Object + Component + Aspect Oriented Software Development: All are interrelated but have Different Development and Practitioner View

Nasib Singh Gill and Pradeep Tomar
Department of Computer Science & Applications
M. D. University, Rohtak-124 001, Haryana, INDIA
nsgill_2k4@yahoo.com, parry.tomar@gmail.com

EXTENDED ABSTRACT:
In the era of software development and software engineering, many companies have started to introduce new technologies (Object to Component, Component to Aspect) into their software development environment. The object oriented, component oriented and aspect oriented approach facilitates and encourages programming practice that increases reusability and reduce testing & maintainability in the code. This paper presents the overview and comparison of object, component and aspect. Object-oriented software development (OOSD) are popular concepts in today's software development environment. Object oriented software development focuses on objects as the primary agents involved in a computation; each class of data and related operations are collected into a single system entity. Object-oriented software development requires a different approach from more traditional functional decomposition and data flow development methods. Component-based software development (CBSD) is an emerging discipline that promises to take software engineering into a new era. Building on the achievements of object-oriented software construction, CBSD aims to deliver software engineering from a cottage industry into an industrial age for Information Technology, wherein software can be assembled from components, in the manner that hardware systems are currently constructed from kits of parts. Current middleware for component-based software development (CBSD) based on the Enterprise Java Beans (EJB) or CORBA Component Model provide good separation of concerns between the business logic (implemented by the components) and the technical infrastructure needed to run the business logic (implemented by the container). Without the dedicated support by the component middleware the implementation, respectively the invocation and orchestration of middleware services, would be scattered around and tangled with the business logic. Component middleware modularizes this crosscutting. Aspect-oriented programming (AOP) is used to build secure software. Secure software could be developed with the crosscutting nature of AOP as it bears the potential to capture security properties by allowing a better Separation of Concerns (SoC). In aspect oriented software development (AOSD) an aspect is a common feature that's typically scattered across methods, classes, object hierarchies, or even entire object models. It is behavior that looks and smells like it should have structure, but you can't find a way to express this structure in code with traditional object-oriented techniques. For example, metrics is one common aspect. To generate useful logs from your application, you have to sprinkle informative messages throughout your code. However, metrics is something that your class or object model really shouldn't be concerned about. After all, metrics is irrelevant to your actual application: it doesn't represent a customer or an account, and it doesn't realize a business rule. It's simply orthogonal. In AOP, a feature like metrics is called a crosscutting concern, as it's a behavior that "cuts" across multiple points in your object models, yet is distinctly different. As a development methodology, AOP recommends that you abstract and encapsulate crosscutting concerns. So our main aim is to present these three different approach with the experimental evaluation, comparison and analysis which help in providing accurate information, guidelines to manager and developer to make decisions, which one is best for different type software during software development.

CONCLUSION:
The goal of present paper is to study the three different approaches OOSD, CBSD and AOSD. The paper concludes that if software engineers and researchers seek to develop a good repositories of object, component and aspect than we can easily find out which is the best approach to develop a good software with low cost and within time. Software will likely need to be tested for each environment so that developers and users can better predict their expected behavior and performance. By using the above approach for development we can easily improve the testing and maintenance of software with the analysis of development and practitioner view.

REFERENCES: