International Journal of Engineering, Science and Mathematics
Vol. 10 Issue 3, March 2021,
ISSN: 2320-0294 Impact Factor: 6.765
Journal Homepage: http://www.ijesm.co.in, Email: ijesmj@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

INSIGHTFUL ARGUMENTATION-BASED INVESTIGATION OF SOFTWARE ARCHITECTURE REASONING DURING DESIGN COLLABORATION

Ranjan Sah

M.Phil, Roll No. : 140439, Session – 2014-15

Department of Computer Science, B.R.A. Bihar University, Muzaffarpur, India

ABSTRACT:

This investigation delves into the profound realm of software architecture reasoning during the design phase, employing a methodology centered on insightful argumentation. Software architecture is the cornerstone of any robust software system, dictating its structure, functionality, and scalability. However, the reasoning processes that guide architects in making critical design decisions are often intricate and multi-faceted. This study adopts an argumentation-based approach, recognizing the inherent complexity of software architecture reasoning and the need for a structured analysis framework. By leveraging argumentation as a tool for exploration and evaluation, the research aims to uncover the nuanced layers of decision-making involved in the architectural design process. The investigation encompasses a comprehensive review of existing software architecture models, design patterns, and best practices, laying the groundwork for a systematic analysis of reasoning patterns. Drawing inspiration from critical thinking and formal argumentation, the research seeks to identify and categorize key arguments that architects employ when making pivotal decisions. Furthermore, the study explores the impact of various contextual factors, such as project requirements, stakeholder preferences, and technological constraints, on the reasoning process.

keywords: *Argumentation, Software, Architecture*

INTRODUCTION

The structures of a computer programme or system are described by the acronym SA, which stands for software architecture. Aside from the software components, these structures also include the externally observable piece features and the relationships between them. Every area of human life has been profoundly impacted by the ubiquitous availability of software. Higher standards of software quality are expected as a result of both the ever-increasing demands of consumers and the abundance of applications. Thus, it is more challenging to understand, oversee, quantify, and regulate software, and even to reduce software complexity, inside the domain of software architecture. Reason being, software design is dynamic and ever-changing. To reduce the amount of time needed for realisation and increase productivity, software architecture plans, implements, and collects the analysis of operations within a family of systems. This is achieved by generating, carrying out, and amassing such measures. Stakeholders gain from SA because it facilitates efficient development, makes it easier to promote reuse, and helps stakeholders understand design decisions. Furthermore, SA facilitates the promotion of reuse.

We are currently methodically reconstructing the most fundamental needs in software development to meet the demands of new prospects that have arisen due to the demand for technologies. One goal of software architecture is to facilitate the automated development of software systems that meet user-specified specifications. Its current focus is on automating the process of designing software systems' architectures and it is also making headway in developing executable programmes. These two events are occurring at the same time. It is feasible to accomplish this due to the programme's architectural designs, the use of standard solutions, and the efficient utilisation of the software system's quality characteristics. With the modular approach, the software architecture may be seamlessly incorporated into an existing business. Using the modular approach makes this possible. When it comes to the IT industry's existing assets and infrastructure, it can provide adaptation and flexibility. Implementations of software architecture are still making noise in the corporate world. The goal of these implementations is to offer a structure for multi-level technical effort integration, new ways to promote and reuse technology, and ways to capitalise on the strengths of existing systems.

Software Architecture in Software Engineering

During software engineering, software architecture is a useful tool. Helping to expose a system's structure while hiding some implementation details is its stated goal. The focus in architecture is similar to that in software engineering: on the relationships between parts and how they work together. Everyone knows that there's a lot of overlap between software architecture and software engineering. Both of these things have been merged into one since many of the rules that control them are the same. It is possible to notice the difference when decisions are made with software engineering in mind and the software architecture is the result of those decisions.

Remember that while technical engineering encompasses software architecture in its entirety, not all engineering does. Bear this in mind; it is important. It is the responsibility of the software architect in software engineering to identify which parts of the programme are fundamental to its inner workings and which are merely technical details. At now, software architecture sits on top of software engineering, which is in charge of choosing the way sophisticated system designs and creations are carried out. When it comes to software engineering, software architecture is now king.

