Journal for Educators, Teachers and Trainers
The PROFESIOLab electronic, peer-reviewed, open-access Magazine

ISSN 1989 – 9572

DOI: 10.47750/jett.2023.14.04.016

Inclusive Educational Review of Software Architectural Styles and Patterns for the Students of the College of Information and Computing Sciences of Cagayan State University

Freddie P. Masuli[1]

Lourdes M. Padirayon[2]

Manny S. Alipio[3]

Daniel T. Ursulum[4]

Grecilia A. Callitong[5]

Segundo D. Pacris Jr.[6]

Journal for Educators,Teachers and Trainers, Vol. 14 (4)

https://jett.labosfor.com/

Date of reception: 10 Jan 2023

Date of revision: 19 Mar 2023

Date of acceptance: 22 Mar 2023

Freddie P. Masuli, Lourdes M. Padirayon, Manny S. Alipio, Daniel T. Ursulum, Grecilia A. Callitong, Segundo D. Pacris Jr. (2023). Inclusive Educational Review of Software Architectural Styles and Patterns for the Students of the College of Information and Computing Sciences of Cagayan State University.*Journal for Educators, Teachers and Trainers*,Vol.14(4).190-202.

[1,2,3,4,5,6]Cagayan State University Sanchez Mira, College of Information and Computing Sciences (CICS) Centro 02, Sanchez Mira, Cagayan, 3518 Philippines

Journal for Educators, Teachers and Trainers
The LabOSfor electronic, peer-reviewed, open-access Magazine

Journal for Educators, Teachers and Trainers, Vol. 14 (4)
ISSN 1989 – 9572
https://jett.labosfor.com/

Inclusive Educational Review of Software Architectural Styles and Patterns for the Students of the College of Information and Computing Sciences of Cagayan State University

Freddie P. Masuli[1], Lourdes M. Padirayon[2], Manny S. Alipio[3], Daniel T. Ursulum[4], Grecilia A. Callitong[5], Segundo D. Pacris Jr.[6]
[1,2,3,4,5,6]Cagayan State University Sanchez Mira, College of Information and Computing Sciences (CICS) Centro 02, Sanchez Mira, Cagayan, 3518 Philippines
*Corresponding Author
Email: desmpadirayon@csu.edu.ph

ABSTRACT
A good architectural design has a high contribution to the success of a system. In addition, this architectural design is useful for the Information Technology (IT) students as their basis of their software development of their capstone project. The utilization of inappropriate architecture can lead to disastrous consequences for IT student researchers. A detailed understanding of software architecture styles is very useful to analyze distributed and complex systems which is the trend of capstone projects. This paper explores the quality attributes of three architecture styles namely shared-nothing, broker, and representational state transfer, which are perceived as beneficial to distributed system architecture that serve as guide to student researchers. This is to provide a picture of the said three key software architecture styles which could be helpful not only for student researchers but also for the software developers by adding references to minimize the uncertainty while selecting the appropriate architectural style for their specific needs. An architectural style must be chosen correctly to obtain all its benefits in the system. In this paper, the three architectural styles are compared on the foundation of various quality attributes derived from ISO 9126-1 standard such as functionality, reliability, usability, efficiency, maintainability, and portability. The results of the study are useful to guide the student researchers in their capstone project and to reduce the number of unsuccessful attempts of software development component of their capstone project.

Keywords: Architecture Styles, Broker architecture, Capstone Project, Shared-nothing, Software developer, Representational State Transfer (ReST)

INTRODUCTION

Capstone Project Mandates of ABET

The Accreditation Board for Engineering and Technology (ABET, 2016) mandates an outcome-based evaluation of graduating engineers' competence to use technical and other professional skills to solving real-world engineering challenges. Widespread implementation of capstone courses has assisted in the development and enhancement of these required skill sets (Omar, 2014).
Engineering programs incorporate capstone projects to combine multi-disciplinary subjects and teach professional abilities that are difficult to convey in a standard lecture course. Due to the fact that these projects assist to transition students into professional engineers, they have a direct effect on the industry reputation and ranking of a university (Ward, 2013).
Todd et al. (1995), who conducted a survey of capstone engineering courses in North America, discovered that many engineering programs use senior design/capstone-type courses to prepare students for engineering practice and that a significant number of institutions partner with industrial clients to sponsor capstone projects. They concluded that the intensive faculty investment produced competent engineering graduates, which was beneficial.
In addition, the nation's present emphasis on preparing undergraduate students for engineering practice has attracted significant attention to the quality of capstone projects, which are used to evaluate the efficacy of an engineering degree. The emphasis on quality capstone projects is part of an effort not just to guarantee that graduates are adequately prepared for engineering practice, but also to raise and broaden the professional competence of the engineering workforce.
Mosher (2015) suggested that in order to develop a high-quality capstone course for technology undergraduates, it is necessary to have appropriate scoping and planning of the project with the client beforehand, ownership and buy-in from students through controlled project and team selection, a high tolerance for ambiguity and

uncertainty as students work through the details of the project, and balanced methods of individual and group accountability. Literature demonstrates that thorough planning and execution are necessary for a successful capstone project.
According to ABET(2016) stresses the importance of engineering programs that provide students with the skills necessary to recognize, articulate, and propose engineering solutions to solve industrial challenges as well as contemporary social or global concerns. In other words, ABET promotes the importance of cooperation, communication, and project-based engineering courses. The culminating courses are meant to help students develop and improve these specific skill sets (Franchetti, 2012). The engineering design process taught to students incorporates the development of analytical, critical thinking, synthesis, and communication skills, which are vital to the industry.
Depending on the course or the general design of the program, capstone design projects may be undertaken independently or in groups. Since cooperation is regarded as an essential ability for success in the working world, the majority of capstone design courses require students to complete a design project in a group context (Zhou & Pazos, 2014).

