IMPROVING THE DESIGN AND IMPLEMENTATION OF SOFTWARE SYSTEMS USES ASPECT ORIENTED PROGRAMMING

Mazen Ismaeel Ghareb
Department of Computer Science
University of Human Development
Sulimanya, Iraq
mazin.ismaeel@uhd.edu.iq
mazen.ghareb@hud.ac.uk

Dr Gary Allen
Department of Informatics
School of Computing and Engineering
University of Huddersfield
Huddersfield, England
g.allen@hud.ac.uk

***Abstract-* Aspect Oriented Programming (AOP) is a technique used to enhance the separation of concern in software design and implementation. To implement AOP developers most commonly use AspectJ, an extension of the Java language. In order to represent the separation of concerns a new design technique is needed for modeling of aspects. Aspect Oriented Design language (ASDL) is a design notation could build on the existing design language UML. It is capable of supporting many aspects-oriented languages. This research aims to investigate where developers can use aspect-oriented programming in the software development process. This includes how to identify Aspects in the design stage, then how they can be applied in the implementation process in software development .This will lead to improve modularity and reusability of the software. The paper will be a position paper abut Aspect Oriented Software Design, also will investigate several case studies of the design and implementation of a software application that shows how we can identify the Aspect in the process.**

Keywords—Aspect Oriented Programming; ASDL (Aspect Oriented Design language)

I. INTRODUCTION:

In the software development process, Aspects are difficult to identify because they are usually tangled and scattered across the entire system. Some aspects are obvious can be identify but others are more subtle difficult to identify. This makes it complex to locate all points of the system where aspects should be applied. To address these aspects in the software development lifecycle, developers need more support to find and analyze aspects in requirements documentation.

AODL (Aspect Oriented design language) is a notation used to show the interaction between traditional UML models of base (Object Oriented) code and Aspect extensions, such as point cuts, join points and advice. The challenge of this study is to find the relationship between design patterns and aspect oriented programming to meet a modular solution for specific issue in the software engineering field. Many studies have been done on Aspect programming techniques during recent years. One of the aspect definitions according to [1] defines Aspect, as “aspects tend not to be the system's functional decomposition, but rather to be properties that affect the performance or the semantics of the components in a systematic way” Kiczales tried to differentiate between the aspects and components [1]. Another definition of Aspect Oriented Programming is to overcome the issues arising from crosscutting concern. It helps developers to change the Object Oriented model dynamically, so the crosscutting enhances code reuse rate and maintainability [2]. Aspect Oriented programming helps developers to overcome the issues associated with code scattering and tangling over multiple system units by reducing the duplication the code. Aspect Oriented Programming supports several crosscutting concerns such as join points, point cuts, and advice. A Join point is one of the several points of the system where concern crosscut a method or constructor, while a point cut is a query about selecting required join points. Consequently, the advice is the construction that takes an action where the join point matched: before, after and around in the specific system [3]. This research will investigate to use AODL aspect Oriented Design Language notation to represent software in the design phase. These notations are a new language proposed by Gary Allen and Saqib Iqbal [11] . This language represents the aspects and usual objects using UML notations and models. Adding to that if could we identifying the Aspects in early design it is possible by applying it on Design patterns. There were several works investigate this issue using the case study of the Car Crash management system [11]. According to [5] studies have shown implemshowions of six GOF design patterns (Observer, Mediator, Prototype, Strategy, State and Abstract Factory) with aspect implementations, the results that shows most aspect oriented programming improves the design of base object-oriented code.

The remainder of this paper will be structured as follows. In Section 2 we describe the state of the art about AOP technology. In Section 3, we introduce UML Language AOP Notation. In Section 4, we describe AOP and its relationship with design patterns. Section 5 we show the result of our survey. In Section 6 we described the methodology approach to identify aspects. In Section 7 we present the experimental results, followed by our conclusion and future work.

