Optimization of Automated Software Contracts Generation by Modified Particle Swarm Optimization

S. V. Gayetri Devi
Research Scholar
Department of Computer Science and Engineering
Bharath Institute of Higher Education and Research
Chennai, India

C. Nalini
Professor
Department of Computer Science and Engineering
Bharath Institute of Higher Education and Research
Chennai, India

Abstract

Incorrect software behaviours are detected using a combination of Programming language contracts and alliedresources of programs namely static source code Analyzers and Runtime verifyingframeworks. To surpass therestrictions of inordinate run-time overheads in the course of verification of software modules with timing constraints, Optimization of the contracts after their automated development is imperative. This is carried out by retrieving the structural and behavioural dependency information of the real time multithreaded Java source code under test, then remodelthem as contract details in a Decision tree after Static and Dynamic analyses. The next step incorporates transformation of the conditions into contracts. The Optimizer logic is implemented as a modification of Particle Swarm Optimization technique during this transformation by adopting an objective fitness function that substantiallylends to handle only the source files that contain valid decisions. Evaluation measures specifically processing time for contracts formation, Memory and CPU Utilization, Number of files processed are used to determine the effectiveness of the proposed system comparable with no optimization applied.

Keywords*–Programming assertions, Source code Analysis, dependency, Particle Swarm Optimization, objective function*

I. INTRODUCTION

Appropriate Verification routines are essential to confirm that dependable software is delivered. Testing forms a very valid, extensive verification method. Nevertheless, it is an extremely exclusive and challenging activity, and along with an exponential development pertaining to the intricacy of softwares, the costing associated with the tests has mounted consequently. Ways to drop the cost with no foregoing on thestandard of verification are necessary. A great deal of that cost can be drawn right away to the human efforts to lead the verification activities. Automation has great capability in the lowering aspect.

The architecture of a system is a pool of interrelated components, where program execution involves crafting objects of various types and communicating with them through operations. The contracts outline the syntactic structure of the operation about what it expects from its arguments and what it works out as an output. During run time checks, the assertions ascertain that the precondition is met on access and that the postconditions are fulfilled on return. Breach of either of these criteria results in an exception. During formal static analysis by a tool, contracts affirm if the prerequisite condition is adequateat each calling on, and that the post condition is satisfied based on the execution of the operation.

Contract programming is thus regarded as encompassing Preconditions, Postconditions, Exception guarantees(which the developers foresee to hold true when a method exits throwing an exception, Class invariants(which express valid states for all the objects of a provided class), Subcontracting(which specify that preconditions cannot be fortified, while postconditions and class invariants cannot be weakened when public methods are overridden by a public method in one or more of its base classes.

Static analysis also referred to as static source code analysismakes use of tools to examine program code, looking for application programming defects, backdoor attacks, or other potential harmful code giving access to crucial business or customer information to the hackers. When static analysis verifies source or object code, the functionality and security of software is evaluatedwithout executing the program modules often by an automated tool. One main merit of Static Analysis is the capability to identify the precise susceptible line in the program code. The flow of data from source to destination can be marked out and the point at which the untrusted data from willlead to aninsecurity can be located making it probable to identify particular lines in source code that are exposed.

Dynamic analysis is the assessment of a software behaviour using real-time information. Complex memory handling defects like indexing outside array limits and resource leaks. Concurrent code can be analysed at execution time identifying possible issues with shared resources access and potential deadlocks. Thereby, both static and dynamic analyses complement each other to reinforce the worthinessin the software and shorten the development costs.

Assertions are usually inserted in the code manually with the help of the software design and user given specifications. It gets very challenging to comprehend the various designs and specifications with increasing complexity of softwares and evolving contracts manually. It takes several iterations and huge time to craft minimal yet effective contracts warranting high code coverage. To alleviate these concerns, an automated means of building programming contracts is required. A provision to independentlyyield contracts using the analyses incorporating analysed information into mining algorithms to produce candidate contracts is proposed.

II. LITERATURE SURVEY OF CONTRACTSFORMATION, VERIFICATIONAND OPTIMIZATION

Thesection belowreviews thegeneral existing approaches in optimization of programming Contractsderivation for software verification.

Hertz et al.,(2019) demonstrated the usefulness of ranking for verification (pre-silicon)and studied the significance of the ranked assertions in the perspective of validation (post-silicon) validation context through outlined and reinstated signal valuesfrom the components netlist of design. The work does not specify how assertions need to be written butgives a framework forassessing them

Zemzami et al., (2019) put forward a schemeon the basis ofPSO using evolutionary neighborhood topology and Parallel computation to project good efficiency in form of lowered time and convergence of Optimality for solutions.

