Computers & Education 70 (2014) 161–172

Contents lists available at ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

Beyond plagiarism: An active learning method to analyze causes behind code-similarity

Feng-Pu Yang [a,*], Hewijin Christine Jiau [a,b], Kuo-Feng Ssu [a,b]

[a] National Cheng Kung University, Taiwan
[b] University of Delaware, USA

ARTICLE INFO

Article history:
Received 11 June 2013
Received in revised form
8 August 2013
Accepted 9 August 2013

Keywords:
Cooperative/collaborative learning
Programming and programming languages
Teaching/learning strategies

ABSTRACT

Code-similarity is the actual indicator of plagiarism in the context of programming assignments. However, the experiences of practical software development have empirically confirmed the existences of other causes for code-similarity. Existing practices usually overemphasis the casual relationship between code-similarity and plagiarism, but ignore the importance to make students understand other causes that also contribute to code-similarity. This paper presents an active learning method to involve students and instructors collaboratively in finding causes of code-similarity occurred in programming assignments. The result shows that most causes occurred in programming assignments are positive. Students can learn the different causes of code-similarity with pros and cons during conducting the active learning method.

1. Introduction

Code-similarity is a frequently observed phenomenon in software. There is a variety of causes behind code-similarity cases, for example, similar expertise level is one cause of code-similarity. That is, novices usually produce codes with similar drawbacks, such as bad smells identified by Fowler, Beck, Brant, Opdyke, and Roberts (1999). On the other hand, experienced programmers usually generate similar codes for recurring problems to maintain the quality of codes, e.g., design patterns (Gamma, Helm, Johnson, & Vlissides, 1994). In addition to those coincidental code-similarity cases, some good practices, such as "software reuse", intentionally create similarity among codes (Krueger, 1992).

Many researchers have used code-similarity as an indicator in plagiarism detection (Ji, Woo, & Cho, 2007; Joy & Luck, 1999; Parker & Hamblen, 1989; Rosales et al., 2008). Though plagiarism is now treated as the most important cause of code-similarity in the context of programming assignments, other causes for code-similarity should not be ignored. Joy et al. remind that the symptoms of code-similarity cannot determine plagiarism cases without further manual confirmation (Joy & Luck, 1999). Since code-similarity is a common phenomenon in software development, code-similarity should be treated with more positive view in programming assignments other than the indicator of plagiarism.

In addition to plagiarism, students should learn the positive aspects of code-similarity. One example of positive aspects is software reuse (Krueger, 1992). Because of the increasing availability of both reusable resources, such as *open source software* (DiBona, Ockman, & Stone, 1999), and practical methods, such as *example-centric programming* (Brandt, Dontcheva, Weskamp, & Klemmer, 2010), most experienced programmers nowadays have recognized the importance of reuse in improving their productivities. However, students are not aware of the meaning of reuse. Some students even treat reuse as a harmful activity because it is one cause of code-similarity, which is the de facto indicator of plagiarism in the context of programming assignments. Although the symptom, i.e., code-similarity, of plagiarism might be very similar as reuse, the causes should be clarified to both instructors and students. Plagiarism is the process of creating similar software from existing software with the reasons of laziness or inability to do the programming assignment. Reuse is the process of creating new software from existing software, which is deliberatively selected from a set of candidates with respect to quality concerns (Gibson, 2009). In this example, the objective of instructors is to foster software reuse but avoid plagiarism.

* Corresponding author. National Cheng Kung University, No. 20, Ln. 863, Sec. 2, Yongcheng Rd., South Dist., Tainan City 702, Taiwan. Tel.: +886 910 828 323.
E-mail addresses: fengpuyang@gmail.com (F.-P. Yang), jiauhjc@mail.ncku.edu.tw (H.C. Jiau), ssu@ee.ncku.edu.tw (K.-F. Ssu).

0360-1315/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compedu.2013.08.005

Understanding causes behind code-similarity cases found in programming assignments is the prerequisite to foster positive aspects of code-similarity and to avoid negative ones. However, except for plagiarism, students and instructors are generally unaware of other causes of code-similarity. One reason resulted in this unawareness is lacking a method to integrate knowledge of instructors and the contextual information possessed by students. Instructors have the knowledge of *how to identify a code-similarity instance* and *what practices are good from programming perspective*. Students have the contextual information, such as *which classmates have been consulted*, *which part of their code has been reused by other classmates*. Both the knowledge and the contextual information are needed to successfully analyze the causes behind code-similarity cases. This paper presents an active learning method, which helps instructors and students collaboratively analyze the causes of code-similarity during programming assignments. The knowledge of instructors about identifying code-similarity instances are provided in terms of two tools, called CST and GVT, for students to semi-automatize the identification. Students can then focus on finding supported evidences based on their contextual information for those identified code-similarity cases. Finally, instructors will confirm the causes concluded by students, and explain the pros and cons of each type of code-similarity for students. This tool-augmented method has been empirically evaluated in a programming course at National Cheng Kung University of Taiwan. The result confirms the capability of this method in helping students learn a variety of causes for code-similarity beyond plagiarism. Most causes found in the empirical study are manually checked by instructors as positive, which also supports our hypothesis about the existences of other positive causes beyond plagiarism in programming assignments.