Software Architecture
Software architecture is the core of software systems defining its sections and their connections (Yang et al., 2021). One of the major design tasks in building enterprise applications is to design good software architecture (Kotusev et al., 2022). Software architecture is the structural solution that accomplishes the general technical and operational requirements for software developments (Bamhdi, 2021). Software architecture is part of the early design stage which comes just after specifying all the requirements in a certain project and just before the design phase (Tian et al., 2022). Software architects incorporate architectural styles in the planning and organizing of the components of a complete system to meet and achieve the requirements of the customer. Additionally, it was developed to solve common problems which arise in developing software systems. Foremost, software architecture decomposes a system into a group of different components and then develops components and related connectors, and finally selects an architecture style and pattern (Bamhdi, 2021). When the architecture style is specified, designers need to determine the extent to which features of the software architecture influence quality attributes.
A quality attribute is the property of a system that can be tested and is used to indicate how well the system satisfies the needs of its stakeholders. Generally, software quality is considered during the early stage of the software development process, to lessen risks and to achieve the success of the overall software system (Sharma et al., 2015) like distributed systems. Also designing achievable architectural representations for detailed software system development, because it allows assessment of the different functional and non-functional properties of a designed system.
Interrelating the computing services of different layers by multiple cloud providers is indispensable for overcoming the exponentially increasing demands of enterprises. Because of a highly dynamic and unpredictable environment, cloud providers have to face numerous challenges related to service provisioning of diversified applications while handling the critical management operations on resource layers. To set up interconnected clouds, proper configuration and uniform development of architectural components and protocols are required, which play an important role in management. Some studies proposed techniques to increase the capabilities of a standalone source. Capabilities that influence interconnected cloud computing management from multiple perspectives. The resource management features of architectural components are from several sources and maintaining service quality metrics in service provisioning. Numerous architectural projects are offered to illustrate the challenges and future perspectives (Latif et al., 2020).
Architectural Approaches are to achieve a common goal as to very much popular today because of the many advantages which users can derive from it as mentioned by (Garcés et al., 2021) but the use of architectural approaches has decreased in the previous decade (Slavin, 2023). There are several architectural styles and patterns available today for distributed systems such as shared-nothing architecture, broker, and representational state transfer (Tian et al., 2022). Designers need to recognize which particular architecture style is suitable for a certain distributed project for them to successfully implement a software project. An architectural style must be chosen correctly to take advantage of all its benefits as applied in a distributed system. Fast release and delivery of next-generation software, which is the software industry's core goal these days, results in a mistake in the software development process (Yang et al., 2021).

The Cagayan State University at glance
The Cagayan State University (CSU) is the sole state university in the province of Cagayan and one of the top higher education institutions in Region 02 in the Philippines. The college is required by its charter to fulfill its thrusts in education, research, extension, and generation. The CSU is made up of 8 empowering campuses that are strategically located throughout the state in order to maximize its human and natural resources. Through the execution of its missions, it acts as a catalyst for regional growth by offering top-notch educational services,

conducting scientific research, and packaging, shipping, and commercializing technologies through outreach programs to the community (Cagayan State University | No.1 Philippines Government Medical College, 2021).

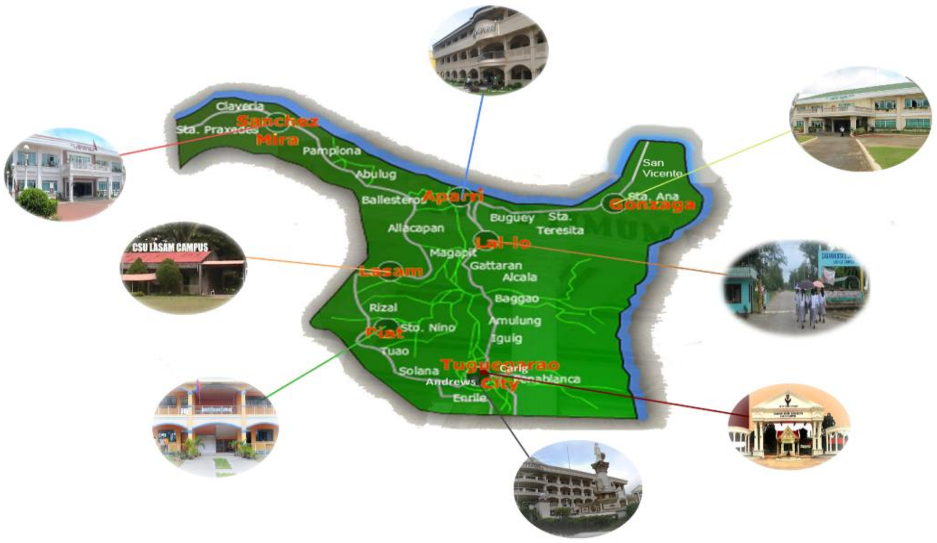

Figure 1: The Map of Cagayan State University (source: (Cagayan State University | Official Website, n.d.)

The fundamental goal of the university is to transform CSU into an agent of positive change that enhances the quality of life for both individuals and communities. The one-liner vision effectively conveys the deep significance and ultimate goal of the university's joint efforts and educational goals. Through excellent instruction and creative research, development, production, and extension, Cagayan State University aims to alter individuals' lives as well as those of their communities. The Cagayan State University, a reputable and eminent center of higher learning in Northern Luzon, is dedicated to enhancing the quality of life for individuals and communities by offering advanced education in the humanities, agriculture, and natural sciences as well as in technological and professional fields. This is done by delivering high-quality instruction and implementing cutting-edge research, resource mobilization, and extension methods. By offering high-quality education and training through instruction, research, extension, and manufacturing, CSU principally contributes to President Aquino's Social Contract to the Filipino people. (Cagayan State University | Official Website, n.d.)

They are guided by the core values like Competence, Critical Thinker, Creative Problem -Solver, Competitive Performer: Nationally, Regionally and Globally, Social Responsibility, Sensitive to Ethical Demands, Steward of the Environment for Future Generations, Social Justice and Economic Equity Advocate. (Cagayan State University | Official Website, n.d.)

One of the campuses that offers Bachelor’s degree in Information Technology was the Sanchez-Mira Campus. There are more than 100 students under this college from 2019-up to present. They also produced graduates who landed a good job related to their fields. The college requires capstone course of students before graduation. A total of 14 groups who were not successful in presenting their capstone projects due to failure to meet the development requirements, very poor features of the system, improper implementation of the software methodology and poor database design.

