Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 92 (2016) 6 – 12

Procedia Computer Science

2nd International Conference on Intelligent Computing, Communication & Convergence

(ICCC-2016)

Srikanta Patnaik, Editor in Chief

Conference Organized by Interscience Institute of Management and Technology

Bhubaneswar, Odisha, India

Nature-inspired approaches in software faults identification and debugging

Florin Popentiu-Vladicescu[a], Grigore Albeanu[b]*

[a]Academy of Romanian Scientists, 54 Splaiul Independentei, Bucharest 050094, Romania
[b]*"Spiru Haret" University, 13 Ion Ghica, Bucharest 030045, Romania*

Abstract

This paper considers software faults identification along the phases from design to testing and debugging. The following subjects are reviewed and extended: bio-inspired concepts for structuring resilient systems, genetic strategies in test data generation, Ant Colony Optimisation (ACO) algorithms for data flow analyzing and testing, artificial immune systems (AIS) based mutation testing, and fault tolerant approaches inspired by immunity principles in order to increase the software dependability. Data collected during software development cycle can be used to understand the software project evolution and its reliability.

Keywords: ACO; AIS; software testing, unit testing; mutation testing; evolutionary testing; software dependability

1. Introduction

Software testing is an important activity in order to achieve and asses the quality of software components/systems, as requested by standards [1, 2]. Quality improvements are obtained through a *test-recognize defects-fix* cycle during software development. Practitioners use also *verification* and *validation* as similar concepts to software testing. As Naik & Tripathy say [3], the verification activities are those that check

* Corresponding author. Tel.: +0-4-021-314-0075.
E-mail address: [a]popentiu@imm.dtu.dk; [b]g.albeanu.mi@spiruharet.ro.

Peer-review under responsibility of the Organizing Committee of ICCC 2016
doi:10.1016/j.procs.2016.07.315

the correctness of a software development phase, while validation activities are related on customer requirements/expectations about the final product. Also, there is a major difference among *testing* (the objective is to find the defect) and *debugging* (the objective is to find the cause of the defect, and remove it.

Software testing deals with the following terms, with apparently similar meaning, but with important differences among them [4-6]: *fault, error, failure*, and *defect*. Fault is a defect within the system (software bug, hardware fault etc.). An error is a deviation from the required operation of system or subsystem, as a consequence of exiting of one or more faults. However, some faults may stay dormant for a long time before generating errors. The presence of an error might cause a whole system to deviate from its required operation, generating a failure. The objective of software testing is to produce *low-risk software (high reliable)* using a small number of *test cases*, where in a simplified model, a test case consists of a pair of (input, expected output). The output may depend not only on input, but also on the system state.

Testing state-oriented software systems is more challenging, the expected output depending both on the current state and the input, which is a similar to biological systems behaviour. A class of state-oriented software is those of *resilient software systems* which behave like natural immune systems by tolerating faults and continue to operate without any failure [7].

In practice, there are at least three levels of software testing [4, 5]: *unit testing* (by programmers working on classes, methods, functions or procedures, in isolation), *integration testing* (more modules are integrated and tested by both software developers and integration test engineers), *system testing* (including functionality testing, load testing, stress testing, robustness testing, performance testing, security testing, and dependability testing (including reliability assessment). Another important level of testing deals with *system acceptance* by customer (a testing level oriented to quality measurement, more than searching for defects) and can be complement to validation activities. The first two levels are important for software developers (programmers/software designers) because they may introduce additional details into the system, which are not covered by requirements and functional specifications. Testing is conducted also oriented through *operational profiles* (ways of software operation by users), and based on *fault classification* (initialization/logic/interface) by *error guessing, fault seeding* and *mutation analysis*. A complete testing should consider coverage analysis taking into account statements, branches, paths, including data-flow paths etc.

Software testing can be conducted through *fuzzy approaches* related to the fault classification (some faults belonging to more than one class), or multilevel input domain partitioning (some inputs belonging to more than one input set), as Madsen et al. described in [8, 9].

In order to increase the speed in test cases generation, software engineers and computer scientists propose the usage of various strategies, some of them inspired by nature: *genetic algorithms* [10-13], *evolutionary strategies* [14-19], *ant-colony systems* [20-29], *artificial immune systems* [30-32], and *other methods* [33-35].

Test data generation consists of generating inputs for the program/system under test in order to evaluate its internal or external behaviour. Automated test generation is a reality nowadays and optimization methods are used to improve the fitness performance (see [14, 19, 20, 23, 27]).

The rest of the paper is organized as follows. Section 2 presents bio-inspired concepts useful to describe genetic algorithms for software testing, and artificial immune systems for fault identification and debugging. Evolutionary strategies, including genetic approaches are described in the third section. The fourth section discusses on ant colony optimization strategies for software testing. Artificial immune systems are considered in the fifth section. The last section is dedicated to conclusions.

2. Bio-inspired concepts

The motivation for using bio-inspired methodologies to software development, testing and debugging comes from the requirements of designing resilient software. Some factors that affect the software resilience and the contexts in which programs might run are related to: the complexity level of software systems, the size of