Science of Computer Programming 89 (2014) 350–373

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

The links between human error diversity and software diversity: Implications for fault diversity seeking

Fuqun Huang [a,*], Bin Liu [a], You Song [a], Shreya Keyal [b]

[a] *Beihang University, Beijing, China*
[b] *University College London, UK*

A R T I C L E I N F O

Article history:
Received 15 October 2012
Received in revised form 27 February 2014
Accepted 14 March 2014
Available online 24 March 2014

Keywords:
Software diversity
Human error
N-version programming
Cognitive style
Personality trait

A B S T R A C T

Software diversity is known to improve fault tolerance in N-version software systems by independent development. As the leading cause of software faults, human error is considered an important factor in diversity seeking. However, there is little scientific research focusing on how to seek software fault diversity based on human error mechanisms. A literature review was conducted to extract factors that may differentiate people with respect to human error-proneness. In addition, we constructed a conceptual model of the links between human error diversity and software diversity. An experiment was designed to validate the hypotheses, in the form of a programming contest, accompanied by a survey of cognitive styles and personality traits. One hundred ninety-two programs were submitted for the identical problem, and 70 surveys were collected. Code inspection revealed 23 faults, of which 10 were coincident faults. The results show that personality traits seems not effective predictors for fault diversity as a whole model, whereas cognitive styles and program measurements moderately account for the variation of fault density. The results also show causal relations between performance levels and coincident faults: coincident faults are unlikely to occur at skill-based performance level; the coincident faults introduced in rule-based performances show a high probability of occurrence, and the coincident faults introduced in knowledge-based performances are shaped by the content and formats of the task itself. Based on these results, we have proposed a model to seek software diversity and prevent coincident faults.

1. Introduction

Software diversity is used as a manner in which to achieve fault tolerance in multiple-version software systems. Independent development is proposed as a way to achieve diversity. People intuitively hope that different versions will fail independently if the versions are developed "independently". The general strategy is to isolate development teams, who are free to choose different programming languages, different algorithms, and different development tools and processes.

Theoretical models have been developed to analyse the effectiveness of diversity on software reliability. The conceptual model first developed by Eckhardt and Lee [1] demonstrates that truly independently developed versions can fail dependently. When different teams solve the identical problem, some parts of the problem are intrinsically more difficult, so the independent teams are more likely to commit errors at these points coincidentally. The generalised model developed by

* Corresponding author at: Department of System & Reliability Engineering, Beihang University, 100191 Beijing, China.
E-mail addresses: huangfuqun@gmail.com (F. Huang), liubin@buaa.edu.cn (B. Liu), songyou@buaa.edu.cn (Y. Song), shreya_keyal@hotmail.com (S. Keyal).

http://dx.doi.org/10.1016/j.scico.2014.03.004
0167-6423/© 2014 Elsevier B.V. All rights reserved.

Littlewood and Miller [2] provides more implications on how to pursue diversity. Using "forced" process diversity, if the difficult parts of the problem under one methodology are consistently easy under another design methodology, the actual reliability of a pair of versions "on average" would be better than that under the assumption of independence.

Empirical evidence also shows that coincident faults arise even if the versions are truly built independently or forced to employ different methods. Knight and Leveson's experiment [3] shows that, when the programmers are isolated and design independently, about one-half of the total faults are coincident faults. Avižienis, Lyu, and Schütz's experiment [4] forced diversity of programming languages although, identical faults were still introduced. Meulen and Revilla's experiment on a large population of programs [5] also shows that, the benefit of language diversity appears to be low. Feldt's experiment [6] employed genetic programming to force the algorithms to differ in various versions; the results showed that the test cases that cause versions to fail with high probabilities are located in several specific areas.

These studies suggest that in some circumstances, programmers are prone to making identical errors, introducing coincident faults at specific function points. However, the underlying mechanisms remain unclear and lack scientific evidence, requiring further study [3,7–9]. Human error, the major factor that introduces software faults, can be the key to understanding fault diversity. However, few empirical studies have explored the links between human error diversity and software diversity.

This paper seeks to address this gap by constructing a theoretical model and conducting an empirical study of the links between human error diversity and software diversity. The rest of this paper is organised as follows: Section 2 comprises the methodology of the study, including the conceptual model between human error diversity and software diversity, the modelling of human error diversity and the design of the empirical study. Section 3 demonstrates the results of the empirical study, Section 4 presents discussions and the implications of seeking fault diversity, and Section 5 contains conclusions and recommendations for further work.

2. Methodology

2.1. Terminology clarification

First, we clarify a few terms that are used in this paper.

A **fault** is a departure from its expected properties in a software product.

A **failure** is the set of one or more symptoms that result when a fault manifests itself in operational use of the software.

An **error** is the human behaviour that introduces a software fault, which is given a more specialized meaning in psychology as a mistake, slip or lapse [10].

Errors may lead to faults in a software system. Such faults are non-conformities to stated requirements and/or to human expectations. Faults may manifest themselves as failures during operational use of the system.

Software diversity is a common term to "describe the properties of the products of the N-version programming efforts, with regards to the goal of design diversity and the improvement of the qualities of the member versions" [11]. Different properties can be specified to software diversity, e.g. Lyu et al. [11] has specified four characteristics of software diversity: structural diversity, fault diversity, tough-spot diversity and failure diversity.

Failure diversity is defined as the probability that versions fail in different ways (including the condition that some versions fail, whereas others do not) on the same demands [7]. Failure diversity has two meanings. The first meaning is relevant to failure occurrence: two versions tend not to fail on the same demands. This meaning can be measured by "number of distinct failures found" divided by "total number of failures found" for a specific set of inputs [11]. The second meaning is relevant to behaviour during failures: although two versions fail together on a certain demand, they do so with different (although both erroneous) behaviours, typically different output values.

Structural diversity refers to structural differences among different versions, which can be measured by complexity and size metrics such as code source line, Halstead's volume, and McCabe's Cyclomatic complexity [11].

Fault diversity is the extent to which different versions contain distinct faults, which is measured by the "number of distinct defects found" divided by the "total number of defects found" by Lyu et al. [11].

Tough-spot diversity: "Tough spots" are first described as "the particular system functions in a specification where a programming team has more trouble in building their software according to the specification" by Lyu et al. [11]. He measures it by dividing the number of faults identified in the different components of the program by the number of faults in the entire program. Tough-spot diversity refers to the differences in difficulty among functions in triggering errors and causing software faults.

Structural diversity, fault diversity and tough-spot diversity belong to the **product diversity** proposed by Popov, Strigini and Romanovsky [7]. "Process diversity" leads to "product diversity", whereas "product diversity" produces "failure diversity".

Human error diversity refers to the differences in humans' propensity to make errors, caused by the variations in human factors.

2.2. Conceptual model

The conceptual model of this paper is based on the software diversity model proposed by Popov, Strigini and Romanovsky [7]. The general idea is that, different factors in the process of software development (process diversity) lead to different