II. State of Art

A. History of AOP

Aspect Oriented programming (AOP) was discovered several years ago, before Demeter team. In 1997, Aspect oriented programming was officially revealed by Gregor Kiczales, with his colleagues in conference name ECOOP97 [6]. AOP Aspect Oriented programming developed methodologies called Subject Oriented Programming. According to [6] AOP is a development methodreducesthat improves software development by capturing the domain related processes in the system, to better fit real domain problems into code; therefore it reduce debugging time and increases readability.

.

B. Aspect Oriented Modelling AOM

Developers should be aware of, and understand the software modelling or architecture before staring implementing AOP. AOM (Aspect Oriented Modelling) is an approach to produce a logical aspect oriented architectural model. Early usage of AOM in development stage will reduce the software development risk of conflicts and undesirable behavior emerging during implementation. The cross cutting element is common in AOP and AOM, but the difference is between the artifices versus source code, it could rise difference technique in representing it. For instance, the code can represent in single functionality, while a model can represent the system with different diagram views. Another difference between AOP and AOM in code is aspect weaving is primarily concerned with inserting functionality at program execution. AOM module consists of major components: primary model, aspect model and composition model Fig.1[7] below shows this model.

.

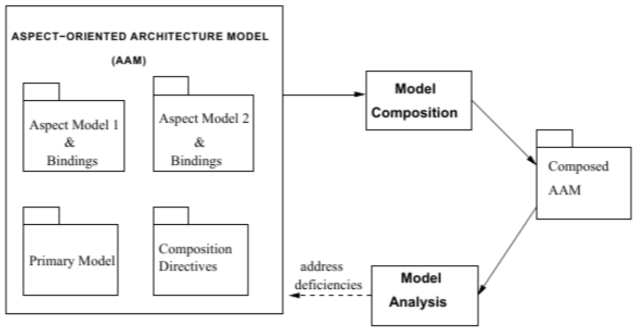


Fig 1 (component of AOM approach)

Fig1 shows a primary model such as a UML diagram to describe a basic architecture class diagram and interactive diagram. The aspect model describes a logical architectural solution. An Aspect Oriented Architecture Model is a logical view of software architecture. Another definition of Aspect Oriented Programming is simplifying of the development process by allowing separation of developmental tasks. In addition, Aspect Oriented Modelling improves an object-oriented application by making it more modular. AOP solves the code scattering problem in OOP. Scattering means the problem of shared the functionality of an application spread among many classes, which tends to slow down the application and make it difficult to maintain. Therefore, AOP solves this problem by bringing together the scattered code in the aspect. An aspect is a cross cutting structure. It implements the functionality such as security, logging and persistence.

C. Development of Aspect-Oriented

AspectJ used to implement AOP, which is a simple aspect oriented programming extension for the Java language. It is an open source programming extension of Eclipse. Moreover, it will support modular implementation of a range of crosscutting concerns [9]. An AspectJ program consists of two major parts, the first part is the base code such as classes and interfaces to carry out a basic functionality of the program, and the second part is the aspect code, which includes the aspects for capturing crosscutting concerns in the program [5]. Aspect supports the main AOP constructs of join points, point cuts and aspects.A join points is a dynamic execution point in the program. Point cuts consist of a collection of join points. An Advice is a somewhat special method attached to the point cuts. Finally, an aspect is a modular unit of AOP. Fig. 2 shows the process of aspect development method [10].

