Volume 2, No. 5, April 2011

Journal of Global Research in Computer Science

ISSN-2229-371X

RESEARCH PAPER

Available Online at www.jgrcs.info

ANTI-RANDOM TEST GENERATION IN SOFTWARE TESTING

Kulvinder Singh[1], Seema Rani[*2] and Rekha Rani[3],
CSE Dept. Kurukshetra University, Kurukshetra,India
kshanda@rediffmail.com[1], seemagure7@gmail.com[2], rekha_dove@rediffmail.com[3]

Abstract: The main purpose of software testing is found a error and then correct it. Random testing selects test cases randomly but it does not explore the previous information. Anti-random testing in which each test applied its total distance from all previous tests is maximum. Anti-Random testing is a variation of random testing, which is the process of generating random input and sending that input to a system for test. In which use hamming Distance and Cartesian Distance for measure of difference.

Keyword: Software Testing, Anti-Random testing, Random Testing

INTRODUCTION

Testing is found out how well something is works. In the human beings, testing says what level of knowledge has been acquired. In computer software development, testing is used as checkpoints in the overall process to determine that objectives are being met. For example in software development, product objectives are sometimes tested by product user. When the design is completed, follows the coding and the finished the code then tested the unit level by each programmer; in the component level the group of programmers involved; and at the system level when all components are combined together. At early stages, a service may also be tested for usability**.** From the point of view testing a script, a error is any circumstance where the script does not know what it is supposed to do. An error may or may not cause the script to actually crash.

Defect can be caused by flaws in the application software or by flaws in the application specification. For example, unexpected or incorrect results can be from the errors made during the construction peroid, or from an algorithm incorrectly defined in specification. Testing is assumed to mean executed software and finded errors. This type of testing is known as dynamic testing, it is not the most effective way of testing. Static testing, inspection and validation of development requirements, is the most effective and cost efficient way of testing. A structured approach of testing should be used both dynamic and static testing techniques.

WHAT IS SOFTWARE TESTING?

The primary purpose of software testing is found a error or defects and then correct it. Software testing is often viewed as either an exercise to show that a program is correct or incorrect. A testing technique can show a program is not correct, but infinite number of testing are required showing a program is correct. Software testing is an investigation conducted to provide stakeholder with information about the quality or services under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of software. Test technique, the process of executing a program or application with the intent of finding software bugs (by Wikipedia)

In software testing technique we create test case for detect a error in the program. After a test case design, if the program is execute successfully that means error is detected. Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results [1]. Software Testing is the process of executing a program with the intent of finding errors [2] Or, it involves any activity aimed at evaluating an attribute of a program and determining that it meets its required results [1]. Software is not other physical processes where inputs are received and outputs are produced. Where software differs is in the manner in which it fails. Most physical systems fail in a fixed set of ways. By contrast, software can fail in many different ways. Detecting all of the different failure modes for software is generally infeasible [3].

Software bugs will almost always exist in any software with different size: not because programmers are careless or irresponsible, but because the complexity of software is generally intractable -- and humans have only limited ability to manage complexity. It is also true that for any complex systems, design defects can never be completely ruled out. Discover the designed defects in software, is totally difficult, for the same reason of complexity.

If a failure during testing and the code is changed, the software may now work for a test case that it didn't work for previous program. But its behavior on pre-error test cases that it passed before can no longer be guaranteed. To account for this possibility, testing should be restarted. The expense of doing this is often prohibitive [3].

An interesting analogy parallels the difficulty in software testing with the pesticide, known as the Pesticide Paradox [4]: Every method you use to find bugs leaves a residue of subtler bugs against which those methods are ineffectual. But this alone will not guarantee to make the software better, because the Complexity Barrier [4] principle states: Software complexity grows to the limits of our ability to manage that complexity. By eliminating the previous bugs you allowed another escalation of features and complexity, but his time you have subtler bugs to face, just to retain the reliability you had before. Society seems to be unwilling to limit complexity because we all want that extra bell, whistle, and feature interaction. Thus, our users always push us to the complexity barrier and how close we can approach that barrier is largely determined by the strength of the techniques we can wield against ever more complex and subtle bugs. [4] . Testing is must than debugging. The purpose of testing can be quality assurance, verification and validation [5].

TESTING METHOD?

Software testing methods are basically divided into two methods that are white box testing and black box testing. There are two approaches are used to describe the point of view that a testing engineer used when designed test cases.

