A Smart Model for Web Programming Agile Testing

Belal Ayyoub[1]

[1] Albalqa Applied University

Received: 14 December 2013 Accepted: 4 January 2014 Published: 15 January 2014

Abstract
A Model that Minimizing the number of test operations on Web Programming Exhaustive Testing is introduced. The proposed model based for the first time to the best of our knowledge, the Hebbian trained matrix algorithm. The model problem is formed as Agility WEB Programming Methodologies optimization problem. Time and web developer efforts will be reduced on white box testing type. A full numerical solved example is introduced. Our numerical experience shows that our approach is promising especially for Web Programming Exhaustive Testing.

Index terms— exhaustive testing, Hebbian rules, white box, software testing, agile.

i.

introduction and related works esting in agile software development should provide the information that stakeholders need to make decisions and steer the development into the right direction. We can increase the value of testing most by improved intelligence earlier. The challenges of testing in agile development have to be solved! What information is the testing based on? What to be tested and what are the expected results? How to make testing, development and business collaborate? How to involve customer and business people in testing? How to test early so we can achieve the customer requirements in time? Can we use a method to do that smoothly? How to match in between exhaustive testing which take time and to be delivered with customer satisfaction?. The ultimate goal of function testing is to verify that the system performs its functions as specified in the requirements and there are no undiscovered errors left. Since proving the code correctness is not feasible, especially for large software systems, the practical testing is limited to a series of experiments showing the program behavior in certain situations. Each choice of input testing data is called a test case. If the structure of the tested program itself is used to build a test case, this is called a white box (or open-box) approach [1]. Several white-box methods for automated generation of test cases are described in literature. For example, the technique of [4] uses mutation analysis to create test cases for unit and module testing. A test set is considered adequate if it causes all mutated (incorrect) versions of the program to fail. The idea of testing programs by injecting simulated faults into the code is further extended in [10]. Another paper [11] presents a family of strategies for automated generation of test cases from Boolean specifications. However, as indicated by [10], modern software systems are too large to be tested by the white-box approach as a single entity. White-box testing techniques can work only at the subsystem level. In function tests that are aimed at checking that a complex software system meets its specification, black-box (or closed box) test cases are much more common. The actual outputs of a black-box test case are compared to expected outputs based on the tester's knowledge and understanding of the system requirements. Since the testers have time for only a limited number of test cases, each test case should have a reasonable probability of detecting a fault along with being non-redundant, effective, and of a proper complexity [6]. It should also make program failures obvious to the tester who is supposed to know the expected outputs of the system. Thus, selection of the tests and evaluation of their outputs are crucial for improving the quality of tested software with less cost. If the functional requirements are current, clear, and complete, they can be used as a basis for designing black-box test cases. Assuming that requirements can be re-stated as logical relationships between inputs and outputs, test cases can be generated automatically by such techniques as cause-effect graphs (see [8]) and decision tables [2]. Another method for automatic generation of test vectors from functional relationships is described in [3]. Several ways are proposed to determine, input-output relationships in tested software. Thus, a tester can analyze system specifications, perform structural analysis of the system's source code, and observe the results of system execution. While available system specifications may be incomplete or outdated, especially in the case of a "legacy" application, and the code may be poorly structured, execution data seems to be the most reliable source of information on the actual functionality of an evolving system. In this paper, we extend the idea initially introduced [1] that input-output analysis of execution data can be automated by Info-Fuzzy Network methodology of data mining [7] [9]. In [7] the proposed concept of IFN-based testing has been demonstrated on individual discrete outputs of a small business program. The current study evaluates the effectiveness of the Hebbian rules in Neural networks to let the system more intelligent (expert-system) and can be learned by previous cases tested methodology on a complex application having multiple continuous outputs. This is also deal with the question of determining the minimal number of training cases required to. The rest of the paper is organized as follows. Section 2 provides the methodology on the process testing and derived required paths. Section 3 presents the notation and definition of the proposed model. Section 4 describes a detailed of the proposed methodology. Finally, Section 5 summarizes the paper with initial conclusions and directions for future research and applications.

ii.

