ISSN: 2278 – 7798
International Journal of Science, Engineering and Technology Research (IJSETR)
Volume 5, Issue 6, June 2016

Study of Evolutionary Testing for Structural Testing of Embedded Systems

Pravin Karmore, Pradeep Butey

***Abstract*— Embedded system testing is most important in settings where small software defects can have a large impact on the safety of human beings. This paper analyses the role of Evolutionary Testing (ET) for test data generation for structural testing of embedded systems. The design of quality System-on-Chip (Embedded System) has many challenges in today's date. Evolutionary algorithms (EA) are normally more straightforward to apply, because no restrictions for the objective function definition exist. Evolutionary testing is an assured approach for the automation of structural test case design. It helps to search, test data that satisfy given structural test criteria by the way of evolutionary computation. Evolutionary testing could directly benefit from the improvements by incorporating new search techniques into test data generation, thus it imparts in an increase in the effectiveness and efficiency of the tests. Evolutionary testing can be applied to statement tests, condition tests, segment tests, and branch tests.**

Index Terms*— *Embedded Systems, Evolutionary Testing, Evolutionary algorithms, Structural Testing.

I. INTRODUCTION

For the development of any embedded system designed for use, testing plays a vital role. Embedded system programming required thousands of lines of code, written by many programmers and months or even years required for development. In order to ensure that the software they produce is without errors and works as it should, developers perform tests on their software while it's development. Embedded system software testing is usually viewed as significant in settings where little software faults can have a large impact on reliability of systems upon which human's lives and livelihoods depend. Embedded systems have unique characteristics that should be mentioned in the test plan. An embedded system testing has its own distinctive flavor [9]. To ensure the reliability, robustness and safety of embedded software, developers carry out a range of tests. These consist of automated solutions, such as automated regression testing and automated integration testing [13]. In industrial applications embedded systems are mainly used for monitoring and controlling technical processes. There are in nearly all industrial areas, for example in motor vehicle technology, railway and aerospace technology, communication technology, process and automation technology, process and power engineering, as well as in defense systems. Almost 95% of all electronic devices produced today are used in embedded systems. In order to accomplish high quality in the development of embedded systems, central significance is attributed to analytical quality assurance. The plan of this research work is to enhance the quality of the tests and to achieve substantial cost savings in system development by providing a high degree of automation of test case design. Evolutionary testing is a promising methodology for completely automating test case design for various test aims [4]. For example, evolutionary tests can be used to structure and automate the testing of non-functional properties and to produce test cases for conventional test methods. Embedded software is often integrated in highly complex devices. Medical device software, automotive software, avionics software, military software and railway software are all used to control devices or vehicles on which people's lives depend. A fault in that software may not just be inconvenient, it could be disastrous.

Manuscript received May , 2016.

***Pravin Y. Karmore**, Research Scholar, Dept. of Electronics and Computer Science, RTM Nagpur University, Nagpur, India, 9422815212.*

***Dr. Pradeep K. Butey,** Dept. of Computer Science, Kamla Nehru College,Nagpur India, 9422110365.*

II. BACKGROUND

An extensive literature analysis is carried out in order to understand the existing methodologies. It was found that, many researchers carried out work in this domain. Some of the works that were studied have been summarized in this section.

A. Code Coverage Analysis

The first method used for software testing was Code Coverage Analysis (CCA), first discussed in 1963 in Miller and Maloney's Communications of the ACM. It was used to test software in every field from avionics to industrial control units. Now a day, code coverage is still one of the most accepted solutions for system testing. Software testing is crucially important for ensuring the reliability and safety of embedded software. And one of the most reliable forms of testing embedded software is Code Coverage Analysis (CCA). Test actions that map to the particular software are not sufficient to thoroughly test software, as they hold to the insignificant paths through the code rather than testing its boundaries and error conditions. Code coverage unravels this issue by telling the tester which parts of the source code were executed in each test, permitting the tester to remove areas and make sure each and every section of code is checked for bugs.

