Aspect-Oriented Software for Testability

Mutum Zico Meetei
Department of Mathematics
Jazan University
Jazan, Saudi Arabia

Abstract— AOP is recently popular for effective technique in modularizing the crosscutting concerns such as exception handling, fault tolerance, error handling and reusability. Modularizing crosscutting concerns has a great impact on testability of software. Testability of software is the degree to facilitate testing in a given test context and ease revealing of faults. Controllability and observability are the important measures of testing non-functional requirements of software. To test software requires controlling the input and observing the output. Controllability provides a concept of probability to handle the software’s input (the internal state) while observability is to observe the output for certain input. This paper presents an overview of the use of aspect-oriented programming (AOP) for facilitating controllability to ease testability of object-oriented software and simulation of well-mixed biochemical systems.

Keywords—Controllability; Observability; Crosscutting concerns; Simulation; AOP; AspectJ;

I. INTRODUCTION

The evaluation of new algorithm is impeded by their limited flexibility, observability and controllability. To ease these limitations, there is required to simulate new algorithms and techniques, especially during initial phase of development. Controllability is the ease of producing a specified output from a specified input [1]. According to Bret Pettichord [2], control is the ability to apply inputs to the software under test or place it in specified states. Controllability is concern with ease to manipulate software in terms of its inputs and, consequently placing the desired state. Inputs would be parameters to class, methods or functions. These inputs come from the users interface. Software is designed so as to handle errors that arise from the software environment and within the software itself. However, some of the error codes are difficult to test, because state need for the execution of error-handling code is difficult to achieve.

Some of the control conditions that are difficult to test are special conditions, exception conditions and error conditions. These conditions are difficult to test from the users interface commands. The difficult to test conditions are arising from the software environment and software itself.

Aspect Oriented Programming (AOP) is a new programming paradigm that implements to Object-Oriented programming. AOP has provided a new mechanism for encapsulating crosscutting concern in a software system, as a module unit called aspect. It is used as an efficient technique for modularizing crosscutting concerns such as logging, exception handling, profiling, synchronization and tracing [3]. Generally, an aspect-oriented program consists of aspects and their base classes (or components) that can be woven into an executable whole [4]. Modularizing crosscutting concerns is done by the weaving process using aspect. This provides a great impact on testability of software, since using aspects overcome tangling and scattering of concerns codes.

This paper focuses on implementation of AOP for the increase of testability in object-oriented software and simulation of well-mixed biochemical systems. The use of AOP is discussed for the control conditions, such as; error handling, exception handling and, fault tolerance. An aspect-base testing tool is also used for simulation process. AOP provides self-checking aspects that helps testability of software. While using AOP, the code related to exception detection and handling is able to reduce considerably. AOP helps in reducing the complexity created by fault tolerance. This is possible by separating and modularizing the extra, crosscutting concerns from the true functionality. Aspect provides advantage over simulation by generating code automatically or semi-automatically, and easy to modify the design models.

The rest of the paper is organized as follows: Section 2 presents the background on testability of software; Section 3 presents an overview of AOP; Section 4 describes the implementation of AOP over testability of object-oriented software and simulations; Section 5 states the conclusion.

II. SOFTWARE TESTABILITY

In software engineering, it is important to acquire knowledge of software testability. Software which is not easy to test would undergo several interactions and test redesign. Software testability is defined as an indication of effort needed to test a system. According to IEEE standard, the term of “testability” refers to “the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met; and the degree to which a requirement is stated in terms that permit the establishment of test criteria and performance of tests to determine whether those criteria have been met.”

Controllability and observability are important measures of testability of software. Controllability is the ease of producing a specified output from a specified input. Binder [5] defined controllability as a probability that users are able to control component’s inputs (and internal state). To test a
component, there is required to control its input and internal state. If we can’t control the input, we wouldn’t be sure what caused a given output. According to Kansomkeat [6], software testers can increase testability in several ways:

- gather information from components without source code
- increase observability to monitor outputs
- increase controllability to support inputs
- choose test criteria and generate test cases for component testing based on the criteria

Testers need to realize that testability is required so as to understand software design. Testability is a design issue and need to be address with the design of the rest of the system.

