AMERICAN JOURNAL OF INTERDISCIPLINARY RESEARCH AND INNOVATION (AJIRI)

ISSN: 2833-2237 (ONLINE)

SPECIAL ISSUE (2nd EIC-2024)

PUBLISHED BY

E-PALLI PUBLISHERS, DELAWARE, USA

2nd E-Palli International Conferences (EIC)
14 April 2024
Kathmandu, Nepal

E-Palli International Conferences on Science Technology, Engineering, and Mathematics (EIC-STEM)
DOI: https://doi.org/10.54536/ajiri.v3i4.3774
www.conferences.e-palli.com

Analysis of Software Patterns to resolve Design Problems

Jubayer Ahamed[1*], Mashiour Rahman[1*], Dip Nandi[1]

ABSTRACT

A reproducible solution to a frequently occurring problem in software design was described as a software pattern, also known as a design pattern. It was a tested method for resolving kinds of software design issues that had been classified and identified by several years of experience and best practices. The key to making safe and accessible software was good software design. There were different types of design problems. Solving those problems was very important to produce good and efficient software. There were some tools, techniques, and patterns to solve these problems. Software design knowledge could be effectively captured and communicated using design patterns. Within a particular context of software design, a software design pattern was a standardized, repeatable solution to a frequently -occurring problem. The patterns were effective for software design on several levels. In this paper, we identified the major design problems. Then we have analyzed those problems and present design patterns to develop an effective software design.

Keywords: Software Design, Design Problems, Behavioral Design Pattern, Creational Design Pattern, Structural Design Pattern

INTRODUCTION

The software design process is a robust and technology-independent idea that involves the ability to fulfill the tasks outlined in the requirement analysis phase. When deciding whether to proceed with an application for development, the consideration of software design becomes significant. Therefore, it is essential to keep to the many stages required to construct an interface that clients would appreciate and prefer. The design phase in the software development process is of the greatest importance as the choices made during this stage will affect the entire program. Software design offers a foundation for constructing software architecture. Establishing an understanding of the internal framework holds significant importance. An effective design will provide an extensive overview of the software's internal structure and internal properties. However, several problems arise throughout the design phase. Design patterns can address design issues. A design pattern can't be easily implemented using a programming language. It is an explanatory or conceptual solution that can be applied in various situations. An efficient software strategy will deal with potential problems that might not become visible until the final stages of execution. Programmers and architects that possess knowledge of design patterns can gain advantages from their reuse, as it allows them to address minor issues before they increase into significant problems and improves the readability of their code. Design patterns deliver comprehensive answers that are explained without the need to disclose details related to a particular situation. There are a wide variety of design patterns. Those identified patterns include Creational, Structural, Behavior, etc. Design patterns represent the most prominent techniques used by experienced software engineers. The design pattern follows issues, solutions, and outcomes. A design pattern refers to a methodology applied to develop software designs that are absent of issues.

LITERATURE REVIEW

The work (Kamal & Avgeriou, 2007) discussed the issues surrounding the attributes of software design. An analysis was conducted to compare the two process theories of software design, and the resulting conclusions were identified. The researchers participated in a discussion regarding the challenges associated with two process theories. A process theory attempts to clarify the sequential occurrence of events, rather than simply considering them within the framework of past or future activity

[1] *American International University-Bangladesh (AIUB), Bangladesh*
[*] *Corresponding author:* jubayer@aiub.edu

sequences. The effectiveness of their two process theory approaches throughout the software development phase was limited (Kamal & Avgeriou, 2007). The utilization of the design pattern will speed up the development process.

The researchers focused on developing the fitness function for the software design in their study (Bontchev, 2020). The primary goal of the fitness function was to establish the external relationship between classes in the design solution on an individual basis, and it is associated with the overall stability of the software design. To evaluate the fitness function, engineers gathered data or information to be utilized as a function. However, in the absence of employing any pattern-based technique, they were unable to achieve ideal efficiency. As a result, the outcome of the sequential functional patterns-based approach was unsatisfactory (Bontchev, 2020).

A software pattern (Gamma *et al.,* 1993) is often employed as a recurrent solution to an often-encountered problem. By combining similar patterns, a language is created that enables a systematic approach to resolving difficulties in software development. The researchers did not prioritize the development of tools or formal systems for expressing patterns. A limited proportion of engineers possess an accurate understanding and consistently apply these essential patterns in their day-to-day tasks.

Bontchev *et al.,* provided an explanation of the many kinds of patterns and their perspective on how patterns might contribute to the development of comprehensive solution techniques for software design challenges. Also, the author evaluated the elements that had identified as crucial aspects of the concept of patterns. The author analyzed the concepts of well-crafted pattern ideas and the approaches employed in software development. It engaged in a discussion regarding the many categories of patterns in the research. The discussion revolved around structural patterns and behavioral patterns (Gamma *et al.,* 1994).