White box testing:-

It is a clear box testing method of testing software that tests internal working of an application as opposed to its black box testing. An internal working of the system or the programming skills, are required and to design test cases. The tester selects the input to exercised paths through the code and determines the correct output. White box testing is performed based on the knowledge of how the system is implemented. White box testing requires knowing what makes software secure or insecure, how to think like an attacker, and how to use different testing tools and techniques [6].

White box testing is very different from black box testing. White box testing is a testing that takes into account the internal mechanism of a system or component [7]. White box testing is also called structural testing, clear box testing, and glass box testing [10]. We use different type of testing in software testing.

Basically six type of testing are used in white box testing that is:-unit, integration, regression, acceptance, function and beta testing, but in case of white box testing we use unit testing, integration testing and regression testing.

- Unit testing:-

  It is a testing of hardware or software groups of related units [7]. A unit is a software component that cannot be subdivided into another component [7]. Unit testing is done on a small piece, or a code of unit. This unit is usually a class. When a unit is integrated into the main code, it is more difficult to find a bug in that unit [12].

- Integration testing:-
  It is a testing it may be hardware component or software component or both are combined and tested to evaluate the integration between them [7]. Integration testing looks at how all components at an application interact [12].
- Regression testing:-

  That is selectively retesting of a system to verify that modifications have not caused unintended effects and that the system or component still complies with its specification requirements [7]. Regression testing verify that modifications to the system have not damaged the whole of the system unit test and integration test can be rerun in regression testing to verifies that modification to the application work properly [12].

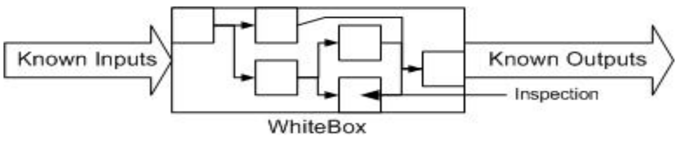

Fig: 1 white box testing [12]

White box test design technique include:-

- Control flow testing:-

  In computer science, control flow refers to the order in which the individual statements, instructions, function or a declaratives program are evaluated.
- Data flow testing:-

  Data-flow testing looks at the life-cycle of a particular piece of data in an application. By looking for patterns of data usage, risky areas of code can be found and more test cases can be applied [52].
- Branch testing:-

  Branch testing A test strategy seeking to choose test data values that lead to the testing of each branch in a program at least once (branching occurring at each decision point). It is equivalent to finding a set of paths through the control flow graph whose union covers all the arcs of the graph. Branch testing normally requires more tests than statement testing but fewer than path testing [53].
- Path testing

  Path testing A test strategy equivalent to finding all possible paths through the control-flow diagram of a program. Testing each path at least once is a typical test strategy, but for much real software complete path test coverage would require an impracticably large test run/time. Path testing almost always requires more test runs than either branch testing or statement testing [53].

Black box testing:-

Black box testing treats the software as knowledge of internal implementation. Black box testing method of testing software that test the functionality of an application as opposed to its internal structure or working (by Wikipedia)

Input
Blackbox
Output

Fig.2 Black box testing (by Wikipedia)

Specific knowledge of the application's code or internal structure and programming knowledge in general is not required. Test cases are built around specification and requirements

The black-box approach is a testing in which test data are derived from the specified functional requirements [8]. It is also termed data-drive [2], or requirements-based [1] testing. Because only the functionality of the software module is of concern, black-box testing also refers to functional testing -- a testing method emphasized on executing the functions and examine of their input and output data [13]. The tester treats the software under test as a black box -- only the inputs, outputs and specification are visible, and the functionality is determined by observing the outputs according to inputs. In testing, various inputs are exercised and the outputs are compared against specification to validate the correctness. All test cases are derived from the specification. No implementation details of the code are considered. The research in black-box testing focuses on how to maximize the effectiveness of testing with minimum cost or the number of test cases. It is not possible to exhaust the input space, but it is possible to exhaustively test a subset of the input space. Partitioning is the common techniques. If we have partitioned the input space and assume all the input values in a partition is equivalent, then we only need to test one represent value in each partition to sufficiently cover the whole input space. Domain testing [7] partitions the input domain into regions, and considers the input values in each domain an equivalent class. Domains can be exhaustively tested and covered by selecting a represent value in each domain. Experience shows that test cases that explore boundary conditions have a higher payoff than test cases that do not. Boundary value analysis [2] requires one or more boundary values selected as representative test cases.

