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

infrastructure under interconnectivity, computer network availability and security, the reliability of open source software, the usage of commercial off-the-shelf software, the rate of software renewal (or the retired software), insufficient testing of the reused software etc.

According to Autili et al [7], biological immunity and software resilience are strong related because the resilient software architecture is based on fundamental concepts, principles and theories developed around the immune systems, those which are self-protected against pathogens being able both to discover and to remove infectious agents. The main elements of a biological immune system are [7, 30]: *Lymphocytes* (TCells: TKillers/THelpers and BCells), *Memory Cells* (the immunological memory), *MHC molecules* (combined with processed antigens recognized by TKillers), *Antigen Presenting Cells* (APC) or macrophages, and *antibodies* (produced by BCells as immune response to an APC presented antigen). THelpers are specialized in recognition of antigens (in order to prevent infections/faults) and TKillers are able to destroy the infectious structures. These elements where used by scientist to design Artificial Immune Systems (AIS) in order to be used in pattern recognition, anomalies discovery etc. Such tasks are possible through immune algorithms based on clonal selection, negative selection, or immune networks theory.

Another motivation in using bio-inspired concepts in software testing comes from evolutionary theories: from an initial population, through evolutionary operators (mutation, crossover etc.) and selection (based on some fitness measure), after a number of steps, a final population is obtained.

Finally, many optimization algorithms were inspired from nature [33]: terrain exploring, in order to find food: ant and bee algorithms, cuckoo search, movement of bats, behaviour of fireflies, swarm optimization etc.

In the following, in the context of software testing only evolutionary strategies and genetic algorithms, ant colony optimization, and immune strategies will be considered.

3. Evolutionary strategies

Evolutionary algorithms are population-based meta-heuristic computational solutions inspired by mechanism of biological evolution like reproduction, mutation, recombination and selection. The most related algorithms to the biological models of computing are genetic algorithms that make use of inheritance, mutation, selection, and crossover. However, general evolutionary operators, and recent evolutionary strategies converge to a powerful evolutionary computation paradigm. The general scheme of any evolutionary algorithm is based on the following steps:

1. Initialize the population P (usually by random candidates);
2. Evaluate (through a metric based procedure) each candidate from P;
3. While a termination criteria is not satisfied do:
   3a) Select parents;
   3b) Recombine pairs of parents;
   3c) Mutate the results obtained through step 3b;
   3d) Evaluate (through a metric based procedure) each offspring obtained through 3c;
   3e) Select the best evaluated candidates and update the population P;
4. Output the obtained population.

An evolutionary algorithm for software testing should evolve as a number of suitable test cases, as initial population. The evaluation metric can use a fitness function. Let be n the total number of domain regions / testing paths, k be the number of regions/paths covered by a test, and then the test case associated fitness/performance is k/n. The recombination and mutation operators applied to test case work like those applied to sequences/strings and depend only on the test case architectural model (representation).

For optimization is considered the following set of attributes: the maximum number of defect detection capability (the detect-ability index), the minimum test design efforts/cost (mainly for the initial population), the largest number of domain-data regions or control flow paths covered by the testing suite etc. Hence the

software testing optimization can be viewed and solved like a multi-objective optimization problem, or multiple criteria ranking problem. Both numeric and linguistic variable based algorithms can be used for test cases ranking.

In some cases, the similarity index among test cases can be used. Based on the normalized matrix associated to the population P, according to the above attributes, a distance (Euclidian, Hamming etc.) matrix D can be computed (d_{ij} is the distance between individuals i and j). Then the distance of an individual i in the population is the sum of distances on the row i, and the quality index of P can be given as the sum of all elements of D. In this way, is easy to extend the approach to a multi-population based algorithm, where populations evolve in parallel, and finally the most suited population of test cases should be selected.

4. ACO strategies

An ACO algorithm can be used for optimal path identification and is a probabilistic technique for exploring graphs according to some well defined objective.

