International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 09 | Sep 2018 www.irjet.net p-ISSN: 2395-0072

Aspect-Oriented Software Development for Real-Time and Internet Applications

Ogbonna J. C.[1], Nwokoma F. O.[2], Nwala K. T.[3], Nwandu I. C.[4]

[1,3]Reseacher, Dept. of Computer Science, Clifford University Owerrinta, Abia State Nigeria
[2,4]Reseacher, Dept. of Computer Science, Federal University of Technology Owerri, Imo State Nigeria

---

Abstract - *Software development evolution has introduced a number of useful software development methods. Efficient modularization of program artifacts that cut across multiple locations during software development called crosscutting concerns have been a major drawback for these software development methods including the almighty Object-Oriented Software Development method. Concerns like recovery, synchronization, logging, encryption, authentication, authorization, validation, verification, caching, transaction processing, monitoring, error detection and correction, optimizations etc are common and cut-across many modules. That is to say that the source codes for these concerns are duplicated in so many modules. In the case of Object-Oriented Software Development method, these concerns are not captured as an aspect but their source codes duplicated across multiple object methods because of the encapsulation principle. These crosscutting concerns reduce the reusability, maintainability and adaptability of software products, hence the need for a software development method called Aspect-Oriented Software Development (AOSD) that gives a better modularization implementation in software development which are used for the composition of the program components more especially to the Object-Oriented Software Development method.*

Key Words: Aspect-Oriented Software Development, Software Modularization, Software Evolution, Joinpoint, Advice, Aspect, Crosscutting Concerns, Weaving.

1. INTRODUCTION

Aspect-Oriented Software Development (AOSD) is an attractive software development model that aimed at complementing and improving a wide variety of modern development approaches such as Object-Oriented approach, Model-Driven Development approach etc. In addition, AOSD offers a wide variety of advanced and unique program development and modularization mechanisms (Albahar, 2015). Research has shown that the development of software applications through aspect-oriented software development mechanisms improves the implementation structure of a software application which has significant influence on a number of important software attributes (Albahar, 2015). AOSD mechanisms improve the quality of software development lifecycle by reducing the complexity and increasing the reusability of a software product, which in turns results in better software development.

AOP is based on the idea that computer systems are better programmed by separately specifying the various concerns (properties or areas of interest) of a system and some description of their relationships, and then relying on mechanisms in the underlying AOP environment to weave or compose them together into a coherent program (Elrad, Filman, & Bader, 2001a). These concerns include Security (verification, validation, authentication, authorization, etc), Quality of Service (performance, reliability, and availability), etc.

The major benefit of an aspect-oriented approach is that it supports the separation of concerns, and they include a definition of where they should be included in a program, as well as the code implementing the crosscutting concern. You can specify that the crosscutting code should be included before or after a specific method call or when an attribute is accessed (Sommerville, 2011a).

Assume we have a requirement that user authentication is required before any change to personal details is made in a database. We can describe this in an aspect by stating that the authentication code should be included before each call to methods that update personal details. Subsequently, we may extend the requirement for authentication to all database updates. This can easily be implemented by modifying the aspect. We simply change the definition of where the authentication code is to be woven into the system. We do not have to search through the system looking for all occurrences of these methods. We are therefore less likely to make mistakes and introduce accidental security vulnerabilities into our program (Sommerville, 2011b).

1.1 The Principle of Separation of Concerns

The principle of separation of concerns (Sommerville, 2006, 2011b) states that:

- Software should be organized so that each program element does one thing and one thing only.
- Each program element should therefore be understandable without reference to other elements.

- Program abstractions such as subroutines, procedures, objects, etc. support the separation of concerns.

