# A method based on computational intelligence for automatic black box test cases generation

**Hindenburgo Elvas Gonçalves de Sá**[1], **Edison Spina**[2]
[1] Engineering School, University of São Paulo, São Paulo, São Paulo, Brazil
[2] Engineering School, University of São Paulo, São Paulo, São Paulo, Brazil

**Abstract -** *A method is presented based on computational intelligence, learning set of rules and fuzzy logic to propose the development of tools that generate and classify test cases for black box tests in order to assist in the activity of test preparation and to achieve a qualitatively superior coverage on the manual creation process. The test technique entity represents a unitary functional test technique, which can be picked from a selection box., using sequential covering algorithms, and a fuzzy inference engine to classify the test cases generated. The test classification outcome results will help the test engineer on his decisions.*

**Keywords:** Software Engineering (Analysis, Tests), Fuzzy (Artificial Intelligence), Computational Learning.

## 1 Introduction

The software development process has the complex task of ensuring its functionality according to specifications or project requirements.

Even with the effort and commitment of developers and software engineers, these systems are susceptible to flaws or defects. According to [13], "the software fails for reasons known and preventable, such as poor documentation or lack of it, inability to meet the requirements, or for not being clear enough or because they are constantly changing, or even for the absence of a key person who has deep knowledge of the process object of the software system.

The validation phase in the process of software development is important in helping to keep the project under control. Functional, or black box, tests aim to test the software without the knowledge of how the internal instructions are developed. The test runs only based on the requirements or specifications. The test engineer prepares a set of test cases to simulate the operation, making a comparison between the entries provided, the expected results and outcomes. This set of test cases is also known as mass test. According to [7] “Many companies have some level of automation for the formation of mass screening " in which test cases are stored in text files for later interpreted and executed by a program.
According to [17], "Generating test data is a challenging process. Without automation, the process is slow, expensive and prone to errors. "However, techniques to automate the generation of test data must respond to a variety of functional criteria and non-functional and must implicitly or explicitly, to solve problems involving the propagation of the state and constraint satisfaction.

## 2 Functional Test Cases

The most common way to detect defects in a piece of software is to make the tester select a set of input data and then run the software with the input data under a particular set of conditions [14]. Which a test case is represented by a set of input data, execution conditions and expected results.

According to IEEE 610.12 (1990), a real test case must contain at least the following information:

- A set of input data for test: This set contains the data received from a source external to the code being tested. The external source can be understood as hardware, software, or human;
- Execution conditions: These conditions are required for the test run, for example, a certain database state, or configuration of a hardware device;
- Expected Outputs: are the expected outputs when the test runs.

A company may decide that additional information should be included in a test case. A test case may be presented in algebraic form T (x, S (x)), where there is a domain (D) composed of all possible input elements (xn) software or program (P) and test cases represented by a function (T) and its input arguments (x) and expected output (S (x)).

Figure 1 demonstrates the use of a test case in a typical scenario of test activities including the figures of the tester, oracle (or any document that person who can decide on outputs produced by the tests) and the results of the tests themselves.

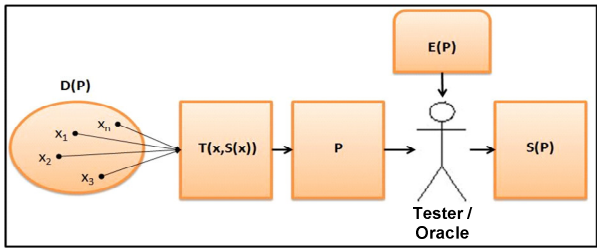


**Figure 1: Representation of a typical scenario for testing activity Adapted from (Delamare, Maldonado, & Jino, p.3, 2007).**

Oracle is "A program, or produce a document that specifies the outputs of a test can serve as an oracle ". Examples include a specification (especially one that contains pre-and post-condition) [14], document development, and a set of requirements. Other sources are the suites of regression tests. The test suites typically contain components with the correct results for earlier versions of the software.

Functional tests are to verify whether the software system meets the complexity of specifications in which test cases are much more common [11].

# 3 The Process of Generating Functional Test Cases

The generation of test cases consists of a Graphical User Interface (GUI) allow the engineer to set the tests to be executed as well as to assemble the knowledge base or to use an existing base.

The knowledge base, shown in Figure 2, is for storing of data entities of the system configurations, from basic training to test cases generated and classified.

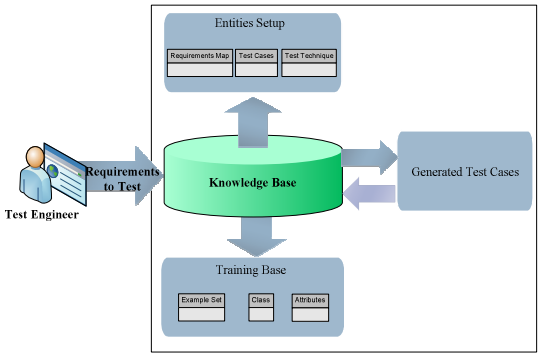


**Figure 2: Representation of Knowledge Base**

Entities' configuration model stores the data relevant to the statement of requirements to test cases and actual test technique to be applied when performing the test.