B. Avionics Software Testing

The embedded software system used in airplanes and flight control hardware which is subject to legally mandated

reliability and safety standards, is necessary by law to meet Federal Aviation Administration (FAA) DO-178B regulations, which sets it apart from traditional embedded software [2]. This software controls on several an aircraft's critical systems, from navigation to the black box, and must be perfect. As most software fails due to code errors, rigorous testing is necessary to guarantee that avionics software is reliable. Failure to do so can result in very expensive blunder, harm or loss of life. Due to the reliability of these testing procedures can be seen every day in the successful takeoff and landings of planes, commercial or otherwise. The standards that avionics software is being tested against the various strictest in the industry, as still one flaw could send a plane thousands of miles off course or cause breakdown that can result in death or other disasters. This high degree of protection not only insures against failure, but provides peace of mind to operators, passengers and pilots, as well as security against liability.

C. Dynamic Testing

Embedded software testing techniques can be broadly divided into two forms: Static testing (ST) and Dynamic testing (DT). Static testing examines the code that has been written by developers to make sure it is coherent, syntactically correct and have no problematic inconsistencies. While performing static testing, the code does not actually need to be run, since it is the code that is being validated. On the other hand, dynamic testing is used to test the functionality of the code while running. This type of code verification frequently involves in running test cases and systematically feeding a range of variables into the program. By examining the software's output and checking whether it is reliable with expectations, it is possible to determine whether the code contains errors. To conduct dynamic testing, the code can be compiled and run. To ensure, an adequate broad range of test cases are used in dynamic testing, while many developers choose to employ automated software testing tools. By automating the dynamic testing process, software developers are capable to conduct thorough, robust tests throughout the development process.

D. Embedded Code Coverage

Embedded software is used in several highly complicated, safety-critical systems, such as avionics, medical devices, railways and defense systems. While creating embedded software, it is very important that developers conduct thorough and efficient testing processes, to ensure their products are free from potentially expensive or dangerous defects. One measure used in this approach is known as Embedded Code Coverage (ECC). The ECC testing examines the software code without running the code and inspecting the resulting output. Several aspects of developer's code are tested during embedded code coverage. These include statement coverage, function coverage, condition coverage and decision coverage. For each, the ECC testing process inspects the software code to ensure every part of the code has been executed.

E. Test Driven Development

Test-driven development (TDD) [18] is a software development approach very much associated with both the set of practices and the agile development framework known as Extreme Programming (EP). The test-driven development (TDD) approach centers around the generation of tests before programmers write each portion of code for a software project [5]. The test-driven development needs that a project can be divided into small iteration, each of which produces a deliverable unit. Test-driven development (TDD) differs from other approaches to testing mainly in that it involves test generation before the program code itself is written [8]. In this approach the developer must think about test cases based on the interface rather than the design of the code. As per a paper published by Matthias Muller and Frank Padberg on the return on investment offered by test-driven development (TDD) focuses on typically results in shorter overall code implementation time.

F. Evolutionary Algorithms

Evolutionary algorithms (EA) are powerful search and optimization heuristics derived from the classic evolution theory, which are applied on computers in the majority of cases. The general idea is that the individuals of a population reproduce, which meet a definite selection criteria and the other individuals of the population expire, then the population will converge to those individuals that meet the selection criteria. If unsatisfactory reproduction is added the population can begin to discover the search space and will move to individuals that have a bigger selection probability and that inherit this property to their descendants. This is based on the basic rule of the Darwinistic evolution theory, which can be described in short as the "survival of the fittest" [21]. The general EA process is described (see figure 1). An EA heuristic follows this basic scheme:

initialize random population A(s=0)
repeat
    evaluate fitness of all a_i from A(s)
    select the fittest a_i as parents B(s) from A(s)
    reproduce descendants C(s) from B(s)
    A(s+1) = C(s)
until break criteria is met

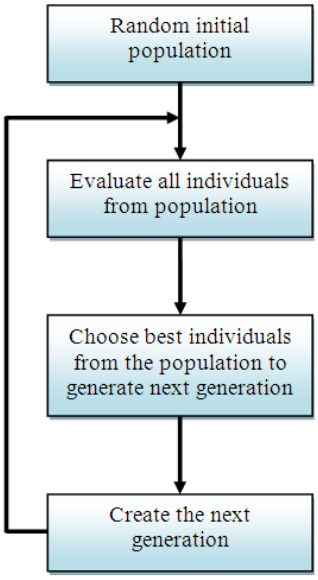

Fig. 1 General EA process

III. THE TESTING FRAMEWORK

Before you start designing tests, it's essential to have a clear understanding of why you are testing. This consideration influences which tests you stress and how early you begin testing. In general, testing done for four reasons:

- To find bugs in software
- To reduce risk to both users and the company
- To reduce development and maintenance costs
- To improve performance

To Find the Bugs. One of the initial essential results from theoretical computer science is a proof, known as the Halting Theorem, that it is not possible to show that an arbitrary program is correct.