1 Methodology

Testing is the process of executing a program with the intent of finding errors." [13]. Assume that there are 520 possible different execution flows. If we execute one test per millisecond, it would take 3.170 years to test this program. See figure (1). All tests should be traceable to customer requirements and the uncover errors will be discovered then quickly. The system configuration can be represented mathematically by a graph, with nodes and representing links. Seefigures (1), (2). Our system will minimize the number of 'things in processes, minimize the size of 'things in processes establish a regular cadence deliver business value early, often and consistently, empower the team to create software that meets the customer needs. The ultimate objective of this paper is to give software developers procedures to enhance their ability to find acceptance testing -by customer-for which testability is an important consideration. Ideally, one would like to generate an acceptance solution algorithms that take as input the characteristics customers requirements as well as needs criteria, and produce as output an optimal path for solution, this is known as acceptance testing, and it is very difficult to achieve. However, we consider set of paths that will execute all statements and all conditions in a program, at least once that is already designed then we try to derive all the test cases, then the customer will select his steps of tests according to his requirements then a test related path will be chosen and get the test result, if the solution face customer satisfaction the test path will be selected and complete the software development. Customer partner ensure customer gets the value they are paying for build a reference first instead of System requirements. Customer gets what he wants and validates the expected results. Developers know the right answer Use the simplest technology team capability ease of use case of Refactoring Flexibility to Change test automation empower developers to run their own tests run tests regularly [13]. Our paper considers the customer to be member in the testing because the expected results depend on his satisfaction and agreement. Our aim is to minimize the time consumed for test according to our new method procedure. The customer choose path steps toward the objective need to be achieved, then we can detect the suitable solution by multiply the steps with the trained matrix, the solution wanted path will be the result iii.

2 Notation and Definitiond

Now we will illustrate in this section, all parameters which we used in our new model and we will define every item: Si: Solution Number

3 Mathematical Model

4 General formulation of the problem:

Analysis function which can be derived for acceptance tests in Extreme Programming Exhaustive testing:?Wi = Si*Pi, n Wt =??Wi i=1

The accepted test will be:Ta = Wt * Pi

Which guarantee the solution to be assess customer visible functionality. We will present her an important assumptions to declare and describe the formulation of our new model. We derive test cases to exercise these paths. And convert each path to its matrix representation and derive the trained matrix. We have paths as follows:Path 1: [1,1,1,0,0,1,1,1] Path 2: [1,1,1,0,0,0,1,1] Path 3: [1,1,0,1,0,0,1,1] Path 4: [1,1,0,1,0,0,1,0] 1 0 ?W1=[1,1,1,0,0,1,1,1]*0= 0 ?W2= [1,1,1,0,0,0,1,1]**= ?W3= [1,1,0,1,0,0,1,1] *= ?W4= [1,1,0,1,0,0,1,0] *= Wt =?W1+?W2+?W3+?W4 =

During the collaboration with customer to output the features and requirements needed a suite solution can be chosen which will achieve the problem and customer requirements. Assume that the customer choose the solution with feature binary: [0, 0,1,0] then to know what is the path to be tested to give solution, it can be derived as follows:

Chosen pattern of solution multiply by trained matrix. By dividing the results on the maximum number and the integer values to be taken the new results will be the matrix value [0 0 1 0], this value represent the solution number 3 which will be under test and expected needs to the customer. The result represents the path of solution wanted the next step will be to be tested directly without any loose of time to discover with uncover error with needless paths. In this method we can minimize the Exhaustive testing and minimize the time taken and efforts which yield to produce the project in quick time and help in the highly iteration and incremental analysis.

v.

5 Conclusion

Exhaustive tests should be planned long before testing begins. The Hebbian trained matrix algorithm applies to Web programming excusive testing as well all testing should be traceable to customer requirements. Most effective testing should be performed by Hebbian trained matrix, time and web developer efforts will be reduced too. [1]

1

Figure 1: Fig. 1 :

