Analyzing effect of Aspect Oriented concepts in design and implementation of design patterns with case study of Observer Pattern

Deepali A. Bhanage[1], Sachin D. Babar[2]

Sinhgad Institute of Technology, Lonavala

***Abstract*— Design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. Several patterns crosscut the basic structure of classes adding behaviour and modifying roles in the classes relationship. Recent studies have shown that several design patterns involve crosscutting concerns where object oriented abstractions failed to handle, this led to decreasing system modularity, reusability supportability, portability, reliability and maintainability. This encourages examining impact of Aspect Oriented Programming on Software development at Design Level. In this paper, effect of Aspect Oriented Programming on Gang of Four (GOF) design pattern is analyzed. This investigation is further continued with implementing case study of Observer Pattern with example of IT Infrastructure Monitoring. This paper also explores use of JAVA Annotations in implementation of Observer Pattern with AOP.**

***Keywords*—Aspect Oriented Programming, AOP, AspectJ, Observer Pattern, Java annotations, GoF design patterns, Crosscutting.**

I. INTRODUCTION

Design pattern is a general repeatable solution to a commonly occurring problem in software design. It is a description or template for how to solve a problem that can be used in many different situations [1]. Design patterns can speed up the development process by providing tested, proven development paradigms. Design patterns helps to produce good design, which helps in producing better softwares. Design Patterns have gained its popularity after the "Gang of Four" book [1] where the first software pattern catalogue containing the 23 Gang-of-Four (GoF) patterns were introduced.

Recent studies shown that several patterns crosscut the basic structure of classes adding behaviour and modifying roles in the classes relationship. Crosscutting represents the situation when a concern is met by placing code into objects through the system but the code doesn't directly relate to the functionality defined for those objects [2]. Examples of crosscutting concerns are logging, synchronization, error and exception handling, scheduling and optimization.

Observer pattern is one of the popular pattern described by GoF. The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. The Observer Pattern Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The observer pattern is used when the change of a state in one object must be reflected in another object without keeping the objects tight coupled. It is also used when framework needs to be enhanced in future with new observers with minimal changes [1].

Abstracting the implementation offers great flexibility. Observers can observe the state change of any subject if that subject implements Subject interface and the observers themselves extend Observer abstract class. But at the same time the responsibilities of classes are also changed. In order to being a subject, an object has to implement the features of maintaining the list of observers, notifying each observer when its state changes. When an observer wants to observe a subject, it has to extend an abstract class to indicate that it has the method receiving the notifications. These features all belong to Observer pattern, are tangled with the core features of the object, obscure the primary concern of objects. This violates the single responsibility feature of OOP [3].

Although the use of Observer pattern brings several benefits, they could "hard-code" the underlying system, making difficult to express changes. To implement Observer patterns described above in a system you may have to modify several classes, affecting their relationships and their clients.

The Object Oriented Programming is most commonly used and very popular methodology employed in software Industry, but it does not do as a good job in address many crosscutting concerns that must be included in multiple modules[2]. In OOP crosscut concerns, code tangling and code scattering will be seen Code tangling is caused when a module handles multiple

concerns simultaneously, the codes from other modules present in a module. Code scattering is caused when a single issue is implemented in multiple modules. Because crosscutting concerns are spread over many modules in nature, related implementations are also scattered over all modules. Code tangling and code scattering together result in lower code reuse and harder evolution [4]. If a module is implementing multiple concerns, it is hard to modify. When a module is scattered in many modules, if the module requires reimplementation, many modules have to be modified. This fails of Open Close Principle [4]. Unfortunately, object-oriented abstractions are often not able to modularize those crosscutting concerns, which in turn decrease the system reusability and maintainability.

Aspect Oriented Programming can help on separating some of the system's design patterns, specifying and implementing them as single units of abstraction [5]. Compared with OOP implementation, Aspect-oriented Programming (AOP) offers one viable solution, duplicated code blocks are modeled in different aspects that crosscut other modules. OOP and AOP complement each other [6]. While aspect-orientation originally has emerged at the programming level, modern software development is more than just coding, there was a need to study applying the aspect orientation also over other development phases, starting from the requirements analysis phase till the implementation phase, which led to the emerge of aspect oriented software engineering. The aspect-oriented programming main goal is to help the developer in the task of clearly separate crosscutting concerns, using mechanisms to abstract and compose them to produce the desired system. The aspect-oriented programming extends other programming techniques (object oriented, structured, functional etc) that do not offer suitable abstractions to deal with crosscutting [5].