Software Architecture Design Tools

The use of software architectural design tools allows for the building of software architecture that is devoid of significant flaws or challenges. This is done through the utilisation of these tools. It is possible to reduce the risk of mistakes occurring during the implementation of the programme or defects in the design that will have implications later on in the development process or when the software is used widely when one makes use of the proper tools. This is because one is able to reduce the likelihood of errors arising during the implementation of the programme.

It is feasible to build software that is free of any security problems by utilising software architectural design tools. This is something that can be accomplished. Considering that there are software risks involved with each and every step of the software development process, this is of the utmost significance. The ability to proceed with self-assurance is afforded to teams in the case that they are successful in avoiding software flaws or difficulties. On the other hand, given that this is not always possible, software architectural design tools not only need to be able to detect problems that arise during the creation of software, but they also need to be able to make the required repairs in a timely manner. When you utilise software architectural design tools that are able to uncover defects, you will have the ability to evaluate the core programme design, estimate the possibility of an attack, identify probable threat elements, and determine any holes or gaps in the security that is now in place. All of these things can be accomplished by using these tools.

Instruments such as those provided by CAST are able to discover and repair design errors across the course of the software architecture design process, which makes them cost-effective. This ability allows them to detect and rectify design problems effectively. This encompasses the first phases, which are the most advantageous period to identify and address problems. Through the utilisation of tools that do architectural risk analysis, threat modelling, and other jobs of a similar kind, the software architecture may be located, rectified, and changed.

Organisations that fail to make use of the right software architectural design tools may be caught aback by the challenges that may reveal themselves at a later period, the consequences of which may even be fatal. These difficulties may manifest themselves at a later time. There are software architectural defects that may go unnoticed for a length of time; nonetheless, they will eventually become visible. These flaws are present inside the framework of software architecture. It is necessary to provide a response to the question of

how much risk was involved and how much work was put in before it was identified. There is a possibility that this will have a detrimental impact not just on the bottom line of businesses but also on their safety and reputation.

If the individuals who are required to make use of the architecture are not aware of what it is, are unable to comprehend it to the extent that it can be applied, or (worst of all) misinterpret it and apply it in an unsuitable manner, then even the most superior architecture, which is the one that is the most perfectly suited for the task at hand, would be almost worthless. Although the architectural team put in a lot of effort, research, and hard work, all of their hard work and sophisticated design will have been for naught. This is because the design will not have the desired effect.

ARCHITECTURE FOCUSED ON SERVICE

In the realm of architecture, the style that is commonly referred to as Service Oriented Architectures (SOA) is one that adheres to the principle of service orientation. The integration of service-based development, service-based terms of service, and service performance objectives may be accomplished through a wide range of different methodological techniques. As a tool for integrating organisational processes in a number of different ways, as well as for promoting reuse while harnessing the value that is already there in legacy systems, SOA implementations continue to make their presence known in web services. This is because SOA implementations present a framework for integrating organisational processes. Due to the capability of service-oriented architecture to reduce inefficiency and inflexibility, the application processes that are crucial to the operation of the company may reap the benefits of this design. In addition to reducing the risk, it maximises the value that is already there in the process. This is a significant advantage. In addition, service-oriented architecture offers the idea that the active operations of a software organisation are not static, and it focuses an emphasis on the dynamic viewpoint in order to get better results. This is done in order to achieve better outcomes.

1. When a service that is comparable already exists, developers could not be receptive to the request. As a result, it is essential to maintain a service directory that is readily accessible, as well as one that makes use of the basic terminology that is utilised throughout the software development organisation.

2. When contemplating the design and execution of services, it is necessary to take into account the boundaries that are not traditionally regarded. For the purpose of promoting their reuse inside the organisation, coarse-grained development and modular services are helpful.

