ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 4, No. 2, 211-220 (2019)

www.astesj.com

Special Issue on Advancement in Engineering and Computer Science

ASTESJ

ISSN: 2415-6698

A Critical Analysis of Topics in Software Architecture and Design

Janet Bishung, Ooreofe Koyejo, Adaugo Okezie, Boma Edosomwan, Sylvester Ani, Abisola Ibrahim, Austin Olushola, Isaac Odun-Ayo*

Department of Computer and Information Sciences, Covenant University, Ogun State, Nigeria

A R T I C L E I N F O

Article history:
Received: 20 January, 2019
Accepted: 07 March, 2019
Online: 27 March, 2019

Keywords:
Software Architecture
Software Design
Service Oriented Architecture
Design Patterns

A B S T R A C T

Software architecture and design is an important component in the software engineering field. This aspect of software engineering covers the functional and non-functional requirements of any system being proposed to be developed, while software architecture deals with non-functional requirements, software design entails the functional requirements.

The objective of this paper is to critically analyze current topics in Software architecture and design. The method of analysis involved the use of inclusion and exclusion criteria of papers published in journals and conferences. These papers were accessed from digital libraries like ScienceDirect, and IEEE explore*, with a quantitative approach of analysis been imbibed. From the analysis, the result showed that, of 35 papers used in analysis, 34.3% discussed stakeholders' involvement and decisions in software design. 17.1% for design quality, 20% examined software reuse while 11.4% discussed software evaluation and 8.6% of papers reviewed discussed software management, evolution and software development life cycle each which should be more focused as it is the fundamentals of software design and architecture. From the analysis derived, stakeholder's involvement and decision in software design is an integral part in software building for effective use. Thereby making researchers dwell more on the topic. The least discussed topics was due to the expectations of researchers. Expecting readers to have a fore knowledge of the fundamentals of design which includes software management, evolution and software development life cycle.*

1. Introduction

Software Architecture gives the high-level description of a software and the discipline of creating the structures and systems [1]. It gives blueprint for the system, laying out tasks to be executed in a logical manner through the design [2]. It is the fundamental structural choices made vis a vis the business needs of the organization which may be costly to change once implemented [3]. Although there are no standard procedures to follow in software architecture that can address all issues of concern in general software development, certain factors should be of utmost non-negotiable fundamentals in software development, to ensure standardization thereby avoiding incessant collapse of systems witnessed in the early years of software developments [4]. Among the factors that will be enumerated briefly is the factor of proper documentation. This facilitates communication among stakeholders, captures decisions about the structures of the task and the design options – focusing on the decisions that must be right from the onset, otherwise, the imminent collapse of such system will be devastating– [5]. Since software architecture is largely driven by the required or expected functionalities, the current insight to software architecture is that required functionalities should reflect or incorporate all the quality attributes which include fault-tolerance, reliability, backward compatibility, extensibility, availability, maintainability, usability, security amongst others. Stakeholders concerns should reflect these quality attributes at both non-functional and functional stages without recourse to extra cost.

Software design envisions and defines software solutions to problem sets. It involves a sequence of steps that describe all aspects of the software in development [6]. Here, solutions to the problem the software is to solve are expressed in a logical sequence with details of their relationships. It begins with describing the total components to be built, then refine them to every detail. It is

*Isaac Odun-Ayo, Covenant University, +2348028829456
isaac.odunayo@covenantuniversity.edu.ng

the physical expression of all the processes that create solutions according to stakeholders' expectations. The designs are taken within the confines of fundamental principles which ensure designs are traceable to requirement analysis, uniformity, integration, structured for change etc. This ensures standardization while addressing the business objectives and stakeholders' needs [7]. In general, software development is dependent on time and cost and the design option should reflect these critical factors.

Software Design addresses all the expected required functionalities of the business objectives. This includes specifications of services, components, integration, data models and algorithms. Meanwhile, Software Architecture addresses design standard ensuring that it aligns with stated strategy as it pertains to business and technology of an organization. This includes considerations such as compliance, technology standards and operational efficiency. An architecture designed is intended to prevent repetitive mistakes in design or inconsistency with other aspects of the organization. It could be said that architecture is global optimization of software and design is local optimization [8]. In general, software architecture provides standardization upon which software designs are tailored.

Software architecture and software design are extremely important for a software project. So, here are brief points highlighting benefits of software architecture & design; solid foundation for software project, scalability of platform, increase performance, identification of area of cost savings, vision implementation, increase quality, better code maintainability, prioritization of goals, higher adaptability, faster platform, risk management and enable quicker changes among many others [9].

This study is aimed at helping researchers to have an in-depth knowledge of the fundamental as well as critical topics involved in software architecture and design of proposed and existing systems. It would also help in uncovering the critical gaps in which many researchers were not able to explore thereby improving knowledge as regards software architecture and design. The objective of this paper is to conduct a critical analysis on current topics in Software architecture and design. Published papers and articles on the topics discussed in the paper were reviewed with the percentages of each topic in relation to others were calculated.

Other parts of the paper are organized as follows: Section 2 gives review of some related work done, Section 3 gives a highlight on software architectural styles, Section 4 discusses Software Oriented Architecture, Section 5 is a discussion on Design Patterns; section 6 presents the results and discussion of the major topics selected for analysis from literatures. Section 7 gives a conclusion of the paper with recommendations for future work.

2. Related Work

In [10], the importance of software architecture as a vital aspect of software development was examined. The paper explored two important components, software evolution and re-usability. These are very critical component that helps curb the huge expenses involved in the development of software. Software architecture that can be evolved and reused should be in high demand, as software evolution and reuse are more likely to receive higher pay-off. [11] considers the fact that researchers cannot overlook the year's technology as well as the fact that software architecture employs fully detailed explorations of notations, techniques, analysis, tools and creation methods. There exist an intersection and interrelationship of software architecture with the study of software design, domain-specific design, program analysis, software families, specific classes of components and component-based reuse. The comparative analysis of software evolution methods in [12] explains the systematic comparison between architecture models and evolution methods centering its base on the scenario-based approach of software architecture.

The review on the successes and failure for software architecture in [13], gives insight into software architecture development and management process. It assesses previous literature and experiences to identify the factors that cause success and failure for software architecture and classifying these factors into subgroups as indicated by practitioners. [14] proposed a different methodology as a guide for practitioners supporting software architecture and design in an agile environment. Highlights of phases in software design process were covered, tools and techniques were proposed to implement those phases. Architectural design decisions and knowledge in [15], examine the essence of reusable architectural knowledge and the importance of documenting quality attributes along with the decisions captured during architectural design.

