Research Article

Aspect Oriented Programming: Trends and Applications

Shko Muhammed Qader[1,3a*], **Bryar A. Hassan**[2b], **Hawkar Omar Ahmed** [1,4c], **Hozan Khalid Hamarashid**[3,d]

1 Department of Information Technology, University College of Goizha, Sulaymaniyah, Kurdistan Region, Iraq
2 Department of Information Technology, Kurdistan Institution for Strategic Studies and Scientific Research (KISSR), Sulaymaniyah, Kurdistan Region, Iraq
3 Information Technology Department, Computer Science Institute, Sulaimani Polytechnic University, Sulaymaniyah, Kurdistan Region, Iraq
4 Department of Information Technology, College of Commerce, University of Sulaimani, Sulaymaniyah, Kurdistan Region, Iraq
E-mail:[a]shko.qader@spu.edu.iq,[b]bryar.hassan@kissr.edu.krd,[c]hawkar.omar@univsul.edu.iq, [d]hozan.khalid@spu.edu.iq

Access this article online		
Received on: 23 September 2021	**Accepted on:** 18 April 2022	**Published on:** 30 June 2022
DOI: 10.25079/ukhjse.v6n1y2022.pp12-20	**E-ISSN:** 2520-7792	

Abstract

The competitive advantage of aspect oriented programming (AOP) is that it improves the maintainability and understandability of software systems by modularizing crosscutting concerns. However, some concerns, such as logging or debugging, may be overlooked and should be entangled and distributed across the code base. AOP is a software development paradigm that enables developers to capture crosscutting concerns in split-aspect modes. Additionally, it is a novel notion that has the potential to improve the quality of software programs by removing the complexity involved with the production of code tangles via the usage of separation of concerns. As a result, it provides more modularity. Throughout its early development, some believed that AOP was easier to build and maintain than other implementations since it was based on an existing one. The statements are predicated on the premise that local improvements are easier to implement. Additionally, without appropriate visualization tools for both static and dynamic structures, cross-cutting challenges may be difficult for developers and researchers to appreciate. In recent years, AspectJ has begun to enable the depiction of crosscutting concerns via the release of IDE plugins. This article explains aspect oriented programming and how it may be used to improve the readability and maintainability of software projects. Additionally, it will evaluate the challenges it presents to application developers and academics.

Keywords: Aspect Oriented Programming, AOP, Logging, Cross Cutting Concerns, Join Points, Point Cut.

1. Introduction and Background

Since the inception of programming languages, a multitude of programming paradigms have been established in response to technological advancements. Nowadays, the complexity of software increased, manufacturers were forced to broaden and adapt their methods to emerging difficulties. As a consequence of the emergence of programming languages, prototypes of procedural and object-oriented programming (OOP) have been built. Of the ongoing innovation that has happened in order for them to stay competitive. In recent years, OOP has established itself as the primary technique for writing successful structure-oriented code. In comparison to OOP, AOP is a relatively recent technique that has made substantial advancements (Zhang, 2011). According to (Rademacher et al., 2019), their study demonstrates that OOP may

fundamentally aid software engineering by providing a better match for the programming challenge through the core object model. However, AOP complements OOP by introducing a novel approach to program structuring. Enable Apart from classes, it provides features that facilitate the modularization of concerns. For instance, transaction management may be applied to a variety of different kinds of objects (Lau et al., 2018). Additionally, it is an innovative software design and implementation approach proposed by Xerox PARC experts (Lemos et al., 2006). Furthermore, the AOP divides the process into two components: the base program and the aspect program.

The base code contains the essential aspects of the system, and object oriented programming may be implemented. Additionally, the aspect program incorporates cross-cutting characteristics via the use of modularization in addition to the basic software programming (Beneken et al., 2005). The latest generation of AOP technology is distinct from technology that is more organized for more effective design and coding. This division may be necessary to show the creators' desire to think about it. Additionally, one of the benefits of AOP is that it is built on current technologies and may provide additional mechanisms for influencing the development of systems in a cross-cutting issue manner. One aspect is capable of assisting in the deployment of a range of activities, components, or objects. It might be homogenous or heterogeneous. For example, while leveraging the homogeneous feature, logging behavior should take specific processes into consideration. By contrast, heterogeneity may be used to implement both sides of a protocol between two distinct classes.