Such an approach belongs to structural testing (white box) which addresses the following types of testing: data flow testing, control flow/coverage testing, basic path testing, and loop testing. Data flow testing focuses on the points at which variables receive values (assignment operator or input data) and the points at which these values are used (or referenced). By data flow testing can be identified improper use of data values (data flow anomalies) due to coding errors.

The ACO algorithm is working in two modes, depending on the moving direction of ants: forward (from source to destination) and backward (from food to nest). The paths under investigation belongs to the control flow graph (CFG) associated to the software under testing. A promised algorithm was described by Mann and Sangwan [21].

Based on ants movement rules and the transformation relationship between ant colony paths and test cases, Yang et al [29] make effective an ant colony algorithm for iterative optimization of software test cases.

Any ACO algorithm iteratively performs a loop based on the following procedures: the solution constructor/updater (iterative/evolutionary operator), and a procedure to update the pheromone trails depending on a parameter modeling the rate of pheromone evaporation. The identified faulting paths will be used to efficiently debug the code.

5. Artificial immune strategies

Artificial Immune Systems should offer both innate (predefined) and adaptive (through learning) immunity. Based on clonal selection (CS) theory, proposed by Burnet [36], the CLONALG is an algorithm proposed by Castro and Zuben [37], which generates a population on N antibodies, each specifying a random solution for the optimization process (in our case the test planning optimization, test case selection and prioritization etc.), and at every iteration, some of the best (according to a specific metric/fitness indicator) existing antibodies are selected, cloned (proliferated) and mutated to form a new candidate population. The original population is enriched by including a percentage of the best new antibodies, while a percentage of the worst antibodies are eliminated. There are possible many variations of CLONALG in order to cover some extensions like polyclonal strategies, new immunity operators to prepare the immune response, new learning operators, new affinity measures etc.).

The negative selection theory provides a mechanism to protect the system against self-reactive entities (for human body those TCells that reacts against self-proteins are destroyed, the remaining TCells, called matured, are free to circulate throughout the body in order to perform immunological functions and protecting the body against external antigens). The first negative selection (NS) algorithm was proposed by Forrest et al [38], generating a set of detectors (by random generation, and removing such detectors that recognized trained self-

data) and use these detectors to anomalies identification. In general, any NS algorithm consists of three steps: self-data definition, candidate detectors generation, usage of the affinity threshold to select those detectors to be used for immunity assurance services.

The immune network theory, proposed by Jerne [39], motivated researchers to use BCells (as vertices) based network models to solve a large plethora of problems by cloning and mutation processes at vertex levels.

Various hybridization techniques there exist. These approaches include soft computing techniques like: artificial neural networks, fuzzy systems, AIS, evolutionary computation and genetic algorithms, including gene expression programming. Main applications of immune network algorithms are connected to the following fields: data analysis, clustering, anomaly detection, machine learning, optimization, robots, computer security etc.

Clonal selection can be used to test generation (a large collection of test cases can be obtained by mutation operator). The size of collection, considered like detectors, can be reduced by simulate a negative selection to eliminate those detectors which are not able to detect faults. The remaining detectors will be cloned and mutated, evaluated and used to create a new population of detectors. In this way a mixed CS + NS approach is obtained. Another application of AIS in software testing may consider the mutation testing and extend the techniques proposed in [12] and [13].

Based on the good behaviour of AIS in optimization, the evolutionary or ant optimization algorithms used in software testing optimization and prioritization can be modified in order to use immune operators and affinity metrics.

6. Conclusions

This paper considers software testing phase and investigates on the applicability of nature inspired approaches to test cases generation, test prioritization, and test planning optimization. Software fault identification can be treated as anomalies identification when using artificial immune systems. The software reliability and dependability can be improved by optimized software testing and adequate debugging.

Acknowledgements

The research on improving software reliability by optimized software testing was conducted by the first author and supported by Academy of Romanian Scientists. Nature-inspired approaches have been used according to the research internal plan of “Spiru Haret” University, namely the “Soft computing methodologies” project coordinated by the second author.