Typical black-box test design techniques include:

- Decision table testing

  Decision tables are a precise yet compact way to model complicated logic. Decision tables, like flowcharts and if-then-else and switches- case statements, associate conditions with actions to perform, but in many cases do so in a more elegant way [by Wikipedia].

- All pair testing

  All-pairs testing or pair wise testing is a combinatorial software testing method that, for each pair of input parameters to a system, tests all possible discrete combinations of those parameters. Using carefully chosen test vectors, this can be done much faster than an exhaustive search of all combinations of all parameters, by "parallelizing" the tests of parameter pairs. The number of tests is typically O(*nm*), where *n* and *m* are the number of possibilities for each of the two parameters with the most choices.

- State transition table

  In automata theory and sequential logic, a state transition table is a table showing what state a finite semi automaton or finite state machine will move to, based on the current state and other inputs. A state table is essentially a truth table in which some of the inputs are the current state, and the outputs include the next state, along with other outputs. A state table is one of many ways to specify a state machine, other ways being a state diagram.

- Equivalence partitioning

  Equivalence partitioning is a software testing technique that divides the input data of a software unit into partitions of data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed.

  In rare cases equivalence partitioning is also applied to outputs of a software component, typically it is applied to the inputs of a tested component. The equivalence partitions are usually derived from the requirements specification for input attributes that influence the processing of the test object. An input has certain ranges which are valid and other ranges which are invalid. Invalid data here does not mean that the data is incorrect; it means that this data lies outside of specific partition. This may be best explained by the example of a function which takes a parameter "month". The valid range for the month is 1 to 12, representing January to December. This valid range is called a partition. In this example there are two further partitions of invalid ranges. The first invalid partition would be <= 0 and the second invalid partition would be >= 13[by Wikipedia].

- Boundary value analysis

Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values. Values on the edge of an equivalence partition or at the smallest value on either side of an edge. The values could be either input or output ranges of a software component. Since these boundaries are common locations for errors that result in software faults they are frequently exercised in test cases.

For an example, if the input values were months of the year expressed as integers, the input parameter 'month' might have the following partitions:[by Wikipedia]

TESTING METHODOLOGY

There are various methodologies available for developing and testing software. The methodology you choose depends on factors such that the nature of project, the project schedule, and resource availability. Most software development projects involve periodic testing, some methodologies focus on getting the input from testing early in the cycle rather than waiting for input when a working model of the system is ready. Those methodologies that require early test involved have several advantages, but also involve tradeoffs in terms of project management, schedule, customer interaction, budget, and communication among team members.

There are some testing methodologies are used:

- Random Testing
- Anti-Random Testing

Random testing:-

Random testing is a form of functional testing that is useful when the time needed to write and run directed tests is too long. Release criteria include a statement about the amount of random testing that is required.

One of the big issues in random testing is to know when a test fails. As with all testing, an oracle is needed. You could rely in assertions in the code as your role oracle. In other situations, common with hardware development. You have two different implementations of the same specification, one is ‘the golden model’, and other is the ‘implementation’. If they both agree to a defined accuracy then the test passes.

Random testing is useful if it does not find many defects per time interval, since it can be performed without manual intervention. An hour of computer time can be much less expensive than an hour of human time When doing random testing you must ensure that your test are sufficiently random, and that they cover the specification repeating the same test for two weeks doesn’t tell you anything[14].

It is often claimed, correct, that random testing is less efficient than directed testing. But you should consider the time needed to write a random test generator and the time to write a set of directed tests. Once you have a random test generator, your computer can work 24 hours a day, at most ,6 productive hours in a day; then the efficiently of the random tests is effectively increased by a factor of 4.

Anti-Random testing**:-**

“The concept of anti-random testing in which each test applied is chosen such that its total distance from all previous tests is maximum[15]”.

Some information are available in black box testing, random testing does not explored that information. This information depends upon the previous tests applied. Now we use a new approach that use this information and allowed like generation to be done automatically. We call this approach Anti-random testing. In Anti-random testing we can test any approach without randomly. In which we can use any test any time.

