International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 1, ISSN 2249 –071X

A New Approach for Improving Quality of Web Applications Using Design Patterns

J.Srikanth[1], R.Savithri[2]
[1]PG Scholar(ME-Software Engineering),
[2]Senior Lecturer, Department of Information Technology
Rajalakshmi Engineering College,
Chennai

***Abstract*- Design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context, they describes the problem and its corresponding solution. Professional software engineers always use Design patterns for introducing abstractions in software and by the way they can build complex web applications. The right adoption of Design Patterns while designing web applications can promote the factors like reusability and consistency of the web application. The existing system detects design patterns automatically or semi-automatically. But any tool based approach for automatic detection of design patterns doesn't suits well for GOF patterns, so the approach taken here is to identify the presence of Design patterns using UML Class diagram. Once we identify the pattern, we can evaluate the quality of the software by using any metric based evaluation technique. So, the approach taken here is to analyze the existing set of patterns in a given framework and injection of new set of patterns to enhance the quality of web applications.**

***Key words*- Design patterns, GOF patterns, UML diagrams, Software metrics**

I. INTRODUCTION

Design patterns are like templates for software development. The purpose of the template is to define a particular behavior or technique that can be used as a building block for the construction of software – to solve the universal problems faced by the developers. Experienced designers/programmers reuse structures and designs that have worked well in the past instead doing it from scratch. The Strength of the programming language is measured by the level of abstraction it provides, for complex web applications, introducing design patterns is considered as the best way to introduce abstraction in it. From developers' point of view, design patterns produce a more maintainable design. Whilst from the users' point of view, the solutions provided by design patterns will enhance the usability of web applications. Programmers use design patterns to organize objects in programs, making them easier to write and modify. Designers use design patterns to provide a solid structure to the product based on the scenario in which it is going to be used. Reusability of design pattern is suitable for any system development which has the similar functions or problems. Object oriented code can be reused among projects by either object composition or class inheritance [6][7], moreover along with design patterns it's even a more convenient way for code reuse. Choosing the most suitable design pattern in the context of Web application is not easy [9], because most developers lack of knowledge on existing design patterns.

Software design is considered as one of the complex part for developing software. Embedding reusability in the root of the software is considered as a best way to enhance the reusability of software. Detecting design patterns from object-oriented program source-code can help maintainers to understand the design of the program. Most of the conventional approach for detecting design patterns from programs analyzes the structural aspects of programs i.e. inter-class relationships. Since these approaches are based on structural aspects, it is impossible to distinguish different design patterns with the same structure [8].

The proposed approach improves quality of web applications by combining existing MVC pattern with new set of patterns. Those new patterns are Aspect Proxy pattern, Data Access Object (DAO) pattern, Inversion of Control pattern, Dependency Injection pattern and Template method pattern. Design patterns are abstractions from programs and systems. Understanding the use of patterns in real world programs requires deep understanding of the context describing a situation where a particular pattern is useful [13].

The rest of the paper is organized as follows: in section 2 proposes the related work and section 3 describes the method for improving quality using proposed architecture model. Section 4 deals with the different viewpoints of quality and finally conclusion and future work is discussed in section 5.

II. RELATED WORK

The great strength of reusability has been introduced within software components, because the Software industry demands the development of products at a very fast rate with most effective solutions. This introduces the metric based study of the design patterns by proposing suitable metrics that are suitable for all the patterns [1]. To enhance the reusability of the software, we need to embed the reusability in the root of software. Programmers use design patterns to organize objects in program, and Designers use the same to

provide solid structure to the product. The analyzed drawback here, there are many methods are available for finding the quality of patterns. Some of the proposed methods are Method Reuse Factor in Pattern, Attribute Reuse Factor of Pattern, Total Operations of Pattern., etc. Too many factors for finding the quality of patterns will ends with the confusing results.

