architecture is a phase in SDLC that describes how to solve customer problems while meeting all non-functional and functional requirements. It is less detailed and serves as the first step toward resolving the issue [6].

Developers use the software architecture style based on project requirements to easily achieve the desired software output [7]. It will be extremely costly for our project if the final system does not produce the desired results due to the inappropriate architectural style. Software developers should map quality attributes to appropriate software architecture styles to achieve the best results [8]. The following factors may have an impact on the system architecture style; i) system acquisition, ii) technical surroundings, iii) organizational structure, and iv) knowledge of masonry. There are a few other important factors to consider when mapping non-functional requirements into system architecture styles: i) the system architecture style should be created in such a way that it meets all the important quality requirements of the customer [9], ii) the system architect should not go into implementation details [10], iii) to maximize production, system architecture should make use of as many system resources as possible [11], and iv) the system architecture should be adaptable to future changes [12]. The design of the system architecture describes how to set up the system, what the system's behavior will be, how risk management is carried out, and how to display the system in terms of components so that programmers can reuse the same components and improve system quality.

2. ARCHITECTURAL STYLES AND NON-FUNCTIONAL SYSTEM REQUIREMENTS

The software architecture body has had many discussions from different angles, like which planning and assessment techniques [13], [14] architecture description languages, and perspectives are best for which cases. Architectural designs are one of the few exceptions where the agreement was reached in the field of programming engineering: their significance is grounded, and they are critical for an engineering portrayal [15]. The relationship between software quality attributes and architecture is not straightforward, easy, or straightforward. It is critical to keep track of all quality attributes and software architecture accurately by keeping this relationship straight and direct [16]. Program coders should have a solid understanding of important architectural styles before mapping non-functional requirements into architectural styles [13]. Structural styles and examples characterize how one can coordinate the parts of the framework to construct a total framework and accomplish the prerequisites of the client. The developer should also understand which architectural style is best suited to our project's non-functional requirements [7].

Generally, all styles offer four things; i) vocabulary (design component or element names), ii) design rules (constraints or conditions), iii) semantic interpretation (well-defined meanings of constraints), and iv) analysis (performed on the system). Finally, there is no single list of architectural examples for programming designers to utilize [17]. All things being equal, there is voluminous and heterogeneous writing of examples, where the different examples vary in their way of thinking and method of portrayal and are regularly not related with regards to an example language. There are various software architectural styles, so one must select a suitable architecture based on the requirements of our project. Table 1 categorizes all the widely used existing software architecture styles [15], [18].

Table 1. List of categories

Categories	Architecture styles	Categories	Architecture styles
Structural architectures	Layered, component-based, monolithic architecture, and pipe-and-filter	Shared memory architectures	Blackboard, rule-based, and data-centric
Distributed system architectures	Client-server, peer-to-peer, object request broker, service-oriented, space-based architecture, shared nothing architecture, and representational state transfer	Messaging architectures	Event-driven, publish-subscribe, and asynchronous messaging
Adaptive system architectures	Reflection, microkernel, plug-ins, and domain-specific languages	Modern systems architectures	Cloud computing, multi-tenancy architecture, big data architecture, and grid computing architecture

3. DIFFERENT SOFTWARE ARCHITECTURE STYLES

A software architectural style is a general, reusable solution to commonly occurring problems in a given context. Architectural styles are similar to software design patterns in scope. Architectural styles advise software developers on how to structure their code in broad strokes. It is the major degree of granularity and specifies the application's layers, high-level modules, and how those modules and levels interact with one another, as well as the relationships between them. A few architectural styles are described below for a better understanding of architectural styles.

3.1. Component-based architecture

The component-based architecture is very much not the same as that of the conventional style [19]. Component-based design necessitates not only a focus on framework detail and improvement, but also additional thought for overall framework setting, individual part properties, and part obtaining and coordination measures. The partitioning of the system is the foundation of the component-based architecture [20]. It divides the system into logical, physical, or functional components that each have a specific function [21]. Each component is concerned with a separate issue and is less reliant on the others. This architecture divides the problem into small sub-problems and allows for high-level analysis [22]. A component can be a module, object, information, resource, or package, among other things [23].

Component types; i) outdated: components that can still be found in the organizational library [24], ii) off-the-shelf: components that are accessible in a third party's library [25], and iii) new: components that are not in the organizational or third-party libraries and must be designed from the star [26]. Since the framework is divided into a few parts, as shown in Figure 1, one could easily reuse parts of one framework in another. In this way, it gives the office reusability. It adheres to the method of high cohesion with low coupling between framework components. Aside from a high cohesion, performs a single related task, whereas a low coupling relies less on one another [27]. A part can be effectively extendable to add greater usefulness or information to it. Since the framework is comprised of free parts, it will not be difficult to discover incorrect components [28]. To keep up with the framework to further develop usefulness or to address errors, one can undoubtedly supplant the old module with another one. One can without much of a stretch discover the necessary parts from outsider libraries, which consequently brings about the simple and expedient improvement of the framework. The reusability idea would not be there if it's an instance of variation of innovations along these lines. The fundamental advantage of this architecture is that it is not utilized in such a case. A few debates like "framework advancement", "similarity", and relocation will consistently be there [29]. The benefits and losses of a component-based architecture are described in Table 2.