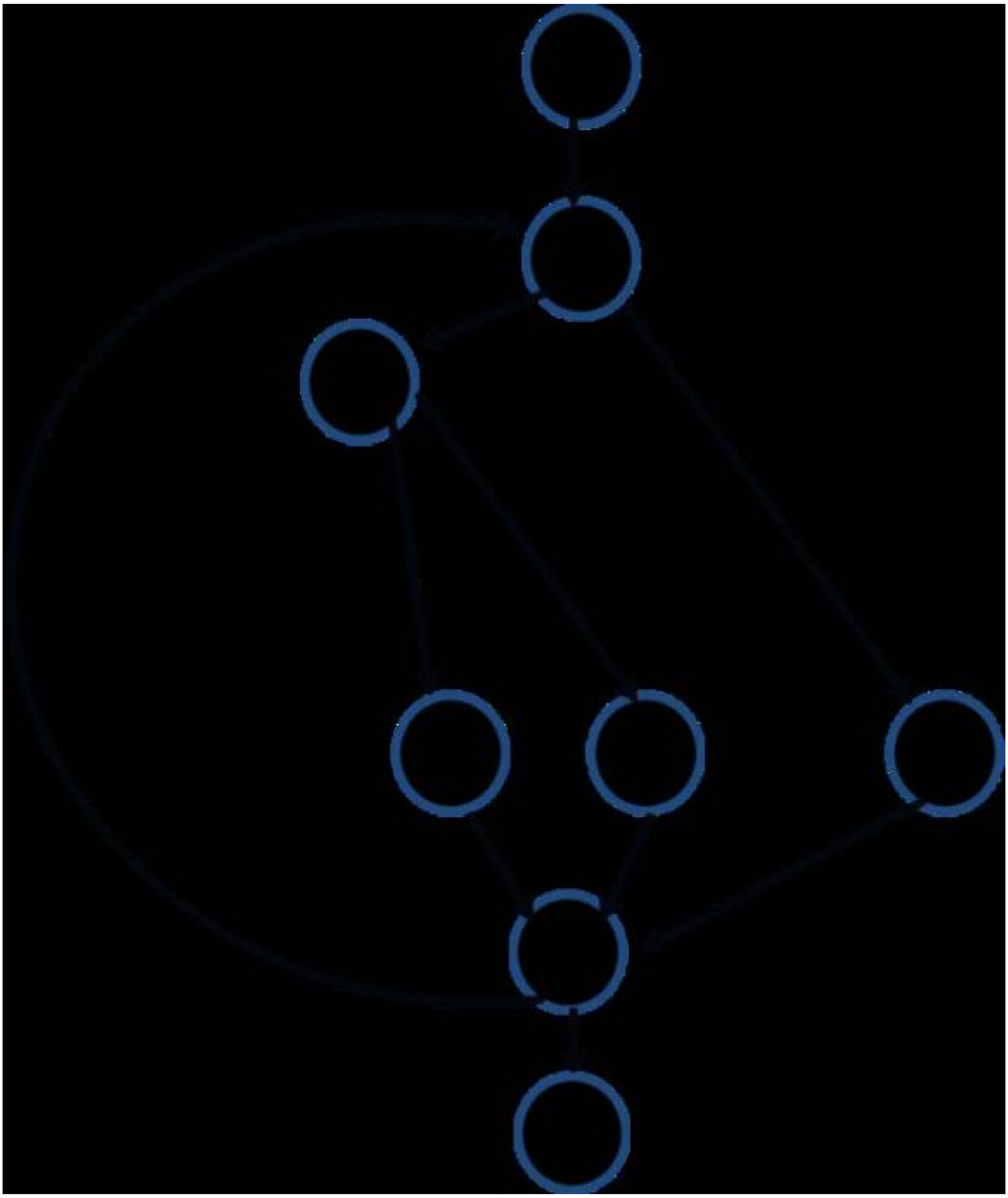

Figure 2:

[Kaner et al. ()] , C Kaner , J Falk , H Nguyen . 1999. Testing Computer Software. Wiley.

[Sharma et al. (2013)] ‘A Survey on Software Testing Techniques using Genetic Algorithm’. Chayanika Sharma , Sangeeta Sabharwal , Ritu Sibal . *IJCSI International Journal of Computer Science Issues* January 2013. 10 (1) .

[Last and Kandel ()] ‘Automated Test Reduction Using an Info-Fuzzy Network’. M Last , A Kandel . *Annals of Software Engineering, Special Volume on Computational Intelligence in Software Engineering*, 2003.

[Zhang and Wang ()] *Automatic generation of test data for path testing by adaptive genetic simulated annealing algorithm*, Bo Zhang , Chen Wang . 2011. IEEE. p. .

[Weyuker et al. ()] ‘Automatically Generating Test Data from a Boolean Specification’. E Weyuker , T Goradia , A Singh . *IEEE Transactions on Software Engineering* 1994. 20 p. .

[Schroeder and Korel ()] ‘Black-Box Test Reduction Using Input-Output Analysis’. P J Schroeder , B Korel . *Proc. of ISSTA ’00*, (of ISSTA ’00) 2000. p. .

[Chauhan ()] Naresh Chauhan . *Software Testing: Principles and Practices*, 2010. Oxford University Press.

[El-Ramly et al. (2002)] ‘From Runtime Behavior to Usage Scenarios: An Interactionpattern Mining Approach’. M El-Ramly , E Stroulia , P Sorenson . *Proceedings of KDD-2002*, (KDD-2002Edmonton, Canada) July 2002. ACM Press. p. .

[Mark et al. (2006)] L Mark , F Menahem , K Abraham . *The Data Mining Approach to Automated Software Testing Proceedings of the 21 st Annual Conference on Computer Assurance*, (Gaithersburg, Maryland) June 2006.

[Voas and Mcgraw ()] *Software Fault Injection: Inoculating Programs against Errors*, J Voas , G Mcgraw . 1998. Wiley.

[Beizer ()] *Software Testing Techniques, 2 nd Edition*, B Beizer . 1990. New York, NY: Van Nostrand Reinhold.

[Beizer ()] *Software Testing Techniques. 2 nd Edition*, B Beizer . 1990. Thomson.

[The Economic Impacts of Inadequate Infrastructure for Software Testing (2002)] *The Economic Impacts of Inadequate Infrastructure for Software Testing*, 02-3. May 2002. National Institute of Standards & Technology (Planning Report)