SOFTWARE TESTING

One of the most important considerations in the process of developing software is the quality of the programme that is being developed. Approaches that may be utilised to improve the quality of software and metrics have been the subject of a significant number of recommendations that have been contributed. A reduction in the number of flaws that

are still there is something that has to be done. A number of alternative approaches have been proposed in attempt to lessen the number of flaws that are still present after the process has been completed. There is a possibility that adequate testing can be carried out in order to cut down on the number of issues that are still present. Software testing is one of the most vital and crucial strategies that can be employed to effectively create high-quality software. It is also one of the most important procedures. One of the primary reasons for the failure of software is the testing of the programme that was carried out in order to determine whether or not faults were there. On the other hand, the process of testing software is one that is not just time-consuming but also expensive. It is also feasible to define software testing as the process of validating and verifying software in order to ensure that the programme fulfils both the technical and business requirements in the manner that is anticipated. This is another way of describing software testing. Verification is carried out in order to ensure that the validation is as close as possible to the functional testing. The Software Under Test (SUT) procedure is used to attain this goal. This procedure assures that the software fulfils the standards and is as close as feasible to the structural testing. There are two sorts of testing methodologies that are often utilised, and they are structure testing and functionality testing. Performing structural testing within the code itself is something that is done. For the purpose of functional testing, the needs of the system's functional capabilities serve as the basis. It is possible to carry out testing in either an automated or manual fashion by making use of various testing tools. The results of a comparison between automated testing and manual testing show that the latter is not better to the former.

SOFTWARE METRICS

At this point in time, it is essential for the methods that are utilised for software testing to develop into a monitor of leading software that is able to carry out an efficient quality and control examination. When it comes to the process of software testing, it is feasible to pinpoint the developments that have taken place as a result of the discovery of fresh techniques to approximate the features of their quality measures. Automatic software testing is performed with the intention of identifying the features that can be quantified and demonstrating the progress that has been made in terms of quality modifications. Developing high-quality software requires a number of components, two of which are the software testing process and an efficient software quality process. Both of these processes are incredibly significant. In the supply sector of the service-oriented business, the software metrics approach is a method that has the potential to be included into the management information system and employed. Because it acts as both a framework and an evaluation for the improvement, the service-oriented software metric is the foundation upon which every corporate improvement is built. Metrics for software products are utilised to evaluate the characteristics of a product that is in the process of being developed, whereas metrics for software processes are utilised to evaluate the quality of software.