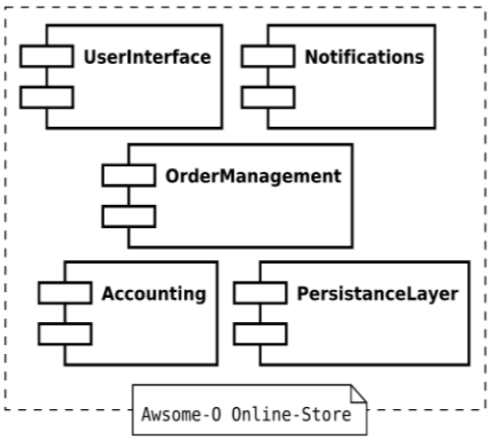

Figure 1. Example of component-based architecture

Table 2. Advantages and disadvantages of component-based architecture

No	Advantages	Disadvantages
1	Component reusability	Expensive initial outlay
2	Low inter-component coupling	Difficult to evaluate
3	Extendable components	Need for middleware because of incompatible technologies
4	Enhanced quality	Compatibility issues
5	Easy maintenance	

3.2. Blackboard architecture

In a blackboard architecture, the complicated assignment is partitioned into more modest sub-errands for which deterministic arrangements are known [30]. The blackboard is a shared repository that utilizes the consequences of its customers for heuristic calculation and stepwise improvement of the arrangement as depicted in Figure 2. A shared repository in the blackboard style is one in which one component serves as the primary data store and is accessed by other systems' independent knowledge sources [31]. The knowledge sources are usually not dependent on one another and each knowledge source attempts to overcome the obstacle. The blackboard style is like that of a lecture room blackboard, and it is used to solve problems.

Blackboard style is divided into two sections; i) blackboard: the primary data storage system and ii) knowledge source: a location where specific domain information is kept. Knowledge sources use the data store (blackboard) to solve problems by writing them down and solving them on the blackboard. All knowledge resources add to or change the answers provided by previous knowledge sources [10]. As a result, all knowledge sources collaborate to solve any problem. The control component is used to control and monitor all knowledge source activities to ensure that they do not deviate from the original problem [32]. The control component oversees and manages all activities. Table 3 lists the advantages and disadvantages of the blackboard architectural style.

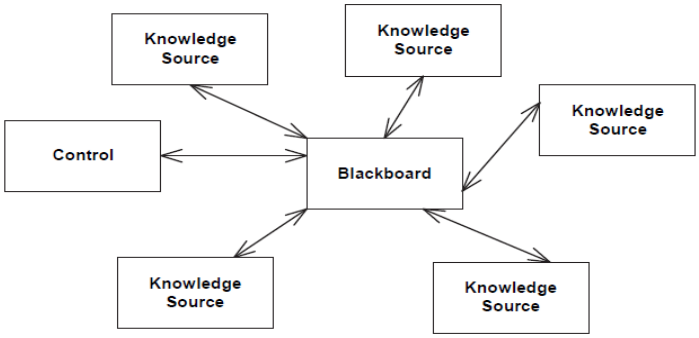

Figure 2. Blackboard architectural style

Table 3. Advantages and disadvantages of blackboard architecture

No	Advantages	Disadvantages
1	Easy to add or update knowledge sources	Reliance on the blackboard and knowledge sources
2	Easy to add or update knowledge sources	Unaware of the termination condition
3	Ensures compatibility	Synchronization is difficult to achieve

3.3. Client-server architectural style

With the progression of innovation, the web is turning out to be particularly more significant in our regular routines, where in all that we do these days includes the utilization of the web. In this way, the use of the web is not restricted to PCs, but it is open to various types of smart computerized gadgets, for instance, versatile ones [33]. Likewise, the design of the web is based on the client-server model, where the correspondence between the customer and worker is the main thing, technologists ought to be worried about. Almost all our web applications are now built on this architecture [34]. The server is the service provider and the client is the service receiver. The client asks for information or service and the server returns the result to the client after data processing to improve performance [35]. Each computer or network is either a client or a server [36]. The client-server architecture is divided into three parts as shown in Figure 3.

a. Client: a client is someone who asks the server for information or a service. Examples include a web browser, an email client, and a chat client [37].
b. Server: the server receives client applications, executes them, [38] collects necessary information, creates a client's call solution, and communicates the requested service or information to the requester. Examples include a web server, a print server, and a database server [35].
c. A communication network is a system that allows the server and client to communicate with one another [6]. Examples include LAN, WAN, and the internet. To proceed with the service request, the client must know the server's address, but the client's location is not required [39]. The server can function as a client and the client can function as a server, depending on the requirements. On occasion, it is possible to work as both a client and a server [40].

For instance, in an organization, a few PCs are associated among which one PC will go about as a print worker (on which printer worker is introduced) and others are customers [41]. The customer PC will demand a print worker PC to print a report. If the print worker is available, it will serve the request as a server and print the record. However, if the print worker is occupied, the customer's solicitation will be the one holding up the line [42].