Some problems are occurring during Anti-random testing. First of all problem of generating Anti-random sequence is considered for Boolean inputs for make each new test as different, we use Hamming Distance and Cartesian Distance for measure of difference. In general, the input variable for a program can be numbers, character etc. Here we use Anti-random testing so we can convert this input code into binary, then this code allow binary anti-random sequence to be decoded into actual inputs, in such cases, Anti-random testing to find defects sooner, and reducing the overall test and debugging time.

Anti-random testing is a variation of random testing, which is the process of generating random input and sending that input to a system for test. In many situations, random test input does not have an associated expected return value. In such situations, the purpose of random testing is to try to generate a system failure of some sort, such as a hang. Research studies have shown that pure random testing is relatively less effective at discovering bugs than other testing paradigms, such as equivalence partition testing and boundary value analysis testing. However, random testing is appealing because it is typically quick and easy to implementation. The idea of Anti-random testing appears to have been introduced by the field of hardware testing. Essentially, Anti-Random testing generates an input set of random values, where the values are as different as possible from each other. The assumption is that similar input to a system will expose similar types of bugs, and therefore an input set that contains values that are very different [56].

Terms used in anti-random sequence:-

- Anti-random test sequence:-

  It is a Anti-random test sequence such that Ti that is satisfied some criteria with respect to all test to Ti…..Ti-1.

- Distance:-

  Measurement of different two vectors like Ti and Tj.

- Hamming distance:-

  That is also the test sequence that is the number of bits in which two binary number are differ.

- Cartesian distance:-

  That is the difference between two vector.

  $A = \{a_N, a_{N-1}, \cdots, a_1, a_0\}$ & $B = \{b_N, b_{N-1}, \cdots, b_1, a_0\}$ is given

  $$CD(A,B) = \sqrt{(a_N - b_N)^2 + (a_{N-1} - b_{N-1})^2 + \ldots + (a_0 - b_0)^2}.$$

  (1)Since the two test vectors are binary, so (1) can be written as

  $$CD(A,B) = \sqrt{|a_N - b_N| + |a_{N-1} - b_{N-1}| + \ldots + |a_0 - b_0|} = \sqrt{HD(A,B)}.$$

- Total hamming distance:-

  In any number is the sum of its hamming distance with respect to all previous number.

- Total Cartesian distance:-

  In any number is sum of its Cartesian distance with respect to all previous number.

- Maximal Distance Anti-random Test Sequence (MDATS):- It is a test sequence such that each test ti is chosen to make the total distance between ti and each of to, tl ... ti-1 is maximum for all possible choices of ti. We used Hamming distance and Cartesian distance to construct MHDATSs and MCDATSs.
- If a sequence B is obtained by reordering the variables of sequence A, then B is a variable- order-variant (VOV) of A.

  Theorem 1: If a sequence B is variable-order-variant of a MHDATS A, then B is also a MHDATS. The theorem follows from the fact that Hamming or Cartesian distance is independent of how the variables are ordered.

- If a binary sequence B is obtained by changing the polarity (i.e. inverting all the values) of one or more variables of a sequence A, then B is termed a polarityvarknt of A.

  Theorem 2: If a sequence B is a polarity-variant of a MHDATS A, then B is also IMHDATS. The theorem follows from the fact that for a pair of vectors the distance remains the same, if the same set of variables in both are complemented

Checkpoint Encoding

Generation of binary anti-random test sequence is also a problem. For reduces this problem we use the checkpoint encoding scheme is introduced.

Checkpoint Encoding is the process of representation of any input domain of a software system into a binary valued domain. Any input data value can now be translated with minimal loss of information into a binary valued string. This abstraction of any application domain into an uniform format allows a variety of testing techniques to be applied consistently and universally. Anti-random testing is one such scheme that takes advantage of binary representation that checkpoint encoding provides and has shown good results. The checkpoint encoding process is usually applied manually and in an arbitrary fashion. The result varies depending on the choices made by individual test engineer carrying out the process [57].

In common cases, the inputs can be numbers; alphanumeric characters as well as data structures composed using them. In such cases also we would like to maximize the effectiveness of testing. It is possible for one to define 'distance' and use them for constructing anti-random sequences in such cases also.

Example 8: Let us consider two real variables x and y which can range from 0 to 1. The following then is a MCDATS.

0 0

1 1

0.5 0.5

So, defining 'distance' can be difficult for data structures. Also for a program, the input variables can be of different types and ranges, which will make construction of anti-random sequences extremely hard.