This reason motivated the researchers to come up with the inclusive educational study to prevent the unsuccessful capstone project. It provided a picture of the said three key software architecture styles for distributed systems which could be helpful for software developers by adding references to minimize the uncertainty while selecting the appropriate architectural style for their specific needs. All the architectural styles are compared based on various quality attributes based on ISO 9126-1 which include functionality, reliability, usability, efficiency, maintainability, and portability. Many businesses require dependable software architectures to meet the quality demands of new developing technologies which is the possible target beneficiaries of the student researchers.

METHODOLOGY

This section describes the process used for performing a systematic literature review of identifying Software Architectural Styles and Patterns. A literature review to assess the collective evidence in particular work by (Snyder, 2019) mainly indeed in Google Scholar, Scopus, and Web ok Knowledge, was conducted. It involves a thorough review of the existing studies in the area of architecture Software Architectural Styles and Patterns. In performing the review, guidelines by Banijamali et al. (2020), the review was based on articles concerning Software Architectural Styles and Patterns. From the research questions, this study derived the keywords. The

keywords were software architecture, architecture for distributed systems, architecture patterns, and styles. The search of the literature on software architecture was not established within a period although most of them date from the five four years. Case reports, review articles, and studies found by keywords and the references taken from the bibliography were short-listed, as well as architectures definitions. A total of 22 candidate software architectures were initially reviewed, however, only fifteen of them met the study requirements. These requirements refer to the description of the software architecture for the distributed system used as the basis for something which is being constructed.
Each of the remaining papers was read in its entirety. Although there is a large body of research related to software architecture in a broader Information Technology context, the papers presented in this literature review are those specifically related to software architecture for distributed systems. After reading each of the selected papers, only 3 remained which are in Table 1 contains a list of the papers included in this literature review.

Table 1. List of Included Publications

ID	Description	Study type	Focus	Paper
P1	The Broker Architectural Framework	Case Study	complex software system	(Varadharajan et al., 2022)
P2	Shared-Nothing Architecture	Case Study	Web applications have an application server, such as Tomcat, Apache + Mod_PHP	(Enia & Martella, 2019)
P3	Applying representational state transfer (REST) architecture to archetype-based electronic health record systems	Case Study	Electronic Health Records (EHRs)	(Sundvall et al., 2013)

RESULTS AND DISCUSSIONS

Architecture Styles and Patterns

Managing shared memory windows can achieve performance levels comparable to state-of-the-art Message Passing Interface (MPI) implementations (Quaranta & Maddegedara, 2021), thus lowering volatility in execution times and boosting process synchronization, especially in conditions with many nodes (Quaranta & Maddegedara, 2021b). Where service-oriented architecture is the most frequently applied and most investigated style and among all applicable quality attributes such as scalability, timeliness, and security. In addition, analyses of the relationship between architectural patterns, styles, views, and evaluation methodologies concerning different quality attributes and application areas. (Banijamali et al., 2020b) are the challenges that need to address. Thus, to address these challenges, software architecture style has become a necessary discipline. Architectures are used to develop a thorough understanding of the system and to ensure acceptable quality.

Architectural Styles in Application Type

Different architectural styles such as software development, Shared Memory, Messaging, Structure, adapt systems, and Modern System are used in distributed systems. Designing software needs a careful selection that supports the understanding of defined architectural approaches across sites.
Shared Memory introduced the capacity to run large-scale simulations with realistic particle shapes on platforms readily accessible to many (Park et al., 2021). A big data tool built upon for computing framework to boost the performance and to implement a scalable error correction algorithm intended for built using commodity hardware (Expósito et al., 2020).
Distributed System is used as a solution to a model that requires long run times and large memory usage for it strongly limiting its application [20]. Distributed Real-Time Architecture platform services for the development of applications establish end-to-end channels over hierarchical, heterogeneous, and mixed-criticality networks respecting mixed-criticality safety and security requirements (Obermaisser, 2018).
Messaging bridge the gap between the abstract representation of communication styles and technologies, stakeholders, and a multitude of application domains (Rouland et al., 2020) where data from new documents is used to create new models, and big data analytics services are used to handle the growing data volumes in all datasets and infer new knowledge from the connected data sources. (Sadek et al., 2022) also, Communication mechanisms are critical for architecture implementation. For various frameworks, many deployment mechanisms and communication patterns appear to be useful.
Adaptable Systems for the architectural style realize flexibility, where its unit adapts behavior and interactions

to operating conditions and copes with another unit (Weyns & Oquendo, 2019) for a system can adapt to a change in its environment (Farshidi et al., 2020) and several interacting cooperatively perform the system tasks (Affonso et al., 2019). Also, an important way to support the development, standardization, and evolution of software systems that aims to support the development of such applications.
Modern System distributed monitoring for reconfigurable computer systems offers nonstop status diagnostics of the computational unit mechanisms for saving the low-productive periods of equipment and for minimization of problems worst-case conditions are sensed (Danilov et al., 2016).

Broker architecture

A broker architecture pattern is used to structure distributed systems by decoupling components that interact by remote service innovations (Ekblom, 2011). Also, a data integration framework is designed to dynamically retrieve and transform heterogeneous data from different sources into a common format to provide an integrated view. Likewise, communication is matched by the component which is responsible for the forwarding of the client’s request to the server and the transmission of results and exceptions (Gruner et al., 2021).
To solve issues with the pattern addresses to large scale system were scaling too many components; such as components must be decoupled and distributed; where more services are required for adding, removing, activating, locating components at run time; and where designers of individual components should not need to know about the others, Brokers are being used to mediate between clients and servers; clients send requests to a broker; brokers locate appropriate servers, forward requests and relay results back to clients; and allow client-side or server-side proxies. Likewise, a digital assistance platform based on an adaptive workflow architecture is designed to individualize worker assistance. This demands a set of operators to adapt the underlying sequence flow and task instructions expressed as workflow models to individualize the resulting assistive action (Oestreich et al., 2021). Lastly, implementation of this architecture includes the Common Object Request Broker Architecture (CORBA) and OLE/DCOM/Active X. Multi-agent systems are often coordinated through brokers such as JADE that provide a standard mechanism for relaying messages, based on a high-level communication protocol. Individual agents may be implemented in any language as long as they can input/output according to the protocol. See Figure 2 for Broker Architecture.