Goal et al. [12] provides controllability mechanism, which facilitates attainment of difficult-to-achieve states during testing. The authors describe the requirements for controllability mechanism, such as (1) identification of control state, (2) creating of control environment, and (3) insertion of control commands in source code. The control commands are externally activated and deactivated.

Observability measures are provisions in the software that facilitate observation of internal and external behavior of the software, to the required degree of detail. It is the ease of determining if specified inputs affect specified outputs. In [3], we use aspect-based testing approach that adapts logging aspect for observing the internal execution details of object-oriented software. The aspect encapsulates method-like code fragment and weave a new functionality into the existing software system. During testing, the woven log aspect facilitates the execution details of the software at unit, integration and system level. We observe that the advantage of using AspectJ [7] is that with a few aspects, we are able to provide details of access to the method calls and method execution including the individual arguments passed to the method.

III. ASPECT ORIENTED PROGRAMMING

Aspect Oriented Programming (AOP) is a new programming paradigm for modularizing concerns that cross-cut the basic functionality of a program. Typical crosscutting concerns include logging, synchronization, context-sensitive error handling, performance optimization and design pattern. AOP provides a language mechanism that explicitly captures crosscutting concerns, which makes it possible to crosscut other concerns in a modular way. AOP introduces a new modular unit called aspect, for encapsulating crosscutting concerns.

AspectJ is an aspect oriented extension to Java. In addition to the regular classes, it introduces some new constructs such as introduction, join point, pointcut, advice and aspect, for better handling of crosscutting concerns. AspectJ provides constructs such as join point, pointcut, advice and aspect are described as follows:

- Join point is a well defined location in the code at which a concern crosscuts the application. Using join point, a programmer can separate crosscutting concern from the objects. AspectJ does not expose to some join points such as for loops and if statement [8]. However in [14], we are able to trace execution details of the loop join points such as if statement and for loop.
- Pointcut is a set of join point that optionally exposes some of the value in the execution of join points.
- Advice is a method like mechanism used to declare the code that should execute at each of the join point in a pointcut. The advice is applied through a process called weaving to a matching join point in the method body. Weaving is a process which allows both aspect and non-aspect code to run together.
- Aspect is a modular unit of crosscutting implementation. Aspect is similar to that of a class in object oriented programming. Its declaration includes pointcut, advice, intertype declaration or interface declaration into a class.

IV. AOP FOR IMPROVING TESTABILITY

Software evolution has long been recognized as an inevitable phenomenon. From the initial stage of development process, to managing, fixing bugs and testing functions, to whole-scale updates like restructuring of the entire code base, the programmer needs to cope with changes that are necessary across software generations. In simple cases such as replacing a function call with a conditional control structure, the modification replaces the original call by an if-then-else statement where either of two functions is chosen depending on the runtime value of a predicate.

Yet the rewriting process may become cumbersome in large programs with a complex adaptive plan. Therefore, this issue of adding new functionality to existing code in a modular way has been recently identified by the object-oriented programming community as one of the important motivations for Aspect-Oriented Programming.

Aspect-Oriented Programming provides a new mechanism for capturing and modularizing crosscutting concerns in software system. The use of aspect helps the crosscutting codes easy to develop, maintain and reuse. This introduces a new concept to the conventional technique and facilitates the testability of software. AOP is an evolutionary step that improves the implementation’s comprehensibility and simplifies incorporating new requirements as well as changes to existing ones [9].

Mao [10] uses aspects to improve components capability of self checking (or self-testing), and also to provide the information on execution trace to facilitate test case selection. AOP is use to enhance the component’s testability by embedding self-checking aspects to check invariants. Aspect is used to crosscut the maintenance concerns and facilitates testing in later stage. The author encountered some advantages of self checking aspects.

Tsang et al. [13] use AspectJ and compare with the use of Object-Oriented real time domain specific language in terms of its effect on system properties related to testability. The authors construct two systems that implemented a real-time sentient traffic simulation, and identify concerns that were
not well encapsulated in real-time Java but possible by using aspects. They also study the use of aspects in improving system properties. Munnelly et al. [11] uses Aspect-Oriented software development modularisation capabilities for improving software quality.

Aspect provides new functionalities to the existing system which can’t be added without invasive changes to the whole system using conventional techniques. Aspect is used to observe the internal details of execution during testing [3]. It is defined without affecting the source code during software development and is woven with the source code during compilation. The following subsections discuss the use of the new functionality of aspects for the following purpose:

1) Exception Handling,
2) Error Handling,
3) Fault Tolerance,
4) Stochastic simulation.