In [16], the concept of sustainability was introduced for software design. It was essential in integrating it into the existing catalog of design quality attributes. This was because design is a key factor in software development and this has been noted by many researchers. The information produced during software design tends to evaporate progressively due to certain conditions like software evolution. [17] considered all developed software need to meet required and specified quality standards as requested by users or stakeholders, quality being a major issue in software systems today. To achieve the quality requirements, different analysis approach was explored and a critical evaluation of the software system was carried out. This was to analyze the architecture, thereby verifying that quality requirements have been duly addressed in the design.

From an in-depth review of System Development Methodologies (SDMs) conducted by [18], a list of the important features in each methodology was made. Despite several SDMs, most of them share similar activities which include well known and practiced requirements-analysis, design, codification-test and implementation, all put together in project management. Four stages in the evolution of SDMs were reported which include pre-methodologies, rigor-oriented methodologies, agile-oriented methodologies and emergent service-oriented methodologies. Well-recognized SDMs in software engineering include Rational Unified Process, Microsoft Solutions Framework and Model-based (system) architecting and software engineering.

The software architecture chosen in the development of a software product is dependent on the software requirements and constraints. A Design Association Theory (DAT) to show the causal relationship of why a design should exist was proposed by [19] for associating design concerns, design problems and design solutions. DAT proposes a five-step association-based design

review process, which includes the extraction of requirements, extraction of design, construction of causal relationship between design elements and design solutions, discovery of potential design issues and verification and confirmation of design issues with architects. In documenting design decisions and design reasoning for objective evaluation, architects also utilize DAT. The DAT model helps designers and reviewers in associating architectural knowledge.

3. Software Architectural Styles

Architectural Styles are principles which shapes an application. It is more of an abstract framework of a system in the area of its organization. There are six major types of architectural styles. Namely:

3.1 Dataflow Architecture

In this, all software systems are categorized as lists of shifts on chronological set of input data, where data and operations are independent of each other. When data enters this system, it flows through the modules one at a time until they are assigned to some final destination [20]. Its aim is to achieve the qualities of reuse and immovability and is suitable for applications involving series of independent data computations on orderly defined input and output. There are three execution sequences between modules that the data flow architecture uses; Batch sequential, Pipe and filter or non-sequential pipeline mode and the Process control [20].

3.2 Data-centered Architecture

Data is centralized in this form of architecture and accessed frequently by other components that modify data. Its main purpose is achieving integrality of data. It contains different components that communicate using shared data repositories. The components access a shared data structure and are independent, meaning, they interact only through the data store. The flow of control sums the types of Data-centered architecture into two types; The repository and the blackboard architecture style. This form of architecture is mostly used in information system [20].

3.3 Hierarchical architecture

This views the whole system as a hierarchy structure whereby software systems are decomposed into subsystems at different levels in the hierarchy. It is mostly used in designing system software such as network protocols and operating systems. [20].

3.4 Interaction oriented architecture

The main aim of the interaction-oriented architecture is to separate users' interaction from data abstraction and business data processing. It divides the system into three major partitions: Data module (which provides the data abstraction and all business logic.), Control module (Which identifies the flow of control and system configuration actions) and the View presentation module (This is responsible for the visual or audio presentation of data output. It has two major styles: Model-View-Controller (MVC) and The Presentation-Abstraction-Control (PAC) [20].

3.5 Component based architecture

It is an architecture that decomposes software designs into functional components with their own methods, events and properties. These components become loosely coupled and reusable to provide modular programs that can be tailored to fit any need. [21]

3.6 Distributed architecture

This is a form of architecture that sits in the middle of a system and manages or supports the different components of that distributed system. Its aim is transparency, reliability, and availability. It hides the way in which resources are accessed and the differences in data platform, the resource location, different technologies from users, failures and resource recovery and a host of others. [20]. It is the most widely used form of architecture as it aligns with the technological advancement of the 21st century. Software development has improved greatly with the introduction of the internet. Software is now been distributed, components been reused, as well as introduction of concurrency and simultaneous change in the modification of data. These are the major advantages of the distributed architecture which has made it a common form of architecture in time past. [20]. There are different types of Distributed Architecture;

3.6.1 Broker Architecture: Mostly used to coordinate and enable the communication between registered servers and clients more like a software bus [20].

3.6.2 Client-server Architecture: Is commonly used by search engines, web servers, mail servers, it is mostly based on the functionality of the clients that is, requesting services of other components. The Service Oriented Architecture is a major sub-division of this. It supports business-driven Information Technology (IT) approach in which an application consists of software services and software service consumers. It has the ability to develop new functions rapidly which makes it mostly used along with its basic features which would be explained in the next section. [22]

Service Oriented Architecture would be further discussed due to its relevance in the current IT revolution. It is a known fact that cloud computing has come to stay and due to the integration ability of Service Oriented Architecture it has become a yardstick in the cloud computing revolution as their technologies have become more like bridges to the cloud.

4. Service Oriented Architecture (SOA)

SOA is a software architecture and design styles that entails the use of services as its main building component [23]. A service (as a software component), is a technique that allows access to several capabilities. SOA is now a mainstream software development mechanism. Despite the introduction of new architectural variants like cloud computing or micro-services [24], SOA is still widely used. This is due to its support for fast building applications using assembling of Internet-accessible services, allowing software organizations to hasten the development of distributed applications as well as a result time-to-market. After all this, a service is simply a distinct unit of performance that specifies a business function. This simply means it relies on Web Services for its implementation [24].

SOA carries out two core functions. Which are creating broad architectural models that explain application goals, including the approaches that help meet the goals. The second function is it defines the implementation specifications, which is mostly

integrated to the Web Services Description Language (WSDL) and the Simple Object Access Protocol (SOAP) specifications [25].

4.1 Major Principles of SOA

The major principles of SOA are [26]:

i. Regulate Service Contract - In this, there must be a form of description that explains what the service is about. This makes it easier for the client applications to understand what the service is meant to do.

ii. Loose Coupling –It entails components having little or no dependency on each other. This is a major characteristic of web services that emphasizes that there should be less dependency between web services and the client initiating this web service. Therefore, if any service functionality changes at any point in time, it should not hinder the client application from working.

iii. Service Abstraction - In this, service is ought to encapsulate its procedures and not expose how it executes its functionality. Explaining to the client application what it does and not how it does it for security purposes.

iv. Service Re-usability - Logic is separated into services with the aim of increasing reuse capability. In any technology, re-usability is a major issue as no one would want to spend time and effort writing the same codes again for multiple applications that require them.

v. Service Autonomy -The service knows everything about the application or system and what functionality it offers so it has complete control over the source code it encompasses.

vi. Service Statelessness - Superlatively, services ought to be stateless. Meaning they should not withhold any form of information from one state to the other.

vii. Service Discoverability - Services can be identified in a service registry. A service registry is a resource that allows controlled access to data for the controlling of SOA.

viii. Service Composability - It splits big issues into little ones. It is to be noted that not all functions should be embedded in an application and moved into one single service. Instead the service should be split into modules each having separate business functionalities.

ix. Service Interoperability - Services should accept and make use of various standards allowing different subscribers to use their various service.