In contrast to previous separation of concerns approaches, AOP's primary goal is to build more modular architectures and code; the majority of concerns are localized rather than distributed. A well-defined interface enables reasoning about a range of issues in relative isolation. Thus, they become (un)pluggable and customizable via independent development (Alshareef et al., 2020). This article will provide an introduction of aspect-oriented programming and explain the notion of a common set of important terminology. Additionally, the possibility for increased software application comprehension and maintainability will be highlighted. Additionally, it will assess the difficulties it poses for both application developers and academics. Finally, a comparison between AOP with OOP in terms of software maintainability will be made.

2. Related Works

AOP utilizes many of the following terminologies, and each of these terms need to be explained in order to have a fundamental comprehensive of the idea of AOP. However, it should be noted that individual implementations and AOP the existance of varites frameworks.

2.3. Cross Cutting Concern

According to (Rajan and Sullivan, 2005), a cross-cutting issue is a dimension within which design decisions are made. When it is recognized in conventional object-oriented designs, it becomes cross-cutting and may result in dispersed (code duplication) or tangled code (significant interdependence across systems), or both. As a result of code dispersion, cross-cutting problems are executed inside the underlying program. For instance, log writing capability is needed in the majority of components. As a result, implementation should spread among them. Additionally, since they are compelled to deal with non-core operations, the responsibilities of the various components will remain unclear. Specifically, the command of components in relation to the number of new features is cross-cutting functionality, sometimes referred to as knotted code. The following Figure 1 illustrates a cross-cutting topic in further detail.

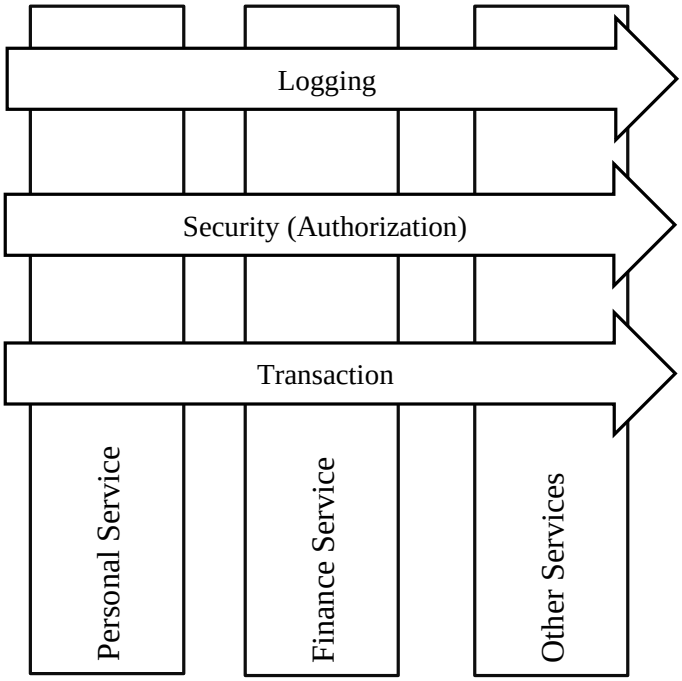

Figure 1. Cross Cutting Concerns (Ju and Bo, 2007)

2.4. Aspect

Aspect is a modular unit that satisfies cross-cutting criteria and may be specified as a class, for example, build. Additionally, it has a number of pointers that are well related. The primary concept of aspect oriented programming (AOP) is to encapsulate cross-cutting concerns functionally separate from the fundamental core programming in order to differentiate specified modules. For instance, the AOP aspect may be used to identify a logging module. Meanwhile, applications might have a variety of aspects, depending on the aspect language's needs. However, aspects may be built hierarchically, and the language may provide methods for expressing an aspect and associating it with a core system.

2.5. Join Point

A join point for the request's execution is a well-defined place in a program that handles cross-cutting. During execution, instructions referred to as join points may be performed. It varies according to the aspect language; for example, joint point might refer to the methods that are executed, such as exception handler execution or modifying the class's properties (Wand et al., 2001). Typically, each problem has a number of common points. However, if there are just a handful, they might easily modify the code manually. AspectJ is an aspect-oriented modification to Java that has been used as an example to aid in comprehension; while constructing an aspect, join points are specified. When specifying point cuts, it must determine which joint point performs the action. Additionally, it has a limited number of accessible join points for doing the following: (Vidal et al., 2015).