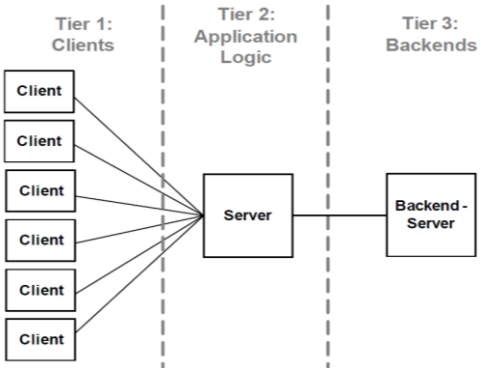

Figure 3. Tier client-server architecture

The client-server architecture types included:

- 2-tier architecture: in this architecture, the client and server speak with one another with no halfway point or hub. Although this architecture offers quick support, it experiences security and execution openings. model: internet explorer utilizes this type of client-server architecture [38].
- 3-tier architecture, another hub called the center tier sits in the middle of the client and server. The center tier receives the solicitation from the center, validates and supports it, and then forwards it to the server. Again, the center tier receives the response from the server and forwards it to the appropriate client after a legitimate check and approval. If the middle tier acts as a heap balancer, it also works on the framework's security [22].

The following are the advantages and disadvantages of the client-server architectural style shown in Table 4.

Table 4. Advantages and disadvantages of client-server architecture

No	Advantages	Disadvantages
1	Easy maintenance	Necessitates expert knowledge
2	Combined resources between different servers	Influence of centralization
3	Efficiency	Servers are expensive
4	Reduces data replication	

3.4. Event-driven architectural style

The event-driven architecture design is a well-known nonconcurrent circulated architecture design that is used to provide extremely versatile applications. It is also profoundly versatile and can be utilized for little applications as well as huge, complex ones. The event-driven architecture is comprised of profoundly decoupled, single-reason event handling parts as demonstrated in Figure 4 that do not concurrently receive and process events. An event is a noteworthy occurrence in the system. To detect changes in any object's state, various identification devices such as motion sensors and controllers are used [43]. For example, when the developer turns off the light, the bulb's state changes from “ON” to “OFF”. With the creation of an event, an event begins to flow. There are four logical layers in event-based architecture [44]:

a. Event generator: each event is generated by a single source. This logical layer oversees event production, and the source of the event is still at this level. The source could be as simple as pressing a keyboard key or clicking a mouse [45].
b. Event channel: the event channel is responsible for transporting events from the event generator's queue to the event user. The event channel could be a TCP/IP connection [46].
c. Event processing: in this layer, events are evaluated, and reasonable actions based on processing rules are generated [47].
d. Downstream event-driven activity: this logical layer displays the event's outcome. For example, a message with no subject line displays an error or warning on the screen [48].

Table 5 identifies the advantages and disadvantages of the event-driven architectural style and demonstrates why it is appropriate for developing interactive systems, but it can take longer to process an event [45].

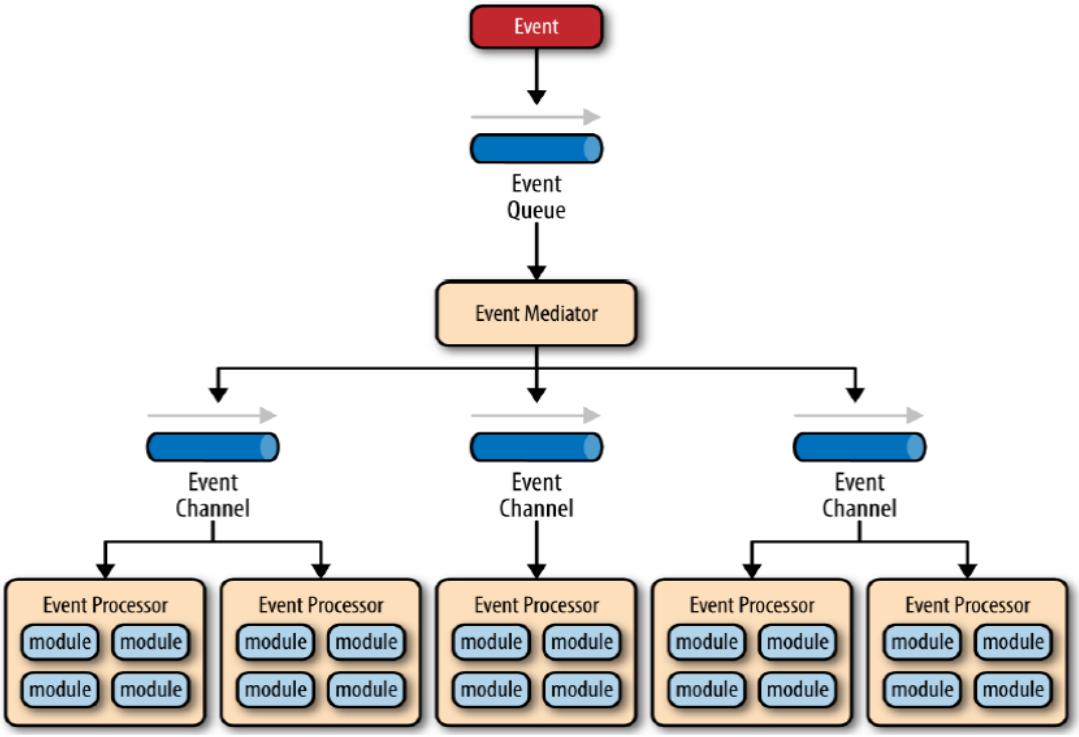

Figure 4. Event-driven architecture

Table 5. Advantages and disadvantages of event-driven architecture

No	Advantages	Disadvantages
1	Simple to maintain	Sometimes take a long to process an event
2	Simple to debug	Needed additional tools
3	Sloppily coupled	Large responses
4	Exceptional performance	