We here propose an encoding approach which will convert the problem to constructing binary anti-random sequences. The approach is based on domain and partition analysis and the concepts of equivalence partitioning, revealing sub domains and homogeneous sub domains. The technique partially encodes an input into binary, such that sample points desired can be obtained by automatic translation.

These sample points, termed checkpoints here, are strategically selected such that they are likely to span most types of variations in the program behavior with respect to each input. To illustrate the approach let us consider this simple example. For convenience, we use a square bracket ("[„ or "I") to indicate inclusion of the endpoint and a parenthesis ("(" or ")") to indicate exclusion. For testing, it is important to test for illegal input values because the program must respond correctly to those inputs, as we see in the following example. The range of illegal values should be regarded as one or more additional equivalent partitions.

If testing is less than exhaustive, then maximum distance anti-random testing (MDAT) is likely to be more efficient than either random or pseudorandom testing. Even when exhaustive testing is feasible, MDAT is likely to detect the presence of faults earlier.

TESTING DATA GENERATION PROCEDURE

Automatic test generation is designed to ease the test effort. Here we have used three different approaches for automatic test generation.

1. Anti-random with checkpoint encoding
2. Random with checkpoint encoding.
3. Random without checkpoint encoding

In Anti-random testing used some basis concepts.

Procedure 1. Construction of a MHDATS (MCDATS)

Step 1. For each of *N* input variables, assign an arbitrarily chosen value to obtain the first test vector. This does not result in any loss of generality.

Step 2. To obtain each new vector, evaluate the THD (TCD) for each of the remaining combinations with respect to the combinations already chosen and choose one that gives maximal distance. Add it to the set of selected vectors.

Step 3. Repeat Step 2 until all 2^N combinations have been used, or until the desired number of vectors have been generated.

This procedure uses exhaustive search. As we will see later, the computational complexity can be greatly reduced. Note that the procedure ensures that a vector will not be repeated.

To illustrate the process of generating MDATS, we consider in detail the generation of a complete sequence for three binary variables.

Procedure 2. Expansion of MCDATS

Step1. Start with a complete MHDATS of *N* variables, $x_{N-1}, x_{N-2}, \dots, x_1, x_0$.

Step2. For each vector t_i, $i = 0, 1, \dots, (2^N - 1)$, add an additional bit corresponding to an added variable x_N, such that t_i has the maximum total HD (CD) with respect to all the previous vectors.

Procedure 3. Expansion and Unfolding of a MHDATS (MCDATS)

Step1. Start with a complete $(N-1)$ variable MHDATS (MCDATS) with 2^{N-1} vectors.

Step 2. Expand by adding a variable using Procedure 2. We now have the first ($2^N/2$) vectors needed.

Step3. Complement one of the columns and append the resulting vectors to the first set of vectors obtained in Step 2. Here, it would be convenient to complement the variable added in Step 2.

The above procedures have been implemented in a program called ATG. It generates MCDATSs which are also MHDATSs. The application of anti-random testing for software has been reported in. Three related approaches termed fast anti-random, random-like and maximum distance testing have recently been proposed that attempt to incorporate some of the features of anti-random sequences.

The CEAR test generation Scheme

The checkpoint encoding Anti-random testing (CEAR) scheme used here was proposed by Malaiya [55].This scheme integrated anti-random testing with checkpoint encoding and design to process input test vector on the automatically and to exercise the software under test. So, this making the scheme cost-effective. The CEAR scheme has three main components:

- The MHDATS (MCDATS) binary sequence generator.
- The Random value generator
- The binary to actual input translator

As shown in the figure, the CEAR is the collection of Software tools that produce actual input vector for the software under test. The MHDATS (MCDATS) binary sequence generator calculates the next binary vector in the anti-random sequence. The appropriate actual input test vector is generated and fed to the software under test.

CONCLUSION

Anti-random testing is a new test generation approach. Here we have demonstrated that it can achieve high-fault coverage much faster than the pseudorandom testing. It can also successfully applied for software testing and also test for VHDL descriptions.Its effectiveness is especially remarkable for finding faults. The scheme is well suited for IDDQ testing because it provides very good coverage with only a few vectors when black-box testing is used. One possible way to exploit the capabilities of anti-random testing is to use it until a suitable high coverage is obtained, and then to switch to deterministic testing.

In this testing we use the anti-random testing in software testing we use the testing with anti-random testing. In anti-random testing we already proposed hamming distance and Cartesian distance and more any sequence terms. Now we use the gray code to generate the anti-random testing.