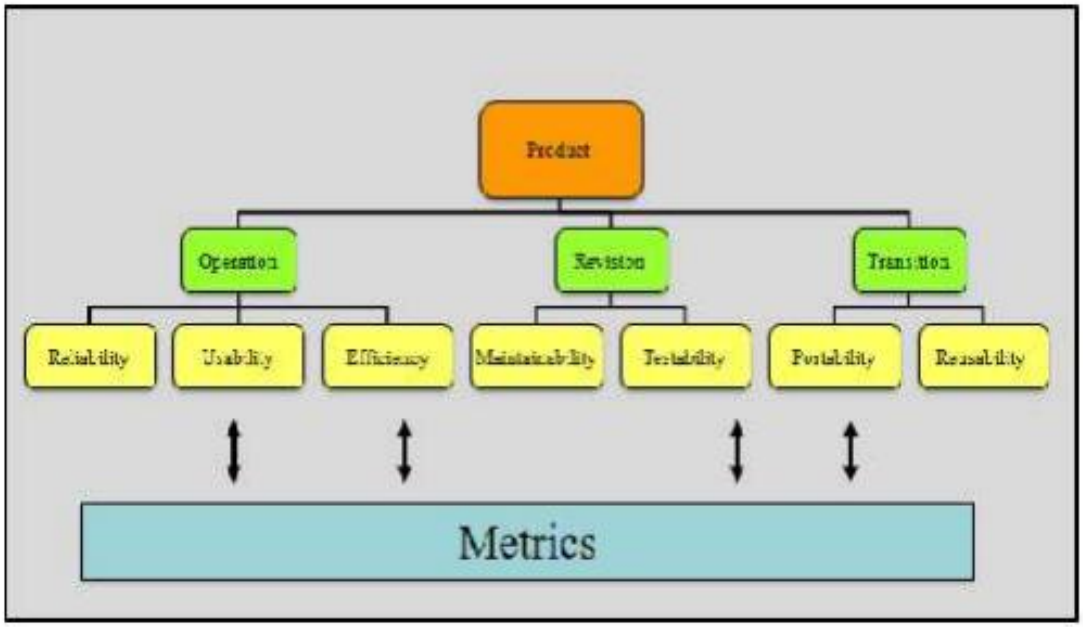

Figure 1 Model for Evaluating Software Quality

Metrics in software development are a way to quantify the observable, measurable, and calculable aspects of a software product or process. The term "software metric" describes this kind of evaluation tool. The program's measurements are utilised to create the software metrics. Figure 1, which is up there, shows these measurements. It makes it easier to put a numerical value on a certain metric. The term "metrics" is used to describe the process of measuring different parts of an activity so that it might be easier to assess how far along the path to success it is. Metrics are frequently domain-specific; in this case, that means they have limited applicability and cannot be used for direct comparisons or analysis outside of the specified domain. To sum up, they have little use outside of that specific field. Both the software development process and the assessment of the software product should be considered when talking about software metrics. You may use them to analyse and evaluate software products, or even to predict how such products will perform. In order to process or compare software solutions, several measurable indicators are utilised. The components must be adequately documented, and a thorough description of the software metrics must be provided when they are being used. Table 1.1 provides a tabular representation of a software metric measuring example. We do this so you may see how far along the track investigation is and how far along the software maintenance activities are.

Table 1 Elements of Software Metrics

Units	Explanation
Entity	Units of measurement
Target rate	Best possible value of the metric
Measurement process	Description of what is measured
Metric	Name of metric
Metric report	Description of what is measured

The software's quality is an important factor to consider while working with procedures that aim to provide services. A system's quality is identical to its functional and non-functional features, not different from them. Technical support, training, and productivity are only a few of the many aspects contributing to the enormous influence of software usability on software industry evaluations as a whole. All of these things are contributing to the already substantial impact. Software firms are preparing to embrace new or updated versions of software because of this, and the usability of software is reaping financial rewards as a result of this. When considering the time and effort needed to do the evaluation, together with objectivity and validity, the usability of software assessment could be a difficult scheme to construct. From the standpoint of finding a middle ground between the three considerations, this is correct. Considerations such as a program's responsiveness or performance, its ease of maintenance, and the amount of time and energy it uses all contribute to its usability.

SOFTWARE ARCHITECTURE CHALLENGES

The Software Architecture acts as the blueprint document, and its primary function is to make it easier for the personnel who are participating in the development process to communicate with one another. In the modern world, the software business is increasing at a tremendous rate; hence, testing and maintenance may begin after the creation of a product has been completed. Throughout the course of my employment, I have engaged in a number of talks with a wide range of team leaders, project managers, and solution architects who are accountable for the execution of software architecture. Throughout the course of these discussions, I have repeatedly seen that the process of managing change in software is one that is loaded with difficulties. The quality assurance process is the second activity that makes a major contribution. It is for this reason that it is asserted that the care, clarity, and completeness of the software development process, together with accuracy and flexibility, without compromising reliability and performance, are extremely important throughout the entirety of the process. At the same time, the software testing, software maintenance, software analysis, software quality assurance, and programme debugging processes are all facing the same problem. Within this particular domain, the utilisation of slicing is employed in order to facilitate the simplification of their task. Following the production and distribution of the product, the provision of service takes on a greater significance, and it is important to the achievement of any individual's goals. One of the

things that really interests me about slicing is the fact that the value of connected services extends beyond the realm of development. It is fairly motivating to achieve anything in this prosperous sector, which is necessary in order to do the task for the thesis. The term "slicing" refers to the process of applying software on several levels, most commonly with programmes. When it comes to large systems, it is applied to software architecture through the use of software architecture. The slicing is computed when the impacting criterion is taken into consideration, and it is also impacted by the criterion that determines whether the application will flow in the forward or backward direction. Depending on the behaviour of the programme, the slicing process can also be carried out in a static or dynamic form.

