SPEED UP OF SOFTWARE DEVELOPMENT USING OBJECT ORIENTED DESIGN PATTERNS

MR. PABBOJU RAMESH ,
Professor, Department of C.S.E
Mahaveer Institute of Science and Technology, Bandlaguda, Hyderabad, T.S,
pramesh.448@gmail.com

DR. R.NAKKEERAN, HOD,
Department of C.S.E
Mahaveer Institute of Science and Technology, Bandlaguda, Hyderabad, T.S,
sudhandhiram64@gmail.com

Abstract: *Software quality is considered to be one of the most important concerns of software production teams. Software design patterns are a better solution for building large Object-Oriented (OO) software systems. They present well-tested and verified solutions to recurring difficulties that developers address. There are many advantages to using patterns. They can speed up the software development method. Design patterns combine learning to perform it more natural for designers to use well-known and strong designs developed from proficient experience. Simultaneously, software design patterns are too abstract and remain an art that has to be learned over time with experience. This study aims to propose a methodology for comparing design patterns to alternative designs with an analytical method. More specifically, the identification of such thresholds can become very useful for decision making during system design and refactoring.*

Keywords: *Design patterns, Object oriented programming, software performance analysis. Quality Attributes.*

I. INTRODUCTION

Design patterns idea is used to identify well-documented reusable solutions to common design problems. An appropriate variety of design patterns for use in software improvement have already been analyzed in the literature [1], [2]. For the realistic implementation of design patterns, a series of tools have been developed to discover and formalize patterns [3]. These patterns and design tools make it easy to understand and create structures. The design model tools aid in the predictable and uninterrupted use of services and the user's assets. Each pattern is described with the help of a pattern template. These form templates reinforce a clear answer to regular inconvenience. Generally, templates are used to capture all sample elements and describe their problems, motivation, strategies, technology, applicable possibilities, responses, and examples. Gamma et al. [4] Also known as Gang of Four (GoF), well-known models suggested twenty-three versions of different types of design patterns. After that, exceptional authors suggested various unique design styles. GoF design patterns are broadly categorized into three categories, construct, structural, and behavioral. Design patterns can also be considered a unique form of program architecture [5]. An object-oriented (OO) model is currently strongly advocated for software development instead of standard and feature-oriented methodologies. The object-oriented method has distinct properties, such as encapsulation, polymorphism, and genetics, making the code reliable and understandable. Building on OO principles, unique metrics are applied to measure program greatness using the item-oriented method.

A hierarchical model for an object oriented design, called Quality Model of Object Oriented Design proposed by Bansiya and Davis relates the quantitable Object Oriented Characteristics give better caliber of Software Quality Attributes. QMOOD model is characterized by four levels and three mappings. Figure 1 illustrates the structure of QMOOD approach.

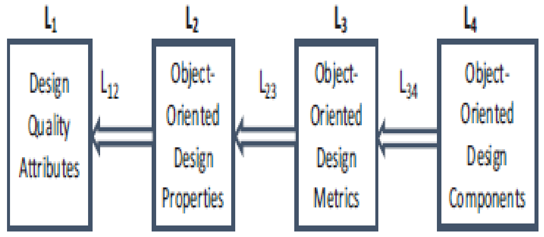
L1
L2
L3
L4
Design Quality Attributes
L12
Object-Oriented Design Properties
L23
Object-Oriented Design Metrics
L34
Object-Oriented Design Components

Fig.1 Quality Model of Object OrientedDesign Model

In general, a lot of the fun can be reused on an architectural level. However, reuse of design patterns does not always lead to direct reuse of algorithms, micro-designs, interfaces, or applications. Systematically explain a preferred design that addresses a routine design problem in object-oriented structures. Design patterns also describe problems and answer during use as well as outcomes for this individual solution. Also, it provides instructions and implementation examples and a diagram or template for a method for troubleshooting a problem that can be used under many unique circumstances. The solution is customized and implemented to solve the problem in a specific context. In Gang of Four (GoF), patterns usually contain these predominant factors: intention, motivation, application, structure, participants, collaboration, consequences, execution, sample code, known uses, and related patterns. Although the terminology may differ more from the author, all object-oriented design patterns are written following the elements above.

II. RELATED WORK

Object oriented design pattern is an active area of research in software development. Ackerman L and Gonzalez C explore the benefits of implementing patterns in software designs [7]. The article introduces developers and architects to the idea of a pattern implementation which is an artifact that allows the codification of a pattern specification for a specific environment and can be created and used for different phases in the software development lifecycle. Rising. L analyzes an example from a small development team and discovers that a novice pattern, called the Mediator, is a perfect fit for the design challenges that they had just spent hours battling.