Separating concerns into independent elements rather than including different concerns in the same logical abstraction is good software engineering practice. By representing cross-cutting concerns as aspects, these concerns can be understood, reused, and modified independently, without regard for where the code is used (Sommerville, 2011b). For instance, user authentication such as password authentication, physical authentication (smart cards, digital certificates etc), biometric authentication (signature analysis devices, fingerprints, retina eye scans and voice analysis devices etc), each can be refactored into an aspect and can be automatically woven into a given program wherever that particular authentication is required.

1.2 Aspect-Oriented Software Development Terminology

The following are some of the terminology used in an aspect-oriented software development.

- **Advice**: The code implementing a concern
- **Aspect**: Aspect is a program abstraction that defines a Crosscutting Concern. It includes the definition of a Pointcut and the advice associated with that concern.
- **Join Point**: Join Point is an event in an executing program where the advice associated with an aspect may be executed.
- **Pointcut**: Pointcut is a statement, included in an aspect, which defines the Join Points where the associated aspect advice is to be executed.
- **Weaving**: Weaving is the incorporation of advice code at the specified Join Points by an aspect weaver.

2. ASPECT-ORIENTED SOFTWARE DEVELOPMENT EVOLUTION

Software development techniques need to continuously evolve to cope with the ever dynamic software requirements (Team, 2018). The first law of software evolution states that "A program that is used must be continually adapted else it becomes progressively less satisfactory" (Mens, Mens, & Tourwé, 2004).

This need for software to evolve continuously poses important challenges for software engineers. Advanced automated software engineering techniques and tools are needed to improve software evolution support.

An essential problem with software development is the tyranny of the dominant decomposition (Elrad, Filman, & Bader, 2001b; Jonckers, 2009; Mei & Gent, 2004; Mussbacher, Amyot, & Weiss, 2007; Suvée, Vanderperren, & Jonckers, 2003). No matter how carefully a software system is decomposed into modular units; there will always be concerns (typically non-functional ones) that cut across the chosen decomposition. The code of these crosscutting concerns will necessarily be spread over different modules, which has a negative impact on the software quality in terms of comprehensibility, adaptability and evolvability (Mens et al., 2004)

Aspect-oriented software development (AOSD) is a solution to this problem. In order to capture crosscutting concerns in a localized way, a new abstraction mechanism (called an aspect) is added to existing programming languages (e.g. AspectJ for Java, AspectC++ for C++ etc). As a result, crosscutting concerns are no longer distributed over different modules. This means that the software is easier to maintain, reuse, evolve and understand (Mens et al., 2004).

2.1 Cross-Fertilization

For aspect-oriented software development to be successful current software programs require being translated to their aspect-oriented equivalent with a continual rephrasing. For the size of industrial software, automated support is required for three important activities which are aspect mining, aspect introduction and aspect evolution (Team, 2018). Given the size and complexity of industrial software systems, this must be achieved with as much automated support as possible as shown in Figure 1. An automated support is needed for three essential activities:

- **Aspect Mining**: Aspect Mining techniques are used to identify the relevant concerns in the source code.
- **Aspect Introduction**: Aspect Introduction techniques are used to define the appropriate aspects for any of the identified concerns, in order to translate the software into the equivalent aspect-oriented version.
- **Aspect Evolution**: Aspect Evolution techniques enable evolvability of aspect-oriented software products.

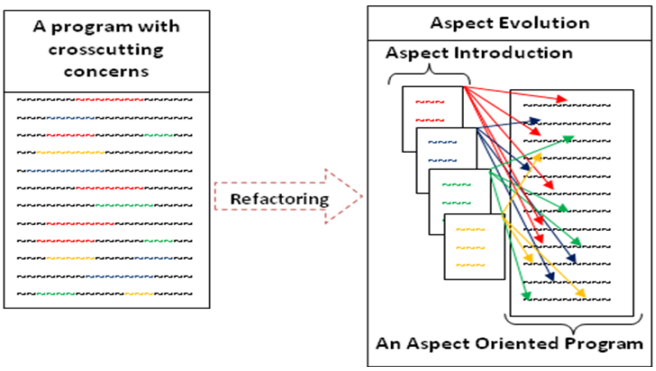

Fig -1: Aspect-Oriented Software Evolution Cross-fertilization

3. IDENTIFYING CROSSCUTTING CONCERNS

The separation of concerns is a key principle of software design and implementation. It means that you should organize your software so that each element in the program (class, method, procedure, etc.) does one thing and one thing only. You can then focus on that element without regard for the other elements in the program. You can understand each part of the program by knowing its concern, without the need to understand other elements. When changes are required, they are localized to a small number of elements (Sommerville, 2011a).