A. Exception Handling

Exceptional handling is a technique to recover or to ignore the system’s behavior during an unsuccessful completion of an operation. It is responsible for performing the recovery actions necessary to bring the software back to the normal state. Using AOP, the exception handling can be defined in either a method or an advice. The around and after advice have the ability to handle exceptions thrown by the methods. An example of AspectJ code for handling exception is shown in Fig. 1.

```java
public abstract pointcut checkException () ;
Object around ():checkException () {
    Object c = null;
    try { c = proceed () ;
    } catch ( Exception e ) {
        String st =
        getmessage ( thisJoinPointStaticPart.getID () )
    } . . . .
}
```

Figure 1. Example of Exception Handling using aspect.

Lippert et al. [16] use an existing framework JWAM reengineered with AspectJ. They use aspect designs that target the contracts and handling of the five most frequently caught exceptions. They found that using of aspects brought several benefits, less interference in the program texts and reduced the code related to exception detection and handling by a factor of four.

Hoffman et al. [15] study refactoring of exception handling concerns for Java applications by using explicit join points. They also observe that reflection could be avoided if AspectJ was enhanced to allow aspects access to the type variables of the join points that are advised.

Ji-Pde et al. [17] discuss the several exception fault problem of AspectJ. They present several examples to analyze the impact of exception fault on program control flow. AspectJ introduce exception softening mechanism which helps to solve problem with Java exception handling.

In [18], the authors compare the Aspect-Oriented and Object-Oriented implementation of three different applications-CheckStyle, Java PetStore and Java Enterprise Edition. The authors use a set of metrics specifically targeting the exception handling concern. AOP promotes implementation of large amount of reusable exception handling while resulting less concern scattering.

B. Error Handling

An error is anything that can cause a system to fail. It is a situation that arises because of unexpected change in resource availability or performance assumptions. The mechanism of dealing error rise in software is to treat them as exceptions which can be thrown by a few codes. Using AOP, the error handling codes are woven with the aspect code and provide a modularising approach. The after advice is often used for handling error codes (catch block) in the aspect.

Filo et al. [19] discuss various classifications of error handling codes and found out have more influence on aspectization. They conclude that use of AOP provides beneficial by allowing separating the error handling codes from the base code. However for a too complex error handling codes, AOP merely adds insult to injury and worsens the overall quality of the system.

Kashif et al. [20] present that aspect oriented design pattern brings additional benefits like the localization of error handling code in terms of definitions, initializations and implementation. Aspect is responsible for all the calling contexts of a safety critical function. Also the error handling codes can be plugged separately and helps to improve reusability.

Pu et al. [21] use aspect orientation to understanding COBOL code by defining the context aspect and error handling aspect. They conclude that Context aspect and error handling aspect are greatly helpful for the comprehension and reuse of legacy COBOL code.

C. Fault Tolerance

Error detection is the basic step in deploying any fault tolerance strategy. To handle, errors have been deployed as a key mechanism in implementing software fault tolerance. The fault tolerance measures using exception handling should produce software with separate error handling code and normal code, and complexity on the system should be minimized. The execution strategy for fault tolerance using AOP is show in Fig. 2.
programs without imposing prohibitively high overheads.

Szentivanyi et al. [22] present successful merging of middleware fault tolerance with AOP by weaving fault tolerance aspects in the application. Aspects are used for improving performance of a fault-tolerant application on top of a standard middleware. An existing FT-CORBA platform is used for the approach.

Alexandersson et al. [23] study memory and execution time overhead associated with implementing fault tolerance using AOP. They implement a fault-tolerant version using AspectC++ and found that memory and execution time overhead is much higher than those obtained in standard C. AOP can be used for implementing fault tolerance in control programs without imposing prohibitively high overheads.

Hashim et al. [24] present a technique of interface fault insertion using AspectJ for Java component-base application. In this technique aspect codes act as wrappers around interface services and perform operation such as raising exceptions. They show how faults can be triggered and easily integrated with Junit testing framework.

Manson et al. [25] present a method for expressing hooks into a virtual machine to provide fault determination and fault tolerance for user code. AOP is used for handling a variety of run-time as dynamic aspects. They propose a system RESCUE-a programming language approach to determine the presence of faults, to express faults as variety of runtime constraints.