To Reduce Costs. In 1990, HP sampled the errors cost in the development of software throughout the year. The answer, $400 million, requires to HP into a completely new effort to eliminate mistakes in writing software. The earlier a bug is found, the less expensive it is to fix. After releasing product, the cost of finding errors and bugs is considerably higher than during unit testing, for example (see fig. 2 below).

To Improve Performance. The performance of the system maximizes by the testing. Finding and eliminating wrong and inefficient code can help ensure that the software makes actual to the hardware and thus it avoids the "hardware re-spin".

A. *Evolutionary Testing*

Evolutionary testing (ET) is demonstrated by the use of Meta Heuristic Search Techniques (MHST) for test case generation (see fig. 3). The transformation of an optimization problem is only aim of such testing. The input domain of the test objective figure out the search space in which one searches for test data that obtained the respective aim of testing. As well as, a numeric consideration of the test aim is required. This numeric consideration is used to define suitable objective functions for the assessment of the generated test data. During the test data evaluation, different objective functions emerge according to test aim is pursued. The outcome of the non-linearity of software (if-statements, loops, etc.) the conversion of test problems to optimization tasks is mostly results in discontinuous, non-linear, and complex search spaces. The search methods like hill climbing are not suitable in such cases. Hence, meta-heuristic search methods are used, e.g. simulated annealing, evolutionary algorithms, or taboo search. In this research work, evolutionary algorithms are considered to generate test data because their robustness and suitability for the resolution of different test tasks has already been proven in earlier work [14], [7], [19].

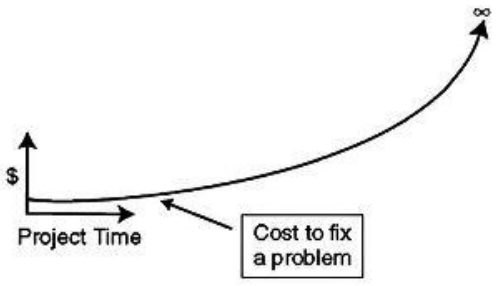

Fig. 2 The cost to fix a problem

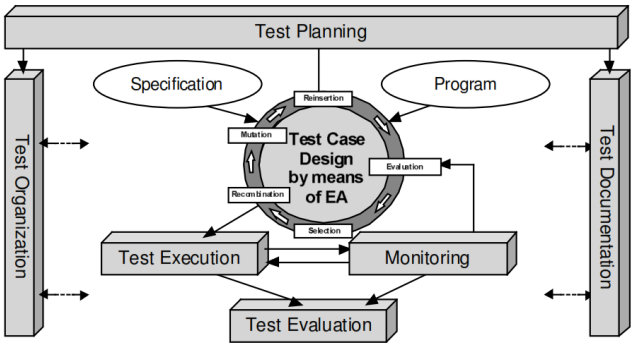

Fig. 3 Structure and interaction of test activities including test case design by means of evolutionary algorithms

In short, evolutionary testing can be used to systemize and automate the testing of non-functional properties and to build test cases for conventional testing methods. Our main focus is on structural testing. An executable test object and its interface specification are required during the application of evolutionary testing. For generation of automated test cases using structural testing the source code of the test object must be available to enable its instrumentation. Joachim Wegener in his paper described the interaction of the evolutionary algorithm with the other testing activity components [4]. Fig. 4 shows the structure of evolutionary testing from regards of the evolutionary algorithm. During the evaluation of the individuals the interaction with the other testing activities occurred. Every individual within the population consider as a test datum with which the test object is executed. By monitoring execution for each test datum and the objective value is determined for the corresponding individual. After this, selection of population members has done according to their fitness and subjected to recombination and mutation to generate new offspring. Make sure that the test data generated are in the input domain of the test object. The evaluations of the offspring individuals are done by executing the test object with the test data in the corresponding domain. A new population is generated by the combination of offspring and parent individuals, according to the defined survival procedures. This complete process repeats itself until the test objective is satisfied or another given the termination condition is reached [4].