System performance may be a concern because users want to have a rapid response from a system; some stakeholders may be concerned that the system should include particular functionality; companies who are supporting a system may be concerned that it is easy to maintain. A concern can therefore be defined as something that is of interest or significance to a stakeholder or a group of stakeholders. It is easier to trace concerns, expressed as a requirement or a related set of requirements, to the program components that implement these concerns. If the requirements change, then the part of the program that has to be changed is obvious (Sommerville, 2011b).

3.1 Types of Stakeholder Concerns

There are a number of concerns depending on their type/category:

- **Functional Concerns** (Primary functional concerns): Functional concerns are related to the specific functionality to be included in a system. For example, in a mobile banking application, specific functional concerns could be: Transfer Fund, Check Account Balance, Buy Airtime, QR Payment, etc.
- **Non-Functional Concerns** (Secondary functional concerns): These features are used to add extra functionalities to the functional concerns that are being integrated into the core system. For example, in a mobile banking application, non-functional concerns could be: keeping a log of all the transactions, statement of account, synchronization etc.
- **Quality of Service (QoS) Concerns**: QoS concerns are related to the non-functional behaviour of a system. These include characteristics such as performance, reliability (including error checking), and availability.
- **Policy Concerns**: Policy concerns are related to the overall policies that govern the use of a system. Policy concerns include security and safety concerns, and concerns related to business rules.
- **System Concerns**: System concerns are related to attributes of the system as a whole, such as its maintainability or its configurability.
- **Organizational Concerns**: Organizational concerns are related to organizational goals and priorities. These include producing a system within budget, reusing existing software tools, maintaining the reputation of the organization, etc.
- **Infrastructure Concerns**: These add functional capabilities to support the implementation of the system on some platform. For instance, Android, IOS, and Windows Phone Mobile Banking Apps might have different specific platform functionalities.

3.2 The Separation of Concerns

Large-scale industrial software applications are inherently complex, and a good separation of concerns within the application is therefore indispensable. Unfortunately, recent insight reveals that the current means for separation of concerns, namely functional decomposition or object-oriented programming, are insufficient. No matter how well large applications are decomposed using current means, some functionality, typically called “crosscutting concerns”, will not fit the chosen decomposition. As a result, implementations of such crosscutting concerns will be scattered across the entire system, and become entangled with other code. In this case, the consequences for maintenance of the system and its future evolution are obviously dire (Bruntink, Van Deursen, & Tourwe, 2004).

Aspect-oriented software development is an improved means for separation of concerns. Aspect-oriented programming languages add an abstraction mechanism (called “aspect”) to existing (object-oriented) programming languages. This mechanism allows a developer to develop crosscutting concerns in a modular way. In order to use this new feature and make the code easier to maintain, existing applications written using traditional programming languages should be evolved into aspect-oriented applications (Bruntink et al., 2004). That is to say that scattered and tangled code implementing crosscutting concerns should be identified, and afterward refactored into aspects.

One proposed approach to managing complex behavioural models is to separate crosscutting concerns from the main behaviour by using aspect-oriented modelling. A crosscutting concern applies throughout multiple locations in the software, and may be crucial to the reliability, performance, security, or robustness of the system (Lindström, Offutt, Sundmark, Andler, & Pettersson, 2017).

Figure 2 shows a Security Concern handled by code in one class. This is a good modularity.