For the implementation of SOA to be a success, you would need a productive SOA method that explains the plans, discoveries, procedures and the selected goals [26].

The integration platform for SOA plays a crucial role in the merging of existing application to cloud services. With SOA, components were split into services that became re-usable and easy to use among several systems. This is similar to the system means used in the automotive industry, where different layouts/systems share the same components e.g. engines. Cloud computing is re-vamping the IT world as we know it. The IT systems are utilized by users and companies. In the automotive industry, owners do not need to buy their own vehicles but can use car-sharing services. These service providers show the cars to several drivers. Comparably, in IT, a cloud service provider acts as a middle man and merges several clouds. Existing IT systems that require combination with new cloud-based solutions or inter-mediated are resolved using cloud service providers [37].

Cloud computing is re-modelling the IT industry and how its services are utilized, just like how petrol is soon going extinct in the automotive industry. Electric cars are now in use even as we use petroleum-based cars. As IT companies continue to make use of more cloud technologies the SOA technologies will continue to serve as bridges to the cloud.

Micro-service-based software architecture is the re-industrialized application of the SOA model. The components are developed as services using Application Programming Interfaces (API), just like the SOA would require. An API broker serves as a mediator to access components, ensuring SOA security and governance practices are followed.

SOA principles have taken us to the cloud and aides the most improved version of cloud software development techniques in use today. [27]

4.2 Advantages of SOA [28]

- Services can be reused in multiple applications independent of their interactions with other services.
- Due to service in-dependency, services can be easily updated or maintained without having to worry about other services.
- SOA-based applications are more reliable since they are small independent services that are easier to test and debug.
- Multiple instances of a single service can run on different servers at the same time.
- It improves Software Quality.

4.3 Disadvantages of SOA [28]

- Every time a service interacts with another service, complete validation of every input parameter takes place. This increases the response time and machine load, and thereby reduces the overall performance.
- There would be high investment cost as implementation of SOA requires a large upfront investment by means of technology, development, and human resource.

5. Design Patterns

Software design could be considered the most important aspect of software development as well as the most difficult process in a software development life cycle. Over the years, based on experience, programmers have embodied and recommended demonstrated results to fulfill the persistent issues that arise during design. Accordingly, the experience-based clarifications are composed and acknowledged as a consistent model for designs patterns [29]. Various design patterns have been presented and classified either as a sanctioned or a variation key to take care of design issues. The current programmed systems for design pattern(s) options help fledgling programmers to choose the more

proper pattern(s) from the rundown of relevant examples, to tackle an issue during the design period of software that is been developed. [29]

In software engineering, a design pattern is a general repeatable result for an ordinarily happening issue in programming structure [30]. Design patterns are used to ensure reuse of software design solutions in the early phases of software development, especially in the requirements engineering phase. Patterns do not provide visible solutions, but they present the concepts from which solutions are derived [31]. Design patterns have been introduced for defining good practices in software design [33]. Design patterns can be used in requirements engineering as patterns exist for core activities of a process. Design patterns are not pure inventions like a light bulb or a car. They are derived patterns that software engineers and architects found, that could be standardized to be used to solve similar problems categorized across three major areas; creational, structural and behavioral. When we determine the proper structure design for a task or issue, it helps us avoid changes that would require budgetary expenses, untenable, multiple and inefficient codes as the system scales up [32].

5.1 Classification and Selection of Design Patterns

The intrigue and involvement of programmers are utilized to present recent arrangement plans for the association of design patterns of specific concerns like object-oriented development and real-time applications [30]. The outline of current efforts describes the number of important categories of design patterns which relies upon the sort and multifaceted nature of target issues. For instance, [33] introduced three important classifications namely behavioral, creational, and structural in the setting of object-oriented advancement to solve recurring issues. In a specific circumstance of working applications, [30] introduced a catalog of thirty-four patterns, which are divided into five categories based on their relevance. In this paper, we would consider the object-oriented advancement categories.

A. Creational Design Patterns

Creational design patterns manage object creation systems attempting to make questions in a way that suits the circumstance. The essential type of object creation could result in design issues or added intricacy to the design. Creational design patterns take care of this issue by controlling this object creation [30]. These patterns can be further divided into class-creation patterns and object-creation patterns. Class-creation patterns use legacy viably in the instantiating of procedures while object-creation designs use assignment adequately to take care of business [35]. Creational design patterns are singleton, abstract factory, prototype, factory method, builder and object pool [31].

B. Structural patterns

Structural Design Patterns are used to ease a design by recognizing a straightforward method to acknowledge relationships [30]. These design patterns are tied in with sorting out various classes and object to frame bigger structures and give new usefulness [35].

C. Behavioral patterns

Behavioral design patterns are design patterns that identify basic correspondence designs among object and understands the patterns. By doing so, these patterns increment adaptability in doing this correspondence [30]. Behavioral patterns are about identifying basic correspondence designs among object and understanding the patterns that exist among them [35].

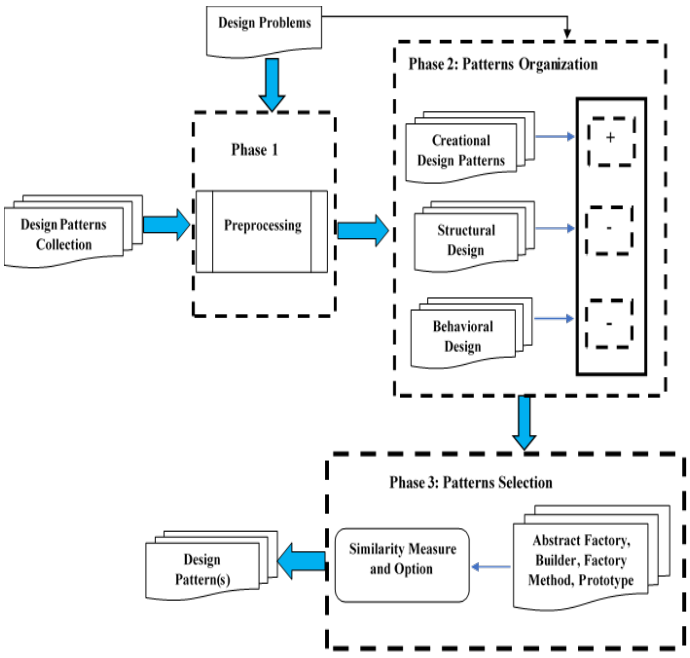

Figure 1: Diagrammatic Representation of Design Patterns Classification and Selection [34].