Abdallah at al., (2018) suggested a Multi-Objectives Evolutionary Algorithm approach (MOEA) using fitness functions to achieve optimized high coverage of source code so thatlesserexpense and superfluity is assured. NSGAII, Random, SMSEMOA, ν and ε-MOEA are said to be part of the MOEA approach and proven to achieve 90 percent of code coverage.

Alakeel (2018) presented a concurrency model to lessen the time for a larger counts of assertions to be probed during Assertion-Based tests while creating testcases. Each assert is converted into a set of nodes with independent execution of each node. Concurrent programming or parallel processing model is used to execute the nodes thus lessening the time required for processing many number of assertions during testing.

Prajapati et al.,(2017) employed an improved form of PSO algorithm to solve Non-Linear Programming Problems without constraints and compared the results of sample problems worked out in SCILAB programming language with improved PSO and regular PSO. The improved PSO with five functions was supposed to be applied to any NLPP such as rastrigin function, banana function, rosenbrook function, square function, etc. with any number of equality constraints.

Stulova et al.,(2016) recommended techniques to improvise the accuracy of software analysis by the semantics of the runtime checking. The method demonstrated the positives and costs of every assertion checking modes suggested. It talked about the problem of reducing runtime overhead on the outlook of verification structures combining both dynamic and static verification. The main aim was to build an automatic verification method for non-trivial, structural properties utilized regularly in production code.

Wang et al.,(2015) provided a novel PSO algorithm where the details of the best particle neighbor and the ideal particle across complete swarm for existing loop is valued. To improvise the global merging speed of the technique, a disordered seeking is implemented in the finest solution of the existing loop. For substantiating the fulfilment of the proposed work, customary testing tasks are utilized. The outcomes display greater competence than ongoingPSO algorithms.

Wahono et al.,(2013)proposed the arrangement of PSO and Bagging technique for refining the correctness of the software defect detection. PSO is used to address the selection of feature, and bagging is utilized to face the issue of class imbalance. The projected method is assessed with the data sets obtained from NASA metric data repository. Outcomes have specified that the method made improvements in prediction efficiency for most of the classifiers.

III. REVIEW FINDINGS

Most software programming languages have noput upmechanism to embraceprogram contracts with the provisionto generating them without any automationconvenience for run time analysis of software. Languages centredondesigns on basis of contractsand their related Integrated Development Environments propose the functionality to deliver methods with pre and post contexts instructions, well recognizedconditionsoninheriting and qualifies thesoftware that transmutes programs from source language level to Object Machine language befortified with preferencesof activating/deactivating rulesverificationwhen the executable is on run. An automated process of arriving atassertionswith appropriate optimization on the candidate assertions derivationis essential.

IV. OPTIMIZATION METHODOLOGY TOAUTOMATE CONTRACTS DEVELOPMENT

The preliminary intent of optimization of programming assertions is generating them in an automated means by analyzing the input software both statically and dynamically. Semantic information pertaining to Structural Dependencies and dependencies those of Behavioral are extracted by appropriate analyses and contrived as Decision Tree. Depending on the specs of the test modules, the constraints on the Decision Tree are built into Programming contracts.

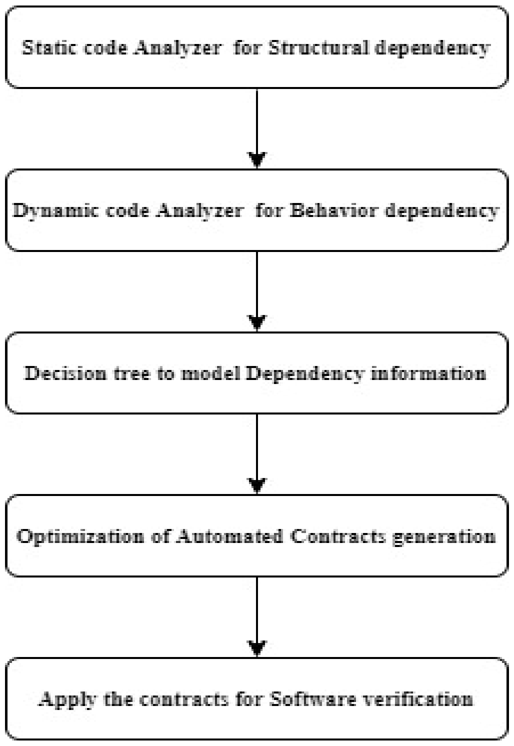
Static code Analyzer for Structural dependency
Dynamic code Analyzer for Behavior dependency
Decision tree to model Dependency information
Optimization of Automated Contracts generation
Apply the contracts for Software verification