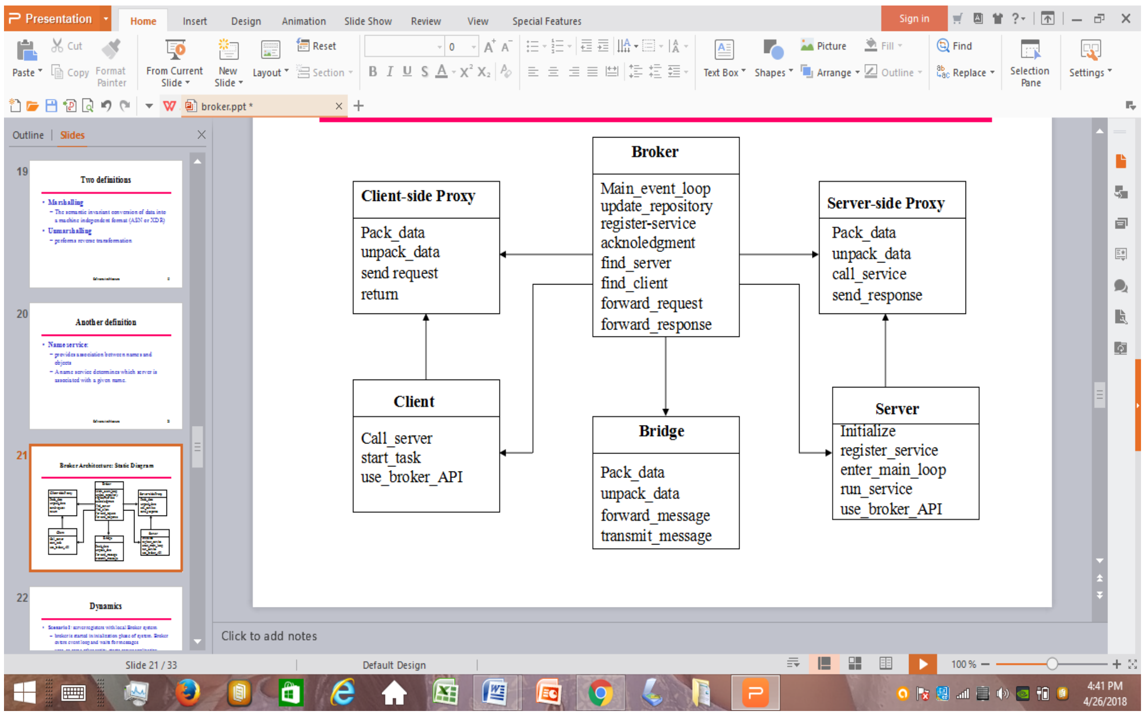

Figure 2: Broker Architecture: Static Diagram (Farshidi et al., 2020b)

A shared-nothing architecture

The Shared Nothing architecture is a distributed computing architecture where nodes are networked to form a scalable system (C. C. Yang et al., 2008). In its environment, each of the systems has its private memory and other available disks.
The study of (Sievi-Korte et al., 2019) described that the clustered processors correspond by passing messages through a network that interconnects the computers and requests from clients are automatically routed to the

system that possesses the resource. Only one of the clustered systems has sole access to and responsibility for a particular resource at a time.
In figure 3, each node has its private memory (RAM), processor (CPU), and storage devices (Disk) which are independent of any other node in the configuration which means that every node stores its lock table and buffer pool. (Karabey Aksakalli et al., 2021)

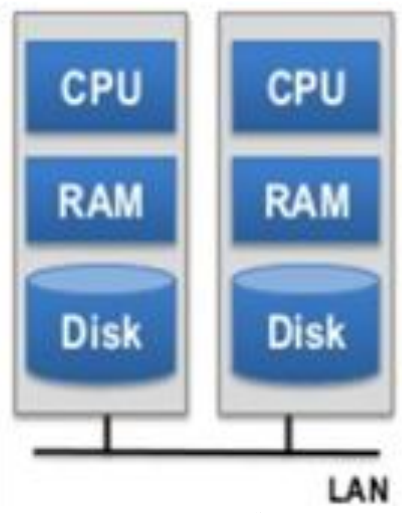

Figure 3: Shared Nothing Architecture ("6 Distributed Systems," 1987)

The study of (Ramirez et al., 2018) identified three of the most popular databases that support a shared-nothing model using different strategies. The first one is the Oracle which operates Range & Hash portioning of tables on shared-nothing (Bednar J & Robertson. D, 2006). Oracle uses a table and table space level segregation of data based either on a hashing algorithm or a range of key values. Another implementation of a shared-nothing model is the DB2 UDB. A partition key is being used to partition the data. Each partition is assigned by rows, and each partition has entire control of that row. If another partition wishes to read or update a row, it must send the request to the owning partition and then the owning partition executes the command on behalf of the requestor. Finally, the third application is the SQL Server which utilizes distributed partitioned views to implement the shared-nothing architecture.

Representational State Transfer (ReST) Architecture

Representational State Transfer (REST) is an architecture style that was originally designed by Fielding in the early 1990s to support the high performance and scalability requirements of the hypermedia environment and as a suggestion to redesign the use of the Hypertext Transfer Protocol and the Uniform Resource Identifier's (Whang et al., 2020). The World Wide Web represents the largest execution of a system compliant with the REST architectural style.
The study of (Lee, S., 2011) discussed that REST behaves like a virtual state machine, where the state transition happens when the user selects links, resulting in the next stage of the application being transferred to the user. In addition, they provided emphasized that the key characteristic of the REST architecture is that it takes a "resource view" of the world. They also described the RESTful principles which are as follows: P1: Resource can be identified by a URI; P2: Separation of the abstract resource and its concrete representations; P3: Stateless interaction, each interaction contains all the necessary context information and meta-data; P4: Small number of operations, with distinct semantics based on HTTP methods: safe operations; non-safe, idempotent operations and non-safe, non-idempotent operation (Post); P5: Idempotent operations and representation metadata support cache; P6: Promote the presence of intermediaries such as proxies, gateways or filters to alter or restrict request and response based on metadata.
RESTs functionality leads to four main subjects (Rajan, S, 2010) such as resources, representations, uniform interface, and State transfer. Resources are given an exclusive identity with a URI. Data sent to or from the resource is designed as a representation. And a set of uniform methods operates the resource. The communication is stateless but the client keeps a state of its workflow by navigating different resources, while the server tracks the state of the values of the resources.