P. Wolfgang and S. Hermann *et al.,* attempted to provide an extensive overview of modern pattern methodologies, including pattern libraries and framework patterns. The authors focused on the fundamental ideas that based frameworks prior to introducing more advanced design principles for constructing these artifacts. Design patterns were being recognized as beneficial for supporting framework development, however, there was still a lack of integration between object-oriented design and the development of frameworks. Understanding the design and implementation specifics of the framework proved to be challenging and required a significant amount of time due to the insufficient documentation. At times, understanding the decisions might be challenging when there are no indications provided (Dong *et al.,* 2009).

Gamma *et al.,* analyzed the differences between frameworks and design patterns. The programming language was explained, and an explanation was given on how frameworks operate. Software design frameworks had a higher level of association with the design patterns. Mature frameworks frequently use multiple design patterns for reuse. To sufficiently clarify the process of framework development at a more abstract level, individuals had to have a knowledge of design patterns (Simons *et al.,* 2010).

Kamal and Paris, (Gomes *et al.,* 2002) identified seven patterns that assist in the high-level design phase and clarified their role in establishing the architecture of systems with specific types of components. The pipeline architectural pattern, data abstraction architectural pattern, communicating processes of architectural pattern, implicit invocation architectural pattern, repository architectural pattern, interpreter architectural pattern, main program, and subroutines architectural pattern, and layered architectural pattern (Gomes *et al.,* 2002) were all topics of discussion.

The solution presented by the authors (Ralph, 2010) includes the Re-Builder framework to suggest suitable patterns, taking into consideration user experience. Re-Builder employed a case-based approach to reasoning, utilizing situations where a pattern has previously been utilized in a software design. In this context, class diagrams were employed to represent cases (Ralph, 2010). However, a notable drawback of this approach was the absence of the sequential process-based diagrams in certain cases, which consequently leads to a lack of success measures.

Initially, delegation, collection, and communication were used to define design patterns, which were divided into three categories: behavioural, structural, and creational patterns (Pree & Sikora, 1997) (Ho & Jézéquel, n.d.). An aggregate in the context of design patterns is an object, like a list, vector, or generator, that offers an interface for building iterators rather than a design pattern itself (Ho & Jézéquel, n.d.).

Table 1: Problem Classifications

1. A process theory attempts to clarify the sequential occurrence of events, rather than simply considering them within the framework of past or future activity sequences. The effectiveness of their two process theory approaches throughout the software development phase was limited (Kamal & Avgeriou, 2007).
2. Understanding the design and implementation specifics of the framework proved to be challenging and required a significant amount of time due to the insufficient documentation. At times, understanding the decisions might be challenging when there are no indications provided (Dong *et al.*, 2009).

3. A notable drawback of this approach was the absence of the sequential process-based diagram in certain cases, which consequently leads to a lack of success measures (Ralph, 2010).
4. Fitness functions describe how close an architecture is to achieving an architectural aim. To check the fitness function, engineers collected data or information to use as a function. But without using any pattern-based technique, they were not able to get the best functions (Bontchev, 2020).

Problem Classifications

Proposed Solution & Discussion

Developers can find well-known answers to recurring issues by using software patterns. Frameworks can be constructed with reusable components by utilizing well-known patterns.

Consider the use of design patterns in a manner like how one might approach changing lanes while driving. Whether you are operating a car, a bike, or a truck, or travelling to work, a museum, or another city, the standard procedure involves activating your turn signal, checking your mirrors, examining your blind spot, and then proceeding to change lanes. The process is constant and serves as a structured approach to guide individuals in completing tasks efficiently. We focus on pattern-based solutions in our study. We are going to propose some effective design patterns in the rest of the part.

Behavioral Design Patterns

Behavioral patterns can be applied to improve the structure of interactions between objects in multiple software systems. One instance is the chain of responsibility pattern, which may be applied to create a workflow system, distributing various jobs across distinct objects inside the chain. A categorized observer is designed to accommodate numerous readers and writers. This pattern expresses information across the entire system. This is working like a blackboard of a software design. It prevents linking the sender of a request directly to its receiver by allowing many objects the opportunity to manage the request. Connect the receiving objects in a chain and transmit the request through the chain until an object processes it. So that, overall design procedure follows a sequential way by using this pattern. Create an object that encapsulates the relationships among a group of objects. The mediator helps loose coupling by preventing items from directly referencing each other, allowing for self-sufficient adjustment of their relationship. We identify some issues like lack of efficiency of a design, absence of sequential way of a design. By using this pattern, developers can overcome these types of issues. Because this pattern follows a sequential way to build a design. As a result, the overall software design will be a good one and the level of efficiency will be high in that case.