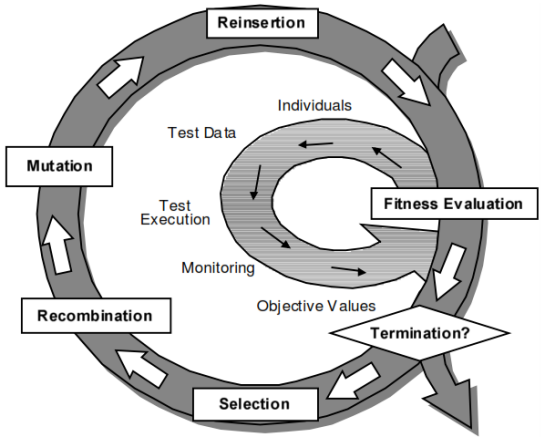

Fig. 4 Structure of Evolutionary Testing

B. *Structural Testing*

Structural testing is well-known in industrial practice and stipulated in many software-development standards [6]. Structural test aims the execution of all statements (statement coverage), all branches (branch coverage), or all conditions with the logical values *True* and *False* (condition coverage). Generally it is applied to unit testing rather than integration testing or system testing. The application of evolutionary testing to structural test is used for automatic generation of a quantity of test data that leads to the best possible coverage of the particular structural test criterion. In the case of statement testing each program statement must execute at least once. In the case of branch testing the empty branches must be executed. The test goals are based on the assumptions where does not include each statement and all branches have been executed at least once, does not present a thorough check of the test object. The estimate level provides a count for an individual that gives the number of branching nodes available between the desired program structure and program structures covered by the individual. The branching nodes that contain an outgoing edge are taken into account for this calculation which results in a miss of the desired program structure. Figure 5 shows the branching nodes of a sample program structure [4]. The program flow chart contains four branching nodes that could result in a miss of the target node. The corresponding approximation level (level 4 to level 1) is assigned to each node. The execution of an individual that misses the target node with the approximation level 2 is shown in the figure. The branching nodes pass by individual in the levels 3 and 4 are desired, but it misses the target node at level 2. How different individuals executing the same program path are distinguish by the calculation of the local distance. Figure 5 illustrates the calculation of distance to the execution of the other program path for the individual by means of the branching statements (conditions) in the branching node in which the target node is missed.

C. *Evolutionary test environment*

Figure 6 presents the organization of the evolutionary test environment and the information exchange between these tools. At the time of test execution the test control and the evolutionary algorithm toolbox (GEATbx) are employed.

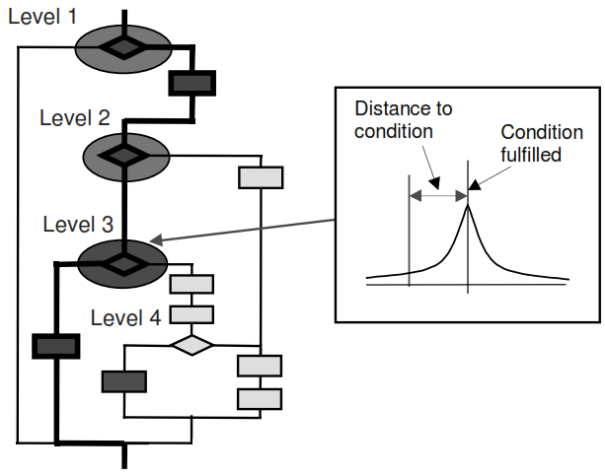

Fig. 5 Local distance calculation (objective function for structural testing) [4]

Following components constitute the test preparation:

1) Parser
2) Interface specification
3) Instrumenter and
4) Test driver generator

1) Parser

The parser is used to identify the functions in the source files that form the possible test objects. It finds out all required structural information on the test objects. Data-flow and control-flow analysis are carried out for every test object determine the interface, the control-flow graph, and the contained branching conditions with their atomic predicates and semantic information on the used data structures.