Fig 1. Optimized automated contracts development

After the programming contracts are generated, the potential rules of programming are then leveraged to Optimization to reduce the cost of software verification in a time constrained environment. This process cuts downs on redundancies of assertions. This requires tailoring the computational aspect of Particle Swarm Optimization algorithm to refine the number of feasible contracts to verify. The particles in exploration space impact each other, conversing details to fine-tune the position and quickness of itself and travel to the optimal solution. PSO initialize a set of particles, terminating the optimal solution by iteration. During every individual iteration, the particle apprises the speed and position by outlining individual extreme point and global extreme point. To begin with, every potential contract is signified by a particle part in PSO technique, inductedarbitrarily over the provided examinedbounds. The algorithm encompasses a swarm (population) of contracts. The position object of everycontract particle assesses excellence of contract and represents the coordinates of the highest or slightestaptness. The position is expected to come closer along the best possibleresolution by providing some shift (velocity vel) along prime contract. Hence each possible contract travel with certain velocity in the direction of the optimal solution.

In every generation or iteration, each possible contract keeps trail of its coordinates in the problem space and the optimization technique looks for optima by updating the particle solution with two "best" values and improves the swarm of contracts towards better position. The first value is the best objective fitness value among all its positions, a contract particle has reachedasyet. The objective fitness content is pbest. A further "best" representation trailed by the optimizer is the best fitness worth (global best gbest), attained among all particles accessed so far within swarm. Thus, the foremost position, a solution personally has determined is the Personal best and the Global best is towards the globally best position

found by the entire swarm. The accelerating constants c1 and c2 are the learning factors that tug each solutionin the direction of personal and global bestlocations. Slighterrepresentations allow solutions to travel far from targetingbounds, while greaterones cause quicker association towards target regions. After determining both, velocity and positions are renewed.
Objective function is evaluated using

$$\boldsymbol{Fitness}(\mathbf{X}) = \sum_{i=0,cdl=0}^{N} \boldsymbol{if}(\boldsymbol{S_i}(\boldsymbol{cdl}) > 0) \rightarrow \boldsymbol{Update}(\boldsymbol{newFitness});$$

V. Results and discussion

The primaryaim of the work is to show that the proposed technique may be used to extract contracts automatically without manually in lining them in code from specification. Performance assessment for classes of the test software with sensitivity to changes in other regions of designs, is made using metrics such as Execution time, Frequency and Popularity rank picked up from code analyzer. Statistical information is obtained by analysis of program code. Behavioral dependencies between classes is assessed from the Trace events. Time of Execution as well as Frequency in order to findthe count of proceduresinvokedare identified. Rank of Popularity is determined to identify the change prone classes. Classes dependency and corresponding relationships is estimated from Background metrics like object classes Coupling, Number of inheriting classes, attributes, instance variables, methods etc.
We can therebyarrive at the relevant automated contracts (assertions). But, the GoldMine tool – a software tool for generating rank register transferlevel (RTL) assertions is not found to extract these mandatory background metrics to generate the candidate assertions and does not derive the contracts / assertions for sensitive / dependent classes and its depending classes.
The following graph given below (Fig2) shows the comparison of assertion generation by GoldMine tool and the proposed generation technique.

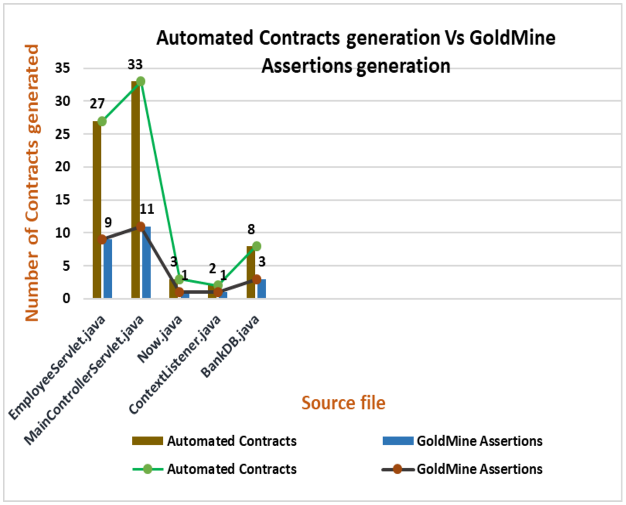

Fig 2. Automated Contracts generation Vs GoldMine tool Assertions generation

The contracts generation phase before optimization processed 14 input files to generate contracts irrespective of whether the source file contained decisions to be validated or not and resulted in Processing time of 6.778 secs, CPU Utilization of 62.17% and Memory Utilization of 78.83%. With