When observer pattern implemented by AOP the subject or the observer doesn't extend an abstract class or implement an interface, an aspect introduces interfaces or classes to existing classes. At the same time when the subject notifies observers and which method of the observer receives notification are defined in the same aspect. Maintaining the list of observers is also finished by this aspect. When the subject state changes, this aspect will notify the observer. In this implementation, if an object wants to observe other objects, it only needs a response method, other modifications are not required. It is not necessary to modify the objects which will be observed. The implementation is clean which easily evolves and helps to address the crosscutting concerns of the OOP [3].

While implementing the Observer pattern using AOP, every object has a single responsibility. The crosscut concern defining the relations between objects is modeled in an aspect, not in classes. Though there are many benefits of AOP as compared to OOP, users need to know Aspect Oriented Programming Language for implementation. Paper discuss about use of JAVA Annotations, to define the subject, the observer and their relations. Users employ these annotations instead of an aspect language to define the pattern concern [4]. This program analyses and generates the aspect, weaves this aspect with target classes by load-time weaving or compile-time weaving.

II. STUDY FORMAT

This paper is divided in following sections. Section III will explain the literature survey which highlights the problems occurring in traditional implementation and related solution that is use of AOP at the time of implementation of GoF Design Patterns. Section IV will give information about implementation of observer pattern with case study of IT Infrastructure event Management system with Aspect oriented concepts. With this example, paper will discuss crosscutting and how to overcome it with the help of AOP. Section V will explore use of Java annotations in implementation. Section VI unfolds the effect of AOP on Observer Pattern measured with respect to different software properties. Finally the conclusion of the paper is presented.

III. RELATED WORK

The majority of applications available in market contain common functionality such as authentication, authorization, caching, communication, exception management, logging and instrumentation, and validation which are described as crosscutting concerns because it affects the entire application, and should be centralized in one location in the code where possible. To compensate for missing tools and languages we need architectural solutions for the problems around crosscutting concerns [9]. Design pattern provide reusable solution in all types of software designing and implementation. It becomes important to verify the effect of Aspect oriented programming to remove the crosscutting concern so that it will improve modularity, maintainability, reusability, uniformity, transparency, extensibility.

Paper analyses comparison between use of Object Oriented and Aspect Oriented concepts at design level. After comparison it has been concluded that in few GoF Design Patterns crosscutting concern is present and it can be addressed by applying the aspect oriented approach at design level [3][8].

TABLE I

Effect of AOP on Design Patterns [3]

Gang of Four Design Patterns	Crosscutting Concern	Improved Modularity	Improved Reusability
Chain of Responsibility	Yes	Yes	Yes
Command	Yes	Yes	Yes
Interpreter	No	No	No
Iterator	No	No	Yes
Mediator	Yes	Yes	Yes
Memento	No	Yes	Yes
Observer	Yes	Yes	Yes
State	No	Yes	Yes
Strategy	No	Yes	Yes
Template Method	No	No	Yes
Visitor	No	Yes	Yes
Adapter	No	Yes	Yes
Bridge	No	Yes	Yes
Composite	Yes	Yes	Yes
Decorator	No	Yes	Yes
Façade	No	No	No
Flyweight	No	Yes	Yes
Proxy	No	Yes	Yes
Abstract Factory	No	Yes	Yes
Builder	No	No	Yes
Factory Method	No	Yes	Yes
Prototype	No	Yes	Yes
Singleton	No	No	No

There were qualitative studies such as the work done by [2][4] which represents effort done in the area of addressing the impact of applying the aspect oriented programming approaches on GOF design patterns based on important software engineering attributes and other quantitative studies [3][8] performed to compare object oriented and aspect-oriented implementations based a metrics of measurements measuring the cohesion, coupling and size of both implementations. There are multiple surveys conducted to see the use of AOP with design patterns at design level. Most of the studies are not extensive and limited to theoretical work.

As discussed in [2] applying aspect oriented concepts on GoF patterns could lead to improved modularity, maintainability, reusability, uniformity, transparency, extensibility even if the pattern doesn't involve crosscutting concerns. Table 1 shows the effect of Aspect Orientation in terms of existence of crosscutting concerns, improved modularity and reusability

In this paper, case study of Observer pattern is done while applying AOP concepts at design as well as implementation level. The distinctive crosscutting elements in the Observer pattern are the roles (Subject and Observer) superimposed to the classes participating in the implementation of the pattern, and the consistent behavior of notifying the observers required from the methods changing the state of the Subject object [6].