Quality Attributes

Software quality in use (QinU) is the perception of software in its context of use (Williamson et al., 2022). In the context of software measurement, ISO/IEC/IEEE 15939 (Souza-Pereira et al., 2022) identifies a process including the definition of a suitable set of measures that address specific information needs, but it does not

provide the set of measures to be used as part of the measurement plan. (Souza-Pereira et al., 2022b) despite the importance that quality has in the development of successful software products (Anon, 2021a), the management of quality requirements is still an open challenge (López et al., 2022) Software quality improvement by enhancing software engineering, advanced software testing, and improvement still with shortcomings (Kokol P, 2021). Thus, measures of quality assurance are determined by standards of software quality lifecycle and a quality model with defined parameters for quality evaluation.
Several models, like, Boehm's, McCall's, FURPS, ISO 9126, and Dromey's, have been developed for quality evaluation using hierarchically related characteristics of quality indicators (Yadav, 2020). Assessing software products involves quality models of a wide range from simple hierarchic decomposition techniques to complex meta-models to cope with the abstract notion of software quality (Galli et al., 2021).
The definition of the quality attributes (Tian et al., 2022) of the ISO 9126-1 standard for software quality measurement which was used to compare the three software architecture styles is shown in Table 2.

Table 2: Characteristics and Sub-Characteristics of ISO 9126-1 Quality Model (Castillo-Salinas et al., 2020)

Attributes	Definitions	Sub-Characteristics
Functionality	The capability of the software product to provide functions that meet stated and implied needs when the software is used under specified conditions (Ikram, A, Masood, I, Sarfraz, T & Amjad, T, 2021).	Suitability, Accuracy, Interoperability, Security, Compliance
Reliability	The capability of the software product to maintain its level of performance under stated conditions for a stated period (Ikram, A, Masood, I, Sarfraz, T & Amjad, T, 2021).	Maturity, Fault tolerance, Recoverability, Compliance
Usability	The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions (Zhu & · H.Pham, 2018).	Understandability, Learnability, Operability Compliance
Efficiency	The capability of the software product to provide appropriate performance, relative to the number of resources used, under stated conditions (Babo et al., 2021)	Time behavior Resource behavior Compliance
Maintainability	The capability of the software product to be modified. Modifications may include corrections, improvements, or adaptations of the software to changes in the environment and the requirements and functional specifications (Fitrisia & Hendradjaya, 2014)	Analysability, Changeability Stability, Testability Compliance
Portability	The capability of the software product to be transferred from one environment to another. The environment may include organizational, hardware, or software environment (Fitrisia & Hendradjaya, 2014)	Adaptability, Installability Co-existence, Replaceability Compliance

Comparison of the Architecture Styles
Table 3 summarizes the comparison of the three architecture styles utilized for distributed systems. They were being compared in terms of functionality, reliability, usability, efficiency, maintainability, and portability.

Table 3: Summary of the comparison of the architecture styles for distributed systems

	Broker	Shared-nothing	REST
Functionality	+	+	+
Reliability	- -	+	++
Usability	+	+	+
Efficiency	- -	+	+
Maintainability	+	++	- -

Portability	++	+	++

Legend:

Attribute Remarks	Measure
++	The attribute (or sub-characteristic) is an asserted advantage of the architectural style
+	At least one of the sub-characteristics is evident in the application of the architecture style to distributed systems
-	No sub-characteristic is evident in the application of the architecture style to distributed systems
- -	The attribute (or sub-characteristic) is an asserted disadvantage of the architectural style

Based on table 3, the broker architecture style is easy to maintain because of its flexibility. It allows dynamic change, addition, deletion, and relocation of objects and automatic activation of applications to scale with the message volume. The advantages of this style are that it is high in portability because the components can be written in different programming languages and related message locking allows more than one instance of an application to process messages from the same queue without explicit synchronization. The functionality and usability of the style are due to the database integration enhancing application performance and simplifying administration. On the other hand, reliability and efficiency are restricted for this style because of indirection, high communications cost, and low fault tolerance which may need object replication to create higher fault tolerance.

Shared nothing architecture has high reliability because of its advantage versus a central entity that controls the network eliminating any single point of failure and allowing self-healing capabilities. Fault tolerance is lofty for failure is local. That if one node fails, the others stay up. Shared nothing is popular for web development because of its scalability allowing almost infinitely simply by adding nodes in the form of inexpensive computers, since there is no single bottleneck to slow the system down. This makes the shared-nothing architecture easy to maintain. Moreover, its portability provides an advantage. With its offering of non-disruptive upgrades, there is no need for reboots for it does not require the whole system to be rebooted when the update process completes. For these reasons (Enia & Martella, 2019) claimed that shared-nothing systems have no apparent disadvantages compared to the other alternatives.

On the other hand, the ReST style is very simple which made it easy to use. It advocates that information should be logically divided into linked resources where each resource is identified by a URI, and where operations on the resources are performed by the methods of HTTP. Content negotiation is used to deliver different types of representations of the resources to the clients which makes a REST-based system efficient (Banijamali et al., 2020). Reliability and portability are the advantages of this style by reusing components that can be managed and updated without affecting the system as a whole. If deployed over HTTP it gets supported by every major programming language and strengthens by well-tested technology, resulting in a system that is ready for work without requiring heavy machinery. The disadvantage of REST becomes apparent when sensitive information is transferred in a more complex system where the developer must strongly consider implementing encryption at the application layer. Because representations are chunks of data, the developer cannot request only parts of a document which makes the ReST architecture style hard to maintain.

CONCLUSIONS

This study provides a picture of shared-nothing, broker, and representational state transfer (REST) architecture styles which are all perceived to be important in distributed system development. The results served as guide not only for system developers but for student researchers. All above discussed architectural styles are compared based on quality attributes such as functionality, reliability, usability, efficiency, maintainability, and portability. Each of which has its qualities and demerits. One style may not fit all types of applications. Quality attributes should be considered depending on the priority and needs of the application. This paper could be further extended with more comprehensive and wide-ranging coverage of all these techniques along with their recent developments. The study's findings can help student researchers complete their capstone projects more successfully and with fewer failed attempts at the software development portion of their capstone projects.