2) Interface specification GUI

The test object interface determination by the parser is required with more accuracy to ensure efficient test data generation and to avoid the generation of unwanted test data. This will be achieved by the graphical user interface of developed tool environment that displays the test objects and their interfaces as they have been identified by the parser. The tester can enter logical dependencies between different input parameters and limit the value ranges for the input parameters, which considered during test data generation. Also there is a possibility to enter initial values for input parameters. As a result, data of an already existing functional test or test data of a previous test run, as well as particular value combinations for single input parameters, can be used as an initial point for test data generation (seeding).

3) Instrumenter

The third component in the environment is the instrumenter that enables test run monitoring. The instrumentation has been carried out in the branching conditions of the program in order to eliminate influences on program behavior. The structural test criterion and the instrumentation of the branching conditions are always same as well as independent of the selected structural test criterion. This also provides information on the program branches and statements executed by an individual.

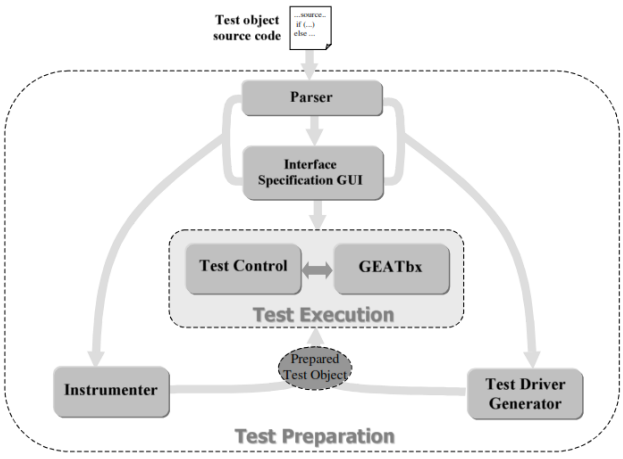

Fig. 6 Components of the evolutionary test environment [20]

4) Test driver generator

The test driver generator makes a test platform that calls the test object and the generated individuals. Test driver generator returns the observed results provided by the

execution of the instrumented test object to the test control. The individuals are mapping onto the interface of the test object when the test object is called by the test driver. The consideration of user specifications for the test object interface is taken into account is important. Individuals that represent invalid input are extracted and assigned a low fitness value.

IV. Conclusion

The detailed test of embedded systems could include a number of demanding testing tasks. The test case design for diverse test objectives is complex to master on the basis of conventional structure-oriented and function-oriented testing methods. In addition, automation of test case design is challenging. The test tool environment has been developed that applies evolutionary testing to C programs to automatic generation of test data for structural tests. The idea behind evolutionary testing is to search for significant test cases in the input domain of the SUT with the help of evolutionary algorithms. In the form of evolutionary testing a new test method for testing embedded systems is provided, that allows the complete automation of test case design for different test objectives. By the full automation of the evolutionary testing, the system could be tested with a large number of different input situations. An interface specification of the system under test (SUT) is needed to guarantee the generation of valid input values is only prerequisite for the application of evolutionary tests. Due to very good results in the application field, evolutionary testing seems to have the potential to increase the efficiency and effectiveness of present test processes. Thus, evolutionary tests contribute to reduction of development costs and to quality improvement. Extended evolutionary algorithms combine global and local search procedures in a number of subpopulations, robust optimization results are obtained for a large number of different testing tasks. Evolutionary testing could directly benefit from the improvements by incorporating new search techniques into test data generation, thus it imparts in an increase in the effectiveness and efficiency of the tests. According to the study, evolutionary testing can be applied to statement tests, condition tests, segment tests, and branch tests. Testing on multiple-condition is yet to complete. The test environment for structural testing of object-oriented programs will be a future work.