5.2 Design Pattern Topics

Six research design pattern topics concluded by [36] includes; pattern usage, quality evaluation, pattern mining, pattern specification, pattern development and miscellaneous issues.

i. Pattern development: This involves any advancements in design pattern research such as:
   a. Proposing a new model or new model language
   b. Reviewing model variants, composing models or elaborating a specific model, model evolution.
   c. Arranging current design patterns into distinct areas.
ii. Pattern usage: this relates to the commitment of utilizing patterns in the software development process. They are characterized into two primary groups:
   a. Pattern utilization
   b. Pattern application
iii. Pattern mining: this includes discovering examples of the pattern in the code or design of a system. It can be characterized into two major classes:
   a. Introduction
   b. Evaluation
iv. Quality Evaluation: the quality and effect of a design pattern on a system after applying it, is one important concern developers have. This can be characterized into two major classes:
   a. Pattern evaluation
   b. Application evaluation
v. Pattern specification: it includes utilizing distinctive strategies and notation of representing the patterns. The two main groups are:

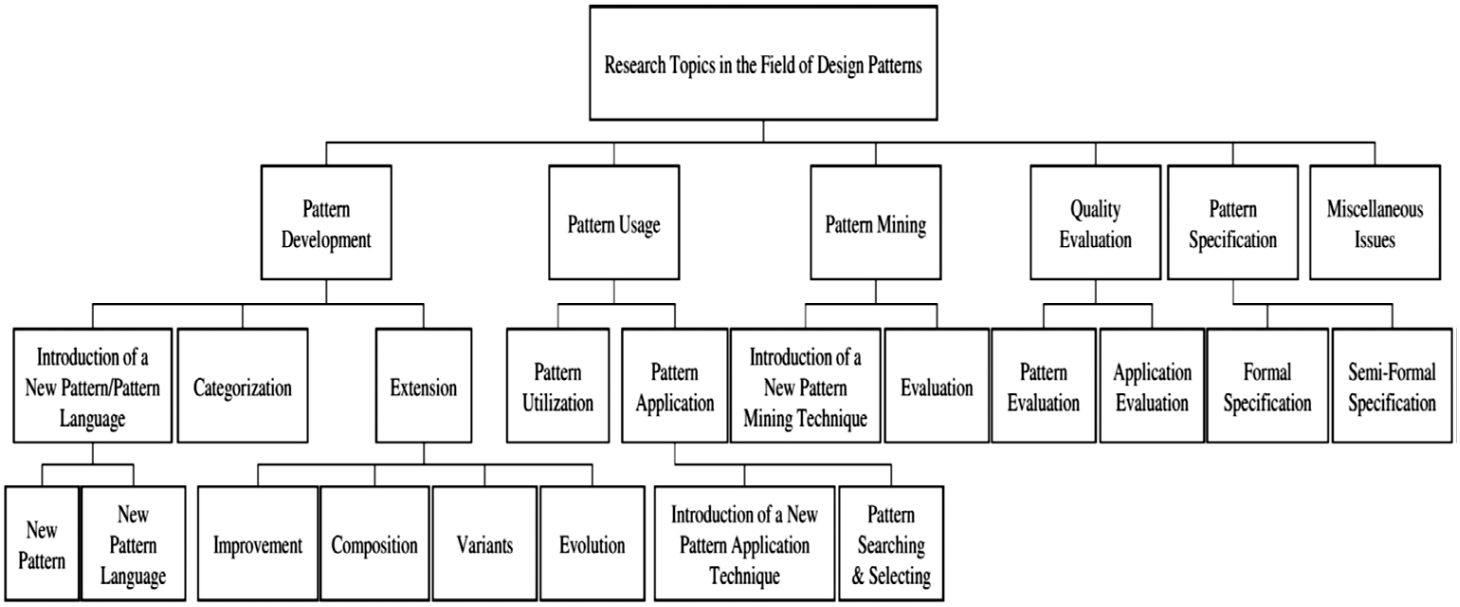

Figure 2: Design Pattern Research Tree [36]

a. Formal specification schemes
b. Semi-formal specification schemes

vi. Miscellaneous issues: this includes other issues that cannot fit into any of the previous classifications' issues such as re-factoring, code smells and anti-patterns. A major challenge of design patterns as discussed in [34,38, 39] is the searching and selection of design patterns before employing a right pattern into the system.

5.3 Pros and Cons of Design Patterns

It is worthy of note that there are very many design patterns available and a lack of understanding of these patterns pose problems for designers in software development especially for the novice [37]. Some of the pros and cons of the design patterns include:

Pros: [40]

- Easy to adapt and very flexible to predictable changes in business needs.
- Easy to test unit and validate individual components.
- Can provide organization and structure when business requirements become very complicated.

Cons: [40]

- Beginner engineers may not understand them, and these can cause a huge delay in development.
- Oftentimes used improperly without a realistic understanding of how the software is likely to change.
- It can add memory and processing overhead, sometimes it is not appropriate for applications such as low-level systems programming or certain embedded systems.

Design pattern, in general, enhances the nature of a product framework by giving a demonstrated solution for repeating design issues [41]. Also, the application of patterns brings about increment in quality and profitability of the software development process [31].

6. Results and Discussion

Table 1 shows several topics in software architecture and design alongside relevant work from several authors. Only one facet was used in this analysis, which are topics that relate to the subject (software architecture and design) discussed by these important authors. These topics were selected using inclusion and exclusion criteria. The essence of the selection criteria was to locate and add all papers that are necessary for the analysis. The inclusion and exclusion criteria were used to eliminate publications that were not significant to the study. It was observed that these topics were the most discussed and they also cut across academics and industry practice. The papers used for this review were access online from digital libraries like ScienceDirect, and IEEE explore. The authors and title of publications are listed in no particular order.