REFERENCES

1. 6 Distributed systems. (1987). A Review of Ada Tasking, 63–72. https://doi.org/10.1007/3-540-18008-7_6

2. ABET (2016). Criteria for Accrediting Engineering Programs. Engineering Accreditation Commission, Accreditation Board for Engineering and Technology, Baltimore, MD. Worldwide web address: http://www.abet.org
3. Affonso, F. J., Passini, W. F., & Nakagawa, E. Y. (2019). A Reference Architecture to support the development of mobile applications based on self-adaptive services. Pervasive and Mobile Computing, 53, 33–48. https://doi.org/10.1016/j.pmcj.2019.01.001
4. Atoum, I.(2020), A novel framework for measuring software quality-in-use based on semantic similarity and sentiment analysis of software reviews, Journal of King Saud University - Computer and Information Sciences, Volume 32, Issue 1, 2020, Pages 113-125, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2018.04.012.
5. Babo, R., Rocha, J., Fitas, R., Suhonen, J., & Tukiainen, M. (2021). Self and Peer E-Assessment. International Journal of Information and Communication Technology Education, 17(3), 68–85. https://doi.org/10.4018/ijicte.20210701.oa5
6. Bamhdi, A. (2021). Requirements capture and comparative analysis of open source versus proprietary service oriented architecture. Computer Standards &Amp; Interfaces, 74, 103468. https://doi.org/10.1016/j.csi.2020.103468
7. Banijamali, A., Pakanen, O. P., Kuvaja, P., & Oivo, M. (2020). Software architectures of the convergence of cloud computing and the Internet of Things: A systematic literature review. Information and Software Technology, 122, 106271. https://doi.org/10.1016/j.infsof.2020.106271
8. Banijamali, A., Pakanen, O. P., Kuvaja, P., & Oivo, M. (2020b). Software architectures of the convergence of cloud computing and the Internet of Things: A systematic literature review. Information and Software Technology, 122, 106271. https://doi.org/10.1016/j.infsof.2020.106271
9. Bednar J & Robertson. D (2006).Architectural Patterns. SAPM Spring 2006: Architecture. Retrieved from http://www.inf.ed.ac.uk/teaching/courses/sapm/2005-2006/slides/architecture_4up.pdf, 2006
10. Berkeley Data Systems and Foundations (DSF) Group. (n.d.). https://dsf.berkeley.edu
11. Cagayan State University | No.1 Philippines government medical college. (2021, June 27). Cagayan State University. https://www.csmu.org/
12. Cagayan State University | Official Website. (n.d.). https://csu.edu.ph/campuses
13. Castillo-Salinas, L., Sanchez-Gordon, S., Villarroel-Ramos, J., & Sánchez-Gordón, M. (2020). Evaluation of the implementation of a subset of ISO/IEC 29110 Software Implementation process in four teams of undergraduate students of Ecuador. An empirical software engineering experiment. Computer Standards &Amp; Interfaces, 70, 103430. https://doi.org/10.1016/j.csi.2020.103430
14. Danilov, I., Dordopulo, A., Kalyaev, Z., Levin, I., Gudkov, V., Gulenok, A., & Bovkun, A. (2016). Distributed Monitoring System for Reconfigurable Computer Systems. Procedia Computer Science, 101, 341–350. https://doi.org/10.1016/j.procs.2016.11.040
15. Dawood, K. A., Sharif, K. Y., Ghani, A. A., Zulzalil, H., Zaidan, A., & Zaidan, B. (2021). Towards a unified criteria model for usability evaluation in the context of open source software based on a fuzzy Delphi method. Information and Software Technology, 130, 106453. https://doi.org/10.1016/j.infsof.2020.106453
16. Dutson, A.J., Todd, R.H., Magleby, S.P. & Sorenson, C.D. (1997). A review of literature on teaching engineering design through project-oriented capstone courses. Journal of Engineering Education, Volume 86, Issue 1, 17–28.
17. Ekblom,R. (2011). Applied Representational State Transfer. Retrieved from http://www8.cs.umu.se/education/examina/Rapporter/RichardEkblom-BT.pdf
18. Enia, M., & Martella, F. (2019). Reducing architecture: Doing almost nothing as a city-making strategy in 21st century architecture. Frontiers of Architectural Research, 8(2), 154–163. https://doi.org/10.1016/j.foar.2019.01.006
19. Eshraghian, E., & Rafe, V. (2015). Performance measurement of models specified through component-based software architectural styles. Measurement, 73, 372–383. https://doi.org/10.1016/j.measurement.2015.05.037
20. Expósito, R. R., González-Domínguez, J., & Touriño, J. (2020). SMusket: Spark-based DNA error correction on distributed-memory systems. Future Generation Computer Systems, 111, 698–713. https://doi.org/10.1016/j.future.2019.10.038