1. Calling methods
2. Calling constructor
3. Writing or reading access to a filed
4. Initializing execution of class and objects
5. Executing exception handler

2.6. Point Cut

A point cut is a collection of one or more entry points used to access the running program's execution. One of the points indicated in the pointcut is also used by guidance and should be implemented. A programmer may specify the time and

location of when and where the additional code shall be executed (Avgustinov et al., 2007). AspectJ's point cut expression is one of the most well-known varieties of point cut expression languages. The model's appearance is shown in Figure 2.

```
point_cut aspectJonPC ():
cflowbelow ( call_function (* coloringClient.*(..)) && this (SomeCallerFunction)) && call (FigureElement
Figure.make*(..))
```

Figure 2. (sample code) Models of AspectJ pointcut (Lehmann et al., 2012)

2.7. Advice

The advise develops an AOP and functional programming community of functions that change other functions. Additionally, it is a guaranteed function, method, or procedure that is implemented at the point of connection in order to facilitate the execution of a common application (Hentunen, 2015). Additionally, it addresses the functional needs for addressing cross-cutting usability issues. Numerous types of advice are provided in conjunction with various modes of communication. It is decided by the action that advise is capable of calling with the target method around a join point, either before, after, or both.

2.8. Weaving

Weaving is a method for sending suitable advice at each execution step or for linking bits of an advised item to other application requests or objects (Hentunen, 2015). These operations occur during the compilation phase, the load phase, or the run phase (Chapman et al., 2013). An isolated aspect compiler is used to weave the code during the compilation process. For instance, AspectJ is one of the most common compile-time aspect language implementations. The class loader is responsible for weaving throughout the virtual machine class loading process in load-time weaving. Additionally, runtime weaving accomplishes the binding via the usage of proxy classes and the building of code libraries. Furthermore, the Spring Framework is a well-known example of a runtime implementation (Nguyen, 2018).

3. Advantages and Shortcomings of AOP

Utilizing aspect-oriented programming has a number of benefits. For instance, it may be an effective method for resolving cross-cutting issues. However, various disadvantages may develop.

3.1 Advantages of AOP

As previously stated, AOP enables the management of cross-cutting business functions by encapsulating the desired system. Alternatively, the function may be divided into discrete modules. Since a consequence, the entire structure may be compromised, as the base program is not accountable for cross-cutting functionality (Raheman et al., 2018). This technique will automatically result in less code duplication and a lower chance of errors. OOP may also be used to expand or modify the usability of it is feasible that fresh extra features can be implemented without modifying the fundamental software. While this is useful when the source code is unavailable for other reasons, it must be deleted since it may result in altering side effects. Additionally, it may be used to evaluate apps through software testing guidance may be necessary to invoke methods and track application counts and execution time. Advice may be validated prior to and after the use of the approaches. Refer to Figure 3.

```
before (): examplePoint_cut()
{
        //excutable code goes here
}
after() returning: examplePoint_cut()
{
        //excutable code goes here
}
```

Figure 3. (Sample code) Simple example of creating advice (before and after) (Gulia et al., 2019)

3Disadvantages of AOP

AOP is a very efficient programming approach that is dependent on the implementation of the aspect language. This procedure may involve the addition of class attribute values that may be used to adjust methods' arguments and return values. Additionally, this might open up new options that can result in an increase in the system's process complexity, resulting in difficult-to-trace faults. According to (Raheman et al., 2018), the use of AOP may add to the system's complexity. This is reason for a programmer to explore any associated classes and characteristics in order to comprehend the system's behavior. Additionally, when applied to AOP, the usability of the base program and any aspect programs that are necessary for comprehending the program in its whole. Additionally, it is necessary to study the characteristics of the super class in order to comprehend its subclass. Additionally, one potential disadvantage arises in the point cuts that connect the join points with the others' advise. For instance, the description of the error point cut may result in guidance to be compelled to enter incorrect join points or to avoid entering at all stages of the join points. Figure 4 illustrates the many types of mistakes that might occur while describing a point cut.