6.1. Software Evolution

Software evolution is the process of developing a software product, employing software engineering principles and methods. It follows the initial development of software and required maintenance. Updates are done till the desired software product is developed, thereby satisfying the (user) expected requirements [42]. This process implements changes to the original software, until the desired software is accomplished. Of the 35 papers used in this analysis, 8.6% of them reported on software evolution. From table 1, the findings show that software evolution though an integral part of software development was not substantially discussed by majority of the literature reviewed during the course of this study. This implies that there may be a decrease in research of software evolution.

6.2. Software Reuse

Software reuse involves creating new software systems from existing software frameworks rather than building software systems from beginning. This simple yet powerful methodology of software development was introduced in 1968 and now widely

Table 1: analysis of software architecture and design topics

Authors	Software Evolution	Software Reuse	Software Management	SDLC	Software Evaluation	Stakeholder Involvement/ Decision Making	Design Quality
Mekni, M., Buddhavarapu, G., Chinthapatla, S. and Gangula, M. (2018) "Software Architectural Design in Agile Environments"	-	-	-	x	x	-	-
Robillard, M.P. (2016) "Sustainable Software development"	-	-	x	-	-	-	x
I. Lytra, G. Engelbrecht, D. Schall and U. Zdun (2015) "Reusable Architectural Decision Models for Quality-driven Decision Support: A Case Study from a Smart Cities Software Ecosystem"	-	x	-	-	-	-	x
A. Ramirez, J. R. Romero and S. Ventura (2018) "Interactive multi-objective evolutionary optimization of software architectures"	-	-	-	-	-	-	-
O. Sievi-Korte, S. Beecham and I. Richardson (2018) "Challenges and recommended practices for software architecting in global software development"	-	-	x	-	-	-	-
N. Hamalainen, J. Markkula, T. Ylimaki and M. Sakkinen (2006) "Success and Failure Factors for Software Architecture"	-	-	x	-	x	-	-
A. Alhar, R. Mazamal and F. Azam (2016) "A Comparative Analysis of Software Architecture Evaluation Methods"	-	-	-	-	x	-	-
David Garlan. (2000) "software architecture"	-	-	-	-	-	-	-
R.Kazman, C.H Lung, (1997) "An approach to software architecture analysis for evolution and reusability"	x	x	-	-	-	-	-
Smrithi Rekha V and Henry Muccini, (2018) "Group decision-making in software architecture: A study on industrial practices"	-	-	-	-	-	x	-
Hans van Vliet and Antony Tang, (2016) "Decision making in software architecture"	-	-	-	-	-	x	-
R.Kazman, C.-H. Lung, s. Bot, and K. kalaichelvan (1997) "An approach to software architecture analysis for evolution and reusability"	x	x	-	-	-	-	-
M.Shaw, and P.Clements(2006) "The golden age of software architecture: a comprehensive survey."	-	-	-	-	-	x	-
H.V. Mohammad, A. Bavar, N.M. Khashayar, and D. Negin(2009)	-	-	-	-	-	-	-
I. Dobrica, and E. Niemela (2002) "A survey on software architecture analysis methods"	-	-	-	-	-	-	-
L. Tan, Y. Lin and H. Ye, (2012). "Quality Oriented Software Product Line Architecture Design,"	-	-	-	-	-	-	x
T. Mens, J. Magee and B. Rumpe(2010). "Evolving Software Architecture Descriptions of Critical System"	x	-	-	-	-	-	-
P. Abrahamsson, M. Ali Babar and P. Kruchen(2010). "Agility and Architecture: Can They Coexist"	-	-	-	x	-	x	-
A.Tang and M. F. Lau (2014) "Software architecture review by association"	-	-	-	-	-	x	-
H.Vlieta and A.Tang(2016) "Decision making in software architecture"	-	-	-	-	-	x	x
P.Y. Reyes-Delgado, M. Mora,H. A. Duran-Limon, L. C. Rodríguez-Martínez,R. V. O'Connorn and R. Mendoza-Gonzalez,(2016)	-	-	-	-	-	-	-
W. Hasselbring (2018) "Software Architecture: Past, Present, Future"	-	x	-	-	-	x	x
S. Orlov and A. Vishnyakov (2017) "Decision Making for the Software Architecture Structure Based on the Criteria Importance Theory"	-	-	-	-	-	x	-
Nitin Upadhyay (2016) "SDMF: Systematic Decision-making Framework for Evaluation of Software Architecture"	-	-	-	-	-	x	-
T. Kim, S. Yeong-Tae, L. Chung and Dung T. Huynh (2015). "Architecture Analysis: A Dynamic Slicing Approach"	-	-	-	-	-	-	-

M. Razavian, B. Paech and A. Tang (2018). "Empirical Research for Software Architecture Decision Making: An Analysis"	-	-	-	-	-	x	-
E. J. Eichwald, E. C. Lustgraaf, & B. Wetzel, (1999). "Transfer of the White Graft Reaction"	-	-	-	-	-	-	-
C. Manteuffel, P. Avgeriou and R. Hamberg (2018) "An exploratory case study on reusing architecture decisions in software-intensive system projects"	-	x	-	-	-	x	-
P.Bengtsson, (1999). "Software Architecture-Design and Evaluation."	-	-	-	-	-	-	-
A. Sharma, M. Kumar and S. Agarwal (2015) "A Complete Survey on Software Architectural Styles and Patterns"	-	-	-	x	-	-	-
M. Ozkaya and M. A. Kose (2018) "SAwUML – UML-based, contractual software architectures and their formal analysis using SPIN"	-	-	-	-	-	-	x
G. Vazquezab, J. Andres, D. Pacec and M Campoab (2014) "Reusing design experiences to materialize software architectures into object-oriented designs"	-	-	-	-	-	-	-
B. Jalendar, A. Govardhan and R. Emchand (2012) "Desiging code level reusable software components"	-	x	-	-	-	-	-
B. Kitchenham, S. Charters (2007) "Guidelines for performing systematic literature reviews in software engineering"	-	x	-	-	-	-	-

adopted all over the world [43]. Only 20% of the papers reviewed, showed software reuse as a major aspect of modern software development**.** More literatures reviewed discussed this subject as indicated in table 1 above, this implies that research still goes on in this area with respect to software architecture and design, as well as in the aspect of software development.

6.3. Software Management

Software management refers to the art and science of leading and planning software projects. From the analysis carried out, software management covers 8.6% of the relationship between software architecture and design and other parameters considered. Like software evolution, the aspect of software management was not widely discussed in literatures reviewed. Substantial research is therefore made in software architecture and design in this area.