LITERATURE REVIEW

Liang et al. (2010). A paradigm shift has occurred in software architecture, with the emphasis moving from detailing the end result of the architecting process to recording architectural details. The primary goal of this field of study has been to record architectural information, including the reasoning behind the decisions made throughout the design process. Furthermore, software architecture is primarily a distributed, global, and collaborative process. A vital and integral part of this system is the sharing and recycling of architectural information. Despite the long-standing acknowledgement of the importance of architectural knowledge, no systematic approach has been developed that prioritises the integration of architectural information into collaborative frameworks. In order to address this issue, this chapter suggests a two-pronged approach: first, a collaborative architecting methodology based on architectural knowledge; and second, a tool suite that provides one method to facilitate this technique. This chapter provides a comprehensive presentation of both of these options. These two parts are covered in detail in this chapter.

Tang et al. (2011). a description of the needs as well as the architecture's design It is especially important to keep this in mind throughout the design phase, which is when both the requirements and the architectural design are simultaneously developing. There may be disagreements and inconsistencies about the guidelines. There are a lot of reasons for this, one of which is that stakeholders do not have current knowledge of each other's work, which stops them from totally appreciating potential problems and inconsistencies. This is only one of the factors that contributes to this situation. Since specifications are typically written in a natural language, it is difficult to automatically track related information. This makes it difficult to find information. Additionally, this makes it harder to track down material that is connected. With the intention of resolving this issue, the goal of this chapter is to provide a general-purpose ontology that we have developed. We give an implementation of a semantic wiki that enables the definition of architecture design in addition to allowing for the traceability of requirements that are constantly growing.

McGregor et al. (2007). The Architecture Expert (ArchE), a tool for software architecture design, is currently being manufactured by the Software Engineering Institute (SEI). ArchE is now being manufactured. Not only does it include information about quality qualities, but it also connects the design of architectural components to the fulfilment of

standards for those traits. Learn how a graduate-level software architecture course at Clemson University made use of ArchE in this in-depth study. Experts in the subject served as the course instructors. This lecture delves into a wide range of topics, including ArchE's roles as a construction tool and an educational tool for the built environment. While the students did not approve of ArchE's immaturity, they were generally supportive of its usage. On top of that, the teacher was enthusiastic about using ArchE in the classroom.

Bass et al. (2004). In Lancaster, United Kingdom, on March 21, 2004, there was a workshop that was held on the subject of Aspect-Oriented Requirements Engineering and Architecture Design. The schedule consisted of both a presentation session and working sessions for participants to participate in. Through the course of the working sessions, particular topics that were associated with the early aspects were discussed. The primary purpose of the workshop was to focus on the challenges that are associated with the establishment of methodical software development processes when it comes to factors that are present at an early stage in the software life cycle. Additionally, the workshop aimed to investigate the possibility of the methodologies and techniques that were presented being scaled up to industrial applications.

Juranić et al. (2019). With the complexity of engineering operations rising at an alarming rate, it is becoming more clear that software assistance for design team communication and improved control of design process dynamics are necessities, especially in highly important scenarios. In cases where the result is really important, this is particularly the case. Therefore, this study aims to present a novel approach that makes use of a set of coloured Petri Nets (CPNs) to facilitate the practical implementation of design activities ontology. In order to do this, taxonomy components must be instantiated at the same time as rules and linkages must be modelled. Three case studies addressing designer cooperation and an in-depth investigation of several long-term development projects executed by a big industrial organisation formed the basis for the presented approach. Problems in achieving interoperability between ontology models and design assistance systems are the primary emphasis of this research. The primary goal of this effort is to streamline the process of integrating ontology models into business operations. In a team setting, the proposed method may pave the way for the instantaneous revision and dissemination of design documents. This has tremendous promise.

McDonnell, J. (2012). This paper will examine a conversation that occurred between two seasoned software engineers as they brainstormed potential features and functions for a new software programme. This interaction occurred while the designers were working on the app's development. How designers maintain a productive design process despite uncertainties and ambiguities regarding the brief and disagreements regarding design elements is the focus of this study, which pays attention to the conversational strategies that manifest the 'web of moves' (Schon, 1985) that characterises expert design behaviour. Paying close attention to them does this. The purpose of this research is to examine how hesitancy can pave the way for productive teamwork and to highlight the conversational

strategies used to make room for differing opinions. In order to capture and accept arguments on how design goals need to be satisfied, there is a specific emphasis on both explicitly mentioning disagreement and using technical language terminology. The work adds to our understanding of the nuanced conversational processes that foster great design collaboration by drawing attention to the need of recording events that benefit the collaboration itself. Furthermore, the study adds to our knowledge of these two processes.