21. Farshidi, S., Jansen, S., & van der Werf, J. M. (2020). Capturing software architecture knowledge for pattern-driven design. Journal of Systems and Software, 169, 110714. https://doi.org/10.1016/j.jss.2020.110714
22. Farshidi, S., Jansen, S., & van der Werf, J. M. (2020b). Capturing software architecture knowledge for pattern-driven design. Journal of Systems and Software, 169, 110714. https://doi.org/10.1016/j.jss.2020.110714
23. Fitrisia, Y., & Hendradjaya, B. (2014). Implementation of ISO 9126-1 quality model for asset inventory information system by utilizing object oriented metrics. 2014 International Conference on Electrical Engineering and Computer Science (ICEECS). https://doi.org/10.1109/iceecs.2014.7045252
24. Franchetti, M., Hefzy, M. S., Pourazady, M., & Smallman, C. (2012). Framework for implementing engineering senior design capstone courses and design clinics. Journal of STEM Education: Innovations & Research, 13(3), 30-45.
25. Galli, T., Chiclana, F., & Siewe, F. (2021). On the Use of Quality Models to Address Distinct Quality Views. Applied System Innovation, 4(3), 41. https://doi.org/10.3390/asi4030041
26. Garcés, L., Martínez-Fernández, S., Oliveira, L., Valle, P., Ayala, C., Franch, X., & Nakagawa, E. Y. (2021). Three decades of software reference architectures: A systematic mapping study. Journal of Systems and Software, 179, 111004. https://doi.org/10.1016/j.jss.2021.111004
27. Gruner, A., Muhle, A., & Meinel, C. (2021). ATIB: Design and Evaluation of an Architecture for Brokered Self-Sovereign Identity Integration and Trust-Enhancing Attribute Aggregation for Service Provider. IEEE Access, 9, 138553–138570. https://doi.org/10.1109/access.2021.3116095
28. Ikram, Amna, Masood, Isma, Sarfraz, Tahira, Amjad, Tehmina. (2021). A Review on Models for Software Quality Enhancement from User's Perspective.
29. ISO 9126: Analysis of Quality Models and Measures. (2010). Software Metrics and Software Metrology, 205–228. https://doi.org/10.1002/9780470606834.ch10
30. Karabey Aksakalli, I., Çelik, T., Can, A. B., & Teki̇nerdoğan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. Journal of Systems and Software, 180, 111014. https://doi.org/10.1016/j.jss.2021.111014
31. Kelly, A., McCreary, D., Borkar, D. (2013). Making Sense of NoSQL: Applying Non-Relational Databases to Business Needs. Kelly-McCreary & Associates, LLC.
32. Kokol P.(2021).Software quality: A Historical and Synthetic Content AnalysisarXiv - CS – Software Engineering (IF), Pub Date: 2021-06-28, DOI: arxiv-2106.14598
33. Kotusev, S., Kurnia, S., & Dilnutt, R. (2022). The practical roles of enterprise architecture artifacts: A classification and relationship. Information and Software Technology, 147, 106897. https://doi.org/10.1016/j.infsof.2022.106897
34. Latif, S., Gilani, S. M. M., Ali, L., Iqbal, S., & Liaqat, M. (2020). Characterizing the architectures and brokering protocols for enabling clouds interconnection. Concurrency and Computation: Practice and Experience, 32(21). https://doi.org/10.1002/cpe.5676
35. Lee, S.(2011). Shared-Nothing vs. Shared-Disk Cloud Database Architecture. International Journal of Energy, Information and Communications Vol. 2, Issue 4, November 2011
36. López, L., Burgués, X., Martínez-Fernández, S., Vollmer, A. M., Behutiye, W., Karhapää, P., Franch, X., Rodríguez, P., & Oivo, M. (2022). Quality measurement in agile and rapid software development: A systematic mapping. Journal of Systems and Software, 186, 111187. https://doi.org/10.1016/j.jss.2021.111187
37. Mikalef, P., & Gupta, M. (2021). Artificial intelligence capability: Conceptualization, measurement calibration, and empirical study on its impact on organizational creativity and firm performance. Information &Amp; Management, 58(3), 103434. https://doi.org/10.1016/j.im.2021.103434
38. Mosher, G. A. (2015). Creating and sustaining high-quality senior capstone experiences. ATMAE Conference Proceedings: Building Bridges. 334-340, Nov 11-14, Pittsburgh, PA, USA.
39. Obermaisser, A. R. M. (2018, September 5). DREAMS Architectural Style | 2 | Distributed Real-Time Architecture fo. Taylor & Francis. https://www.taylorfrancis.com/chapters/edit/10.1201/9781351117821-2/dreams-architectural-style-obermaisser-abuteir-ahmadian-balbastre-barner-coppola-coronel-crespo-balbastre-fohler-gala-grammatikakis-larrucea-ortube-koller-owda-weber

40. Oestreich, H., da Silva Bröker, Y., & Wrede, S. (2021). An Adaptive Workflow Architecturefor Digital Assistance Systems. The 14th PErvasive Technologies Related to Assistive Environments Conference. https://doi.org/10.1145/3453892.3458046
41. Omar, M.A. (2014). Design and implementation of a capstone course to satisfy the industry needs of virtual product development and ABET engineering criteria. Education Research International.
42. Padayachee, Indira & Kotzé, Paula & Van, A & Van der Merwe, Alta. (2010). ISO 9126 external systems quality characteristics, sub-characteristics and domain specific criteria for evaluating e-Learning systems.
43. Park, E. H., Kindratenko, V., & Hashash, Y. M. (2021). Shared memory parallelization for high-fidelity large-scale 3D polyhedral particle simulations. Computers and Geotechnics, 137, 104008. https://doi.org/10.1016/j.compgeo.2021.104008
44. Perkins, J (2016). Architectural Pattern: Broker. Software Architecture. http://slideplayer.com/slide/4703089/
45. Quaranta, L., & Maddegedara, L. (2021). A novel MPI+MPI hybrid approach combining MPI-3 shared memory windows and C11/C++11 memory model. Journal of Parallel and Distributed Computing, 157, 125–144. https://doi.org/10.1016/j.jpdc.2021.06.008
46. Quaranta, L., & Maddegedara, L. (2021b). A novel MPI+MPI hybrid approach combining MPI-3 shared memory windows and C11/C++11 memory model. Journal of Parallel and Distributed Computing, 157, 125–144. https://doi.org/10.1016/j.jpdc.2021.06.008
47. Rajan, S (2010). Cloud Database Design, Scale Out Using Shared Nothing Pattern. @CloudExpo Journal, SYS-CON Media, Inc. Retrieved from http://cloudcomputing.sys-con.com/node/1586119#related
48. Ramirez, G. M., Collazos, C. A., & Moreira, F. (2018). All-Learning: The state of the art of the models and the methodologies educational with ICT. Telematics and Informatics, 35(4), 944–953. https://doi.org/10.1016/j.tele.2017.10.004
49. Romero, M., Guédria, W., Panetto, H., & Barafort, B. (2022). A framework for assessing capability in organisations using enterprise models. Journal of Industrial Information Integration, 27, 100297. https://doi.org/10.1016/j.jii.2021.100297
50. Rouland, Q., Hamid, B., & Jaskolka, J. (2020). Formal specification and verification of reusable communication models for distributed systems architecture. Future Generation Computer Systems, 108, 178–197. https://doi.org/10.1016/j.future.2020.02.033
51. Sadek, J., Craig, D., & Trenell, M. (2022). Design and Implementation of Medical Searching System Based on Microservices and Serverless Architectures. Procedia Computer Science, 196, 615–622. https://doi.org/10.1016/j.procs.2021.12.056
52. Sharma, A., Kumar, M., & Agarwal, S. (2015). A Complete Survey on Software Architectural Styles and Patterns. Procedia Computer Science, 70, 16–28. https://doi.org/10.1016/j.procs.2015.10.019
53. Sievi-Korte, O., Richardson, I., & Beecham, S. (2019). Software architecture design in global software development: An empirical study. Journal of Systems and Software, 158, 110400. https://doi.org/10.1016/j.jss.2019.110400
54. Slavin, B.B. (2023). An architectural approach to modeling artificial general intelligence, Heliyon, Volume 9, Issue 3, 2023, e14443, ISSN 2405-8440, https://doi.org/10.1016/j.heliyon.2023.e14443.
55. Snyder, H. (2019). Literature review as a research methodology: An overview and guidelines. Journal of Business Research, 104, 333–339. https://doi.org/10.1016/j.jbusres.2019.07.039
56. Souza-Pereira, L., Pombo, N., & Ouhbi, S. (2022). Software quality: Application of a process model for quality-in-use assessment. Journal of King Saud University - Computer and Information Sciences, 34(7), 4626–4634. https://doi.org/10.1016/j.jksuci.2022.03.031
57. Souza-Pereira, L., Pombo, N., & Ouhbi, S. (2022b). Software quality: Application of a process model for quality-in-use assessment. Journal of King Saud University - Computer and Information Sciences, 34(7), 4626–4634. https://doi.org/10.1016/j.jksuci.2022.03.031
58. Sundvall, E., Nyström, M., Karlsson, D., Eneling, M., Chen, R., & Örman, H. (2013). Applying representational state transfer (REST) architecture to archetype-based electronic health record systems. BMC Medical Informatics and Decision Making, 13(1). https://doi.org/10.1186/1472-6947-13-57