2. The cause investigation method of code-similarity

This work designed an active learning method to help students learn causes of code-similarity through four steps as follows:

- Step 1: Measure code-similarity relations among programs.
- Step 2: Group programs based on measured code-similarity.
- Step 3: Elicit and check causes of code-similarity groups.
- Step 4: Instructors review found causes and discuss their pros and cons with students.

To ease the burden of cause analysis, two tools, CST and GVT, were provided to students to assist Step 1 and Step 2, respectively.

- A code-similarity measurement tool (CST): Step 1 focuses on helping students to measure code-similarity of given programming assignments. Since manually analyzing code-similarity is impractical, a code-similarity measurement tool is needed. However, it is difficult for students to development a code-similarity measurement tool from scratch. To make Step 1 practical, a code-similarity measurement tool, CST, is designed and developed for students.
- A group visualization tool (GVT): Step 2 is for students to integrate code-similarity pairs found in Step 1 into code-similarity groups. We believe that group is a better granularity than pair in analyzing causes of code-similarity. The details of this granularity decision can be found in latter section named "Design decisions of the Learning Method." GVT provides instant and visual feedbacks of grouped results to students. Students can iteratively refine their grouping criteria based on the visualization provided by GVT.

2.1. Step 1: measure code-similarity relations among programs

As Fig. 1 depicts, programs delivered by students are the inputs in CST, and code-similarity relations are the outputs from CST. To ease the burden of investigating code-similarity, we implement CST with four code-similarity views. Table 1 lists these four code-similarity views: *Class Structure*, *Class Relation*, *Class Content* and *Logic Flow*. CST is implemented as an Eclipse plug-in, and it accepts inputs written in Java programming language. After inputting all assignment works into CST, four types of code-similarity relations are measured for each pair of assignment works. The output of CST is a text file consisted of involved students and the relation with a value of weight. Fig. 1 shows an example of code-similarity relations file. To reduce students' burdens of interpreting measured relations, all the four code-similarity

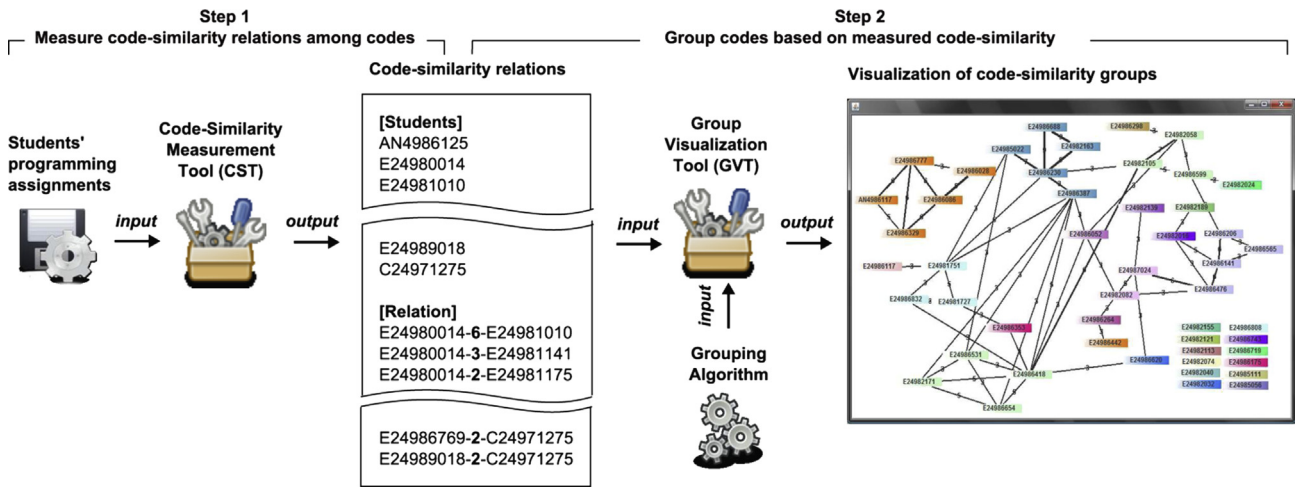

Fig. 1. Steps of cause investigation method with corresponding tools and materials.