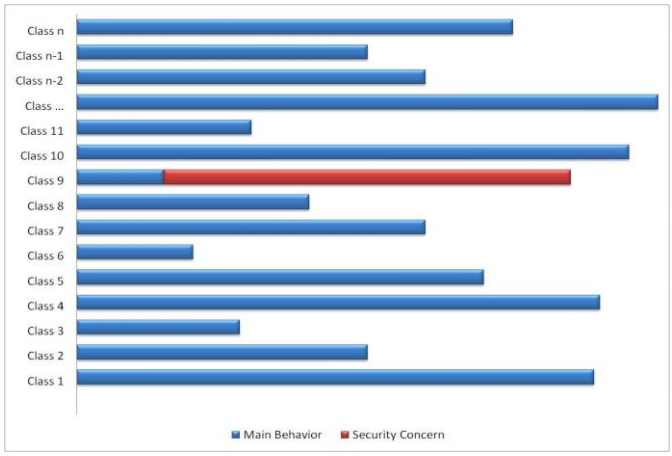

Fig -2: Security Concern handled by code in one class

Figure 3 shows Security Concern handled by code in two classes related by inheritance (i.e. Class9 extends Class6). This is a good modularity.

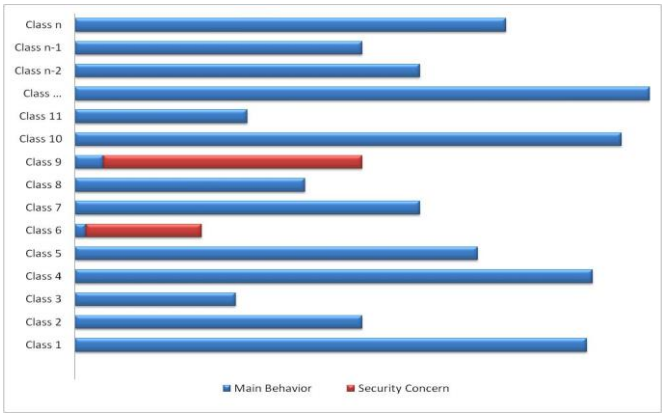

Fig -3: Security Concern handled by code in two classes related by inheritance

In a Real-Time and Internet Application, cryptography plays an important role in data security during data transmission. That is to say that during data transmission, all confidential data should be encrypted before transmission so that:

- The data cannot be accessed by unauthenticated and unauthorized persons (Confidentiality).
- The data cannot be altered during data transmission (Integrity).
- The sender cannot deny the authenticity of his/her signature on a message he/she originated (Non-repudiation).

Figure 4 shows a Cryptography Concern handled by code that is scattered over almost all classes. This is a bad modularity.

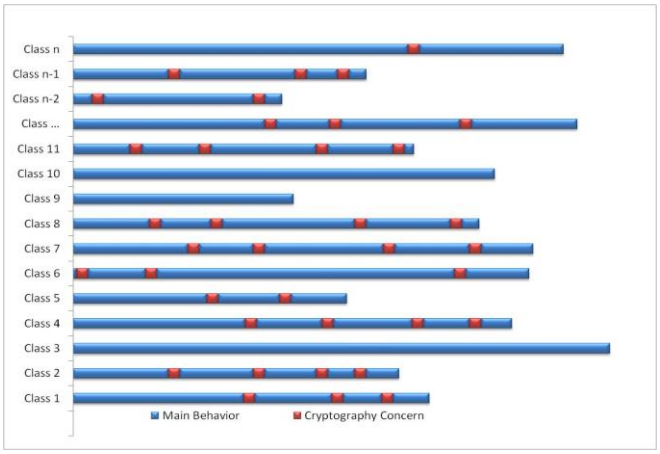

Fig -4: Cryptography Concern handled by code that is scattered over almost all classes

At the same time, figure 5 shows Cryptography Concern handled by Aspect-Oriented Programming, which is a good modularity.