3.5. Microkernel architecture

Consider a framework family where various forms of a framework should be upheld. In every rendition, parts can be made distinctively, and different subtleties, like the offered administrations, public APIs, or UIs, may be unique. In any case, the framework family should be recognized for utilizing a standard architecture to simplify programming upkeep and encourage reuse. A microkernel acknowledges administrations that all frameworks, gotten from the framework family, need a fitting and-play foundation for the framework explicit administrations [49]. To implement product-based software, mostly the microkernel architecture is used. It allows you to add new application features as plugins to the basic application while also providing stability and feature separation. The microkernel architecture is made up of two parts as illustrated in Figure 5.

– The core system provides application features with flexibility, stability, and isolation. It only includes the bare minimum of functionality required to get the system up and running [50].
– Add-ons: plugins are self-contained modules that contain additional features and special processing to enhance or expand the core system and generate additional product capabilities. The core system should be aware of the plugins that are available and how to use them. Plugins are generally self-contained, but you can create plugins that require the presence of other plugins [51]. They must maintain communication to avoid dependency issues.

The benefits and drawbacks of microkernel architecture are summarized in Table 6. Which demonstrates that it allows you to add new application features as plugins to the basic application while also providing stability

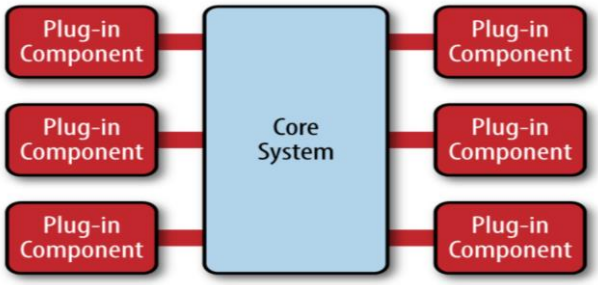

Figure 5. Microkernel architecture

Table 6. Advantages and disadvantages of microkernel architectural style

No	Advantages	Disadvantages
1	Simple to maintain	Sometimes take a long to process an event
2	Simple to debug	Needed additional tools
3	Sloppily coupled	Large responses
4	Exceptional performance	

3.6. Cloud computing architecture

This is a new computing model in which services are provided as regular utilities that users can rent and deliver via the internet. Cloud computing is appealing to entrepreneurs because it eliminates the need for clients to plan for provisioning and allows endeavors to start small and expand assets only when there is an increase in interest. Nonetheless, notwithstanding the way that cloud computing offers gigantic freedoms to the IT business, the improvement of cloud computing innovation is at present in its early stages, with many issues still to be tended to. As depicted in Figure 6, the cloud computing architecture consists of four layers that are required for cloud computing [52].

a. The hardware layer: the hardware layer manages cloud resources such as power, physical servers, and switches.
b. The infrastructure layer, also known as the virtualization layer, creates a reserve of storage and computing resources by distributing physical resources via virtualization technology [28].
c. The platform layer: to reduce the deployment burden directly on virtual machines, the platform layer consists of software frameworks and operating systems.
d. The application layer: this layer is constructed on top of the actual cloud applications [53]. Examples include Google Maps, YouTube, and Facebook.

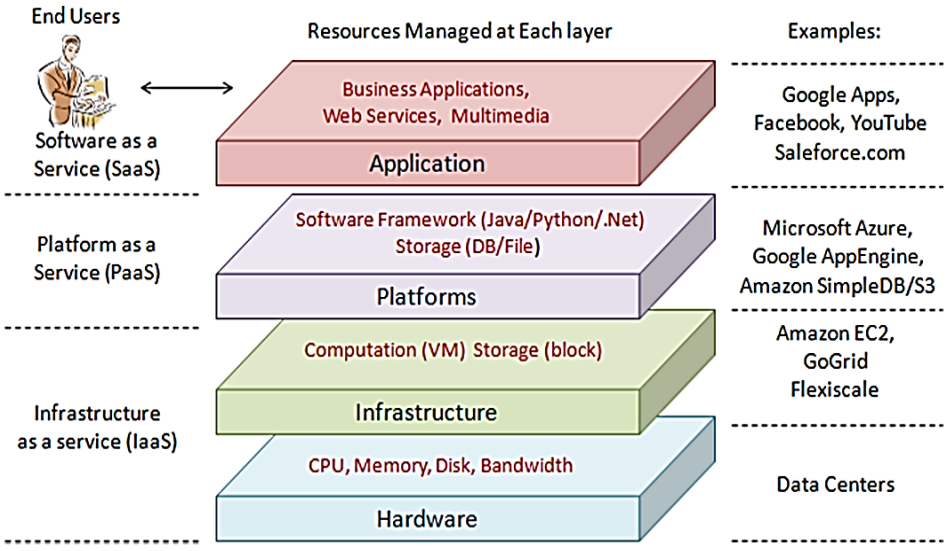

Figure 6. Cloud computing architecture

Cloud services are classified into three types:

a. Software as a service (SaaS): refers to the provision of on-demand applications via the internet. Examples include Google Docs and Microsoft BPOS [54].
b. Platform as a service (PaaS): refers to the provision of platform layer resources such as development frameworks and operating system support. Examples include Google App Engine and IBM Blue Mix, [55].
c. Infrastructure as a service (IaaS): refers to the provision of on-demand infrastructure resources [18], typically in the form of virtual machines. Examples include Zadora storage and Google Drive [56].