Optimization in place, improvement in the values was observed since only the source files with decisions were considered for contracts extraction. This led to improved Processing time of 2.916 secs, CPU Utilization of 33.14% and Memory Utilization of 41.78% being attained.

TABLE I

PARAMETERS ASSESSMENT WITH AND WITHOUT OPTIMIZATION OF AUTOMATED CONTRACTS GENERATION

Parameters	**Without Optimization**	**With Optimization**
Contracts generation Processing Time(sec)	6.778	2.916
CPU Utilization (%)	62.17	33.14
Memory Utilization (%)	78.83	41.78
Input files processed for Contracts generation	14	5

The graphical outcomes are shown in the figures provided below (Fig3, Fig4,and Fig 5).

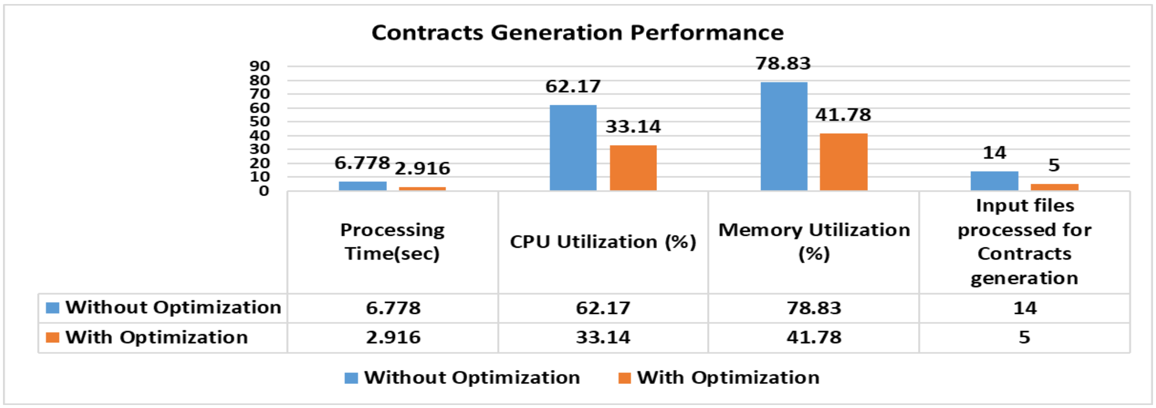

Fig 3. Contracts generation Processing Time, CPU and Memory Utilization and Number of sourcefiles processed

