Abstract

Object-oriented programming (OOP) is a dominant programming methodology for software development. OOP manages the complexity of software by breaking the functionality of the software into a set of objects. It efficiently handles the core concerns of software. However, it is not sufficient for handling crosscutting concerns such as logging, testing, authorization, security, etc. Handling of crosscutting concerns result in code that is intertwined with the code handling the actual functionality of the software, resulting in tangling of code. The tangled code is difficult to understand, maintain and modify.

Aspect-Oriented programming (AOP) is an emerging software development paradigm that aims at the separation of crosscutting concerns of the software from the core concerns. It helps in identification and modularization of the crosscutting concerns. With the use of AOP, a few aspects (modularizing units) are implemented that eases the tangling of code. Logging, a crosscutting concern, is a systemic method for code validation and debugging at the operating system level as well as at the application level.

During testing of software, there is a need to observe the internal execution details of the software. Provisions are required to be made in the software to facilitate observation of the internal execution details. Observability mechanisms are provisions in the software that facilitate observation of internal and external behavior of the software, to the required degree of detail.

Several techniques exist that facilitate observation of internal execution details of the software. Logging of software is one of the techniques used for observing the internal behavior of software.

During testing of software, the internal execution details required at unit, integration and system levels are different. Goel et al. [9] observe the internal execution details at unit, integration and system levels in object-oriented software using source-code instrumentation, which is intrusive in nature.

We focus on incorporating observability in object-oriented software at the programming language level, using AOP, required during the testing of software.

We present an *aspect-based testing technique* that uses AOP to facilitate observing of internal details of execution at different levels of abstraction: unit, integration and system levels, during testing of object-oriented software. Several techniques exist that use AOP to improve testing of object-oriented software. But none of them address the issue of observing internal execution details at different levels of abstraction during testing. Our technique adapts logging aspect and an observability mechanism, to suit the testing needs of object-oriented software. Our technique defines the structure of logging aspect that facilitates observation of internal execution details at unit, integration and system levels. The logging aspect defined by our technique is generic in nature. The Log File generated using our technique is useful for future analysis. Test coverage reports are generated at different levels like, method, class, inheritance and dynamic binding levels. The logging aspect defined using this technique, can be later, included in the standard library of the Aspect oriented software.