Table 7 sums up the advantages and disadvantages of cloud computing architecture. Table 8 compares various architectural styles in terms of the most important quality attributes such as functionality, complexity, efficiency, reliability, scalability, usability, maintainability, and portability.

Table 7. Advantages and disadvantages of cloud computing style

No	Advantages	Disadvantages
1	Reduced operating costs	Security issues
2	Accessibility	Limited control
3	Easy to use	Bandwidth issues
4	Reliability	

Table 8. Comparison of architectural styles

Quality attributes	Architecture style					
	Component-based	Black board	Event-driven	Client-server	Microkernel	Cloud computing
Functionality	High	High	Average	High	High	Average
Complexity	Low	High	Average	Average	High	High
Efficiency	High	High	Average	High	Average	Average
Reliability	Average	Average	Low	High	High	Average
Scalability	High	High	Low	Average	Low	High
Usability	Average	Low	Average	High	Average	High
Maintainability	Average	Average	Average	Average	High	High
Portability	High	Low	Low	Average	High	High
Application area	Structured	Artificial intelligence	Distributed systems, messaging	Web apps	Product-based apps	Distributed systems
Example	Pluggable apps	Speech recognition	Earthquake-detection	Email	Mobile phones os	Google app engine

4. CONCLUSION

This review provides a comprehensive overview of the most important architectural styles. All the architectural styles described are distinguished by quality attributes. This conclusion summarizes all the important architectural styles to help you decide which architectural style is best for your project requirements. For example, if your primary architectural concern is complexity, you can see from this table that component-based architecture is most likely a good choice. They are likewise thought of based on application regions.

Since blackboard architecture is mostly utilized in AI, it brings about high intricacy because computerized reasoning applications are computerized and mischievous to construct. It is a reasonable computerized man-made brainpower application like hearsay II, copycat, OCR text recognition, and triage. Along these lines, the application space of the chalkboard building can be used for robotic frameworks, speech acknowledgment frameworks, text acknowledgment frameworks, and earth perception satellites. It provides a high level of usefulness and productivity in the application. The utilization of the internet is expanding step by step, bringing about substantial utilization of client-server compositional styles in electronic applications. Ace Project, Gant tic, and Celosias of online software are available. The client-server architecture model is used by a server-based application that supports multiple clients and is accessed via an internet browser. It is also relevant in the application of concentrated asset frameworks (such as administration capacities and stockpiling) that will be used by many clients. Email, the world wide web, and FTP solubilization are popular and widely used client-server architecture application models. The complexity of such applications is not nearly as great as that of slate-based applications. Client-server applications retain their high utility and proficiency. Because of the center tier, which handles all security-related tasks, 3-tier client-server applications have excellent consistency. Component-based architecture ought to be utilized when there are a few comparative tasks, so modules or parts created for one venture can be effectively used in another. Part-based architecture ought to be embraced when appropriate parts are free to foster a venture by reusing. The fundamental component of a component-based architecture is reusability. Existing parts can

be reused when the timetable for fostering the venture is short. This architecture will be suitable for developing pluggable applications. Because the entire image of the framework is as far as parts are concerned, it results in high viability and compactness. Similarly, on request, software engineers can easily add another part. Such a framework will be difficult to expand. Because of the event-driven compositional style, it is appropriate for widely disseminated applications and applications that include the utilization of events between software components. The java swing API, which is based on an event-driven architecture, is one example. It demonstrates normal execution for most of the quality credits.

This research paper will assist and guide you in choosing the appropriate architectural style for your system. When deciding on an architectural style, you must consider all aspects of your organizational environment, such as developer skillset, project cost, project size, and project deadlines. The selection of the appropriate architectural style is critical because changing architectural organization is difficult and expensive. This paper could be additionally stretched out with a more comprehensive and comprehensive inclusion of these methods alongside their organizational events.

REFERENCES

[1] A. Kamilaris, "A lightweight resource-oriented application framework for wireless sensor networks," ETH Z¨urich, 2009, doi: 10.3929/ethz-a-005816888.

[2] Q. Feng, J. Liu, and J. Gong, "UAV remote sensing for urban vegetation mapping using random forest and texture analysis," *Remote Sensing*, vol. 7, no. 1, pp. 1074–1094, Jan. 2015, doi: 10.3390/rs70101074.

[3] I. Malavolta, H. Muccini, and M. Sharaf, "A preliminary study on architecting cyber-physical systems," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, Sep. 2015, pp. 1–6, doi: 10.1145/2797433.2797453.

[4] J. Berni, P. J. Z.-Tejada, L. Suarez, and E. Fereres, "Thermal and narrowband multispectral remote sensing for vegetation monitoring from an unmanned aerial vehicle," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 3, pp. 722–738, Mar. 2009, doi: 10.1109/TGRS.2008.2010457.