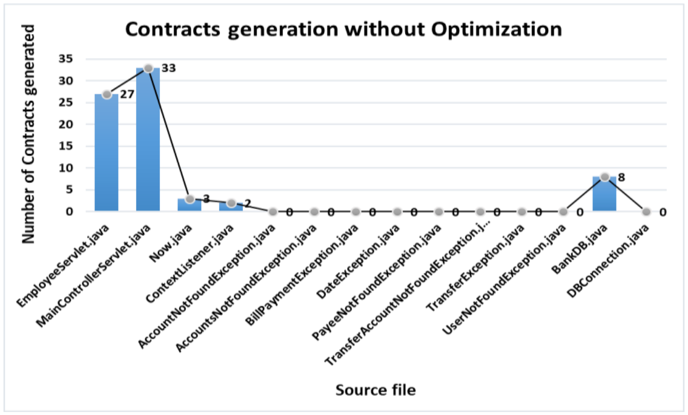

Fig 4. Automated Contracts generation without Optimization

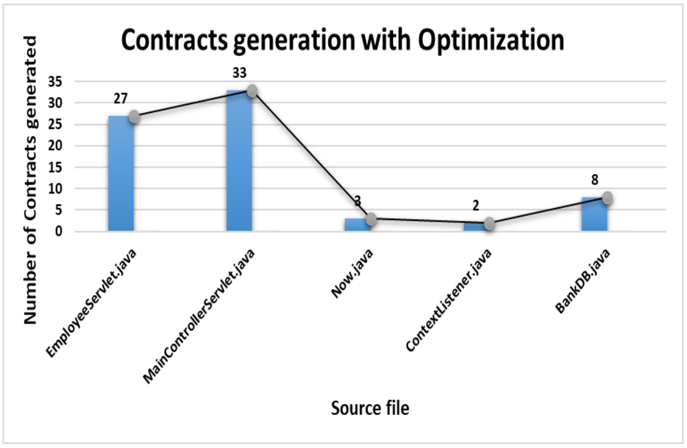

Fig 5. Automated Contracts generation with Optimization

V. CONCLUSION

The widespreadcontract derivationtechniquesare studiedand a straightforward methodology to form contracts with notypicalinvolvement of developers is set forth. The coreobjective of the providedmethod is to augmentadequacies of Verification and Validationamidst the existence of enormouscontracts.Also, thetechniquesupports software developersfor time efficientcontracts-basedtesting to their software modules with the evaluation of measures namely Contracts generation processing time, Memory and CPU Utilization. Optimization is performed to identify source files to process for contracts appropriate to Test environment with Time associatedrestrictionsduringfindings ofseveral programming bugsencompassingrare bugs in multi-threaded software.

REFERENCES

[1] B. Meyer. Applying "design by contract". Computer, 25(10):40–51, Oct. 1992.
[2] C. Hurlin. Specifying and checking protocols of multithreaded classes. In Proc. of SAC'09, pages 587–592. ACM, 2009.
[3] B. Meyer. Object-Oriented Software Construction, 2nd edition. Prentice Hall, 1997
[4] G. T. Leavens, A. L. Baker, and C. Ruby. Preliminary design of JML: a behavioral interface specification language for Java. SIGSOFT Softw. Eng. Notes, 31(3):1–38, 2006.
[5] Scott A. Carr, Francesco Logozzo, Mathias Payer. Automatic Contract Insertion with CCBot. IEEE Transactions on Software Engineering (Volume: 43, Issue: 8, Aug. 1 2017), Page(s): 701 – 714
[6] F. Logozzo, "Practical Specification and Verification with Code Contracts," in Proceedings of the 2013 ACM SIGAda Annual Conference on High Integrity Language Technology, ser. HILT '13. New York, NY, USA: ACM, 2013, pp. 7–8. [Online]. Available: http://doi.acm.org/10.1145/2527269.2534188
[7] Schmidt, R. F.: Software engineering architecture-driven software development. Amsterdam: Elsevier: doi:https://doi.org/10.1016/B978-0-12-407768-3.00015-X, (2013)
[8] Knutson, C., & Carmichael, S.: Verification and Validation. Embedded Systems Programming, Vol. 25 (2001).

[9] e Silva, R. A. B., Arai, N. N., Burgareli, L. A., de Oliveira, J. M. P., & Pinto, J. S.: Formal verification with frama-C: A case study in the space software domain. IEEE Transactions on Reliability, Vol. 65(3), (2016) 1163-1179.

[10] Podgurski A, Clarke LA. A formal model of program dependences and its implications for software testing, debugging, and maintenance. IEEE Transactions on Software Engineering 1990; 16(9): 965–979. DOI: 10.1109/32.58784.

[11] Callo Arias TB, Spek P, Avgeriou P. A practice-driven systematic review of dependency analysis solutions. Empirical Software Engineering 2011; 16(5): 544–586. DOI: 10.1007/s10664-011-9158-8.

[12] EtentingHub – Online Free Software Testing Tutorial. (n.d.). Software Testing Verification. Retrieved January 27, 2018, from http://www.etstnghub.com/testing_verification.php

[13] Quinlan, J. R. (1987). "Simplifying decision trees". International Journal of Man-Machine Studies. 27 (3): 221. doi:10.1016/S0020-7373(87)80053-6.

[14] Perry,W., 2006: Effective Methods for Software Testing, Third Edition. John Wiley & Sons, Inc., New York, NY, USA.

[15] Gopinath, R., C. Jensen, and A. Groce, 2014: Mutations: How close are they to real faults? 25th International Symposium on Software ReliabilityEngineering, 189–200

[16] Anand, S., E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen,W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, et al., 2013: An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 86, no. 8, 1978–2001

[17] G. Myers, “The Art of Software Testing,” John Wiley & Sons, New York, 1979.