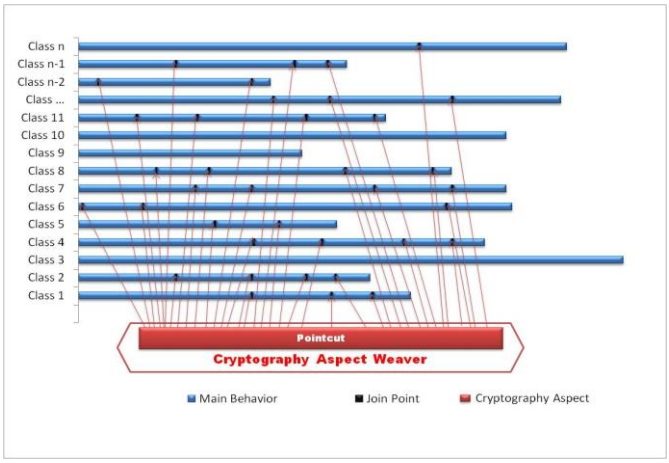

Fig -5: Cryptography Concern handled by Aspect-Oriented Programming

3.3 Scattering and Tangling Crosscutting Concerns

- The implementation of a concern is scattered if its code is spread out over multiple modules (code duplication). The concern affects the implementation of multiple modules and its implementation is not modular.
- The implementation of a concern is tangled if its code is intermixed with code that implements other concerns (i.e. code in one region addresses multiple concerns). The module in which tangling occurs is not reliable.
- Scattering and tangling often go together, even though they are different concepts (Malekzad, 2013).

4. ASPECT WEAVING

Weaving of aspects is normally done by executing the advices defined in the aspect at a resolved join point (Suganantham, Babu, & Raju, 2017).

Figure 6 and figure 7 shows the Transfer Fund concerns of a mobile banking application and Load Airtime concerns of the same mobile banking application respectively. In each case, we can see the crosscutting concerns that cut across 'Transfer Fund' and 'Load Airtime' modules; and in Figure 8, an Aspect-Oriented Software Development strategy is used to capture these concerns as Aspects, and finally uses an Aspect Weaver to weave the aspects into their respective Join Points.