[5] M. Azam and M. Nouman, "The similarity between disease and drug network in link prediction," *ICST Transactions on Mobile Communications and Applications*, vol. 7, no. 2, pp. 1–10, Sep. 2022, doi: 10.4108/eetmca.v7i2.2667.

[6] P. R. Lewis *et al.*, "A survey of self-awareness and its application in computing systems," in *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*, Oct. 2011, pp. 102–107, doi: 10.1109/SASOW.2011.25.

[7] P. J. Z. -Tejada, R. D. -Varela, V. Angileri, and P. Loudjani, "Tree height quantification using very high resolution imagery acquired from an unmanned aerial vehicle (UAV) and automatic 3D photo-reconstruction methods," *European Journal of Agronomy*, vol. 55, pp. 89–99, Apr. 2014, doi: 10.1016/j.eja.2014.01.004.

[8] M. Azam, M. Nouman, and A. R. Gill, "Comparative analysis of machine learning techniques to improve software defect prediction," *KIET Journal of Computing & Information Sciences [KJCIS]*, vol. 5, no. 2, pp. 41–66, 2022, doi: 10.51153/kjcis.v5i2.96.

[9] B. W. Boehm and R. Valerdi, "Achievements and challenges in cocomo-based software resource estimation," *IEEE Software*, vol. 25, no. 5, pp. 74–83, Sep. 2008, doi: 10.1109/MS.2008.133.

[10] M. Frat, A. Kynar, C. Kankava, İ. T. Frat, and T. Tuner, "Prediction of Pentacam image after corneal cross-linking by linear interpolation technique and U-NET based 2D regression model," Compute. Biol. Med., vol. 146, p. 105541, Jul. 2022, doi: 10.1016/j.compbiomed.2022.105541.

[11] D. Guinard, V. M. Trifa, and E. Wilde, "Architecting a mashable open world wide web of things," Zürich, Swiss, p. 663, 2010.

[12] D. J. Mulla, "Twenty five years of remote sensing in precision agriculture: key advances and remaining knowledge gaps," *Biosystems Engineering*, vol. 114, no. 4, pp. 358–371, Apr. 2013, doi: 10.1016/j.biosystemseng.2012.08.009.

[13] L. Chen, "Towards architecting for continuous delivery," in *2015 12th Working IEEE/IFIP Conference on Software Architecture*, May 2015, pp. 131–134, doi: 10.1109/WICSA.2015.23.

[14] M. Azam, M. Nouman, and A. R. Gill, "Impacts of information technology on society in the new century," *Preprints*, pp. 1–9, 2022, doi: 10.20944/preprints202207.0283.v1.

[15] B. Rao, A. G. Gopi, and R. Maione, "The societal impact of commercial drones," *Technology in Society*, vol. 45, pp. 83–90, May 2016, doi: 10.1016/j.techsoc.2016.02.009.

[16] D. Guinard and V. Trifa, "Towards the web of things: web mashups for embedded devices," *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009)*, vol. 15, pp. 1–8, 2009.

[17] P. -J. Bristeau, F. Callou, D. Vissière, and N. Petit, "The navigation and control technology inside the AR.drone micro UAV," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 1477–1484, Jan. 2011, doi: 10.3182/20110828-6-IT-1002.02327.

[18] M. Nouman and M. Azam, "The role of laboratory technicians and the client public-private collaboration against COVID-19," *Academia Letters*, vol. 3284, pp. 1–7, Aug. 2021, doi: 10.20935/AL3284.

[19] M. Nouman, M. Azam, and A. R. Gill, "A systematic review of blockchain technology in current epoch: applications, adoption challenges, and opportunities," *Preprints*, no. July, pp. 1–13, 2022, doi: 10.20944/preprints202207.0259.v1.

[20] A. Hernandez, H. Murcia, C. Copot, and R. D. Keyser, "Towards the development of a smart flying sensor: illustration in the field of precision agriculture," *Sensors*, vol. 15, no. 7, pp. 16688–16709, Jul. 2015, doi: 10.3390/s150716688.

[21] J. Chen, X. Wang, and X. Xu, "GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction." *Applied Intelligence*, vol. 52, pp. 7513-7528, 2021, doi: 10.1007/s10489-021-02518-9.

[22] J. Dandois, M. Olano, and E. Ellis, "Optimal altitude, overlap, and weather conditions for computer vision UAV estimates of forest structure," *Remote Sensing*, vol. 7, no. 10, pp. 13895–13920, Oct. 2015, doi: 10.3390/rs71013895.

[23] A. M. Khaleghi *et al.*, "A DDDAMS-based planning and control framework for surveillance and crowd control via UAVs and UGVs," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7168–7183, Dec. 2013, doi: 10.1016/j.eswa.2013.07.039.

[24] T. Chen *et al.*, "The handbook of engineering self-aware and self-expressive systems," *Arxiv-Computer Science*, pp. 1–83, 2014, doi: 10.48550/arXiv.1409.1793.

[25] A. Kamilaris, A. Pitsillides, and V. Trifa, "The smart home meets the web of things," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 7, no. 3, pp. 145–154, 2011, doi: 10.1504/IJAHUC.2011.040115.