This paper explains the Observer pattern with the help of AOP and example of IT Infrastructure Event Management. The Information Technology Infrastructure Library (ITIL) is a set of practices for IT service management (ITSM) that focuses on aligning IT services with the needs of business. Event Management and monitoring is process added in ITIL which is responsible to manage all events that occur in the infrastructure to ensure normal operation and also to assist in detecting and escalating exception conditions. Event Management assists with the early identification of incidents through providing a baseline of automated operations in order to allow service management staff to focus on become more proactive in their daily operations. Events are normally notifications/warnings created by an IT Service, Configuration Item (CI) or any type of monitoring tool around the status or availability.

IV. IMPLEMENTATION OF OBSERVER PATTERN WITH AOP

Observer pattern maintains consistency among several objects that depends on a model data in a way that promotes reuse and keep a low coupling among classes. In this pattern, every time the Subject state changes, all the Observers are notified. The main problem with the object-oriented Observer pattern is that you should modify the structure of classes that participate in the pattern. So, it is hard to apply the pattern into an existing design as well as remove from it.

The design of the Observer pattern is changed in order to represent it as classes and aspects. There is no Observer role neither a Subject one. Both structure and behavior of these two roles are expressed in the MonitoringObserverPattern aspect. ITSMSubject describes the interface that all the concrete subjects must be in accordance to (enforce by the MonitoringObserverPattern and MonitoringConcreteObserverPattern). When implemented, the ITSMSubject will contain a reference to its observers, and allow the dynamic addition and deletion of observers. Observer describes the interface

that all the concrete observers must be in accordance to (enforce by the MonitoringObserverPattern and MonitoringConcreteObserverPattern). They are notified every time the state of the subject changes. ITSMConcreteSubject Store state information to be used by ConcreteObservers. It does not, however, send notifications to its Observers. This responsibility is part of the MonitoringObserverPattern role. ConcreteObservers serve as basis to field and method's introduction performed by the MonitoringObserverPattern. MonitoringObserverPattern is an abstract aspect that encapsulates the behavior of the Observer pattern. The MonitoringObserverPattern contains the fields and methods to be included in the classes that are affected by the MonitoringConcreteObserverPattern. Concrete ObserverPattern specifies in what situations the ConcreteObservers are going to be notified as well as what is going to be executed when the ConcreteObservers are notified.

In the implementation of ITSM event management, different devices, applications, hardware, database and services are subjects. ITSM Administrators and different application can serve as Observes. When some incident occurs for configurable item, popularly known as CI, sends notifications to the observer as event. Sometime observer may propose to receive events for analysis and information. The list of all the observers is maintained in Aspect MonitoringObserverPattern. Pointcut and advices in MonitoringObserverPattern helps to waive the application. When the classes corresponding to concrete subjects and concrete observers are weaved together with the MonitoringConcreteObserverPattern, the following methods and fields are attached to the ITSMConcreteSubject and ConcreteObservers: fields Observer observers (ITSMConcreteSubject), Subject subject (Concrete Observers) methods void add(Observer obs), void remove(Observer obs) (Concrete Subject), void setSubject(Subject s), Subject getSubject() (Concrete Observers) These attachments are specified in the abstract aspect (the MonitoringObserverPattern). The ITSMConcreteSubject implements the ITSMSubject interface. The concrete observers implement the observer interface. The other modifications are done to the dynamic nature of the observer and subject classes, telling that every time that the state of the subject changes the update method of the observers is invocated.

V. USE OF JAVA ANNOTATION IN IMPLEMENTATION

However, using this method you have to know how to define pointcuts, advices, inter-type declarations in aspects, how to weaving aspects with classes. This paper provides a method using Java annotations to define pattern participants and their relations, and then those annotations are used to generate aspects. In this

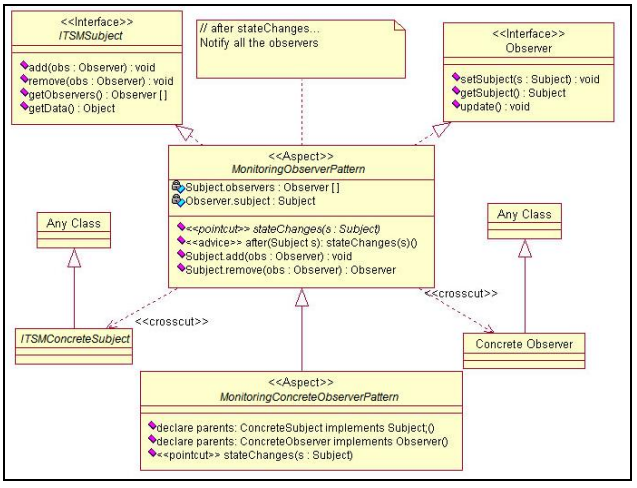

Figure 1: Implementing Observer Pattern with AspectJ[2]

implementation, users only use Java annotations rather than Aspect language; they don't have to know an Aspect language.