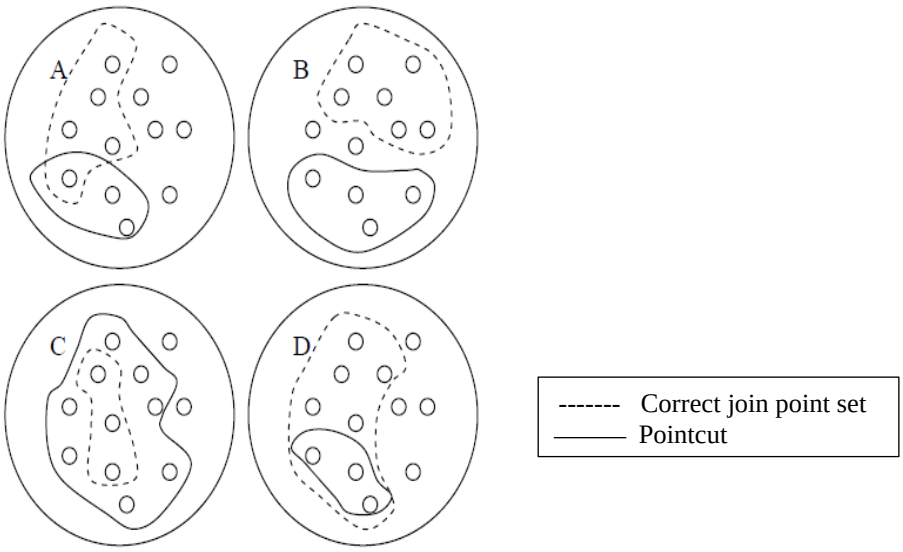

Figure 4. Potential Error Situations with Point Cuts.

The following mistake scenarios are possible with point cuts (Lemos et al., 2006); see Figure 4:

1. Point cut selects a subset of the join points offered.
2. No join points were identified by the point cut.
3. A point cut selects anything that is desired or incidental.
4. Point cut deliberately selects many, but not all.

Inaccuracies in the thumbnail description may result in difficult-to-detect behavior. If the system has connected pointcut faults, the effective implementation of an aspect-based transaction management technique may result in a number of system difficulties.

4. Evaluating AOP against OOP

The comparison of AOP vs OOP focuses mostly on cross-cutting concerns. AOP is used to modularize dispersed code; creating modules from scattered code simplifies logging and improves the readability of the code. In contrast to AOP, OOP cannot be represented physically. However, AOP is a seldom utilized technique that may incur considerable runtime costs.

```
Public class Log {
        public static void new (String message){
                System.out.println(message);
        }
}
Public class MyClass{
        int number = 0;
        public void addOne () {
                Log.new("Beginning of addOne");
                number = number+1;
                Log.new("End ofaddOne");
        }
        public void subtractOne () {
                Log.new("Beginning of subtractOne");
                number = number -1;
                Log.new("End of subtractOne");
        }
}
```

Figure 5. (Sample code) A logger implemented in Java (Lehmann et al., 2012)

```
public aspect Log {
        point_cut log(): execution (void add*(*)) && execution (void subtract*(*));
                before (): log () {
                        System.out.println("Beginning of "+thisJoin
                        Point.getTarget().getClass().getName());
                }
                after () : log() {
                        System.out.println("End of" + thisJoin
                        point.getTarget().getClass().getName();
                }
}

public class MyClass {
        int number = 0;
                public void addOne() {
                        number = number +1;
                }
                public void subtractOne() {
                        number = number -1;
                }
}
```

Figure 6. (Sample code) A logger implemented in Java using AspectJ (Lehmann et al., 2012)

There are two methods in Figure 5, 'addOne' and'subtractOne', as well as two log calls that have no influence on the functionality of the calling method. These techniques may need extensive and sophisticated code as a result of the implementation of this kind of cross-cutting issue; one advantage is that the logging may be abstracted through class functions. Additionally, the log aspect abstracts whole logs. Demonstrates how to use the logger. By comparison, Figure 6 Additionally, 'MyClass' methods have a unique and suitable functionality where all handling is contained inside a single function.

5. AspectJ in AOP

AOP's primary goal is to try to resolve code tangling and scattering difficulties by modularizing and encapsulating crosscutting concerns. It introduces a novel concept referred to as an element in order to include a variety of cross-cutting problems (Mcheick and Godmaire, 2018). Additionally, AspectJ may communicate with the underlying code through pointers, advice, and type declarations. Refer to Figure 7, which has an example of Java code.

```
aspect TracingExample {
        //capture the execution of methods in
        // classes named * in methods named *
        // with any parameters in package
        // "example"

point_cut trace () : execution (* * (. .)) && within (example);

before () : trace () {
        // Excute this code
        // before the above point is reached
        System.out.println ("Entering"+thisJoinPoint.getSignature().getName());
}
```

Figure 7. (Sample code) AspectJ code snippet (Lehmann et al., 2012)

In terms of intertype declarations or intro- durations, they are a method that enables application developers to crosscut concerns in a static manner. For instance, extending a class with additional methods or characteristics (Panwar et al., 2019).