6.4. Software Development Life cycle

Software Development Life Cycle (SDLC) gives a description of the development process of a system from the initial study until the time it is updated or replaced. There are six steps that make up the SDLC [44]. The major function of the SDLC is to neatly lay out the process of system development. Despite being a popular and well-discussed topic in practice and theory. 8.6% of the reviewed papers discussed the topic. Unlike other topics, this is a major aspect of software, from the reviewed literature there was no substantial amount of discussion made in this aspect**.**

6.5. Software Evaluation

Comprehensively, non-systematic checklists can be applied to a program in the software evaluation process [45]. In recent times, software assessment using theory-based approaches which incorporates relevant criteria derived from psychological, linguistic and pedagogical models of language learning and teaching has been proposed. 11.4% of the 35 articles reviewed, discussed the importance of software evaluation in developed systems. There has been current research going on in the aspect of software evaluation, hence making it one of the major discussed topics from table 1 above.

6.6. Stakeholder Involvement/ Decision Making

The nature of the design problem also determines the form of decision that will be made. As reported in [46], a structured design problem makes the decision-making process better and easier. In this analysis, 34.3% of the articles reviewed, considered stakeholder involvement and in some cases, decision-making as a factor in software architecture and design, making it the most emphasized topic. This had the highest number of literatures discussing the topic, this implies that currently, more researchers are gearing towards this aspect during the course of their research to emphasize the need of stakeholders' involvement during software development.

6.7. Design Quality

If the quality of a design is not considered properly, it could lead to a negative impact on the product being developed [47]. The quality of any software is dependent on how well it conforms with the design plan of that product, it determines if the product would deliver the requirements desired properly and efficiently. 17.1% of the papers used for analysis discussed this topic, either directly or by evaluation of some design quality factors like quality attributes or design decisions. Also, this was among the topics discussed substantially in the selected literatures, which makes design quality a high recommended aspect of software architecture and design both in terms of academics as regards research and also in the industry.

From the results, the most discussed topics gotten from the analysis of selected literatures were stakeholder involvement and design, software reuse, design quality and software evaluation while the least discussed topics include software management, software evolution and software development life cycle.

7. Conclusion

Software architecture and design is an important component in the software engineering field. For success in the software engineering field both the architecture and design of software must be considered. Hence, various fundamental topics as regards software architecture and design have been analyzed.

The objective of this paper was to critically analyze current topics in software architecture and design. The method of analysis

adopted was the collection of published papers and articles on the topics discussed in the paper and the percentages of each fundamental topic was calculated. From the analysis, the result showed that, of 35 papers used in analysis, 34.3% discussed stakeholders' involvement and decisions, 17.1% for design quality, 20% examined software reuse while 11.4% discussed software evaluation and 8.6% of papers reviewed discussed software management, evolution and software development life cycle each.

From the analysis, it can confidently be concluded that aspects of software architecture and design such as software evolution, management, re-usability and building software which are fault tolerant, reliable, backward compatible, maintainable and secured are under-discussed. Several authors addressed various aspects of software architecture and design, but there are no standard procedures to follow that addresses all issues of concern in general software developments. As earlier stated, some factors should be non-negotiable in software development to ensure standardization thereby reducing incessant collapse of systems witnessed in the early years of software developments. This research stressed the significance and rigorous work involved in the development of software and outlined major factors that should be considered.

Therefore, it is important to note that a critical and rigorous analysis of software architecture and design is required to overcome the overall failure or crash of software in software development process and to also identify relevant gaps in the architecture and design styles or methods. Software architecture and design as an ever-growing field of software engineering, calls for further analysis to test and validate principles as they evolve. This study would help other researchers in the quest of knowing more about software development and the need to research further on the least discussed topics which are software management, software evolution and software development life cycle (SDLC).

Conflict of Interest

The authors declare no conflict of interest.

References

[1] C. Paul, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford, (2010). "Documenting Software Architectures: Views and Beyond", Second Edition. Boston: Addison-Wesley. ISBN 978-0-321-55268-6, pp. 592.

[2] D. E. Perry, A. L. Wolf, (1992). "Foundations for the study of software architecture". ACM SIGSOFT Software Engineering Notes. 17 (4): CiteSeerX 10.1.1.40.5174. doi:10.1145/141874.141884, pp. 40.

[3] "SoftwareArchitecture".(2018), https://www.sei.cmu.edu/educationoutreach/courses/course.cfm?coursecode=p34. Retrieved 2018-07-23.

[4] B. Len, P. Clements and R. Kazman, (2012). "Software Architecture in Practice", Third Edition. Boston: Addison-Wesley. ISBN 978-0-321-81573-6.

[5] M. Fowler, (2003). "Design – Who needs an architect?". IEEE Software. 20 (5): 11–44. doi:10.1109/MS.2003.1231144

[6] F. Peter and D. Hart, (2004). "A Science of design for software-intensive systems". Communications of the ACM. 47 (8): 19–21 [20]. doi:10.1145/1012037.1012054.

[7] A. Davis, (2004)" Principles of Software Development", McGraw-Hill, Inc. New York, NY, USA, ISBN:0-07-015840-1.

[8] J. Spacey, (2017) "Software Design Vs Software Architecture.", https://simplicable.com/new/software-design-vs-software-architecture , Retrieved 2018-07-23.

[9] E. Novoseltseva, (2016). "https://apiumtech.com/blog/15-benefits-of-software-architecture/", Retrieved 2018-07-23.

[10] R.Kazman, C.-H. Lung, s. Bot, and k. kalaichelvan, (1997) " an approach to software architecture analysis for evolution and reusability", center for advanced studies conference, pp. 144-154.

[11] M.Shaw, and P. Clements, (2006)" the golden age of software architecture: a comprehensive survey.", institute for software research international school of computer science. Camegie mellon university, Pittsburgh, PA,0740-7459/06/.

[12] A.Alhar, M. Liaqat and F.Azam, (2016). "A Comparative Analysis of Software Architecture Evaluation Methods" Journal of Software Vol 11, No. 9. DOI:10.17706/JSW.11.9.934-942.

[13] N. Hämäläinen, J. Markkula, T. Ylimäki and M. Sakkinen, (2006)"Success and Failure Factors for Software Architecture". Available: https://jyx.jyu.fi/bitstream/handle/123456789/41384/Article_Success_and_Failure_Factors_for_SA.pdf;sequence=1 , Retrieved: 2018-08-21.

[14] M. Mekni, G. Buddhavarapu, S. Chinthapatla and M. Gangula, (2018) "Software Architectural Design in Agile Environments," Journal of Computer and Communications, 6, pp. 171-189. Available: https://doi.org/10.4236/jcc.2018.61018 .