D. Stochastic Simulation

Simulation is a process of describing a real system and using this model for experimentation, with the goal of understanding the system behaviour or to explore alternative strategies for its operation. Using simulations we can improve our understanding of systems behaviour and the effects of interactions among components. In complex and larger scale systems, require simulations with a large number of simulated entities and a high level of detail (such as cloud computing platforms or peer-to-peer networks).

Consequently, scalability in terms of memory is an additional requirement for current simulation frameworks. At the same time, the bundling of computing power in a networked environment using cloud computing or cluster computing, together with the appearing limitations of single machine execution, drives the demand for simulation in a distributed environment. In bio-chemical system, spatial stochastic simulation is an extremely computationally intensive task. This is due to large number of molecules, which, along with the refinement of the spatial domain, results in a large number of diffusive transfers between sub-volumes.

In case of chemical master equation [27], there involve $N$ molecular species $\{S_1, S_2, \ldots, S_\nu\}$ represented by the state vector $X(t) = \{X_1(t), \ldots, X_\nu(t)\}$, where $X(t)$ is the number of molecules of species $S_i$ at time $t$. There are $M$ reaction channels, labelled $\{R_1, R_2, \ldots, R_m\}$ in the system. It is assumed that the system is well mixed and in thermal equilibrium. The dynamic of reaction channel $R_j$ is characterised by the propensity function $a_j$ and state change vector,$$v_j = \{v_{j_1}, v_{j_2}, \ldots, v_{j_m}\}.$$ denotes the probability that, given $X(t) = x$, one $R_j$ reaction will occur in the next infinitesimal time interval $[t, t+dt]$, and $v_j$ with the change in $X_j$ induced by one $R_j$ reaction. The system states are updated by simulation then proceeds to the time of the next reaction. Since the stochastic simulation algorithm simulated all reaction events in the system, there exits tangling of codes in the software. Using of an aspect can ease such tangling of code and can provide efficient simulation of reaction time event.

The Fig. 3 shows sequence diagram of performance concerns of a system consists of three tiers: reaction chain, $R_1$, and $R_2$. It shows the sequential steps of “time” scenario. The AspectJ pointcuts are used with and before () and after () advice to get the responds time of the scenario.

Aspect-based mechanism can fill a gap in the sequence diagram by inserting time constraints to the section of the AspectJ code. The use of aspect can facilitate construction of interoperating collections of scientific tools and models[28]. The aspect-based method is used for creating tool federations for scientific simulation tools written in Java. Aspects are effectively used for input/output statements in the tools and models of a federation.

The use of aspects incorporate observability efficiently during scientific simulations. Fig. 4 is a snap shot of the aspect-based testing tool[3] showing the internal execution details of the Moti Carlo program to compute value of $\pi$. It is found that aspects can be easily, integrated and help to observe the behaviour of a system.

In [26], the authors propose a simulation-based design-level performance analysis method based on aspect-oriented programming. The performance models are separated from design models, and then inject performance requirements into skeleton code generated from design models. The method has the advantages that (1) code for the simulation is generated automatically, and (2) it is relatively easy to
modify design models or performance models independently when they are changed.

In many simulation systems, implementation of number of concerns happen such as event scheduling, event handling, keeping a track of simulation’s state and creating random number generators. These concerns crosscut over multiple modules in the system, increase complexity and reduce the maintainability. The use of AOP refector these complexities in simulation system by modularizing the concerns that crosscut other modules.

V. CONCLUSION

The difficult-to-test conditions that arise during testing, such as exception conditions, error conditions and fault tolerance are discussed. AOP provides a new modularizing approach that wove with error handling codes with aspect codes and reduces the additional logical complexity. Aspect-oriented software is widely used in the area of fault tolerance, error handling and reusability. This work also focuses on the use of aspect-based testing tool in scientific simulations and its advantages over the difficult-to-test conditions in software. AOP can be efficiently applied to the simulation-based analysis in design phase. The use of aspect eases the complexity arise in simulation process by modularizing crosscutting codes. As a future work, we would focus on the role of AOP over stochastic simulation of cellular process using reaction-diffusion master equation.

REFERENCES


Figure 4. Internal execution details of the Monti Carlo program to compute π.