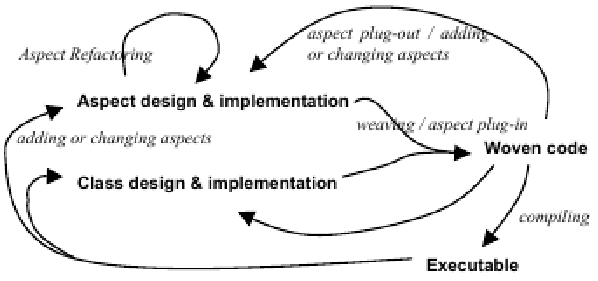

Aspect Refactoring
aspect plug-out / adding or changing aspects
Aspect design & implementation
adding or changing aspects
weaving / aspect plug-in
Woven code
Class design & implementation
compiling
Executable

Fig. 2 (process of aspect-oriented development)

Recent research shows that there is not a tool to support an aspect-programming notation such as UML (Unified Modelling Language) . One of the approaches to design an aspect notation is to extend the UML notation to support aspect oriented units called aspect oriented design model AODM. Therefore, this approach will help to show the aspect programming weaving mechanism and represented in UML, which will help developers to develop aspect programming notation language [11]. AODM might show an aspect as a modular unit of crosscutting implementation, which acts as a container of the given members in the piece of source code [12].

According to Stein “AspectJ is an implementation for aspect oriented programming in Java language”, adding that cross cutting is a part of the aspect that specifies where the crosscutting code has been woven into base classes. Join Points in AspectJ are standard points in executable dynamic programs. Join points present many actions such as calls to constructors and method execution. In addition, they call classes and object initialization. In AspectJ Point cuts consist of joint points. It specifies at which of the join points particular crosscutting behavior should execute. In terms of a designator point cuts are 'if', 'this', 'target', 'urges, or 'flow'. Developers will select Join points depending on the dynamic context during execution of the base code[13] .

The designer should specify at what time on the execution the advice is to execute for instance before, after, or around specific source code. Another important unit in Aspect J is introducing an additional member type of classes such as methods, constructor and another field for the class.

In addition, it may change the super class type of super interface by inserting new initialization and generalization relationship to the class structure.

III. UML Language AOP Notation:

UML language is object-oriented programming notation language. UML provides the basic building blocks to model software systems such as abstraction, relationships and diagrams. Adding to UML will give extended UML notation such as tagged values used to attach arbitrary information to a model element. Besides that, the extension totally supports new building blocks that drive from existing ones. This new building called stereotyping, have the same structure (attributes, association and operations) as the base system block that thrive on it.

Therefore, a UML extension is able to represent an AspectJ basic abstraction such as a Join point, Point cut and pieces of Advice. Fig.3 illustrates a UML representation for Join Points only, and shows the communication links to create or destroy an instance. Therefore, UML cannot assign or represent the Join Point. AODM suggests solving this problem by representing the communication as a pseudo operation that can only write and read for a specific field. This makes sure no execution might happen without calling a constructor or the initialization. Fig.4 shows that UML could represent a message that pass between two instances. As it seems that the join points indicate the special kinds of stereotypes such as <<execute>>, <<initialize>>, <<set>> and <<get>>.

In AODM, point cuts are represented as special stereotype operations named as <<Pointcuts>>. As it is shown in Fig. 5. (Stein, D., Hanenberg, and S. And Unland R., 2002).

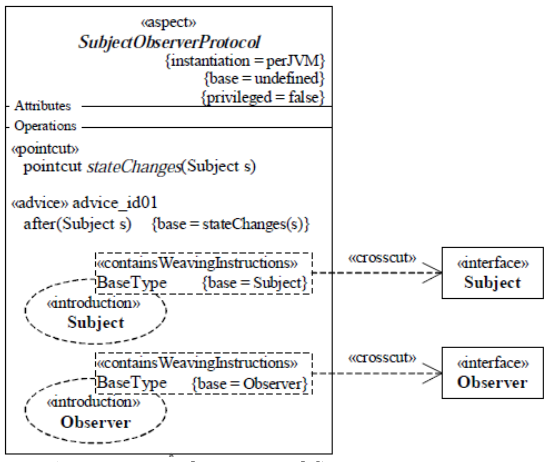


Fig. 3 (Aspect Oriented Design Model)

While in UML notation point cuts have an operational definition that has an arbitrary number of (output-only) parameters and their declaration and implementations as it shows in Fig. 4.