In [9], an analysis was done to verify the reusability of design patterns and software packages, which uncovered some advantages and disadvantages of design patterns. Examples here developed by employing design pattern, alternative design, and compared, the result shown that patterns provide a solution that is most extensible. A team at Erricson developed frameworks based on design patterns for two years, according to. The history of design patterns and some developments characterizing the advantages and disadvantages of design patterns are presented in [9]. Appostollos 2006 evaluates usage of object oriented design patterns in game development, proving maintainability although the research was biased towards game development only.

Yacoub, S. A., Ammar, H. H., &Mili A., teaches developers about the abstraction benefits of design patterns. The paper explains the benefits that include understanding the dependencies and collaboration between participating patterns while hiding implementation details. Douglas C. Schmidt and Paul Stephenson present a case study illustrating the implementation of design patterns (Reactor and Acceptor) in object-oriented telecommunication software framework across UNIX and Windows NT OS platforms and discuss the techniques, benefits, and limitations of applying a design pattern-based reuse strategy to commercial telecommunication software systems.

There are 3 basic classes of design patterns.

a) **Creational Design patterns**
b) **Structural Design Pattern**
c) **Behavioral Design Pattern.**

A.Creational Design patterns

It is further categorized into

1) Object-Creational Patterns
2) Class – Creational Patterns

1. Object creational Patterns

It deals with Object creation. It defers part of its Object creation to another Object**.**

2. Class Design patterns

It deals its object creation to subclasses. The creational design patterns are abstract factory, builder, factory method, singleton and prototype.

These design patterns are basically concerned with class instantiation and are composed of two dominant ideas. One is encapsulating knowledge about which concrete classes the system uses and the other hides how instances of these concrete classes are created and combined. Creational design patterns are further categorized into Object-creational patterns and Class-creational patterns, where Object-creational patterns deal with Object creation and Class-creational patterns deal with Class-instantiation. In greater details, Object-creational patterns defer part of its object creation to another object, while Class-creational patterns defer its object creation to subclasses Gang of Fourq. The examples of creational design patterns are; abstract factory, builder, factory method, singleton and prototype.

b) Structural design patterns

These patterns ease software design by identifying a way to realize relationships between entities. Such patterns gives Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. But the structural object-patterns define ways to compose the objects to obtain new functionality. Examples are adapter, decorator, bridge, composite, flyweight, façade and proxy

c) Behavioral design patterns

Behavioral design patterns are going to find the common communication patterns in the between objects and realizes the assignment of responsibilities between related objects. By doing so, these patterns increase flexibility in carrying out this communication. The behavioral design patterns are Observer, chain of responsibility, interpreter, iterator, memento, template method, command, visitor, strategy and mediator.

III. HOW TO SELECT DESIGN PATTERNS

With many design patterns to choose from; it may be difficult to find one that addresses a particular problem, especially if the list is new and unfamiliar. The problem of searching patterns means the effort of getting information about existing patterns. The Selection of a particular pattern is described as the problem of deciding which pattern to choose among all available solutions. Below is an outline of approaches that can be used to find the design pattern that suits a particular problem.

According to the GoF e-book, there are five main methods to research and choose design patterns, which are; Sample repositories, pattern catalogs, advisory structures, official languages, search engines, and other tactics. Authors create online pattern repositories to increase pattern availability. In such repositories, patterns can be retrieved using search criteria and with the help of manual browsing between the different patterns. Moreover, various consulting systems have been proposed to indicate the exact pattern according to the inconvenience the developer wants to clarify. There are also many articles describing the processes that use the official languages so that one can represent design patterns and choose patterns compatible with any illustration. Pattern selection by search engines such as Google and Yahoo is compatible with keyword searches on slow-moving engines that index sample descriptions. Finally, there are many different technologies that cannot be categorized into any of the above categories made up of.

Accordingly, methods have been proposed that allow software developers to find the correct design samples for a particular design problem. So a degree is maintained that avoids the influence of the large number of design patterns found in exclusive design sample catalogs.

Consider how design patterns solve design problems: According to the GoF textbook discussion, design patterns help find appropriate objects, determine object granularity, specify object interfaces, and several other ways in which design patterns solve design problems.