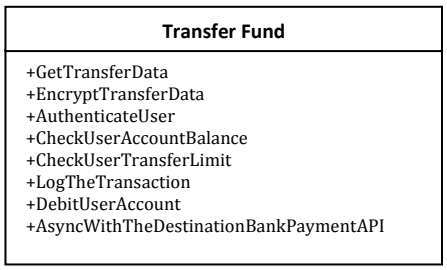

Fig -6: Transfer Fund Concerns

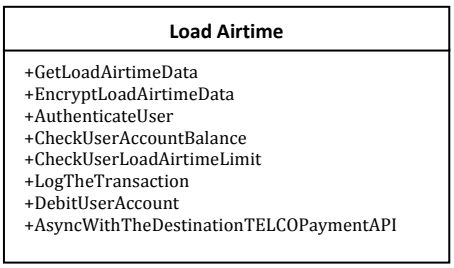

Fig -7: Load Airtime Concerns

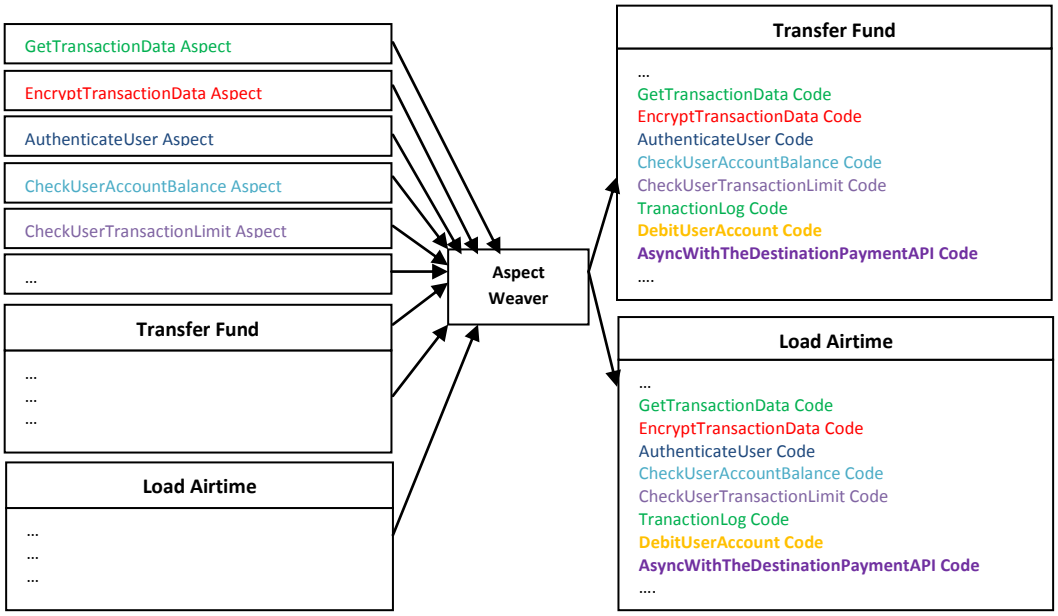

Fig -8: Real-Time and Internet Application Aspect Weaving

Examples of contexts where advice can be woven into a program include the following contexts:

- Before a specific method executes
- After a method call
- When one of the properties of an object is modified (Event Handler)

5. ASPECT-ORIENTED SOFTWARE DEVELOPMENT FRAMEWORKS

Some of the tools and frameworks used for the development of software products using AOSD method include the following:

- **AspectJ** (Eclipse Foundation, n.d.), is a seamless aspect-oriented extension to the Java programming language that clean modularization of crosscutting concerns, such as error checking and handling,

synchronization, context-sensitive behaviour, performance optimizations, monitoring and logging, debugging support, and multi-object protocols.

- **AspectC++** project, according to (Xerox Corporation, n.d.), is a set of C++ language extensions that extend the AspectJ approach to facilitate aspect-oriented programming with C/C++.
- **XWeaver** is a tool for aspect oriented programming for C/C++ and Java applications. The weaving process is especially designed to be compatible with the needs of applications that, like on-board applications, must undergo a qualification process. The XWeaver tool is built as a command line tool and a Plug-in for the Eclipse platform and is provided with a complete set of documentation and a large number of sample aspect programs (XWeaver, n.d.) .
- **JBoss-AOP** is a framework for Organizing Cross Cutting Concerns, and is a 100% Pure Java Aspect-Oriented Framework usable in any programming environment. JBoss-AOP is not only a framework, but also a prepackaged set of aspects that are applied via annotations, pointcut expressions, or dynamically at runtime. Some of these include caching, asynchronous communication, transactions, security, remoting, and many more (JBoss AOP, n.d.).
- **AspectWerkz** language has been moving closer and closer to that of AspectJ. The key difference is that whereas AspectJ uses a syntax that complements Java as defined in the original JLS (Java Language Specification), AspectWerkz supports both annotation and XML based styles of development. In addition, AspectWerkz has focused largely on close integration of load-time weaving into J2EE application environments while AspectJ has focused more on (static) compilation and weaving, runtime performance, and tools support (AspectWerkz, n.d.).
- **JAC** (Java Aspect Components) a framework for Aspect-Oriented Programming in Java, is an open-source software developed by the AOPSYSTM company with the collaboration of the LIP6, the CEDRIC, and the LIFL laboratories; and is a direct application of Renaud Pawlak's PhD Thesis (ObjectWeb Consortium, 2005).
- **LOOM.NET** project aims to investigate and promote the usage of Aspect Oriented Programming (AOP) in the context of the Microsoft .NET framework.
- **Nanning Aspects** is a simple and scalable Aspect Oriented Framework for Java based on dynamic proxies and aspects implemented as ordinary Java-classes.
- **CLAW** is a .NET a dynamic weaver implemented in C++ and using the Common Object Model (COM) to extend the CLR (Common Language Runtime) by linking in to the profiling mechanism supplied with the runtime. With this mechanism, it is possible to add a new method at runtime, inject new CIL (Common Intermediate Language, originally known as Microsoft Intermediate Language (MSIL)) code at runtime for an existing method body, relocate methods from one type to another, and recompile existing methods (Blackstock, 2004).
- **Aspect.NET** implementation is based on Microsoft Phoenix state-of-the-art multi-targeted optimizing infrastructure for developing compilers and other language tools, in particular, comfortable for creating and editing .NET assemblies. The weaver uses Phoenix IR for scanning target applications and weaving aspects (Safonov, Gratchev, Grigoryev, & Maslennikov, 2006)