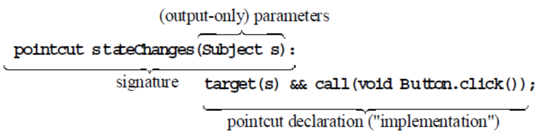

(output-only) parameters
pointcut stateChanges(Subject s):
signature
target(s) && call(void Button.click());
pointcut declaration ("implementation")

Fig. 4(Similarities between points cut in Aspect and Operation in UML)

Similar to Point cuts, Advice can be represented as an operation, but one semantic difference is that Advice does not have a unique identifier. Therefore, this might be a big conflict in Aspect. Therefore, AODM has solved this issue by defining by pseudo identifier that cannot be overdriven. As it appears in Fig. 5 .

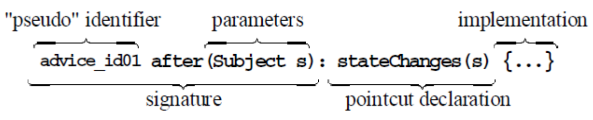
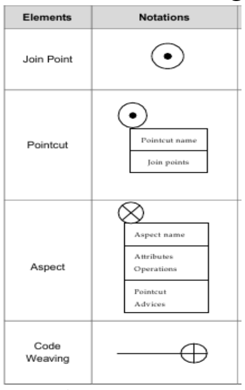


Fig. 5 (Similarities between Advice and Operation)

In [4] the authors propose a new notation of AODL Aspect Oriented Design Language, this language helps to model the Aspects with their attributes and characteristics along with a traditional UML object diagram. Both aspects and objects can be used within a design for single framework. This reduces designer operating cost to work with two different platforms. Therefore, developers chose UML to extend to contain aspects for many reasons. One important reason is that UML is most used tool for modelling. Secondly, it is easier for developers to use one tool rather two tools together. Finally, it is easy to use UML extensibility to design Aspects because it is easy to define a new notation and use them with the core notation. In AODL uses an aspect notation similar to class notation in UML to model other components such as Aspects and point cuts. However, there is an interaction between these components . For instance, point cuts contain the join point which advice directly depends on. As a result, each part has its own characteristics; therefore [4] claim that AODL should represent each notation as unique notation, as shown below in Fig.6:



Elements	Notations
Join Point	
Pointcut	Pointcut name Join points
Aspect	Aspect name Attributes Operations Pointcut Advices
Code Weaving	

Fig. 6 (AODL component notations)

In the AODL design notation, join points are represented as a hook. They connect the other parts of the program with the point cuts. Point cuts are explained as a rectangle box with a collection of related join points. The box symbol is used for Aspects because of them having similar characteristics to classes in their behavior, as it shows above. Aspect notation looks like class notation also it has same similarities of class and the cross circle shows the cross cutting concern of the aspect. Code weaving is associated which connects the aspect with classes where aspect code is woven in. Moreover, there are two models to design weaving process, aspect static diagram and aspect dynamic diagram. Aspect Programming has to show the join points in the programming. Sequence diagram in UML will show the join point in early design phase. This diagram called a join point identification diagram. The behavior of join points is modeled using an activity diagram, which shows the place of join point and the system activity. Fig.7 shows them below.

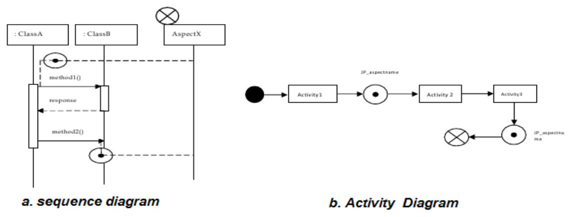


Fig. 7 (Joint point identification and behavior diagram)

The main AODL notation is the Aspect notation, which is represented as a big rectangle with many attributes and operations. It has the aspect name at the top and circle cross to show the cross concern behavior. Fig.8 shows a typical representation of an Aspect in AODL.