The impact of design patterns on quality attributes in the context of software maintenance and evolution [2]. Three design patterns, Abstract Factory, Composite and Flyweight have been evaluated with three quality factors, reusability, understandability and expandability respectively. The reason behind the selection of patterns is to illustrate our respondents' assessment because of their popularity- they are among the most commonly used patterns and thus they felt that their evaluation would be more accurate. Two types of Result analysis done by them are Quantitative analysis and Qualitative analysis, so it's hard to come for conclusion based on null hypothesis results. The analysis of the results of their study reveals that in contrary to common lore, design patterns do not always impact quality attributes positively.

Reverse Engineering activities efficiently supports both software maintenance and evolution [3], but two important tasks pursued by reverse engineering are design pattern detection and software architecture reconstruction. The proposed Eclipse plug-in called MARPLE (Metrics and Architecture Reconstruction Plug-in for Eclipse), which supports both the detection of design patterns and software architecture reconstruction activities through the use of basic elements and metrics that are mechanically extracted from the source code. MARPLE plug-in is used in the context of system modernization and in particular for what concerns systems migration to SOA. But here the context is about Web Applications not the Service Oriented Architecture.

The best practice of software design by means of a tool named DPLab[4] to assist in recognizing and comprehending design patterns. Teaching design patterns for undergraduate students is a challenging task, because design patterns are abstractions from programs and systems. People tend to understand them by using a pattern instance they know as a blueprint, and relating it to other instances they know. On the other hand, understanding the use of patterns in real world programs requires deep understanding of the context describing a situation where a particular pattern is useful. Since design patterns are used by professional software engineers, it's considered tough for beginners to work the tool based approach, because of the used tool here is Eclipse.

The Decomposition of design patterns into simpler elements; it may reduce significantly the creation of variants in forward engineering, while it increases the possibility of identifying applied patterns in reverse engineering [5]. There are few reverse engineering tools that exploit the decomposition of patterns (i.e., FUJABA, SPQR). FUJABA (From UML to Java And Back Again) and SPQR (System for Pattern Query and Recognition). The reason of decomposing design patterns into sub components in the context of the two tools is different, however both obtain significant results. FUJABA is a forward and reverse engineering tool exploiting sub-patterns to reduce the dimension of the design pattern catalogue and the complexity of the elements searched in the source code, as well as to improve the detection algorithm. SPQR is an automatic tool for design pattern detection for C++.

III. METHOD

Identifying the presence of Design pattern is considered as preliminary step of our approach, by knowingly or unknowingly almost all software projects may contain Design patterns. Consider the purpose of the famous Singleton pattern, Ensure a class has only one instance, and provide a global point of access to it [6]. This simple pattern may unknowingly been used by most of the web developers by having no knowledge about design patterns. As mentioned earlier design patterns are used by experienced software designers/programmers, because they only provide framework for junior developers. A framework is a set of cooperating classes that make up a reusable design for a specific class of software. For example, a framework can be geared toward building graphical editors for different domains like artistic drawing, music composition and mechanical CAD [6]. A framework provides boundary to developers by the architect to satisfy user requirements neither exceeding nor below the boundary.

The original Model-View-Controller pattern is restructured here with some additional patterns like Template, Inversion of control, Dependency injection and Aspect proxy. Model-View-Controller pattern has three components; Model, View and Controller. Model summarizes the core application data and functionality. View displays information to client on the screen. Controller handles the interaction between user interface and user inputs, and initiates the creation of the application's new view [10]. Web application that adopts MVC pattern will be maintainable and scalable in web applications [11]. The restructured view of an existing MVC pattern meant for Web applications is shown below.

