OpenAOP: Analysing Cross-Cutting Concerns in Open Source Software

Tony Kong
University of British Columbia
Vancouver, BC
y2k0b@ugrad.cs.ubc.ca

Raymond Situ
University of British Columbia
Vancouver, BC
r7p0b@ugrad.cs.ubc.ca

Kevin Wong
University of British Columbia
Vancouver, BC
b2j0b@ugrad.cs.ubc.ca

Benjamin Hwang
University of British Columbia
Vancouver, BC
r6o0b@ugrad.cs.ubc.ca

James Yoo
University of British Columbia
Vancouver, BC
l4k0b@ugrad.cs.ubc.ca

ABSTRACT

Aspect-oriented Programming is a programming paradigm for improving the modularity of a system via the *Aspect*, which encapsulates the cross-cutting in a system and enforces a separation of concerns between logic which would otherwise be tangled without it. In *Open AOP*, we analyze an open-source Java software system and measure the extent of tangling in the system prior to refactoring using AspectJ. We then analyze the evolved system, with respect to tangling and cohesion.

KEYWORDS

Aspect-oriented Programming, Cross-cutting concern, Joinpoint, Pointcut, Advice, Tangling, Cohesion, AspectJ

ACM Reference Format:
Tony Kong, Raymond Situ, Kevin Wong, Benjamin Hwang, and James Yoo. 2018. OpenAOP: Analysing Cross-Cutting Concerns in Open Source Software. In *Proceedings of Intro. PL (CPSC 311).* ACM, New York, NY, USA, Article 4, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 MOTIVATION

Modern software systems are extremely complex and often require large amounts of logic which may be unrelated to their core concerns. Classical examples of this include logging and exception handling. *Tangling* occurs when these cross-cutting concerns are scattered throughout modules. This tangling lowers the cohesion of modules. As a result, the system becomes more difficult to reason about and evolve.

2 METHODOLOGY

(1) Select a medium-sized [1] open-source project and extract the following metrics.
   - Measure how much of a the project's concerns are cross-cutting [2]
   - Cohesion
(2) Refactor the project via encapsulating cross-cutting concerns with Aspects
(3) We revisit the metrics we extracted in (1)

[1] $\leq$ 6000 SLOC
[2] 1 Murphy et al: Identifying, Assigning, and Quantifying Crosscutting Concerns

CPSC 311, 2018W1, UBC Vancouver
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 PROJECT TYPE

We understand that our project type is rather unique in the sense that it is a blend of two of the classical types presented in CPSC 311, namely

(1) **Type 3**: Applying an existing, interesting analysis (static or dynamic) to a substantial program and illustrating the value of the analysis.
(2) **Type 4**: Creating a substantial program in a potentially useful but esoteric (i.e., "weird" and ideally cutting-edge) language that you have not already studied. Your program should illustrate the particular value of the language.

We would fulfill (1) by using the methodology of *Murphy et al* to measure how much of a program's concerns are indeed cross-cutting - this would be a static analysis of the program we would chose to refactor using AspectJ. This leads us to how our project would fulfill (2). Reafactoring/creating a program with AspectJ would be a non-trivial task, especially as none of us have worked extensively with the language. We would illustrate the value of AOP by introducing Aspects in our program which enable the encapsulation of cross-cutting concerns.

4 HIGH LEVEL OBJECTIVES

Our primary goals for this project are as follows

(1) Refactor an open-source software system using AspectJ
(2) Compare and contrast the degree of crosscutting and cohesion present in the prior system to that of the evolved system

Successful completion of the objectives above would map to the 100% project completion milestone.

5 PROJECT MILESTONE OUTLINE

Although we have committed to the 100% completion goal, below we detail a high-level overview of what our project would look like at different levels of completion.

