Science of Computer Programming 119 (2016) 31–50

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Avoiding code pitfalls in Aspect-Oriented Programming

Adriano Santos [a,*], Péricles Alves [a], Eduardo Figueiredo [a], Fabiano Ferrari [b]

[a] Computer Science Department, Federal University of Minas Gerais, Belo Horizonte, Brazil
[b] Computing Department, Federal University of São Carlos, São Carlos, Brazil

ARTICLE INFO

Article history:
Received 15 March 2015
Received in revised form 1 December 2015
Accepted 2 December 2015
Available online 9 January 2016

Keywords:
Aspect-Oriented Programming
Mistakes
Code pitfalls
Empirical study

ABSTRACT

Aspect-Oriented Programming (AOP) is a maturing technique that requires a good comprehension of which types of mistakes programmers make during the development of applications. Unfortunately, the lack of such knowledge seems to represent one of the reasons for the cautious adoption of AOP in real software development projects. This paper reports on the results of a series of experiments whose main goal is to analyze and catalogue code pitfalls that are likely to lead programmers to make mistakes in AOP refactoring. Each experiment consists of a task that requires the aspectization of a crosscutting concern in one object-oriented application. Eight rounds of the experiment provided us with data of 98 AOP implementations from four crosscutting concerns in four different applications. Each participant of the experiment produced one implementation. Based on the analysis of these implementations, we (i) document six categories of recurring mistakes made by programmers, (ii) correlate these mistakes with the programmer expertise in object-oriented programming, years of software development, and pair programming, and (iii) derive a catalogue of code pitfalls which are likely to lead programmers to make the documented mistakes. We apply significance tests in order to statistically evaluate our results. We also present a prototype tool to warn programmers of the code pitfalls during refactoring activities.

1. Introduction

Aspect-Oriented Programming (AOP) [25] is a software development technique that aims to improve software modularity through the separation of crosscutting concerns into modular units called aspects. A concern is any consideration that can impact on the design and maintenance of program modules [38]. It is well known that novice programmers need special guidance while learning how to program in a new programming language [41,42,44]. Therefore, to be widely adopted in practice, we need a good comprehension of the kinds of mistakes made by AOP programmers when learning this development technique and the situations that lead to these mistakes.

The IEEE Standard Glossary for Software Engineering [22] defines a *mistake* as "*a human action that produces an incorrect result*". In the context of this paper, a refactoring mistake occurs when a developer performs an inconsistent code refactoring, which may have as a consequence, the introduction of a fault into the application code. A *fault*, on the other hand, consists in an incorrect step, process, or data definition in a computer program [22]. In other words, a fault occurs when there is

* Corresponding author.
E-mail addresses: adrianolages@dcc.ufmg.br (A. Santos), periclesrafael@dcc.ufmg.br (P. Alves), figueiredo@dcc.ufmg.br (E. Figueiredo), fabiano@dc.ufscar.br (F. Ferrari).

http://dx.doi.org/10.1016/j.scico.2015.12.003
0167-6423/

a difference between the actually implemented software product and the product that is assumed to be correct. A mistake may lead to one or more faults being inserted into the software, although it may not necessarily do so.

Previous research [3,6,11,14] has investigated mistakes that are likely to be made by AOP programmers either to build systems from scratch or to refactor existing ones. Other studies have identified aspect-oriented code smells [15,28,31,35] and fault-prone scenarios [46]. However, these studies target at complex software systems developed by experienced programmers. In such complex scenarios, it is difficult to reveal the factors that hinder the learning of basic AOP concepts, such as pointcut and advice, by novice programmers. This situation gives us a lack of understanding of the scenarios that could lead novice programmers to make mistakes while refactoring existing code to AOP. We call these scenarios *code pitfalls*. In turn, code pitfalls may represent one of the reasons for the cautious adoption of AOP in real software development projects [26,33,36].

This paper extends our previous study [4] that investigated a preliminary set of recurring mistakes made by novice AOP programmers. Previously, we derived mistakes by running six rounds of an experiment with 80 fresh AOP programmers [4]. This paper extends our previous work by reporting two additional rounds of the same experiment (resulting in eight rounds in total) with a different application and 25 additional participants. Therefore, this paper relies on an extended dataset of 98 refactored code samples of four applications (Section 2). With the new replications, we identified additional instances of mistakes and code pitfalls.

In this paper, we not only further investigate and confirm our previous findings, but also seek to better understand the categories of common mistakes made by AOP programmers (Section 3). An interesting result found in our study is that more experienced programmers in OOP made more mistakes than novice programmers in some categories, such as incorrect advice type and duplicated crosscutting code. In addition, we also perform further statistical analysis of the data in this follow up study, for instance, to understand how skills of programmers impact on the mistakes they make (Section 4). We rely on a full factorial experiment design with application of one-way Analysis of Variance (ANOVA) [23,45]. Based on the new statistical analysis, this paper extends the catalogue of code pitfalls (Section 5); i.e., situations that appear to lead programmers to make the identified categories of recurring mistakes. Code pitfalls were observed by inspecting the original source code focusing on code fragments that were incorrectly or inconsistently refactored to AOP.

To support the automatic detection of the documented code pitfalls, we developed a prototype tool called ConcernReCS (Section 6). ConcernReCS is an Eclipse plug-in that aims to warn AOP programmers of situations that may lead to refactoring mistakes. It is based on static analysis of Java code and builds up on knowledge of our experiment results. Section 7 discusses some limitations of our study and Section 8 summarizes related work. Section 9 concludes this paper and points out directions for future work.

2. Experimental setup

This section presents the configurations of the experimental study we conducted. Section 2.1 presents and discusses the research questions that we aim to answer in this paper. Section 2.2 describes the background of all participants. Section 2.3 introduces the design of the target applications and the basic characteristics of the refactored crosscutting concerns. Section 2.4 explains the experimental tasks.

2.1. Research questions and hypotheses

This study aims at investigating the types of mistakes made by students and junior professionals when using AOP. Additionally, we target at uncovering and documenting code pitfalls that lead programmers to make these mistakes. Based on these aims, we formulate the following three research questions.

R1. What kinds of mistakes do junior AO programmers often make while refactoring crosscutting concerns?
R2. What are the code pitfalls in the source code that lead to these mistakes?
R3. Which skills should a programmer have to make fewer mistakes in AOP?

To answer R1, we first identify and classify the recurring categories of mistakes made by AOP programmers. Then, we document error-prone situations as a catalogue of code pitfalls to address R2. Finally, we analyze whether and how the programmer skills impact on the observed mistakes to answer R3. In this case, the common sense is that the background of programmers impacts on the number of mistakes they make. Therefore, we decomposed the last research question (RQ3) in the set of hypotheses below. To validate our hypotheses, we rely on a full factorial experiment design, including all possible combinations between the levels of the experiment factors. Statistical significance was tested by the analysis of variance (ANOVA) of experimental data.

H1.0: Level of experience in OOP does not impact the number of mistakes in AOP.
H1.1: Level of experience in OOP impacts the number of mistakes in AOP.

H2.0: Work experience does not impact on the number of mistakes in AOP.
H2.1: Work experience impacts on the number of mistakes in AOP.