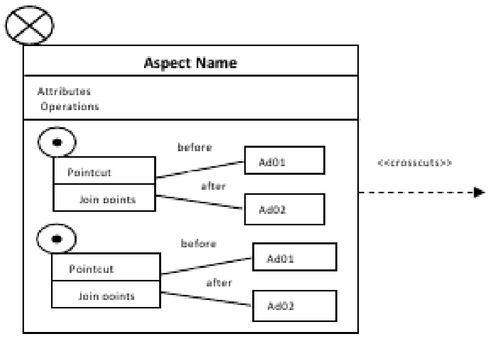


Fig. 8(Aspect Representation in AODL)

Aspects can be identified in the early stages of development using use case diagrams, as shown by Ivor Jacobson [10]. However, Ivor argued that aspects could not be implemented using use case diagram because tangling problems of the component in the use case diagram. While Iqbal suggests that to redundant the calling other component of use case and develop use case component separately. In this way, it is easy to find a crosscutting concern in use a case study [14]. He has shown an example of ATM system the withdraw cash use case needs to add logging aspect to the ATM use case as appears in Fig.9.

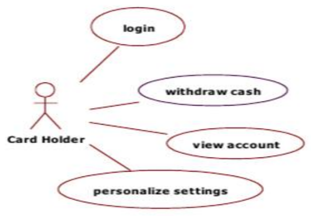


Fig. 9(ATM Use Case)

When Draw this use case diagram of with cash draw with sequence diagram it show the interaction with all parts of aspects joint point , point cuts and aspects. It could identify the aspects and also can show aspect characteristics such as calling joint point and point cut (before, after and around)[14] .The aspect identification and showing properties of it in Fig.10.

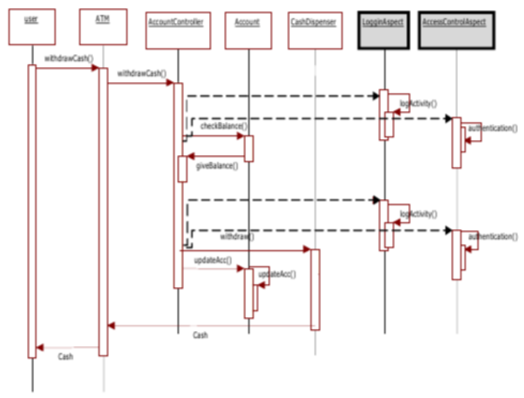


Fig. 10 (Sequence diagram showing aspect)

IV. AOP VS DESIGN PATTERN

[15] stated that Aspect oriented programming complements the Object oriented programming by giving powerful constructs to handle composition and modularity. This will help develop the best modularity of design patterns of these concerns. The patterns consist of two parts, part one identified the aspects, classes, relationships and operations related to the solution. Second part is concerned with the number of signifying behaviors and structural relations between components. For instance, in the adapter design pattern there are two classes which cannot use the same interface that share components, while an Aspect oriented, model will allow that by extending the interface of the Adoptee as is shown Fig. 11.