Structural Design Patterns

Structural design patterns establish the connections among the components of a system. Like other design patterns, structural patterns address the most regular design issues. By applying them, it can reduce the application design process by providing answers for common issues related to component structure. There are seven types of structural patterns. Those are decorator, composite, flyweight, bridge, proxy, façade, and adapter. By using the types, we can improve the efficiency of a software's design and architecture. Also, some indications are provided during the design process to the decision makers. This will be very helpful for the decision makers. The overall design of the software will be a good one and the structure of the software will be effective as well.

Creational Design Patterns

The Creational Design Patterns will focus on the method by how things are produced. It eliminates complications and

Table 2: Creational Design Patterns

Solution A: Behavioral Design Patterns	Increase the efficiency of a design, It follows a sequential way to build an architecture by using programming language.
Solution B: Structural Design Patterns	It produces an effective structure. The components of an architecture are well connected to each other.
Solution C: Creational Design Patterns	It decreases the usage of process-based diagrams in software design. Therefore, the success metrics increase in that scenario. Engineers must gather data or knowledge to design architecture. Engineers will connect the fitness functions to this pattern in order to achieve an improved outcome.

uncertainty by the intentional creation of objects. This pattern reduces the coupling and increases the cohesion in a software environment and design. So that, the design procedure will be effective of any software. When the overall design is effective, then the structure of a software will be more efficient. It reduces the usage of process-based diagrams in the software design. As a result, the success measures are high in that case. Engineers will need to collect data or information to create architecture. Engineers will relate the fitness functions to this pattern to get a better result.

CONCLUSION

Design issues deteriorate software in various ways. This causes the software to be ineffective for users. Given the fundamental issues of the software, users have faced difficulties and feel it is challenging. The software doesn't have user-friendliness for users when it comes to structural-based issues. As a result, the software's effectiveness in the marketplace is limited. We acknowledge these concerns and seek to offer possible solutions. That is the reason why we focused our efforts on this specific issue. We have implemented many design patterns in an attempt to address these issues. Our objective is to offer software engineers effective alternatives. These solutions have the potential to reduce this difficulty and contribute to the improvement of software design. Initially, we attempted to identify the precise issues. Upon identifying the issues, we proceed with analysing various categories of design patterns. Next, we established six design patterns to address those issues. The following design patterns are included: Creational design pattern, Structural design pattern, Command design pattern, State of art design pattern, Abstract design pattern, and Alexander design pattern. After that, the solution for each challenge is established by employing these six design patterns in our suggested solutions and subsequent debate. As a result of insufficient resources, we are unable to carry out this implementation. In the future, we aim to offer a more precise solution that goes beyond the options currently available. If there is any complexity in these solutions supplied by us, then we will work on them in the future to provide the proper solutions.

REFERENCE

Bontchev, B. (2020). On the usability of object-oriented design patterns for a better software quality. *Cybernetics and Information Technologies, 20*(4), 36–54. https://doi.org/10.2478/cait-2020-0046

Dong, J., Zhao, Y., & Peng, T. (2009). A review of design pattern mining techniques. *International Journal of Software Engineering and Knowledge Engineering, 19*(6), 823–855. https://www.worldscientific.com/doi/abs/10.1142/S021819400900443X

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). Design patterns: Abstraction and reuse of object-oriented design. In *Proceedings of the ECOOP'93 Conference, Kaiserslautern, Germany. Springer Verlag.* https://cseweb.ucsd.edu/~wgg/CSE210/ecoop93-patterns.pdf

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: Elements of reusable object-oriented software. *Addison-Wesley*. https://www.javier8a.com/itc/bd1/articulo.pdf

Gomes, P., Pereira, F. C., Paiva, P., Seco, N., Carreiro, P., Ferreira, J. L., & Bento, C. (2002). Using CBR for automation of software design patterns. In *Proceedings of the 6th European Conference on Advances in Case-Based Reasoning* (pp. 534–548). https://www.researchgate.net/publication/220831615_Using_CBR_for_Automation_of_Software_Design_Patterns

Ho, W. M., & Jézéquel, J. M. (n.d.). Object-oriented frameworks for distributed systems.

Kamal, A. W., & Avgeriou, P. (2007, January). An evaluation of ADLs on modeling patterns for software architecture. Department of Mathematics and Computer Science, University of Groningen, The Netherlands. https://www.researchgate.net/publication/228814559_An_Evaluation_of_ADLs_on_Modeling_Patterns_for_Software_Architecture

Pree, W., & Sikora, S. (1997). Design patterns for object-oriented software development. In *Proceedings of ICSE 97, ACM, Boston, USA.* https://www-public.imtbs-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/PreeSikora97.pdf

Ralph, P. (2010). *Comparing two software design process theories.* Springer-Verlag Berlin Heidelberg. https://link.springer.com/chapter/10.1007/978-3-642-13335-0_10

Simons, L., Parmee, I. C., & Gwynllyw, D. R. (2010). Interactive, evolutionary search in upstream object-oriented class design. *IEEE Transactions on Software Engineering, 36*(6), 798–816. https://ieeexplore.ieee.org/abstract/document/543222