59. Symeonidou, N., Leiponen, A., Autio, E., & Bruneel, J. (2022). The origins of capabilities: Resource allocation strategies, capability development, and the performance of new firms. Journal of Business Venturing, 37(4), 106208. https://doi.org/10.1016/j.jbusvent.2022.106208
60. system by utilizing object-oriented metrics, DOI: 10.1109/ICEECS.2014.7045252
61. Tian, F., Liang, P., & Babar, M. A. (2022). Relationships between software architecture and source code in practice: An exploratory survey and interview. Information and Software Technology, 141, 106705. https://doi.org/10.1016/j.infsof.2021.106705
62. Todd, R. H., Magleby, S.P., Sorensen, C.D., Swan, B.R. & Anthony, D.K. (1995). A survey of capstone engineering courses in North America. Journal of Engineering Education, vol. 84, no. 2, 165–174.
63. Varadharajan, C., Hendrix, V. C., Christianson, D. S., Burrus, M., Wong, C., Hubbard, S. S., & Agarwal, D. A. (2022). BASIN-3D: A brokering framework to integrate diverse environmental data. Computers &Amp; Geosciences, 159, 105024. https://doi.org/10.1016/j.cageo.2021.105024
64. Verkaik, J., Hughes, J., van Walsum, P., Oude Essink, G., Lin, H., & Bierkens, M. (2021). Distributed memory parallel groundwater modeling for the Netherlands Hydrological Instrument. Environmental Modelling &Amp; Software, 143, 105092. https://doi.org/10.1016/j.envsoft.2021.105092
65. Viswanathan, S. and G. Bright (2014). Effective capstone project in manufacturing design engineering program. Presented at the 121st ASEE Annual Conference & Exposition, June 15-18, Indianapolis, IN.
66. Viswanathan, S. and H. Evans, (2005). Effective capstone/masters projects – Do's and don'ts. Presented at the ASEE Annual Conference, June 12-15, Portland, Oregon, USA.
67. Ward, Thomas A. (2013). Common elements of capstone projects in the world's top-ranked engineering universities. European Journal of Engineering Education, 38(2), 211-218.
68. Weyns, D & Oquendo, F.(2019). An Architectural Style for Self-Adaptive Multi-Agent Systems,2019. arXiv:1909.03475 [cs.MA] (or arXiv:1909.03475v1 [cs.MA] for this version) https://doi.org/10.48550/arXiv.1909.03475
69. Whang, K. Y., Na, I., Yun, T. S., Park, J. A., Cho, K. H., Kim, S. J., Yi, I., & Lee, B. S. (2020). Building social networking services systems using the relational shared-nothing parallel DBMS. Data &Amp; Knowledge Engineering, 125, 101756. https://doi.org/10.1016/j.datak.2019.101756
70. Yadav, S. (2020). Analysis and Assessment of Existing Software Quality Models to Predict the Reliability of Component-Based Software. International Journal of Emerging Trends in Engineering Research, 8(6), 2824–2840. https://doi.org/10.30534/ijeter/2020/96862020
71. Yan, et. al. (2021).Yang, T., Jiang, Z., Shang, Y., & Norouzi, M. (2021). Systematic review on next-generation web-based software architecture clustering models. Computer Communications, 167, 63–74. https://doi.org/10.1016/j.comcom.2020.12.022, Computer Communications, Volume 167, 2021, Pages 63-74, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2020.12.022.
72. Yang, C. C., Chen, C. K., Chang, Y. H., Chung, K. H., & Lee, J. K. (2008). Software architecture design for streaming Java RMI. Science of Computer Programming, 70(2–3), 168–184. https://doi.org/10.1016/j.scico.2007.07.003
73. Zhou, Z., & Pazos, P. (2014). Managing engineering capstone design teams: A review of critical issues and success factors. IIE Annual Conference Proceedings, 3006-3011.
74. Zhu & · H.Pham, (2018). A software reliability model incorporating martingale process with gamma-distributed environmental factors, https://doi.org/10.1007/s10479-018-2951-7S
75. Zou, J., Mei, J., & Wang, Y. (2010). From Representational State Transfer to Accountable State Transfer Architecture. 2010 IEEE International Conference on Web Services. https://doi.org/10.1109/icws.2010.56