6. CONCLUSION

In this paper, we have reviewed Aspect-Oriented Software Development (AOSD), which is a software development paradigm that complements Object-Oriented Software Development (OOSD) and other approaches to software engineering by providing a different way of thinking about the development process of software products. The key unit of modularity in OOSD is the class, whereas in AOSD the unit of modularity is the aspect. AOSD enable the modularization of concerns such as transactions that cut across multiple objects. The idea of separating concerns that underlies AOSD is important and thinking about the separation of concerns is a good general approach to software engineering. Research has shown that the development of software applications through aspect-oriented software development mechanisms improves the implementation structure of a software application, improves the quality of software development lifecycle by reducing the complexity and increasing the reusability and maintainability of software products.

REFERENCES

[1] Albahar, M. A. (2015). Aspect Oriented Software Engineering, (4), 29–31.

[2] AspectWerkz. (n.d.). AspectJ and AspectWerkz to Join Forces. Retrieved from http://www.eclipse.org/aspectj/aj5announce.html

[3] Blackstock, M. A. (2004). Aspect Weaving with C # and . NET. Retrieved from https://www.researchgate.net/profile/Michael_Blackstock/publication/228949568_Aspect_Weaving_with_C_and_NET/links/544166060cf2e6f0c0f61c80.pdf

[4] Bruntink, M., Van Deursen, A., & Tourwe, T. (2004). Identifying Cross-Cutting Concerns in Embedded C Code. European Research Consortium for Informatics and Mathematics, (58), 39–41.

[5] Eclipse Foundation. (n.d.). AspectJ project. Retrieved July 23, 2018, from http://eclipse.org/aspectj

[6] Elrad, T., Filman, R. E., & Bader, A. (2001a). Aspect-oriented programming: Introduction. Communications of the ACM, 44(10), 29–32.

[7] Elrad, T., Filman, R. E., & Bader, A. (2001b). Aspect-Oriented Programming. Communications of the ACM, 44(10), 28–32. https://doi.org/10.1145/317665.317679

[8] JBoss AOP. (n.d.). JBoss AOP: Framework for Organizing Cross Cutting Concerns. Retrieved September 18, 2018, from http://jbossaop.jboss.org/

[9] Jonckers, V. (2009). Aspect Oriented Software Development. Vrije Universiteit Brussel.

[10] Lindström, B., Offutt, J., Sundmark, D., Andler, S. F., & Pettersson, P. (2017). Using mutation to design tests for aspect-oriented models. Information and Software Technology, 81, 112–130. https://doi.org/10.1016/j.infsof.2016.04.007

[11] Malekzad, M. (2013). Aspect Oriented Software Development. PUCSD. Retrieved from https://www.slideshare.net/mmalekzad/aspect-oriented-software-development-16283495

[12] Mei, T. M., & Gent, H. P. (2004). Software Evolution and Aspect-Oriented Programming. In Scientific Research Network on Foundations of Sotware Evolution. Retrieved from http://prog.vub.ac.be/FFSE

[13] Mens, T., Mens, K., & Tourwé, T. (2004). Aspect-Oriented Software Evolution. ERCIM News, 58, 36–37.