Scan Intent sections from all the patterns in the catalog: Each pattern's intent is to be considered to find one or more that sound relevant to the current problem. A classification scheme can be used to narrow the search

IV. HOW TO USE A DESIGN PATTERN

Once a design pattern has been selected, the question is now on the approach that is to be used to apply the design pattern effectively. The following are the steps that can be employed in the use of a design pattern:

The design pattern has to be read through at least once for an overview. While reading, pay particular attention should be to the applicability and consequences sections to ensure that the pattern is right for that specific problem. The structure, participants and collaborations sections must be revisited and an understanding of how the classes and objects in the pattern relate to one another needs to be achieved. Studying the code helps in learning how to implement the pattern, and this is done by looking at the sample code section to find concrete examples of the design pattern in the code.

Choice of names for pattern participants is very important because they ought to be meaningful in the application context. They are usually too abstract to appear directly in an application. Nevertheless, it's useful to incorporate the participant name into the name that appears in the application, as this helps make the pattern more explicit during the implementation. For example, usage of the Strategy pattern for a text composition algorithm might mean having classes such as Simple-Layout-Strategy or Text-Layout-Strategy. The next stage involves defining the classes, declaring their interfaces, establishing their inheritance relationships, and defining the instance variables that represent data and object references. Identifying existing classes in the application that the pattern will affect and modifying them accordingly can also be included.

Defining application-specific names for operations in the pattern generally depends on the application. Responsibilities and collaborations associated with each operation should be used as a guide. Consistency in the naming conventions is important, for example, using the "Create-" prefix consistently denotes a factory method.

Implementing the operations is now done to carry out the responsibilities and collaborations in the pattern. The Implementation section offers hints to guidance in the implementation. The examples in the Sample Code section may be of help as well.

V. ADVANTAGES OF DESIGN PATTERNS

The following are some of the benefits of OO design patterns.

They can reduce development time as known solutions are used instead of reinventing the wheel thereby improving delivery speed.

Design patterns promote Discovering and learning with a view to making it easier for designers to use popular and successful designs developed from professional experience (Chang, 2011. According to Rising (2010), there may be a debate about the advantages of providing formal education to developers versus simply having access to a large search repository in which to search for a sample. To tackle some challenging problems. According to Blestin (2003), engineers are under the increased tension to provide answers with an excessive degree of exception in time. The Triangle project (2011) describes that task management can focus more effectively on the following three aspects of an improvement attempt: (a) Speed, (b) satisfactory and (c) value. " Each sample describes a problem that occurs over time and over again in our environment, then describes half the answer to that problem, that way I can use the answer a million times, without doing the same thing.

Design patterns provide flexibility and extensibility.

Tichy (1998) describes the idea of flexibility such as how design patterns can improve software architecture, speed of implementation, simplify maintenance, and help prevent architectural slippage. Future continuous adjustments (extensibility) with these types of hinges are surprisingly less expensive, but forcing the program to work in different ways is like bending the elbow backwards; the device is commonly broken

[12]. It's also helpful to have thoughtful answers that can be tried and tested.

Patterns make the communication of development teams easier.

Because design patterns capture distilled joy, they can provide a conversational tool at some point in the software improvement life cycle and in various communities of designers and programmers (Cline, 1996). This improved conversation between software developers is a feature that may allow less experienced developers to offer great designs. According to Fowler (2003), a group expert can use written design methods to help educate other team participants as they draw through software, design, and review requirements. Since design patterns make fun of distilled cheer, they can make for a chatting device. Throughout the software development life cycle and in numerous groups of designers and programmers (Cline, 1996). This constant conversation between software developers is a feature that can allow less experienced builders to offer great designs. According to Fowler (2003), an expert on the team can use written design methods to help educate other stakeholders on the team as they draw, design, and evaluate program requirements.

Patterns are underspecified

Design patterns are indeterminate as they generally do not restrict implementations too much. This is useful because modules allow flexible solutions that can be customized to take into account software requirements and limitations imposed by the use of tool optimization. Since it is not specific enough, identifying your pattern for a specific reason is one of the best ways to identify it. [4].

Design patterns capture knowledge that is implicitly understood

Once developers are exposed and motivated well by design patterns, they may yearn to embrace naming and pattern ideas. This stems from the fact that patterns encode knowledge that has already been intuitively understood. Therefore, once the basic parameters, codes, and formatting of prototypes are mastered, it becomes easy to record and approximate the many parts of the structure and design use patterns.

Promote an approved approach to documenting software architectures. Styles not specified

Planning styles are not specific enough considering they generally don't restrict implementations much now. This is useful because patterns allow flexible responses that can be customized to take account of tool needs and limitations imposed with the help of tool development. Since it has not been identified, implementing a pattern to your employees for a specific reason is one of the best ways to find out.