REFERENCE

[1] Hetzel, William C., *The Complete Guide to Software Testing, 2nd ed.* Publication info: Wellesley, Mass. : QED Information Sciences, 1988

[2] Myers, Glenford J., *The art of software testing*, Publication info: New York : Wiley, c1979.

[3] http://www.rstcorp.com/definitons/software-testing.html.

[4] Boris Beizer, Software Testing Techniques. Second edition. 1990

[5] jiantao pan ,jpan@emu.edu,Dependable Embedded Systems
[6] cigital, 2005-09-26,updated 2009-07-27 by Ken van Wyk
[7] Beizer, Boris, Black-box Testing: techniques for functional testing of software and systems. Publication info: New York : Wiley, c1
[8] A standard for testing application software, William E. Perry, 1990
[9] William E. Howden. Functional program Testing and Analysis. McGraw-Hill, 1987.
[10] IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990), IEEE Computer Soc., Dec. 10, 1990.
[11] Computer Science Dept.Colorado State University,Yashwant K.Malaiya.
[12] Williams.Land Heckman.S,By:Sarah Heckman Updated:2008-08-25
[13] W. Howden, Software Engineering and Technology: Functional Program Testing, McGraw-Hill, New York, New York, 1987.
[14] *Dave whipp,2005 mailto:dave@whipp.name.*
[15] IEEE 2002Malaiya S.Jayasumana.Y.K Dept. of Comput. Sci., Colorado State Univ., Fort Collins, CO
[16] "Random testing revisited" Information and Software Technology, Volume 30, Issue 7, September 1988,Tsai WK,Loo.PS.
[17] "Adaptive random testing" 'the art of case diversity' Journal of Systems an*d* Software, Volume 83, January 2010, Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, T.H. Tse
[18] "Mirror Adaptive Random testing" Information and Software Technology, Volume 46, 1 December 2004, T.Y. Chen, F.-C. Kuo, R.G. Merkel, S.P. Ng
[19] "On the use of uniform random generation of automata for teaching" Electronic Notes in Theoretical Computer Science, Volume 253, 17 October 2009, Frédéric Dadeau, Jocelyn Levrey, Pierre-Cyrille Héam
[20] "Adaptive random testing based on distribution metrics" Journal of Systems and Software", Volume82,Issue9, September2009, Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu.
[21] "A uniform random test data generator for path testing" Journal of system and software, volume 83,Issue 12, December 2010,Petit M,Gotlieb.
[22] "Distributing test cases more evently in adaptive random testing, Journal of Systems And Software, Volume 81, Issue 12, December 2008, Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu
[23] "A more general sufficient condition for partition testing to be better than random testing" ,Information Processing Letters, Volume 57, Issue 3, 12 February 1996.
[24] "On the inverse power laws for accelerated random fatigue testing" ,International Journal of Fatigue, Volume 30, Issue 6, June 2008.
[25] "System for automated fatigue crack growth testing under random loading" ,International Journal of Fatigue, Volume 7, Issue 1, January 1985,R.Sunder.
[26] "A fuzzy representation of random variables: An operational tool in exploratory analysis and hypothesis testing" ,Computational Statistics & Data Analysis, Volume 51, Issue 1,1 November 2006, Gil González-Rodríguez, Ana Colubi, María Ángeles Gil
[27] "Testing For unit root processes in random coefficient autoregressive models",Journal of Econometrics, Volume 142, Issue 1, January 2008.
[28] "Testing the correlated random coeffient model", journal of Econometrics, Volume 158, Issue 2, October 2010, James J. Heckman, Daniel Schmierer, Sergio Urzua
[29] "Testing for random walk", Physics Letters A, Volume 362, Issues 2-3, 26 February 2007, Tomomichi Nakamura, Michael Small
[30] ."Testing the random walk hypothesis through robust estimation of correlation" Computational Statistics & Data Analysis, Volume 52, Issue 5, 20 January 2008, Andrei Semenov
[31] "Testing parallel random number generators" ,Parallel Computing, Volume 29, Issue 1, January 2003, Ashok Srinivasan, Michael Mascagni, David Ceperley
[32] "Testing for random effects and serial correlation in spatial autoregressive models" , Journal of Statistical Planning and Inference, Volume 140, Issue 4, April 2010, Gabriel V. Montes-Rojas
[33] "Examining random and designed tests to detect code mistakes in scientific software" ,Journal of Computational Science, Available online 22 December 2010
Diane Kelly, Robert Gray, Yizhen Shao
[34] "Bootstrap techniques and fuzzy Random variables: synergy in Hypothesis Testing with fuzzy data",Fuzzy Sets and Systems, Volume 157, Issue 19, 1 October 2006, Gil González-Rodríguez, Manuel Montenegro, Ana Colubi, María Ángeles Gil
[35] "Effectiveness and benefit-cost of peer-based workplace substance abuse prevention coupled with random testing", *Accident Analysis & Prevention, Volume 39, Issue 3, May 2007*, Ted R. Miller, Eduard Zaloshnja, Rebecca S. Spicer
[36] "Relative density of the Random r-factor proximity catch digraph for testing spatial patterns of segregation and association", Computational Statistics & Data Analysis, Volume 50, Issue 8, 10 April 2006, Elvan Ceyhan, Carey E. Priebe, John C. Wierman
[37] "The use of domination number of a random proximity catch digraph for testing spatial patterns of segregation and association", Statistics & Probability Letters, Volume 73, Issue 1, 1 June 2005, Elvan Ceyhan, Carey E. Priebe
[38] "Testing random number generators for microcomputer applications of Monte carlo simulation", Environmental