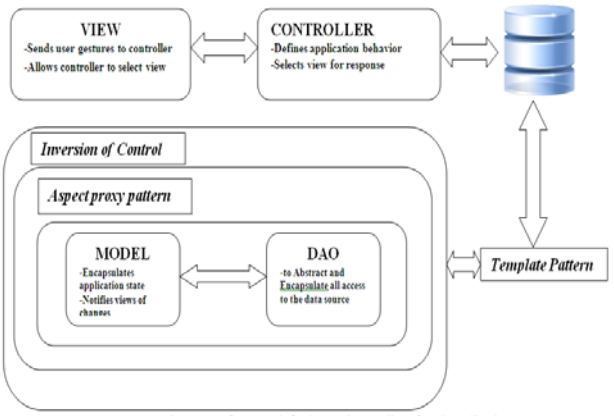

Figure 1: A framework for improving quality of Web applications

In this new view, Model is combined with DAO (Data Access Object) as shown in figure 1, it's a core j2ee pattern helps developers to access persistent data. Almost all enterprise applications need to use persistent data at some point. For many applications, persistent storage is implemented with different mechanisms. Some other applications may need to access data that resides on separate systems, for example, the data may reside in mainframe systems, Lightweight Directory Access Protocol (LDAP) repositories, and so forth [12]. The solution for the mentioned problem is to use a Data Access Object (DAO) to abstract and encapsulate all access to the data source. The DAO manages the connection with the data store to obtain and store data. Above to this combination we have Aspect Proxy pattern.

Aspect proxy pattern is the collaboration of Aspect oriented programming with proxy. Aspect-oriented programming (AOP) is a programming paradigm which aims to increase modularity by allowing the separation cross-cutting concerns; A concern is a particular goal, concept, or area of interest. In technology terms, a typical software system comprises several core and system-level concerns. For example, a credit card processing system's core concern would process payments, while its system-level concerns would handle logging, transaction integrity, authentication, security, performance, and so on. Many such concerns known as crosscutting concerns tend to affect multiple implementation modules. Using current programming methodologies, crosscutting concerns span over multiple modules, resulting in systems that are harder to design, understand, implement, and evolve. Proxy pattern provide a surrogate or placeholder for another object to control access to it [6]. One reason for controlling access to an object is to defer the full cost of its creation and initialization until we actually need to use it. For example, consider a document editor that can embed graphical objects in a document, some graphical objects like large raster images can be expensive to create, but opening a document should be fast, so we need to avoid creating larger objects. Proxy offers a solution for this problem is to use another object, an image proxy, which acts as a stand in for the real image.

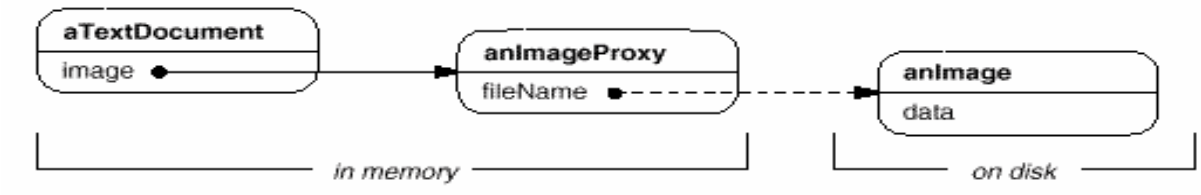

Figure 2 taken from [6]