Babar et al. (2007). The following is a description of a tool that may be utilised for the purpose of managing architectural comprehension and reasoning. Providing support for a framework that enables the gathering and utilisation of architectural information was the primary objective behind the development of the tool, which was created with the intention of strengthening the architecture process. The fundamental architectural components of the tool as well as its functionality are going to be explored within the confines of this article. Additionally, the article provides examples of how the tool may be utilised to assist well-known architectural design and research approaches. These examples are included in the paper.

RESEARCH METHODOLOGY

In comparison to other forms of architecture, service-oriented architecture often necessitates a greater level of architectural design quality. The objective of system testing is to ensure that any changes made to the architectural design will not have an effect on the functionality that has been described for the design. This is the goal of system testing. Through the utilisation of test cases, the tester is able to assess the level of quality that is present in the architectural design. It is possible that the procedure of carrying out all of the test cases will be challenging and time consuming. As a result, automated testing is utilised in order to get beyond these limitations imposed by the system. Through the utilisation of automated testing, the quality of a particular application software solution is confirmed. Automated testing is performed with the purpose of determining the proportion of test cases that include mistakes. Last but not least, the improved test cases result in a more positive conclusion for the testing process.

DATA ANALYSIS

The main focus of the methodology known as Service-Oriented Software Architecture (SOSA) is the creation of service-oriented reusable services. This approach is used to construct software architecture. Typically, different service providers handle these tasks. Since composition is very similar to component-based software architecture design, it is considered to be an element of SOA. For this reason, composition is an integral part of SOA (service-oriented architecture).

The goal of software quality evaluation is to determine the efficiency of application software. The term "software evaluation process" might describe this assessment as well. When evaluating the quality of application software, a wide range of quality metrics are used. This document presents a few of these measures. The two most common measures

used to evaluate software are its usability and its portability. The quality components are evaluated using these two measures. The metrics used in this process include the number of errors, the time it takes to run, the time it takes to finish the work, and the time spent on the task itself.

SERVICE DESIGN ESSENTIALS

Using Service Orientation Design Principles (SODP) to construct the solution logic of services that are housed within service-oriented architectures (SOAs) is something that has been recommended as a potential course of action. It is possible to employ any particular design paradigm as the basis for the development of software in order to get superior outcomes. When it comes to SOA, initial analysis of a significant nature is necessary in the majority of situations. Consequently, there is a risk that a service-oriented architecture that is built without actual processes would fail. This is because of the fact that this is the case. The path towards service-oriented architecture (SOA) would be a good transition that delivers on the benefits that were promised when it was adopted, and it would be helpful to design a set of rules in order to assure that this change would take place.

SODP is the application of these design principles that creates technology independent software. These design principles serve as a guideline for identifying and evaluating the software.

System architecture

When it comes to software development, the Service-Oriented Architecture (SOA) solution design pattern is a process that involves evaluating and validating the public interfaces in order to create a system that is capable of delivering service to either an end user or another service. The usability of the software may be reviewed with the aid of a range of metrics, such as the completed tasks, the amount of time spent on tasks (use time), the number of mistakes, and the satisfaction ratings. This evaluation can be carried out with the assistance of a process that makes use of six sigma approaches. Moreover, the matrix technique may be utilised to assess the program's portability, which is a significant asset.

Figure 1 System Architecture Diagram

Algorithm Performance Evaluation

In this section, we will cover the performance evaluation of the proposed Binary PSO algorithm, as well as the comparison of the recommended algorithm to the Genetic algorithm (GA) that is already in use. In order to effectively implement the system that is being proposed, an object-oriented programming language is utilised. An instrument that is employed for the goal of discovering the connections that are present between the classes is referred to as a bespoke source analyzer. Let us take into consideration two projects in order to evaluate the suggested technique in software clustering. These projects are as follows: Along with Marvin, Java2HTML A detailed explanation of the recommended classes and the connections between them can be found in Table 4.1. Additionally, the explanation is broken down into its component parts.