[15] I. Lytra, G. Engelbrecht, D. Schall and U. Zdun, (2015) "Reusable Architectural Decision Models for Quality-driven Decision Support: A Case Study from a Smart Cities Software Ecosystem". Available: https://eprints.cs.univie.ac.at/4330/1/paper.pdf . Retrieved: 2018-08-21.

[16] M. P. Robillard (2016). "Sustainable Software development,". Available: https://www.cs.mcgill.ca/~martin/papers/fse2016.pdf . Retrieved: 2018-08-21.

[17] l. Dobrica, and E. Niemela, (2002) "a survey on software architecture analysis methods", IEEE Transactions on software engineering, vol 28, no. 7. Available: https://www.researchgate.net/publication/3188246. Retrieved: 2018-07-23.

[18] Antony Tang and Man F. Lau, (2014) "Software architecture review by association", Journal of Systems and Software", Volume 88, pp 87-101, ISSN 0164-1212.

[19] P.Y. Reyes-Delgado, M. Mora, Hector A. Duran-Limon, L. C. Rodríguez-Martínez, R. V. O'Connor and R. Mendoza-Gonzalez, (2016) "The strengths and weaknesses of software architecture design in the RUP, MSF, MBASE and RUP-SOA methodologies: A conceptual review, Computer Standards & Interfaces", Volume 47, pp 24-41, ISSN 0920-5489.

[20] "Software architecture and design tutorial," Tutorialpoints, 2019.[Online]. Available: https://www.tutorialspoint.com/software_architecture_design/index.htm. Retrieved:27-Feb-2019.

[21] R. J. de Paula, V. Falvo and E. Y. Nakagawa (2016), "Architectural Patterns and Styles" [Online]. Available:https://edisciplinas.usp.br/pluginfile.php/977101/course/section/268862/S4-%20Architectural%20Patterns%20and%20Styles.pdf.

[22] S. Boyd, M. D'Adamo, C. Horne, N. Kelly, D. Ryan and N. Tsang (2013), "SOFTWARE ARCHITECTURAL STYLES" SENG 403-W2013 Paper Project (Group4) [Online]. Available:http://kremer.cpsc.ucalgary.ca/courses/seng403/W2013/papers/04ArchitectureStyles.pdf

[23] D. Ameller, X. Burgués, D.Costal,C.Farré, X. Franch, (2018) "Non-functional requirements in model-driven development of service-oriented architectures"Science of Computer Programming,Volume 168, Pp 18-37.

[24] G.Rodríguez, J. A. Díaz-Pace and Á.Soria (2018) "Information Systems A case-based reasoning approach to reuse quality-driven designs in service-oriented architectures", Information Systems Volume 77, pp 167-189.

[25] "What is service-oriented architecture (SOA)? - Definition from WhatIs.com," SearchMicroservices, 2014. [Online]. Available: https://searchmicroservices.techtarget.com/definition/service-oriented-architecture-SOA. Retrieved: 13-Dec-2018.

[26] "SOA (Service Oriented Architecture) Principles," Meet Guru99 - Free Training Tutorials & Video for IT Courses, 2018. [Online]. Available: https://www.guru99.com/soa-principles.html. Retrieved: 13-Dec-2018.

[27] S. Pulparambil and Y. Baghdadi (2018) "Service oriented architecture maturity models: A systematic literature review", Computer Standards & Interfaces, Volume 61, pp. 65-76.

[28] "Advantages and Disadvantages of Service-oriented Architecture (SOA)" Techspirited, 2018. [Online]. Available: https://techspirited.com/advantages-disadvantages-of-service-oriented-architecture-soa. Retrieved:27-Feb-2019.

[29] S. Hussain, J. Keung, and A. A. Khan, (2017). "Software design patterns classification and selection using text categorization approach". Applied Soft Computing Journal, vol.58, pp. 225–244. http://doi.org/10.1016/j.asoc.2017.04.043

[30] "Design Patterns and Refactoring", (2018). [Online]. Available: https://sourcemaking.com/design_patterns. Retrieved: 13- Dec- 2018.

[31] J.L. Barros-Justo, B.V.B. Fabiane, L.C. Ania, (2018) "Software patterns and requirements engineering activities in real-world settings: A systematic mapping study", Computer Standards & Interfaces 58, pp. 23-42.

[32] J.O. Coplien, N.B. Harrison, (2005) "Organizational Patterns of Agile Software Development", Pearson Prentice Hall, ISBN-13: 978-0131467408.

[33] E. Gamma, R. Helm, R. Johnson and J.Vlissides , (1995) " Design Patterns: Elements of Reusable Object-oriented Software". Boston: Addison-Wesley Longman Publishing Co., Inc.

[34] S. Hussain, J. Keung, M. K. Sohail, A. A. Khan, and M. Ilahi (2019). "Automated framework for classification and selection of software design patterns". Applied Soft Computing Journal, vol.75, pp.1–20. http://doi.org/10.1016/j.asoc.2018.10.049

[35] "Design Patterns | Set 1 (Introduction) - GeeksforGeeks", GeeksforGeeks, (2018). [Online]. Available: https://www.geeksforgeeks.org/design-patterns-set-1-introduction/. Retrieved: 13- Dec- 2018.

[36] B. Bafandeh Mayvan, A. Rasoolzadegan and Z. G. Yazdi (2017) "The state of the art on design patterns: A systematic mapping of the literature", Journal of Systems and Software, Volume 125, pp. 93-118.

[37] J. Kress, B. Maier, H. Normann, D. Schmeidel, G. Schmutz, B.Trops, C. Utschig-Utschig and T. Winterberg. (2014). "SOA and Cloud Computing", https://www.oracle.com/technetwork/articles/soa/ind-soa-cloud-2190513.html Retrieved: 2018-01-09

[38] A. Birukou, (2010) A survey of existing approaches for pattern search and selection, in: Proceeding of PLoP DISI - University of Trento, Italy, ACM 978-1-4503-0259-3.

[39] P. Velasco-Elizondo, R. Marín-Piña, S. Vazquez-Reyes, A. Mora-Soto and J. Mejia, (2016), Knowledge representation and information extraction for analyzing architectural patterns, Sci. Comput. Program. Vol. 121 pp. 176–189.

[40] 2018. [Online]. Available: https://www.quora.com/What-are-some-pros-and-cons-of-using-Design-Patterns-to-describe-your-business-model. Retrieved: 2018-12- 13.