Design patterns capture tacitly understood information

Once developers discover and are encouraged through the use of design patterns, they are eager to embrace naming ideas and patterns. This is due to the fact that the patterns encode knowledge that has already been intuitively understood. Therefore, once you master the basic concepts, notations, and pattern template codecs, it becomes easy to record and use virtually many parts of the tool architecture and design use of patterns. Well exposed and stimulated by design patterns, they can be enthusiastic to embrace naming and pattern ideas. This stems from the fact that patterns encode knowledge that has already been intuitively understood. Therefore, once the basic standards, codes, and formatting of prototypes are mastered, it becomes easy to record and create the many parts of the machine structure and design use patterns.

Promote an approved approach to documenting software architectures. Styles not specified

Design styles are nonsensical since they generally don't restrict implementations much now. This is useful because patterns allow flexible responses that can be customized to take account of tool needs and limitations imposed with the help of tool development. Since it has not been identified, implementing a pattern to your employees for a specific reason is one of the best ways to find out.

Design patterns capture tacitly understood information

Once developers discover and are encouraged through the use of design patterns, they are eager to embrace naming ideas and patterns. This is due to the fact that the patterns encode knowledge that has already been intuitively understood. Therefore, once you master the basic concepts, notations, and pattern template codecs, it becomes easy to record and use virtually many parts of the tool's architecture and design use of patterns.

They promote a structured means of documenting software architectures

This is completed by portraying the structure and collaboration of contributors to the architecture of a software program at a stage above the source code. This abstraction action is beneficial because it captures important architectural interactions while suppressing irrational information. Design patterns make successful designs and structures difficult to reuse. The expression of validated strategies as design patterns makes them more available to builders of new systems. Design patterns help design options that make the device reusable and avoid options that put reusability at risk. Design patterns can even improve the documentation and maintenance of existing structures by providing explicit specification for class-object interactions and their primary purpose. In fact, design patterns help a dressmaker to get the 'right' styling faster. Patterns improve the documentation and maintenance of existing structures by providing explicit specifications for the evolution and interactions of organisms and their primary component.

VI. CONCLUSION

From the analysis carried out based on the previous published work, a conclusion is reached that design patterns are not a panacea. In as much as there are advantages, they also tend to pose a lot of disadvantages if not applied correctly. In this paper a brief overview of OO design patterns is given with catalog of design patterns, how to select and use them and their advantages and disadvantages in Object oriented analysis and design. Much work is still to be done on design patterns to make them understandable to new users because currently usage depends on the developer's expertise.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[2] D. Alur, D. Malks, J. Crupi, G. Booch, and M. Fowler, Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies, 2nd ed. Prentice Hall, 2003.

[3] H. Zhu and I. Bayley, "An algebra of design patterns," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 22, no. 3, pp. 23:1–23:35, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2491509.2491517.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[5] A. K. Dwivedi and S. K. Rath, "Selecting and formalizing an architectural style: A comparative study," in Contemporary Computing (IC3),2014 Seventh International Conference on, Aug 2014, pp. 364–369.

[6] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," Software Engineering, IEEE Transactions on, vol. 28, no. 1, pp. 4–17, 2002.

[7] Ackerman. L and Gonzalez. C, 'The value of pattern implementations', The World of Software Development Journal, Computer Science Vol.32 Issue. 6, pp. 28-32, 2011.

[8]ApostolosAmpatzoglou, postolosKritikos, George Kakarontzas and IoannisStamelos, 'An Empirical Investigation on the Reusability of Design Patterns and Software Packages', Department of Informatics, Journal of Systems and Software, Vol. 84, 2011.

[9] J. Niere, W. Sch¨afer, J. P. Wadsack, L. Wendehals, and J. Welsh, "Towards pattern-based design recovery," in Proceedings of the 24th international conference on Software engineering. ACM, 2002, pp. 338–348.

[10] H. Zhu and I. Bayley, "An algebra of design patterns," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 22, no. 3, pp. 23:1–23:35, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2491509.2491517

[11] I. Issaoui, N. Bouassida, and H. Ben-Abdallah, "Using metric-based filtering to improve design pattern detection approaches," Innovations in Systems and Software Engineering, Springer, December 2014.

[12] C.-H. Chang, C.-W. Lu, and P.-A. Hsiung, "Pattern-based framework for modularized software development and evolution robustness," Information and Software Technology, vol. 53, no. 4, pp. 307–316, 2011.