Table 1: Classes and their relationships

project	**Classes**	**Relationships**			
		Aggregation	**Generalization**	**Association**	**Dependency**
Java2HTML	43	21	5	22	11
Marvin	27	23	8	5	13

For the purpose of the evaluation, the two factors that are taken into consideration are the amount of time that is spent computing and the quality of the architecture.

Table 2 has a table that contains a list of the parameters that were used for the experimental evaluation. It was decided to make some adjustments to the parameters in order to improve the results and acquire a more in-depth understanding of the model.

Table 2: Parameter setting for the algorithm

Parameter	Binary PSO	GA
Population size	200	200
Termination criteria	MQ=1	MQ=1
No. of clusters	3	3

Java2 HTML: In order to evaluate the performance of each system with regard to the amount of time required for calculation and the quality of the design, the Binary PSO and GA algorithms were utilised.

Quality of the Solution: Figure 2 is a table that illustrates a comparison of the quality of the solutions that are created by two different algorithms, namely Binary PSO and GA. The table depicts a comparison of the quality of the solutions provided by both algorithms. The better fitness function value is a measurement that is employed in the process of evaluating the quality of the solution (also known as the solution quality evaluation). If we compare the performance of the GA technique to the graph of the fitness function of the Binary PSO algorithm, we find that the performance of the GA method is much inferior than the performance of the Binary PSO algorithm.