[26] C. Hung, Z. Xu, and S. Sukkarieh, "Feature learning based approach for weed classification using high resolution aerial images

from a digital camera mounted on a UAV," *Remote Sensing*, vol. 6, no. 12, pp. 12037–12054, Dec. 2014, doi: 10.3390/rs61212037.
[27] F. A. Vega, F. C. Ramírez, M. P. Saiz, and F. O. Rosúa, "Multi-temporal imaging using an unmanned aerial vehicle for monitoring a sunflower crop," *Biosystems Engineering*, vol. 132, pp. 19–27, Apr. 2015, doi: 10.1016/j.biosystemseng.2015.01.008.
[28] T. Chen and R. Bahsoon, "Symbiotic and sensitivity-aware architecture for globally-optimal benefit in self-adaptive cloud," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems-SEAMS 2014*, 2014, pp. 85–94, doi: 10.1145/2593929.2593931.
[29] M. Bernardo, P. Ciancarini, and L. Donatiello, "Architecting families of software systems with process algebras," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 4, pp. 386–426, Oct. 2002, doi: 10.1145/606612.606614.
[30] D. G. -Candón, A. I. D. Castro, and F. L. -Granados, "Assessing the accuracy of mosaics from unmanned aerial vehicle (UAV) imagery for precision agriculture purposes in wheat," *Precision Agriculture*, vol. 15, no. 1, pp. 44–56, Feb. 2014, doi: 10.1007/s11119-013-9335-4.
[31] T. S. López, A. Brintrup, M. -A. Isenberg, and J. Mansfeld, "Resource management in the internet of things: clustering, synchronisation and software agents," in *Architecting the Internet of Things*, Berlin, Heidelberg: Springer, 2011, pp. 159–193, doi: 10.1007/978-3-642-19157-2_7.
[32] E. Wilde, "Putting things to REST," *UC Berkeley Journals*, vol. 15, pp. 1–13, 2007.
[33] F. Oquendo *et al.*, "ArchWare: architecting evolvable software," in *Software Architecture*, Berlin, Heidelberg: Springer, 2004, pp. 257–271, doi: 10.1007/978-3-540-24769-2_23.
[34] Å. Rosenqvist, A. Milne, R. Lucas, M. Imhoff, and C. Dobson, "A review of remote sensing technology in support of the Kyoto Protocol," *Environmental Science & Policy*, vol. 6, no. 5, pp. 441–455, Oct. 2003, doi: 10.1016/S1462-9011(03)00070-4.
[35] H. D. Mills, M. Dyer, and R. C. Linger, "Cleanroom software engineering," *IEEE Software*, vol. 4, no. 5, pp. 19–25, Sep. 1987, doi: 10.1109/MS.1987.231413.
[36] M. Shahbazi, J. Théau, and P. Ménard, "Recent applications of unmanned aerial imagery in natural resource management," *GIScience & Remote Sensing*, vol. 51, no. 4, pp. 339–365, Jul. 2014, doi: 10.1080/15481603.2014.926650.
[37] S. Sankaran *et al.*, "Low-altitude, high-resolution aerial imaging systems for row and field crop phenotyping: A review," *European Journal of Agronomy*, vol. 70, pp. 112–123, Oct. 2015, doi: 10.1016/j.eja.2015.07.004.
[38] M. Wang, L. Qi, and X. Wang, "A Survey on Knowledge Graph Embeddings for Link Prediction," *Symmetry*, vol. 13, no. 3, p. 485, Mar. 2021, doi: 10.3390/sym13030485.
[39] J. T. Luxhøj, "A socio-technical model for analyzing safety risk of unmanned aircraft systems (UAS): an application to precision agriculture," *Procedia Manufacturing*, vol. 3, pp. 928–935, 2015, doi: 10.1016/j.promfg.2015.07.140.
[40] L. Zongjian, "Uav for mapping—low altitude photogrammetric survey," *International Archives of Photogrammetry and Remote Sensing*, vol. 37, pp. 1183–1186, 2008.
[41] E. R. Hunt, W. D. Hively, S. Fujikawa, D. Linden, C. S. Daughtry, and G. McCarty, "Acquisition of NIR-green-blue digital photographs from unmanned aircraft for crop monitoring," *Remote Sensing*, vol. 2, no. 1, pp. 290–305, Jan. 2010, doi: 10.3390/rs2010290.
[42] C. G. Sorensen *et al.*, "A user-centric approach for information modelling in arable farming," *Computers and Electronics in Agriculture*, vol. 73, no. 1, pp. 44–55, Jul. 2010, doi: 10.1016/j.compag.2010.04.003.
[43] L. Wallace, A. Lucieer, Z. Malenovský, D. Turner, and P. Vopěnka, "Assessment of forest structure using two UAV techniques: a comparison of airborne laser scanning and structure from motion (SfM) point clouds," *Forests*, vol. 7, no. 62, pp. 1–16, Mar. 2016, doi: 10.3390/f7030062.
[44] S. A. O'Shaughnessy and S. R. Evett, "Developing wireless sensor networks for monitoring crop canopy temperature using a moving sprinkler system as a platform," *Applied Engineering in Agriculture*, vol. 26, no. 2, pp. 331–341, 2010, doi: 10.13031/2013.29534.
[45] K. C. Swain, H. P. W. Jayasuriya, and V. M. Salokhe, "Low-altitude remote sensing with unmanned radio-controlled helicopter platforms: a potential substitution to satellite-based systems for precision agriculture adoption under farming conditions in developing countries," *Agricultural Engineering International: the CIGR Ejournal*, vol. 9, no. 12, pp. 1–16, 2007.
[46] F. Agüera, F. Carvajal, and M. Pérez, "Measuring sunflower nitrogen status from an unmanned aerial vehicle-based system and an on the ground device," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, pp. 33–37, 2012, doi: 10.5194/isprsarchives-xxxviii-1-c22-33-2011.
[47] I. Braud, "Spatial variability of surface properties and estimation of surface fluxes of a savannah," *Agricultural and Forest Meteorology*, vol. 89, no. 1, pp. 15–44, Jan. 1998, doi: 10.1016/S0168-1923(97)00061-0.
[48] M. T. Chilinski, K. M. Markowicz, and J. Markowicz, "Observation of vertical variability of black carbon concentration in lower troposphere on campaigns in Poland," *Atmospheric Environment*, vol. 137, pp. 155–170, Jul. 2016, doi: 10.1016/j.atmosenv.2016.04.020.
[49] S. Anusuyya and M. N. Jothy, "A secure color image using integer wavelet transform with linde-buzo gray algorithm," *International Research Journal of Engineering and Technology (IRJET)*, vol. 3, no. 4, pp. 395–398, 2016.
[50] V. Varsha and R. Singh Chhillar, "Data hiding using advanced LSB with RSA algorithm," *International Journal of Computer Applications*, vol. 122, no. 4, pp. 41–45, Jul. 2015, doi: 10.5120/21691-4796.
[51] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1996.
[52] L. Ross, "The image processing handbook," *Microscopy and Microanalysis*, vol. 17, no. 5, pp. 843–843, 2011, doi: 10.1017/s1431927611012050.
[53] M. Kumar, D. Dinesh, and D. Naveen, "Improvisation of security aspect of steganographic system by applying RSA algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 7, pp. 245–249, 2016, doi: 10.14569/IJACSA.2016.070733.
[54] M. Petrou and C. Petrou, *Image processing: the fundamentals*. West sussex, UK: John Wiley & Sons, 2010, doi: 10.1002/9781119994398.
[55] E. Z. Astuti, D. R. I. M. Setiadi, E. H. Rachmawanto, C. A. Sari, and M. K. Sarker, "LSB-based bit flipping methods for color image steganography," *Journal of Physics: Conference Series*, vol. 1501, no. 1, pp. 1–9, Mar. 2020, doi: 10.1088/1742-6596/1501/1/012019.
[56] J. R. Winkler, "Numerical recipes in C: the art of scientific computing, second edition," *Endeavour*, vol. 17, no. 4, p. 201, Jan. 1993, doi: 10.1016/0160-9327(93)90069-F.