Software, Volume 6, Issue 4, December 1991, A.A. Jennings, S. Mohan
[39] "Testing for random effects in panel data under cross sectional error correlation-A bootstrap approach to the Breusch pagan test", Computational Statistics & Data Analysis", Volume 50, Issue 12, August 2006, Helmut Herwartz
[40] "Non-specification based approaches to logic testing for software", Information and Software Technology, Volume 44, Issue 2, 15 February 2002, Noritaka Kobayashi, Tatsuhiro Tsuchiya, Tohru Kikuno
[41] "Testing- based process for evaluating component Replaceability", Electronic Notes in Theoretical computer science,volume 236, 2 april 2009, Andres flores, Macario polo
[42] "Adaptive Random Testing: the ART of test case diversity" , Journal of Systems and Software, Volume 83, Issue 1, January 2010, Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, T.H. Tse
[43] "Adaptive software Testing with fixed-memory feedback", Journal of Systems and Software, Volume 80, Issue 8, August 2007, Kai-Yuan Cai, Bo Gu, Hai Hu, Yong-Chao Li
[44] "Robust performance testing for digital forensic tools" , Digital Investigation,Volume 6, Issues 1-2, September 2009, Lei Pan, Lynn M. Batten
[45] "Detecting buffer overflow via automatic test input data generation" , Computers & Operations Research, Volume 35, Issue 10, October 2008, C. Del Grosso, G. Antoniol, E. Merlo, P. Galinier
[46] "ShenHui Wu Malaiya, Y.K. Jayasumana, A.P." ,Dept. of Comput. Sci., Colorado State Univ., Fort Collins, IEEE oct 1998
[47] "High-Assurance System Engineering Symposium",1998. Proceedings, Third IEEE International IEEE Nov 1998.
[48] "PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering", IEEE Nov1997
[49] "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions",Journal of Systems and Software, Volume 79, Issue 5, May 2006, Yuen Tak Yu, Man Fai Lau
[50] Shen hui Wu, Sridhar jandhyala, Yashwant K. Malaiya and Anura P.Jayasumana computer science department, Colorado state university, January 2008
[51] Fast Anti- Random(FAR) Testgeneration to Improve The Quality of Behavioral Model Verification Tom Chen, Andre Baj, Amjad Hajjar, Anneliese K.Amschler Andrews and Anderson,2002.
[52] Laurie Williams and Sarah Heckman, Sarah Heckman, 2008-08-25
[53] JOHN DAINTITH."branch testing. "A Dictionary of Computing. 2004. *Encyclopedia.com.*
[54] ANTIRANDOM TESTING: GETTING THE MOST OUT OF BLACK-BOX TESTING Yashwant K. Malaiya Computer Science Dept. Colorado State University
[55] Y.K Malaiya, "Anti-random testing: Getting The Most Out Of Black Box Testing," Proc. International symposium on software Reliablity Engineering, Oct, 1995.
[56] Partial Antirandom string Testing by James McCaffrey, Oct. 2009.
[57] Issues in Automation of Checkpoint encoding for anti-random testing (2002) by Sanjay R.Pillai