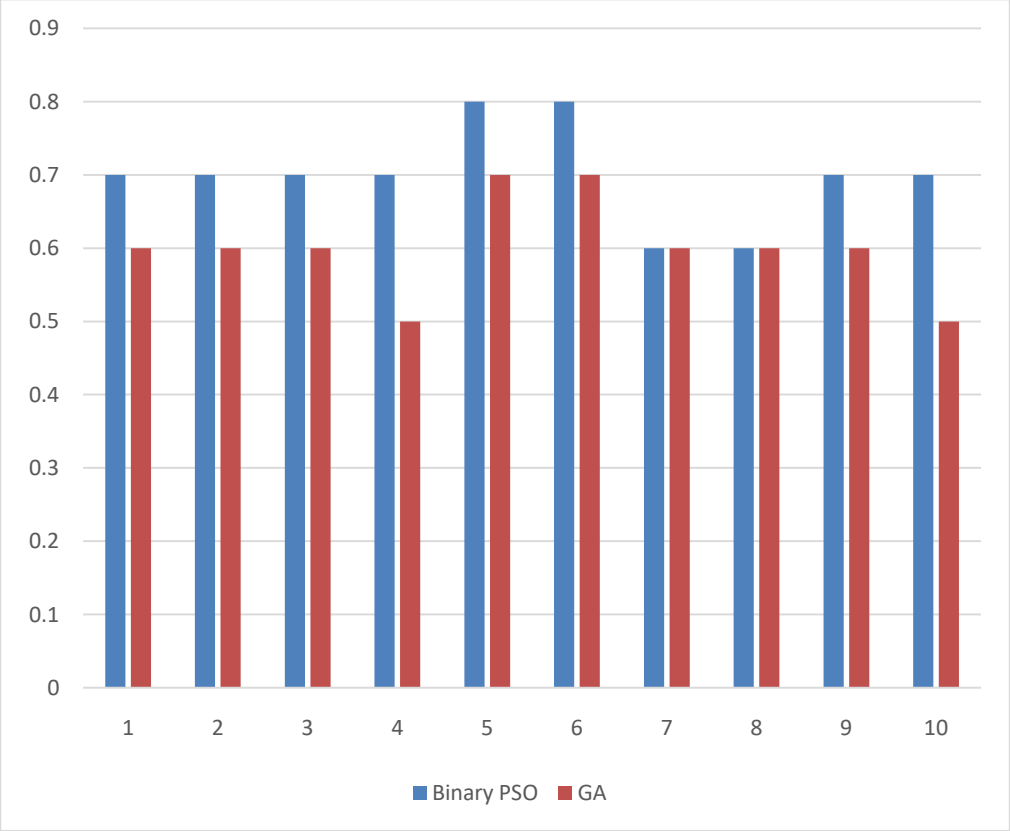

Figure2 : Fitness function Vs Iterations

CONCLUSION

There is a growing need for the creation of a variety of apps that are related to service as a result of the increasing prevalence of technology in a wide range of applications being utilised in day-to-day life. As a consequence of this, service-oriented design in software has evolved into an exciting research topic within the realm of information and culture. An efficient software architecture model that is based on software design and a pattern that is based on Service Oriented Architecture (SOA) was the objective of this research project. The quality metrics were the major focus of attention during the course of this investigation. In terms of the overall effectiveness of the architectural model and the quality of the building itself, the plan that was provided ended up being more successful than the other options. The primary objective of this thesis is to devote its attention to the investigation of service-oriented architecture and the open difficulties that it poses with regard to the quality of software, including usability and portability. It has been suggested that it might be beneficial to use a Hybrid Particle Swarm optimisation approach in order to test case optimisation. In addition, the concepts of six sigma are employed in order to compute the quality of a service-oriented architectural design model by utilising software quality criteria along the lines of usability and portability. The framework for applications that are built on service and application architecture provides support for two extra elements of service quality: availability and performance. These features are considered to be additional aspects of service quality. The optimisation of test cases has been achieved by the application of the genetic algorithm in conjunction with hybrid particle swarm optimisation. In the context of optimisation, the high-performance swarm optimisation (HPSO) approach is a technique that incorporates the use of both the concept of particle swarm optimisation and the genetic algorithm.

REFERENCE

[1] Capilla, R., Jansen, A., Tang, A., Avgeriou, P., & Babar, M. A. (2016). 10 years of software architecture knowledge management: Practice and future. Journal of Systems and Software, 116, 191-205.

[2] Tang, A., Liang, P., Clerc, V., & van Vliet, H. (2011). Supporting coevolving architectural requirements and design through traceability and reasoning. Relating Software Requiremens and Software Architecture.

[3] Liang, P., Jansen, A., & Avgeriou, P. (2010). Collaborative software architecting through knowledge sharing. Collaborative Software Engineering, 343-367.

[4] McGregor, J. D., Bachman, F., Bass, L., Bianco, P., & Klein, M. (2007, July). Using an architecture reasoning tool to teach software architecture. In 20th Conference on Software Engineering Education & Training (CSEET'07) (pp. 275-282). IEEE.

[5] Mangalaraj, G., Nerur, S., Mahapatra, R., & Price, K. H. (2014). Distributed cognition in software design: An experimental investigation of the role of design patterns and collaboration. MIS Quarterly, 38(1), 249-274.

[6] López, C., Codocedo, V., Astudillo, H., & Cysneiros, L. M. (2012). Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. Science of Computer Programming, 77(1), 66-80.

[7] Bass, L., Klein, M., & Northrop, L. (2004). Identifying aspects using architectural reasoning. Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 51.

[8] Tang, A., Avgeriou, P., Jansen, A., Capilla, R., & Babar, M. A. (2010). A comparative study of architecture knowledge management tools. Journal of Systems and Software, 83(3), 352-370.

[9] Mohsin, A., & Janjua, N. K. (2018). A review and future directions of SOA-based software architecture modeling approaches for System of Systems. Service Oriented Computing and Applications, 12(3-4), 183-200.

[10] Dobrica, L., & Niemela, E. (2002). A survey on software architecture analysis methods. IEEE Transactions on software Engineering, 28(7), 638-653.

[11] Olaf Zimmermann, Jana Koehler, Frank Leymann, Ronny Polley and Nelly Schuster.(2009),"Managing Architectural Decision Models with Dependency Relations, Integrity Constraints and Production Rules", The Journal of Systems and Software, vol.82, pp.1249–1267.

[12] Peng Liang, Anton, Jansen, Paris Avgeriou, Antony Tang and Lai Xu. (2011),"Advanced Quality Prediction Model for Software Architectural Knowledge Sharing", The Journal of Systems and Software, Vol.84, pp.786–802.

[13] Babar, M. A. Zhu, L and Jeffery R. (2004), "A Framework for Classifying and Comparing Software Architecture Evaluation Methods", Proceedings of the International Conferences of the Agent oriented Software Engineering Conference, Portugal, pp. 309-318.