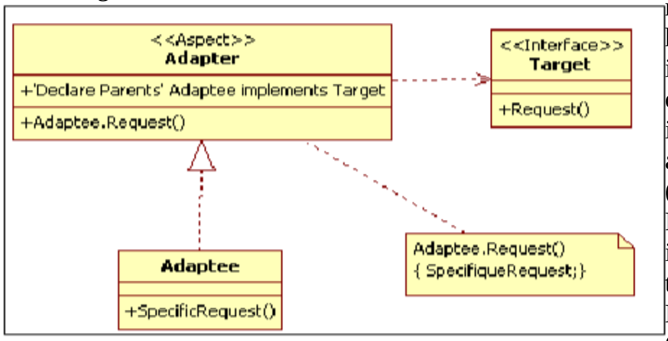
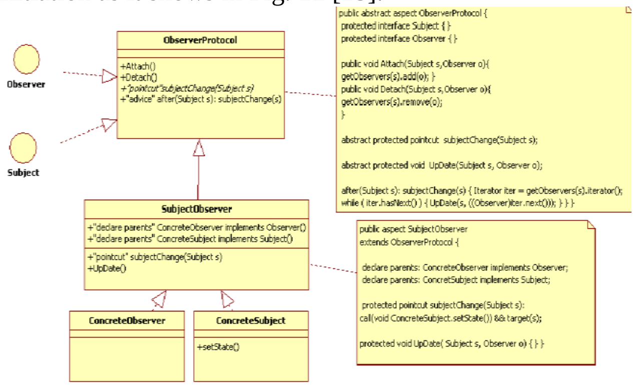


Fig. 11 (Adapter Pattern in aspect class diagram)

Another example is Observer pattern, which defines one to many dependent objects. If one of the objects is changing all the depended objects should be notified and updated automatically. While in the Aspect class diagram some part is common to all potential initiation of the pattern, and other specific to each initiation as it shows in Fig. 12 [15].


Observer
Subject
ObserverProtocol
+Attach()
+Detach()
+"pointcut"subjectChange(Subject s)
+"advice" after(Subject s): subjectChange(s)
SubjectObserver
+"declare parents" ConcreteObserver implements Observer()
+"declare parents" ConcreteSubject implements Subject()
+"pointcut" subjectChange(Subject s)
+UpDate()
ConcreteObserver
ConcreteSubject
+setState()

```
public abstract aspect ObserverProtocol {
 protected interface Subject { }
 protected interface Observer { }

 public void Attach(Subject s,Observer o){
 getObservers(s).add(o); }
 public void Detach(Subject s,Observer o){
 getObservers(s).remove(o);
 }

 abstract protected pointcut  subjectChange(Subject s);

 abstract protected void  UpDate(Subject s, Observer o);

 after(Subject s): subjectChange(s) { Iterator iter = getObservers(s).iterator();
 while ( iter.hasNext() ) { UpDate(s, ((Observer)iter.next())); } } }
```

```
public aspect SubjectObserver
extends ObserverProtocol {

 declare parents: ConcreteObserver implements Observer;
 declare parents: ConcretSubject implements Subject;

 protected pointcut subjectChange(Subject s):
call(void ConcreteSubject.setState()) && target(s);

protected void UpDate( Subject s, Observer o) { } }
```

Fig. 12 (Observer aspect class diagram)

Another important case study was (AGS) Antenna Group Server. This system is used to drive the antenna systems. It consists of 37 classes implemented in the C/C++ language. It had discovered some design patterns in implementing the system such as Observer, command, states, singleton and chain of responsibility. [16] Show that the drawback of implementing AGS using objects oriented programming with design pattern are these points:

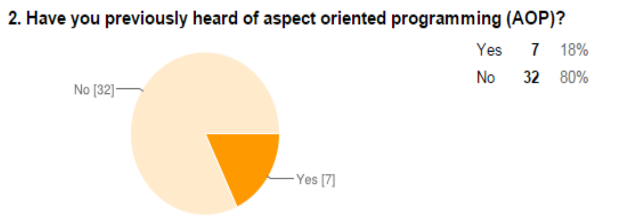
Inheritance relationships: The Inheritance relationships between classes are static when apply the design pattern because some classes to be concrete cannot be reused in the system. Pattern overlapping: This has happened when more than one pattern instance, has one concrete class, and this leads to cut in pattern traceability and class reusability. Encapsulation violation: composite and observer patterns force to the developer to expose internal objects to another object to handle requested computations.