5.1 Open-source Project Selection

We would select an open-source project by the 90% completion deadline at the very latest. Notice that the 80% and the 90% completion goals do not require us to refactor an existing project with AspectJ, but instead pertain to getting familiar with Aspects and/or writing a *small* project with it (i.e. about the size of the previous CPSC 210 term project). That being said, we will take into account the following when selecting some possible refactoring candidates

(1) Size of system (SLOC), we want a moderately-sized system, no larger than 8000 lines of code.
(2) Documentation: we would ideally select a project with a large amount of readily available documentation. This would be *integral* in getting our team to understand the codebase as fast as possible

5.2 80% Completion

For this milestone, we expect each member to be comfortable with the following

(1) Identify a cross-cutting concern in the context of AOP
(2) Write an AspectJ aspect to address the concern above

This would be achieved by consulting the literature in our weekly meetings and each member "owning" a single facet of AspectJ and teaching others. Using the competencies above, we would perform a static analysis of a very small Java software system and generate a report on the cross-cutting concerns existent in it, and *how* AspectJ could be utilized to mitigate them.

5.3 90% Completion

This milestone assumes that all group members are comfortable with the competencies outlined in **5.2**, but takes it further by

(1) Writing a small-scale software system (e.g. small game, some tool) in Java
(2) Evolving said system by introducing aspects via AspectJ
(3) Comparing the evolved system with the prior system

The 90% completion project will fulfill the following

(1) Includes at least 4 cross-cutting concerns handled by AspectJ
(2) Should be demoable via computer
(3) Could be an implementation of a simple Java program, e.g. Space Invaders

5.4 100% Completion

This milestone is similar to the 90% milestone, but applied to an open-source software system in a larger scale. We expect this to be significantly more challenging than the 90% target due to the fact that not only do we have to learn AspectJ, but we also have to become familiar enough with a foreign system to refactor and evolve the system appropriately. The 100% completion project will fulfill the following requirements

(1) Contains at least 1 cross cutting concern that is not logging or exception handling
(2) Should be a refactoring performed on a system which is at most 5000 lines of code
(3) Contains few dependencies for easy set up on development machines

5.5 Poster

This is the part of the project which other CPSC311 classmates will have the most exposure to. Therefore, we plan to take a high-level approach to describing AspectJ. We will detail the primary language features of AspectJ, such as the *Aspect*, *Pointcut*, *Joinpoint*, and the different types of *Advice* made possible. In addition to the features above, we will also enumerate some common cross-cutting concerns which are targeted by AspectJ, and provide a simple demo of AspectJ in action using an IDE. Below are some specific details regarding our poster and its presentation

(1) Code examples which directly show how the Aspect enables developers to encapsulate cross-cutting concerns.
(2) We will have a computer with an IDE which allows people to see AspectJ in action.

6 SELECT REFERENCES

This section details a few sources which we will utilize as a "jumping-point" for our project. Note that these are not the complete set of sources which we will consult, but are a subset which are especially appropriate for getting familiar with AspectJ and AOP.

(1) Eclipse Foundation: *Getting Started with AspectJ*

This is a webpage which contains some key information about the AspectJ language, containing information about Pointcuts, Joinpoints, and the different types of Advice which are available as part of the language. It is especially appropriate as a primary language specification due to the fact that the Eclipse foundation is strongly associated with AspectJ.

(1) Kiselev, Ivan: *Aspect-Oriented Programming with AspectJ*

This is a book which is accessible to people with a general knowledge of the Java programming language. It eases readers into AspectJ with concrete code examples which are small enough to be written into an IDE at the same time.

(1) Murphy et al: *Identifying, Assigning, and Quantifying Cross-cutting Concerns*

This paper will deal with researching how we can measure the amount of cross-cutting which is present in a system. This is important as we will require a metric in order to measure the efficacy of AspectJ in mitigating cross-cutting.

7 CITATIONS

(1) Kiczales et al: *Aspect-Oriented Programming*, from the Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997
(2) Murphy et al: *Identifying, Assigning, and Quantifying Cross-cutting Concerns*, in ACoM '07 Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques
(3) Baeldung, Intro to AspectJ: *https://www.baeldung.com/aspectj*, from Baeldung, Blog.
(4) Eclipse Foundation: *Getting Started with AspectJ*, from the Eclipse Foundation, Website
(5) Kiselev, Ivan: *Aspect-Oriented Programming with AspectJ*, published by *Sams Publishing*

(6) o7planning, Java Aspect Oriented Programming Tutorial: *goo.gl/BDVAEk*, from o7Planning, Website
(7) Eclipse Foundation: *AspectJ Language Semantics*: *goo.gl/iCSLdu*, from the Eclipse Foundation, Website.