[18] Maria Zemzami, Norelislam Elhami, Mhamed Itmi, Nabil Hmina, 2019:A modified Particle Swarm Optimization algorithm linking dynamic neighborhood topology to parallel computation. International Journal of Advanced Trends in Computer Science and Engineering, Volume 8, No.2, March - April 2019

[19] Raju Prajapati, Om Prakash Dubey and Randhir Kumar, 2017: Improved Particle Swarm Optimization for Non-Linear Programming Problem with Barrier method, International Journal of Students’ Research in Technology & ManagementeISSN: 2321-2543, Vol 5, No 4, 2017, pp 72-80

[20] Stulova, Nataliia & Morales, Jose & Hermenegildo, Manuel V.. (2016). Reducing the overhead of assertion run-time checks via static analysis. 90-103. 10.1145/2967973.2968597.

[21] El-agouz, Samar & Moawad, Ramadan & Fawzy, Esaam. (2018). An Optimization Approach for Automated Unit Test Generation Tools using Multi-objective Evolutionary Algorithms. Future Computing and Informatics Journal. 3. 10.1016/j.fcij.2018.02.004.

[22] D. Rosenblum, “Toward A Method of Programming With Assertions,” Proceedings of the International Conference on Software Engineering, 1992, pp. 92-104.

[23] Ali M. Alakeel, “A Testability Transformation Approach for Programs with Assertions,” Proceedings of the Sixth International Conference on Advances in System Testing and Validation Lifecycle, Nice, France, pp. 9-13, October 2014.

[24] Ali M. Alakeel, “Intelligent Assertions Placement Scheme for String Search Algorithms,” Proceedings of the Second International Conference on Intelligent Systems and Applications, Venice, Italy, pp. 122-128, April 2013

[25] Ali M. Alakeel, "Assertion-Based Software Testing Metrics Approach Based on Fuzzy Logic," Proceedings of the 22nd International Conference on Software Engineering and Data Engineering (SEDE–2013), Los Angeles, California, USA, pp. 9-12, September 2013.

[26] B. Goetz, et al., “Java Concurrency in Practice,” Addison-Wesley Professional, 2006.

[27] Fu, H., Wang, Z., Chen, X. et al, “A systematic survey on automated concurrency bug detection, exposing, avoidance, and fixing techniques,” Software Quality Journal, No. 26, 2018, pp.855–889.

[28] “How to test your concurrent software: an approach for the selection of testing techniques,” Proceedings of the 4th ACM SIGPLAN International Workshop on Software Engineering for Parallel Systems, Vancouver, Canada da, pp. 42-43, 2017.

[29] Guodong Qu, Song-Tao Guo, and Hongqun Zhang, “A practical approach to assertion testing framework based on inner class, IEEE 2nd International Conference on Software Engineering and Service Science, pp. 133-137, 2011.

[30] B. Korel, A. Al-Yami “Assertion-Oriented Automated Test Data Generation,” Proc. 18th Intern. Conference on Software Eng., Berlin, Germany, 1996, pp. 701-80.

[31] B. Korel, , Q. Zhang, L. Tao, “Assertion-Based Validation of Modified Programs,” Proc. 2009 2nd Intern. Conference on Software Testing, Verification and Validation, Denver, USA, 2009, pp. 426-435.

[32] Ali M. Alakeel, "Using Fuzzy Logic in Test Case Prioritization for Regression Testing Programs with Assertions," The Scientific World Journal, vol. 2014, Article ID 316014, 9 pages, 2014. doi:10.1155/2014/316014.

[33] Ali M. Alakeel, “Using Concurrency to Improve Assertions-Based Software Testing " International Journal of Computer Science and Information Security (IJCSIS), Vol. 16, No. 12, December 2018

[34] Alakeel AM. Using Fuzzy Logic Techniques for Assertion-Based Software Testing Metrics. ScientificWorldJournal. 2015;2015:629430. doi:10.1155/2015/629430

[35] Wang, Chun-Feng & Liu, Kui. (2016). A Novel Particle Swarm Optimization Algorithm for Global Optimization. Computational Intelligence and Neuroscience. 2016. 1-9. 10.1155/2016/9482073.

[36] Wahono, R.S., & Suryana, N. (2013). Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction.

[37] Samuel Hertz, Debjit Pal, Spencer Offenberger, and Shobha Vasudevan. 2019. A figure of merit for assertions in verification. In Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASPDAC ’19). Association for Computing Machinery, New York, NY, USA, 675–680. DOI:https://doi.org/10.1145/3287624.3287660