The study tries to show the different results when applying AOP, two implementation ways used to carry out the case study: A Lazy implementation (L): This implantation used to move all inter-type declaration to Aspects such as methods and fields for all patterns then re-inject them into original places at load time. This will help to better modularize the scattered code .A Unplug implementation of AOP, the logical Design pattern code encapsulation into aspects. A reusable AOP implementation: improving the aspects where two large aspect use large dynamic properties of AOP to get greater (UN) plug ability and reusability.
Finally, to prove that AOP in Design patterns is improving the implementation [16] used metrics to check the AOP implementation. The metric parameters are
Depth of inheritance (DIT).
Coupling method calls (CMC).
Weighted Operation in Module (WOM).
Coupling between Modules (CBM).
Lack of Cohesion in Operations (LCO).
Response for Module (RFM)

The study confirms that implementation the design patterns using AOP improve the quality and modularity of the software. It also helps to avoid cross cutting concerns cause by implementing the Design pattern with object oriented code scattering. The AOP solution of coupling and cohesion are critical issues because of the interception mechanism at run time [16].

V. PRELIMINARY ANALYSIS

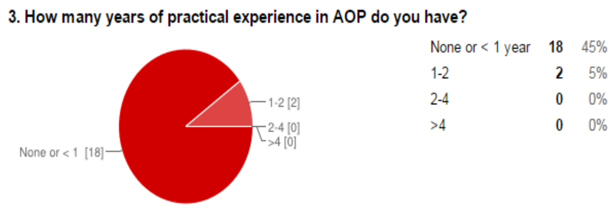
Aspect Oriented Programming is not currently widely known among developers and researchers. The previous sections illustrates, there are various attempts to use AOP for software developer.A survey has been carried out to investigate the awareness between developers and other researcher about AOP. 40 responses have been collected. The result shows that 80% of the participant did not hear about AOP, as it appears In Fig.13



Yes	7	18%
No	32	80%

Fig. 13 (Using AOP)

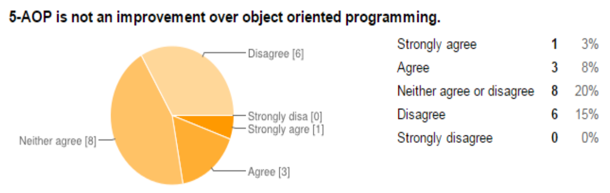
Therefore, it is clear that there is a gap for using AOP in many stages of the development process. Another important question was do you work with AOP and how many years do you work with it, 45% do not work with it or have less than 1 year of experience . As it is shown in Fig 14.



None or < 1 year	18	45%
1-2	2	5%
2-4	0	0%
>4	0	0%

Fig 14 (AOP Experience)

Moreover, many developers do not have enough knowledge about the differences between OOP and AOP. In Fig 15 shows that participants not sure, whether AOP is better than OOP on software development, or have neutral opinion about it.



Strongly agree	1	3%
Agree	3	8%
Neither agree or disagree	8	20%
Disagree	6	15%
Strongly disagree	0	0%

Fig 15 (AOP vs OOP)

VI. Solution Approach

This research tries to investigate and find the best approach to identifying Aspects in the early stage of software design. The idea is to make a global rule or regulation to make it systematic across all aspect oriented components and software design. For instance, in requirement stage either from the stockholder or business analysis. These requirements are functional requirements such as the activities of the business needs. However, there are other non-Functional requirements such as logging, security, performance and transaction management which should be taken into account during the development stage. AOP can implement these non-functional requirements separately and can span across the entire business model. This makes it easier to change or maintain this part later in the system lifecycle [17]. Another approach to identifying aspects is to define stakeholder concerns, refine the stakeholder related concerns, define cross cutting concern, separate cross cutting concern and finally weave these cross cutting concern across the system [18]. There are also several other approaches that we have mentioned in state of art section, they also show that it is possible to identify cross cutting in UML design diagrams in the design stage. However, there is not a unique approach to identify ,where Aspects should be or when they should be triggered.