A. *Defining Annotation @subject and @observer*

@Subject is applied to specify the subject, @Observer is used to define the observer. Both annotations are used to annotate a class or an interface. Sometime there are many groups of different observing associations between objects, so the id property is defined to identity which group objects are in; its default value is o. The same class may be annotated with different id. Other annotations defined later also have this property.

B. *Annoting State Change and Update Method*

@SubjectChange is applied to a method, indicting after executing this method, the subject's state changes, and observers will be notified.

C. *Starting and Stopping Observing*

After annotating pattern participants, the subject and the observer, we need to establish the association between participants to start observing. It is apparent that only when we can get both the subject and the observer, we may establish their association.

VI. RESULT OF STRUCTURED ANALYSIS

In this paper, structured analysis is done for applying Aspect Oriented concepts on Observer Pattern to solve the problem emerging from the existence of Crosscutting concerns. It is realized that after applying AOP to Observer pattern leads to improvement in some key software properties. Table 2 shows the effect of AOP on different software properties when applied to Observer Pattern.

After Applying AOP to Observer Pattern, paper analyzes the impact of using JAVA Annotation in Observer Pattern.
Results of the JAVA Annotation analysis is as follows

- Annotations provide a novel method to define participants and their relations by annotations.
- JAVA Annotation can be used to generate Aspects.
- No need to understand AOP to overcome shortcomings of Observer Pattern.
- Use of annotation provides clean and simple solution to complex implementations.

TABLE II
Structured Analysis of Observed Pattern with AOP

Sr No	Software Properties	Improved with AOP
1	Modularity	Yes
2	Reusability	Yes
3	Maintainability	Yes
4	Uniformity	Yes
5	Transparency	No
6	Extensibility	Yes
7	Portability	No
8	Supportability	Yes
9	Reliability	No
10	Efficiency	No
11	Usability	Yes
12	Adaptability	Yes
13	Redundancy	Yes

CONCLUSION

When applying Aspect Oriented Programming to design pattern lead to efficacious push towards building effective and intelligent solutions. Implementing ITSM Event Management with Observer Pattern under the umbrella of Aspect Orientation reduces the crosscutting concern and improves the maintainability, reusability, uniformity, transparency, extensibility. Use of Java annotations in Observer Pattern helps to deliver clean and powerful solution.

REFERENCES

[1] Gamma, E. et al Design Patterns- *Elements of Reusable Object Oriented Software.* Addison-Wesley, 2009.

[2] Eduardo Kessler Piveta; Luiz Carlos Zancanella;, "Observer Pattern using Aspect- Oriented Programming" *SugarloafPLoP Conferenece, 2003.*

[3] El Maghawry, N.; Dawood, A. R.; ,"Aspect oriented GoF design patterns," *Informatics and systems (INFOS), 2010 The 7th International Conference on, vol., no., pp.1-7, 28-30 March 2010..*

[4] Liu Jicheng; Yin Hui; Wang Yabo; , "A novel implementation of observer pattern by aspect based on Java annotation," *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on , vol.1, no., pp.284-288, 9-11july 2010.*

[5] Ramnivas Laddad . AspectJ in Action: Practical Aspect Oriented Programming. Addison-Wesley,2005.

[6] Lei Zhang; , "Study on comparison of AOP and OOP," *Computer Science and Service Syatem (CSSS), 2011 International Conference on , vol., no., pp.3596-3599, 27-29 Jane 2011.*

[7] Christiansson, B.; Forss, M.; Hagen, I.; Hansson, K..; Jonasson, J.; M. Jonasson, Lott, F.; Olsson S.; and Rosevall, T. "GoF Design Patterns- with example using Java and UML2 ",2009.

[8] Wang Xiaohuan; Zhang Shu; Tang Wanmei. "A New Programming Fashion in OOP and AOP," *Journal of Chongqing University of Arts and Sciences (Natural Science Edition), 2007, v.26 n. 14, pp.41-44.*

[9] Christa Schwanninger; Egon Wuchner; Michael Kircher "Encapsulating Crosscutting Concerns in System Software". *In Proceedings of the Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software 2004.*

[10] Marin, M.; Moonen, L. ; van Deursen, A.," A Classification of Crosscutting Concerns", *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on 26-29Sept2005.*