[41] D. K. Kim, L. Lu, and B. Lee, (2017). "Design pattern-based model transformation supported by QVT". Journal of Systems and Software, vol.125, pp.289–308. http://doi.org/10.1016/j.jss.2016.12.019

[42] Tutorials.point. "Software engineering overview" available:https://www.tutorialspoint.com/software_engineering/software_engineering_overview.html, Retrieved: 2018-12-13

[43] C.W. Krueger, (1992) "Software reuse", School of Computer Science, G'arnegie Mellon University, Pittsburgh, Pennsylvania. Available: https://dl.acm.org/citation.cfm?doid=130844.130856

[44] En.wikibooks.org, (2012) "Computer Revolution". [Online]. Available: https://en.wikibooks.org/wiki/The_Computer_Revolution/MIS/SDLC. Retrieved: 2018-11-2.

[45] IGI Global, (2012) "Software Evaluation". [Online]. Available: https://www.igi-global.com/dictionary/software-evaluation/27677 Retrieved: 2018-11-3.

[46] C. Zannier, M. Chiasson and F. Maurer, (2007) A model of design decision making based on empirical results of interviews with software designers. Inf. Softw. Technol. Vol. 49 (6), pp. 637–653.

[47] L. Tan, Y. Lin and H. Ye, (2012). "Quality Oriented Software Product Line Architecture Design," Journal of Software Engineering and Applications, vol. 5, pp. 472-476, [Online]. Available: http://www.SciRP.org/journal/jsea

[48] A. Ramírez, J. R. Romero and S. Ventura, (2018)" Interactive multi-objective evolutionary optimization of software architectures" Information Sciences, Vol 463–464, pp 92-109 Available: www.elsevier.com/locate/ins

[49] O. Sievi-Korte, S. Beecham and I. Richardson, (2018)"Challenges and recommended practices for software architecting in global software development" Information and Software Technology, In press, Available: www.elsevier.com/locate/infsof

[50] D. Garlan, (2000), "software architecture", School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA.

[51] S. B. chung-Horng Lung, (1997) "An Approach to software Architecture Analysis for Evolution and Reusability," in Centre for Advanced Studies Conference, Toronto, Canada.

[52] V. S. Rekha and H. Muccini, (2018) "Group decision-making in software architecture: A study on industrial practices", Information and Software Technology, Vol. 101, pp. 51-63, ISSN 0950-5849.

[53] M. Unterkalmsteiner, T. Gorschek, R. Feldt and E.riks Klotins, (2015) Assessing requirements engineering and software test alignment—Five case studies, Journal of Systems and Software, Vol. 109, pp. 62-77, ISSN 0164-1212.

[54] H. Vlieta and A.Tang (2016)."Decision making in software architecture", Science Direct-Journal of Systems and Software,Vol 117,pp. 638-644, https://doi.org/10.1016/j.jss.2016.01.017 .

[55] A. Tang and M. F. Lau (2014). "Software architecture review by association", Journal of Systems and Software, Vol 88, pp.87-101, https://doi.org/10.1016/j.jss.2013.09.044 .

[56] T. Mens, J. Magee and B. Rumpe (2010). "Evolving Software Architecture Descriptions of Critical System", IEEE Computer Society, vol 43, pp. 42-48. DOI: 10.1109/mc.2010.136

[57] P. Abrahamsson, M. Ali Babar and P. Kruchen (2010). "Agility and Architecture: Can They Coexist", IEEE Computer Society, pp16-22.

[58] S. Orlov and A. Vishnyakov (2017). "Decision Making for the Software Architecture Structure Based on the Criteria Importance Theory", Procedia Computer Science, Vol 104, pp. 27-34, ISSN 1877-0509.

[59] Nitin Upadhyay (2016). "SDMF: Systematic Decision-making Framework for Evaluation of Software Architecture", Procedia Computer Science, Vol 91, pp. 599-608, ISSN 1877-0509.

[60] T. Kim, S. Yeong-Tae, L. Chung and D. T. Huynh (2015). Architecture Analysis: A Dynamic Slicing Approach. Dept. of Computer Science University of Texas at Dallas https://doi.org/10.1017/CBO9781107415324.004

[61] M. Razavian, B. Paech and A. Tang (2018). "Empirical Research for Software Architecture Decision Making: An Analysis", Journal of Systems and Software, ISSN 0164-1212.

[62] C. Manteuffel, P. Avgeriou and R. Hamberg (2018). "An exploratory case study on reusing architecture decisions in software-intensive system projects", Journal of Systems and Software, Vol 144, pp. 60-83, ISSN 0164-1212

[63] E. J. Eichwald, E. C. Lustgraaf, & B. Wetzel, (1999). Transfer of the White Graft Reaction. Proceedings of the Society for Experimental Biology and Medicine, vol. 126(3), pp. 619–620. https://doi.org/10.3181/00379727-126-32521

[64] P.Bengtsson, (1999). Software Architecture-Design and Evaluation.University of Karlskrona PerOlof Bengtsson Department of Software Engineering and Computer Science. https://pdfs.semanticscholar.org/6c1c/a7056fbb2ee1f82f04c3ae2b7b7e16f41c6c.pdf

[65] A. Sharma, M. Kumar and S. Agarwal,(2015) "A Complete Survey on Software Architectural Styles and Patterns," Procedia Computer Science, Vol. 70, pp. 16-28 https://doi.org/10.1016/j.procs.2015.10.019

[66] M. Ozkaya and M. A. Kose, (2018) "SAwUML – UML-based, contractual software architectures and their formal analysis using SPIN,"Computer Languages, Systems & Structures,Vol 54, pp. 71-94, https://doi.org/10.1016/j.cl.2018.04.005

[67] G. Vazquezab, J. Andres, D. Pacec and M. Campoab, (2014), "Reusing design experiences to materialize software architectures into object-oriented designs,"Information Sciences Vol. 259, pp. 396-411 https://doi.org/10.1016/j.ins.2010.03.013

[68] B. Jalendar, A. Govardhan and R. Emchand, (2012) "Desiging code level reusable software components", International Journal of Software Engineering & Applications, Vol. 3, n. 1, pp. 219-229.

[69] B. Kitchenham and S. Charters, (2007) "Guidelines for performing systematic literature reviews in software engineering", EBSE,Software Engineering Group, School of Computer Science and Mathematics, Keele University, Keele, Staffs, ST5 5BG, UK and Department of Computer Science, University of Durham, Durham, UK .

[70] W. Hasselbring (2018). "Software Architecture: Past, Present, Future in: Gruhn V., Striemer R. (eds) The Essence of Software Engineering". Springer, Cham. https://doi.org/10.1007/978-3-319-73897-0_1]