References

[1] The Institute of Electrical and Electronics Engineers. “29119-1-2013 - Software and systems engineering —Software testing.” IEEE Standards Association. http://standards.ieee.org/findstds/standard/29119-1-2013.html

[2] International Organization for Standardization. “What Is a Standard?” ISO Standards. http://www.iso.org/iso/home/standards.htm>http://standards.ieee.org/findstds/standard/29119-1-2013.html

[3] Naik K, Tripathy P. *Software Testing and Quality Assurance. Theory and Practice*. Wiley, 2008, 616p.

[4] Myers G, Badgett T, Sandler C. *The Art of Software Testing*. 3rd Edition. Hoboken, NJ: J. Wiley & Sons; 2014.

[5] Spillner A, Linz T, Schaefer H. *Software Testing Foundations*. 4th Edition.. Santa Barbara: Rocky Nook Inc.; 2014.

[6] Meyer B. Seven Principles of Software Testing. *Computer* 2008, **August**:99-101.

[7] Autili M, Di Salle A, Gallo F, Perucci A, and Tivoli M., Biological Immunity and Software Resilience: Two Faces of the Same Coin?, in A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, DOI: 10.1007/978-3-319-23129-71, 1–15, 2015.

[8] Madsen H, Thyregod P, Burtschy B, Albeanu G, Popentiu F. A Fuzzy Logic Approach to Software Testing and Debugging. In: Guedes Soares, Zio E, editors. *Safety and Reliability for Managing Risk* , London:Taylor & Francisc Group; 2006, p. 1435-1442.
[9] Madsen H, Thyregod P, Burtschy B, Albeanu G, Popentiu F. On using soft computing techniques in software reliability engineering. *International Journal of Reliability, Quality, and Safety Engineering* 2006;**13(1)**:1–12.
[10] Sharma C, Sabharwal S, Sibal R. A Survey on Software Testing Techniques Using Genetic Algorithm. *International Journal of Computer Science Issues* 2013;**10(1)**:381-393.
[11] Kamboj S, Mohinder Singh M. Survey Paper on Optimum Selection of GA Algorithm's Parameters for Software Test Data Generation. *International Journal of Science and Research* 2014;**3(6)**:46-49.
[12] Hanh LTM, Tung KT, Binh NT. Mutation-based Test Data Generation for Simulink Models using Genetic Algorithm and Simulated Annealing. *International Journal of Computer and Information Technology* 2014;**3(4)**:763-771.
[13] Mishra KK, Tiwari S, Kumar A, and Misra AK, An Approach for Mutation Testing Using Elitist Genetic Algorithm, 3rd IEEE International Conference on Computer Science and Information Technology, doi:10.1109/ICCSIT.2010.5564072, 426-429, 2010.
[14] Chauhan D, Sehgal A. Automated test data generation using soft computing techniques. *International Journal of Advanced Research in Computer Engineering & Technology* 2015;**4(4)**:1165-1169.
[15] Manoj Kumar M, Sharma A, and Kumar R, Optimization of Test Cases using Soft Computing Techniques: A Critical Review, *WSEAS Transactions on Information Science and Applications* 2011;**11(8)**:440-452.
[16] Pandey B, Jain R. Soft Computing Based Approaches for Software Testing: A Survey. *International Journal of Soft Computing and Engineering* 2014;**4(2)**:4-8.
[17] Parnami S, Sharma KS, and Chande SV, A Survey on Generation of Test Cases and Test Data Using Artificial Intelligence Techniques, Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering, doi:10.3850/978-981-07-1403-1483, 196-198, 2012.
[18] Ribeiro JCB, Zenha-Rela MA, de Vega FF. Test Case Evaluation and Input Domain Reduction strategies for the Evolutionary Testing of Object-Oriented software. *Information and Software Technology* 2009;**51**:1534–1548.
[19] Singla S, Kumar R, Kumar D. A Review on Techniques Used for Automatic Generation of Software Test Cases. *International Journal of Electrical, Electronics and Computer Engineering* 2014;**3(1)**:69-73.