6. Concluding Remarks

Through the use of AOP, a novel and inventive technique for resolving cross-cutting issues has arisen. AOP provides an alternate answer to issues that are difficult to handle with conventional OOP. However, as the system evolves and becomes more complicated, AOP has yet to make a fundamental breakthrough that will answer all cross-cutting issues in the near future. At the moment, AOP is unable to function successfully due to a lack of developing materials and resource information from other initiatives. Despite these difficulties, it is expected to be included into future programming standards. AOP is used to develop and construct dynamic and structured applications. AOP paired with AOP may create a novel approach for programming in which the AOP may provide tools for managing cross-cutting difficulties. As a result, the combination of object and aspect orientation is advantageous. The modularized nature of cross-cutting activities may result in a succinct and ordered structure, which can assist decrease the likelihood of mistake. This is made feasible by the ability to incorporate the functionality of several components into software. As a consequence, less code is required than would be necessary with pure OOP software. Additionally, AOP is more effective. Additionally, to what has been said, despite the many benefits of AOP, there are some worries concerning its transparent and nonintrusive functions. For instance, characteristics may be adjusted independently of source code access information using the present essential program features. Additionally, researchers assert that programmers who improve their comprehension and maintainability of application components will get superior outcomes in the era of programming languages. We have noticed that in recent years, AOP is still considered one of the most promising programming languages in terms of being integrated with other technologies. This means that the changes and developments in it are continuing to come up with new methods that can be more effective and capable of yielding more satisfying results. For future reading, the authors advise the reader could optionally read the following research works (Bryar A.Hassan, 2020; B. A. Hassan, 2020, 2021; B. A. Hassan, Ahmed, Saeed, and Saeed, 2016; B. A. Hassan and Qader, 2021; B. A. Hassan and Rashid, 2019, 2021a; B. A. Hassan, Rashid, and Hamarashid, 2021; B. A. Hassan, Rashid, and Mirjalili, 2021; B. Hassan and Dasmahapatra, n.d.; Maaroof et al., 2022; Saeed, Hassan, and Qader, 2017)(B. A. Hassan and Rashid, 2021b)

References

Alshareef, S. F., Maatuk, A. M., Abdelaziz, T. M., and Hagal, M. (2020). Validation Framework for Aspectual Requirements Engineering (ValFAR). *Proceedings of the 6th International Conference on Engineering & MIS 2020*, 1–7.

Avgustinov, P., Hajiyev, E., Ongkingco, N., de Moor, O., Sereni, D., Tibble, J., and Verbaere, M. (2007). Semantics of static pointcuts in AspectJ. *ACM SIGPLAN Notices*, *42*(1), 11–23.

Bryar A.Hassan and T. A. R. (2020). Formal Context Reduction in Deriving Concept Hierarchies from Corpora Using Adaptive Evolutionary Clustering Algorithm. Complex & Intelligent Systems.

Beneken, G., Marschall, F., and Rausch, A. (2005). A Model Framework Weaving Approach, In: Proceedings of the First Workshop on Models and Aspects-Handling Crosscutting Concerns in MDSD at the 19th European Conference on Object-Oriented Programming (ECOOP 2005). *19th European Conference on Object-Oriented Programming (ECOOP 2005)*.

Chapman, M. P., Colyer, A. M., and Dalziel, B. J. (2013). *Monitoring dynamic aspect oriented applications at execution time*. Google Patents.

Gulia, P., Khari, M., and Patel, S. (2019). Metrics analysis in object oriented and aspect oriented programming. *Recent Patents on Engineering*, *13*(2), 117–122.

Hentunen, D. (2015). *Detecting a return-oriented programming exploit*. Google Patents.

Hassan, B. A. (2020). CSCF: a chaotic sine cosine firefly algorithm for practical application problems. Neural Computing and Applications, 120.

Hassan, B. A. (2021). Analysis for the overwhelming success of the web compared to microcosm and hyper-G systems. ArXiv Preprint ArXiv:2105.08057.

Hassan, B. A., Ahmed, A. M., Saeed, S. A., and Saeed, A. A. (2016). Evaluating e-Government Services in Kurdistan Institution for Strategic Studies and Scientific Research Using the EGOVSAT Model. Kurdistan Journal of Applied Research, 1(2), 1–7.