[14] Mussbacher, G., Amyot, D., & Weiss, M. (2007). Visualizing Early Aspects with Use Case Maps. In Transactions on aspect-oriented software development III (pp. 105–143). Springer, Berlin, Heidelberg.

[15] ObjectWeb Consortium. (2005). The JAC Project. Retrieved September 18, 2018, from http://jac.ow2.org/

[16] Safonov, V., Gratchev, M., Grigoryev, D., & Maslennikov, A. (2006). Aspect .NET—aspect-oriented toolkit for Microsoft .NET based on Phoenix and Whidbey. .NET Technologies.

[17] Sommerville, I. (2006). Aspect-oriented Software Development. In Software Engineering (8th ed.).

[18] Sommerville, I. (2011a). Aspect-oriented software engineering. In Software Engineering (9th ed., pp. 565–590).

[19] Sommerville, I. (2011b). Software Engineering (9th ed.). Pearson Education.

[20] Suganantham, S., Babu, C., & Raju, M. (2017). A Quantitative Evaluation of Change Impact Reachability and Complexity Across Versions of Aspect Oriented Software. International Arab Journal of Information Technology, 14(1), 42–52. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85007575619&partnerID=40&md5=77c66e8467e120c4adda2ccfe89c2989

[21] Suvée, D., Vanderperren, W., & Jonckers, V. (2003). JAsCo : an Aspect-Oriented approach tailored for Component Based Software Development. In Proceedings of the 2nd International Conference on Aspect-Oriented Software Development (pp. 21–29). ACM.

[22] Team, A. (2018). The Evolution of Aspect-Oriented Software Development. Retrieved July 18, 2018, from http://aosd.net/the-evolution-of-aspect-oriented-software-development-aosd/

[23] Xerox Corporation. (n.d.). The Home of AspectC++. Retrieved July 23, 2018, from http://www.aspectc.org/

[24] XWeaver. (n.d.). The XWeaver Project. Retrieved September 18, 2018, from https://www.pnp-software.com/XWeaver/

BIOGRAPHIES

Ogbonna James Chinyere obtained his B.Tech. and M.Sc. degrees in Computer Science from Federal University of Technology Owerri (FUTO) in 2009 and 2016, respectively. He is an academic staff of Computer Science Dept., Clifford University Owerrinta, Abia State. His research interests include Mobile Computing, Information Security, Database Management Systems, and Software Engineering. He is a member of Nigeria Computer Society (NCS).

Nwokoma Francisca Onyinyechi obtained a National Diploma in Computer Science from Abia State Polytechnic Aba in 2005; B.Tech. and M.Sc. in Computer Science from Federal University of Technology Owerri in 2010 and 2016 respectively. She is an academic Staff of Computer Science Depts., FUTO. Her research interest includes Data Communication and Networking, Software Engineering, and Human Computer Interaction. She is a member of IEEE.

Nwala Kenneth Tochukwu obtained a Bachelor of Science (B.Sc.) and a Master of Science (M.Sc.) in Computer Science from Babcock University, Nigeria in 2011 and 2016 respectively. He is an academic staff of Computer Science Dept., Clifford University Owerrinta, Abia State.. His research areas include Human Computer Interaction, Networking, and Telecommunications. He is a member of Nigeria Computer Society (NCS).

Nwandu Ikenna Caesar obtained a Bachelor of Science (B.Sc.) from University of Benin, Nigeria and a Master of Science (M.Sc.) from Federal University of Technology Owerri, Nigeria all in Computer Science. He is currently working on his Ph.D at Federal University of Technology Owerri, Nigeria. His research areas include Software Engineering Operations Research and Data Communication.