REFERENCES

[1] B .Broekman, E. Notenboom, "Testing embedded software," Addison-Wesley, 2003.
[2] RTCA/DO-178B Software Considerations in Airborne Systems and Equipment Certification.
[3] Capability Maturity Model for Software, Software Engineering Institute, Carnegie Mellon University.
[4] Joachim Wegener, Kerstin Buhr, Hartmut Pohlheim, Daimler Chrysler AG, "Automatic Test data Generation for Structural Testing of Embedded Software Systems by Evolutionary Testing", vol. 21, pp. 324 – 332, 2002.
[5] David Astels, Test Driven Development: A Practical Guide, Upper Saddle River, NJ: Prentice Hall PTR, 2003.
[6] IEC 65A Software for Computers in the Application of Industrial Safety-Related Systems (Sec 122).
[7] Jones B. F., Eyers D. E., Sthamer H. H., A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing. The Computer Journal, vol. 41, no. 2, pp. 98 – 107, 1998.
[8] M. Alles, D. Crosby, C. Erickson, B. Harleton, M. Marsiglia, G. Pattison, C. Stienstra. "Presenter First: Organizing Complex GUI Applications for Test-Driven Development," accepted at Agile 2006 conference, Minneapolis.
[9] Pravin Karmore and Pradeep Butey, "Analysis of Model-based Testing Methodology for Embedded Systems", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 6, Issue 5, pp. 308-314May- 2016,.
[10] Pohlhem H., "Genetic and Evolutionary Algorithm Toolbox for Matlab". http://www.geatbx.com/, 1994-2002.
[11] Pohlhem H., "Evolutionäre Algorithmen – Verfahren", Operatoren, Hinweise aus der Praxis. Berlin, Heidelberg: SpringerVerlag, 1999. http://www.pohlheim.com/eavoh/
[12] Pohlhem H., "Testing the Temporal Behavior of Real-Time Software Modules using Extended Evolutionary Algorithms". in Banzhaf, W. (ed.): GECCO'99 - Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA: Morgan Kaufmann, p. 1795, 1999.
[13] Pravin Karmore and Pradeep Butey, "The Unique Challenges of Test Automation on Embedded Systems", International Journal on Information Technology Management, DMIETR, vol. 1, pp. 18-25, Dec., 2015.
[14] Sthamer H. H., "The Automatic Generation of Software Test Data Using Genetic Algorithms". Ph D Thesis, University of Glamorgan, Pontyprid, Wales, Great Britain, 1996.
[15] Tracey N., Clarck J, "An Automated Framework for Structural Test-Data Generation", Proceedings of the 13th IEEE Conference on Automated Software Engineering, Hawaii, USA.
[16] Watkins A, "A Tool for the Automatic Generation of Test Data Using Genetic Algorithms", Proceedings of the Software Quality Conference '95, Dundee, Great Britain, pp. 300-309, 1995.
[17] Wegner J, "Evolutionärer Test des Zeitverhaltens von Realzeit-Sytemen (Evolutionary Testing of the Temporal Behaviour of Real-Time Systems)". Shaker Verlag, 2001.
[18] Pravin Karmore and Pradeep Butey, "Study of Testing Approaches for Heterogeneous Designs of Embedded Systems", Proceedings of 2nd National Conference on Advancement in the Era of Multi Disciplinary Systems, TERII, Kurukshetra, Hariyana, India, Elsevier publication, pp. 703, 2013.
[19] Wegner J, Grochtmann M, "Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing". Real-Time Systems, 15, pp. 275-298, 1998.
[20] Wegner J, Baresel A, "Evolutionary Test Environment for Automatic Structural Testing. Information and Software Technology, Special Issue devoted to the Application of Metaheuristic Algorithms to Problems in Software Engineering, vol. 43, pp. 841 – 854, 2001.
[21] Darwin, C.R.: "The Origin of Species", London (John Murray) 1859.

Authors Profile

Pravin Y. Karmore pursuing Ph.D. in Computer Science from RTM Nagpur University, Nagpur. He is obtained Master in Computer Applications degree from RTM Nagpur University, Nagpur. And M.Phil. degree in Computer Science from Alagappa University, Karaikudi. At present he is working as Assistant Professor at Dept. of Computer Applications, Shri Ramdeobaba College of Engineering and Management, Nagpur.

Dr. Padeep K. Butey obtained M.Sc. degree and PGDCS&A from RTM Nagpur University, Nagpur. He obtained his Ph.D. degree in Computer Science from RTM Nagpur University, Nagpur. Now he is working as Associate Professor and Head at Dept. of Computer Science, Kamla Nehru College, Nagpur. He has published more than 35 research papers in various national and international conferences and journals.