[20] Agarwal S, Gupta S, Sabharwal N. Automatic Test Data Generation-Achieving Optimality Using Ant-Behaviour. *International Journal of Information and Education Technology* 2016;**6(2)**:117-121.
[21] Mann M, Sangwan OP. Generating and prioritizing optimal paths using ant colony optimization. *Computational Ecology and Software* 2015;**5(1)**:1-15.
[22] Panda M, Sarangi PP. Performance Analysis of Test Data Generation for Path Coverage Based Testing using Three Meta-Heuristic Algorithms. *International Journal of Computer Science and Informatics* 2013;**3(2)**:34-41.
[23] Sharma P. Automated Software Testing Using Metahurestic Technique Based on Improved Ant Algorithms for Software Testing. *International Journal on Recent and Innovation Trends in Computing and Communication* 2014; **2(11)**: 3505-3510.
[24] Singh A, Garg N, Saini T. A hybrid Approach of Genetic Algorithm and Particle Swarm Technique to Software Test Case Generation. *International Journal of Innovations in Engineering and Technology* 2014;**3(4)**:208-214.
[25] Singla S., Kumar D, Rai HM, Singla P. A Hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with Dominance Concepts. International Journal of Advanced Science and Technology 2011;**37**:15-26.
[26] Srivastav AK, Supriya NS. Refinement of Data-Flow Testing using Ant Colony Algorithm, *International Journal of Combined Research & Development* 2015;**4(2)**:266-270.
[27] Srivastava PR, Baby K. Automated Software Testing Using Metaheurestic Technique Based on An Ant Colony Optimization. 2010 International Symposium on Electronic System Design (ISED), Bhubaneswar, 20-22 Dec. 2010, pp. 235-240.
[28] Suri B, Singhal S. Implementing Ant Colony Optimization for Test Case Selection and Prioritization. *International Journal on Computer Science and Engineering* 2011;**3(5)**:1924-1932.
[29] Yang S, Man T, Xu J. Improved Ant Algorithms for Software Testing Cases Generation. *The Scientific World Journal (Hindawi)* 2014; http://dx.doi.org/10.1155/2014/392309.
[30] Timmis J, Knight T, Castro L, Hart E. An Overview of Artificial Immune Systems, In: Paton R, Bolouri H, Holcombe M, Parish JH, Tateson R, editors. *Computation in Cells and Tissues: Perspectives and Tools for Thought*, Natural Computation Series, Springer; 2004, p:.51-86.
[31] WebAIS. The Online Home of Artificial Immune Systems (http://www.artificial-immune-systems.org/algorithms.shtml)
[32] Liaskos K, and Roper M, Hybridizing Evolutionary Testing with Artificial Immune Systems and Local Search, IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08), doi:10.1109/ICSTW.2008.21, 211-220, 2008.
[33] Bouchachia A, Mittermeir R, Sielecky P, Stafiej S, Zieminski M. Nature-inspired techniques for conformance testing of object-oriented software. *Applied Soft Computing* 2010;**10**:730–745.
[34] Li K, Zhang Z, Kou J. Breeding Software Test Data with Genetic-Particle Swarm Mixed Algorithm. *Journal of Computers* 2010;**5(2)**: 258-265.

[35] Malhotra R, Anand C, Jain N, Mittal A. Comparison of Search based Techniques for Automated Test Data Generation. *International Journal of Computer Applications* 2014;**95(23)**:4-8.
[36] Burnet FM. *The Clonal Selection Theory of Acquired Immunity*. Cambridge University Press; 1959.
[37] Castro L de, Zuben F. Learning and Optimization Using the Clonal Selection Principle. *IEEE Transactions on Evolutionary Computing* 2002; 6(3)::239-251.
[38] Forrest S, Perelso AS, Allen L, Cherukuri R. *Self-Nonself Discrimination in a Computer*. In Proceedings of IEEE Symposium on Research in Security and Privacy, Los Alamitos, CA: IEEE Computer Society Press;1994, p. 202-212.
[39] Jerne NK. *Towards a Network Theory of the Immune System*. Ann. Immunology 1974; 125C: 373-389.