VII. Conclusion and Future Work

In this paper, we presented initial investigations into identifying Aspects in software design stage. The research shows that there are many approaches to identifying AOP in software design, but not all approaches are applied in all cases or have specific rules or standards that can easily found cross cutting concern in any system easily. Adding more we have shown that in Kurdistan region most software engineering and academic staff do not have enough knowledge about this new approach. Therefore, we thought it was important to start working on how to find a standard approach to identifying a crosscutting concern and Aspect in the early stage of the system on requirement stage.

In future work we will try to find rules in software requirement and design that automatically will specify cross cutting of aspect in early stage. We will develop a new stage of the Aspect Specification approach in all systems and extract the possible aspect of the system. We will do that by applying it to different case studies in the real world.

VIII. Acknowledgment

We thanks University of Human Development Staff for answering my survey also all postgraduate students of Huddersfield University. Thanks to Dr. Gary for his usual support.

IX. References

[1] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, “Getting started with AspectJ,” Communications of the ACM, vol. 44, no. 10, pp. 59–65, 2001.

[2] H. Li, M. Zhou, G. Xu, and L. Si, “Aspect-oriented Programming for MVC Framework,” in Biomedical Engineering and Computer Science (ICBECS), 2010 International Conference on, 2010, pp. 1–4.

[3] J. D. Gradecki and N. Lesiecki, Mastering AspectJ: aspect-oriented programming in Java. John Wiley \& Sons, 2003.

[4] S. Iqbal and G. Allen, “Designing Aspects with AODL,” International Journal of Software Engineering, vol. 4, no. 2, pp. 3–18, 2011.

[5] C. Sant’Anna, A. Garcia, U. Kulesza, C. Lucena, and A. Von Staa, “Design patterns as aspects: A quantitative assessment,” Journal of the Brazilian Computer Society, vol. 10, no. 2, pp. 42–55, 2004.

[6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, Aspect-oriented programming. Springer, 1997.
[7] R. France, I. Ray, G. Georg, and S. Ghosh, "Aspect-oriented approach to early design modelling," IEE Proceedings-Software, vol. 151, no. 4, pp. 173–185, 2004.
[8] R. Pawlak, L. Seinturier, J.-P. Retaillé, and H. Younessi, Foundations of AOP for J2EE Development. Springer, 2005.
[9] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of AspectJ," in ECOOP 2001—Object-Oriented Programming, Springer, 2001, pp. 327–354.
[10] I. Jacobson, "Use cases and aspects-working seamlessly together," Journal of Object Technology, vol. 2, no. 4, pp. 7–28, 2003.
[11]S. Iqbal, "Aspects and Objects: A Unified Software Design Framework," 2013.
[12]D. Stein, S. Hanenberg, and R. Unland, "A UML-based aspect-oriented design notation for AspectJ," in Proceedings of the 1st international conference on Aspect-oriented software development, 2002, pp. 106–112.
[13] S. A. Khan and A. Nadeem, "UML extensions for modeling of aspect oriented software: a survey," in Proceedings of the 2010 National Software Engineering Conference, 2010, p. 5.
[14] S. Iqbal and G. Allen, "On Identifying and Representing Aspects.," in Software Engineering Research and Practice, 2009, pp. 497–501.
[15] M. Berkane, M. Boufaida, and L. Seinturier, "Reasoning about design patterns with an Aspect-Oriented approach," in Information Technology and e-Services (ICITeS), 2012 International Conference on, 2012, pp. 1–7.
[16] M. L. Bernardi and G. A. Di Lucca, "Improving Design Patterns Modularity Using Aspect Orientation," STEP 2005, p. 209, 2005.
[17]K. Sirbi and P. J. Kulkarni, "Stronger enforcement of security using aop and spring aop," arXiv preprint arXiv:1006.4550, 2010.
[18]A. Rashid, "Aspect-oriented requirements engineering: An introduction," in International Requirements Engineering, 2008. RE'08. 16th IEEE, 2008, pp. 306–309.