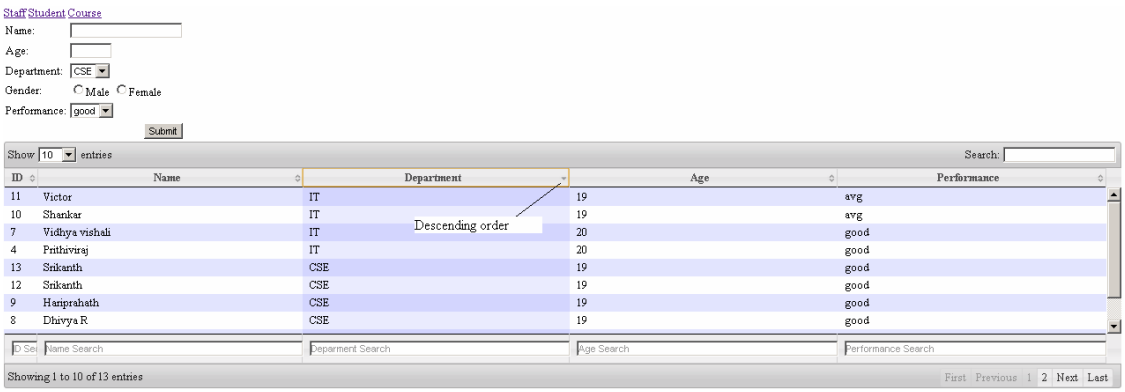

Figure 3: Sort operation

The image proxy creates the real image only when the document editor asks it to display itself by invoking its Draw operation. The proxy forwards subsequent requests directly to the image. Filename can be used as a reference to real object in case it's stored in separate files. By including some dynamism with proxy and combining it with AspectJ, we shall name it as Aspect proxy pattern. AspectJ is a java extension to Aspect oriented programming. Above to that we have Inversion of control pattern, it is used to remove dependencies in a code, because unnoticed dependencies may lead to the tight coupling problem. For web applications it's so obvious to have dependencies between the different modules, so by combining Inversion of control pattern with Dependency injection pattern it is possible for the developers to remove dependencies. Template pattern define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. Here Template pattern is used

as a mediator between database and the mentioned wrapped version of patterns. In other end we have view and controller, which serves the same purpose mentioned above.

IV. DISCUSSION FROM THE VIEWPOINT OF QUALITY

Once we apply the mentioned set of patterns in an existing web application, surely we can increase the quality at least by reducing Lines of Code (LOC) in application. The other possible metric is to calculate McCabe Cyclomatic Complexity, MCC measures the number of independent paths in the program control flow.

An implementation of a proposed system is shown in Figure 3, we have taken a web application from an educational institution, the data table displays the details of staffs, students, and courses respectively, apart from the details it have the features like insert, update, delete, findAll, findbyId, sorting, individual search, global search, etc. We have taken a sample of 13 records and applied descending sort operation on the field Department. A query is generated based on the above operation as shown in figure 4.

```
/* select
        count(id)
    from
        Student  */ select
            top ? count(student0_.id) as col_0_0_
        from
            student student0_
from
    Student
order by
    dept desc */ select
        top ? student0_.id as id0_,
        student0_.age as age0_,
        student0_.dept as dept0_,
        student0_.name as name0_,
        student0_.performance as performa5_0_
    from
        student student0_
    order by
        student0_.dept desc
```

Figure 4: select query

Figure 5 shows the search operation, the local search gives the flexibility of searching records on the basis of id/name/department/age/performance or any combination. The selected search uses department, age and performance and its corresponding generated query is shown in figure 6, this complex automated query generation is implemented with Template pattern. The successful deployment of this pattern can reduce the code complexity upto 70 percentage.

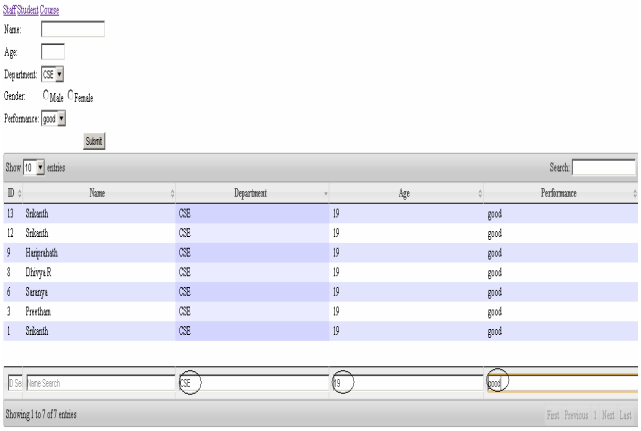

Figure 5: Search operation

```
from
    Student
where
    dept like '%CSE%'
    and age like '%19%'
    and performance like '%good%'
order by
    dept desc */ select
        top ? student0_.id as id0_,
        student0_.age as age0_,
        student0_.dept as dept0_,
        student0_.name as name0_,
        student0_.performance as
performa5_0_
    from
        student student0_
    where
        (
            student0_.dept like
'%CSE%'
        )
        and (
            student0_.age like
'%19%'
        )
        and (
            student0_.performance
like '%good%'
        )
    order by
        student0_.dept desc
```