BIOGRAPHIES OF AUTHORS

Muhammad Nouman earned a BS in Computer Science from COMSATS University Islamabad in 2017 and an MS in Software Engineering from the University of Agriculture Faisalabad in 2021. He is currently employed as a Lecturer in the Department of Computer Science at the National Textile University Faisalabad. He is a researcher who specializes in SE, data mining, deep learning, blockchain, and suicidal behavior. He has a background in both academia and industry, and he has completed numerous national and international projects. In addition, he has papers on blockchain technology, machine learning, public-private collaboration against COVID-19, and crop yield estimation. He can be contacted at email: m.nouman909@gmail.com.

Muhammad Azam earned MS in Software Engineering from the University of Agriculture Faisalabad in 2021. He is currently a Ph.D. Student University of Missouri Columbia. He is a researcher who specializes in link Prediction, ML, SE, data mining, deep learning, blockchain, and social network analysis. He has a background in both academia and industry, and he has completed numerous national and international projects. In addition, he has papers on social network analysis, blockchain technology, machine learning, public-private collaboration against COVID-19, and crop yield estimation. He can be contacted at email: writetoazamkhalid@gmail.com.

Ashraf Mousa Saleh completed her graduate studies in 2017 with a Ph.D. in Software Engineering from Universiti Sains Islam Malaysia. He is an Assistant professor in software engineering. Currently, He is working at Faculty of Computer Sciences and Informatics, Software Engineering Department at Amman Arab University, Amman, Jordan. He can be contacted at email: asaleh@aau.edu.jo.

Abdullah Asaeda received a B.Sc. in computer science from the College of Computer Science and Engineering, Taibah University (Madinah, Saudi Arabia, in 2008), M.Sc. in advanced software engineering from the University of Sheffield, Department of Computer Science (Sheffield, UK, in 2011), and Ph.D. in computer science from the University of Sheffield (the UK, in 2016). He is currently an Associate Professor at Taibah University, Madinah, Saudi Arabia. He can be contacted at email: aasaeedi@taibahu.edu.sa.

Hayfa Yousef Abuaddous completed her graduate studies in 2017 with a Ph.D. in Software Engineering from Universiti Sains Islam Malaysia and her undergraduate studies at Yarmouk University, Jordan with M.A and B.A. in Computer Information Systems. In 2017, she joined Amman Arab University as an Assistant Professor in Software Engineering Department. Her main research interests are human-computer interaction, UX, and usability. She can be contacted at email: Haddose@aau.edu.jo.