Hassan, B. A. and Qader, S. M. (2021). A New Framework to Adopt Multidimensional Databases for Organizational Information System Strategies. ArXiv Preprint ArXiv:2105.08131.

Hassan, B. A. and Rashid, T. A. (2019). Operational framework for recent advances in backtracking search optimisation algorithm: A systematic review and performance evaluation. Applied Mathematics and Computation, 124919.

Hassan, B. A. and Rashid, T. A. (2021a). A multidisciplinary ensemble algorithm for clustering heterogeneous datasets. Neural Computing and Applications. URL: https://doi.org/10.1007/s00521-020-05649-1

Hassan, B. A. and Rashid, T. A. (2021b). Artificial Intelligence Algorithms for Natural Language Processing and the Semantic Web Ontology Learning. ArXiv Preprint ArXiv:2108.13772.

Hassan, B. A., Rashid, T. A., and Hamarashid, H. K. (2021). A Novel Cluster Detection of COVID-19 Patients and Medical Disease Conditions Using Improved Evolutionary Clustering Algorithm Star. Computers in Biology and Medicine, 104866.

Hassan, B. A., Rashid, T. A., and Mirjalili, S. (2021). Performance evaluation results of evolutionary clustering algorithm star for clustering heterogeneous datasets. Data in Brief, 107044.

Hassan, B. and Dasmahapatra, S. (n.d.). Towards Semantic Web: Challenges and Needs.

Ju, K. and Bo, J. (2007). Applying IoC and AOP to the Architecture of Reflective Middleware. *2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC 2007)*, 903–908.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. (2001). Getting started with AspectJ. *Communications of the ACM*, *44*(10), 59–65.

Lau, R. Y. K., Zhang, W., and Xu, W. (2018). Parallel aspect-oriented sentiment analysis for sales forecasting with big data. *Production and Operations Management*, *27*(10), 1775–1794.

Lemos, O. A. L., Ferrari, F. C., Masiero, P. C., and Lopes, C. V. (2006). Testing aspect-oriented programming pointcut descriptors. *Proceedings of the 2nd Workshop on Testing Aspect-Oriented Programs*, 33–38.

Mcheick, H. and Godmaire, S. (2018). Designing and implementing different use cases of aspect-oriented programming with AspectJ for developing mobile applications. *Proceedings of the 7th International Conference on Software Engineering and New Technologies*, 1–8.

Maaroof, B. B., Rashid, T. A., Abdulla, J. M., Hassan, B. A., Alsadoon, A., Mohamadi, M., … Mirjalili, S. (2022). Current Studies and Applications of Shuffled Frog Leaping Algorithm: A Review. Archives of Computational Methods in Engineering, 1–16.

Nguyen, T. (2018). *Java Spring Framework in developing the Knowledge Article Management application: A brief guide to use Spring Framework*.

Panwar, P., Agarwal, D., Vyas, P., Jadhav, P. A., and Joshi, S. D. (2019). Evolution of Testing with Respect to the Programming Paradigms. *International Journal of Mechanical Engineering and Technology*, *10*(3).

Rademacher, F., Sachweh, S., and Zündorf, A. (2019). Aspect-oriented modeling of technology heterogeneity in microservice architecture. *2019 IEEE International Conference on Software Architecture (ICSA)*, 21–30.

Raheman, S. R., Maringanti, H. B., and Rath, A. K. (2018). Aspect oriented programs: Issues and perspective. *Journal of Electrical Systems and Information Technology*, *5*(3), 562–575.

Rajan, H. and Sullivan, K. J. (2005). Classpects: unifying aspect-and object-oriented language design. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 59–68.

Saeed, M. H. R., Hassan, B. A., and Qader, S. M. (2017). An Optimized Framework to Adopt Computer Laboratory Administrations for Operating System and Application Installations. Kurdistan Journal of Applied Research, 2(3), 92–97.

Vidal, C., Benavides Cuevas, D. F., Leger, P., Galindo Duarte, J. Á., and Fukuda, H. (2015). Mixing of join point interfaces and feature-oriented programming for modular software product line. *BICT 2015: 9th EAI International Conference on Bio-Inspired Information and Communications Technologies (2015), p 433-437*.

Wand, M., Kiczales, G., and Dutchyn, C. (2001). A semantics for advice and dynamic join points in aspect-oriented programming. *SAIG*, 45–46.

Zhang, L. (2011). Study on comparison of AOP and OOP. *2011 International Conference on Computer Science and Service System (CSSS)*, 3596–3599.