Figure 6: select query

V. CONCLUSION AND FUTURE WORK

The approach taken here is to enhance the quality of web applications by using design patterns by using some additional combinational patterns to existing MVC patterns. Software Designers/Architects can improve the working ability of developers by enhancing the boundary of the framework. The proposed approach encourages the concept of reusability. The selected pattern increases upto 70 percentage of the performance when it compare with other patterns. In future we planned to include more number of patterns with this framework for accomplishing operations like FindByEmailId, FindByFirstName, FindByLastName, etc.

REFERENCES

[1] Parvinder Singh Sandhu, Parvinder Pal Singh, Anil Kumar Verma, "Evaluating Quality of Software Systems by Design Pattern Detection" , International Conference on Advanced Computer Theory and Engineering, 2008. ICACTE '08. 20-22 Dec 2008, pp 3 – 7.

[2] Foutse Khomh and Yann- Gael Gueheneuc, "Do Design Patterns Impact Software Quality Positively?", CSMR '08 Proceedings of the 2008 12th European Conference on Software Maintenance and Re-engineering.

[3] Francesca Arcelli Fontana, Marco Zanoni, "A tool for design pattern detection and software architecture reconstruction"

[4] Jens Dietrich and Elizabeth Kemp, "Tool Support for Teaching Design Patterns", 19th Australian Conference on Software Engineering.

[5] Francesca Arcelli, Stefano Masiero, Claudia Raibulet, "Elemental Design Patterns Recognition in Java", Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05).

[6] E. Gamma, R. Helm, R. Johnson and J.M. Vlissides, Design Patterns; Elements of reusable object-oriented software, Addision- Wesley, MA, 1995.

[7] Hironori Washizaki, Kazuhiro Fukaya, Atsuto Kubo, Yoshiaki Fukazawa, "Detecting Design Patterns using Source code of Before Applying Design Patterns ", 2009 Eighth IEEE/ACIS International Conference on Computer and Information Science.

[8] J. Borchers, "A Pattern Approach to Interaction Design", New York: John Wiley & Sons, 2001.

[9] M. Veit and S. Herrmann, "Model-view-controller and object teams: a perfect match of paradigms," Proceedings of the 2nd International Conference on Aspect oriented software development, pp.140-149, ACM, March 2003.

[10] GuangChun Luo, Yanhua Wang, Xianliang Lu, and Hanhong, "A novel web application frame developed by MVC", ACM SIGSOFT Software Engineering Notes, vol. 28(2) (March 2003), New York, NY, USA, 2003.

[11] http://java.sun.com/blueprints/corej2eepatterns/Patterns/ DataAccessObject.html.

[12] Jens Dietrich, Elizabeth Kemp, "Tool Support for Teaching Design Patterns", 19th Australian Conference on Software Engineering.

AUTHOR'S PROFILE

J. Srikanth

received the B.Tech degree in Information Technology from Anna University, Chennai, India. He is pursuing M.E-P.T(Software Engineering) at Rajalakshmi Engineering College affiliated to Anna University, Chennai, India. He has 5 years experience in teaching. He has presented a paper in National Conference. His area of interests includes Design Patterns, Object Technology and Data structures.

R. Savithri

received the B.E degree in Computer Science and Engineering from Bharathidasan University and M.E degree from Anna University, Chennai, India. Presently she is working as a Senior Lecturer in the Department of Information Technology, Rajalakshmi Engineering College, Chennai, India. She has 8 years of experience in teaching. She has presented a paper in National Conference. Her area of interests includes Software Engineering and Data Mining.