The statement of requirements, or requirements document, will contain software requirements details such as data entry, processing methods form and expected values as output.

The test case will have a strong relationship with the entity-map requirements in view of the dependency between them. The entity test case will be responsible for storing the data for the test case name, the type used in input arguments, return values and status after the test execution.

The approved testing technique is a method technique for functional testing which can be selected in a box.

Figure 3 shows the process flow to generate test cases. The first step is the configuration of the tests by the test engineer, the second step is performed by generating test cases that analyze the data stored in the knowledge base and create situations (Test cases) undergoing the test, then the running through the module tester. The latter accounts for applying the generated test cases and for capturing the output results.

After the capture and execution of test cases the qualifying round of tests begins. The uses a fuzzy inference machine. The test results are classified into three groups: test case "Cling", which has a great chance of finding or pointing flaws. A test case is classified as "Little Compliant" when the result is equal to the expected result. A test case is classified as "non-adherent" when it does not apply to the set of input arguments of the software being tested.

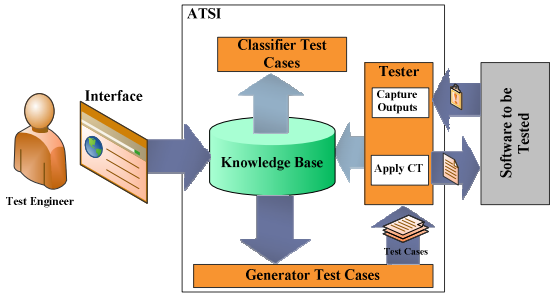


**Figure 3: Representation of Automatic Generation Process of Test Cases.**

# 4 Generator Test Cases

The method used for generating test cases is an automated technique based on a learning set of rules, algorithms, sequential coverage. Other methods such as Info-fuzzy network (IFN), as in [11], are also used in order to generate masses of tests automatically. However, the use of sequential covering algorithms for automatic generation of test cases is more effective as compared to IFN algorithms due to their lower computational complexity.

Machine Learning are computational techniques that provide arguments for the computer that simulates human behavior and that under [19] "is used to build systems capable

of automatically acquiring knowledge."The learning of a set of rules is one of the common techniques of machine learning.

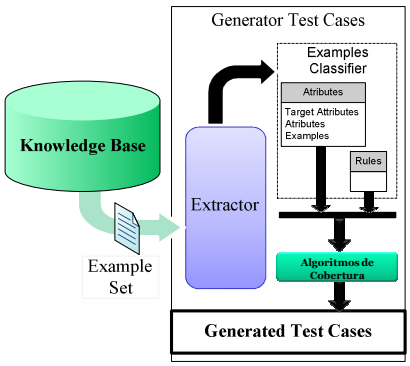


**Figure 4: Representation of Test Cases Generator.**

The test cases generator, shown in Figure 4, consists of three distinct parts, namely: a data extractor knowledge base, a binder of examples and a covering algorithm.

The extractor accounts for seeking knowledge in the basic data about the configuration of the tests and the set of examples to start the cycle test generation.

The process of classifying examples accounts for approved data brought by the extractor and arranges them in a decision tree, separating the attributes, rules and classes.

# 5 The Classifier Test Cases

The use of fuzzy logic to analyze complex systems and decision-making process is grounding in the approach outlined by Zadeh which was based on premises "[..] that the key elements of human thought are not just numbers, but labels Fuzzy Sets [..]" [12]. In connection with the use of fuzzy logic in decision-making processes or in decision support systems and considering the fact that fuzzy logic is not restricted to the control system the fuzzy logic was used to classify the test results generated.

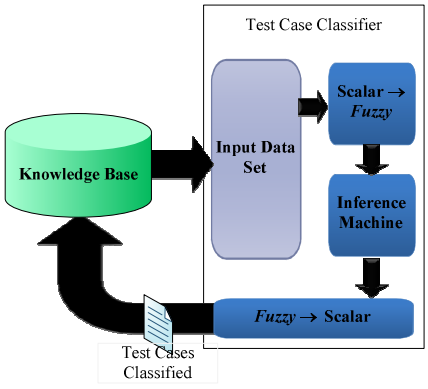


**Figure 5: Representation of Test Cases Classifier**

The process of classifying test results consists of a fuzzy inference machine and is represented in Figure 4. This process consists of a set of input values that are produced by the tests, the transformation of scalar values, or crisp, fuzzy values (fuzzification). The second stage is called inference machine which, based on Mamdani's model (min and max), makes associations with the inference rules, and consequently generate the output values that undergo one more transformation - from fuzzy to scalar (defuzzification)..

The fuzzification consists of the input variables (linguistic variable), which are the set of input values or values crisps, the input labels and the fuzzy production rules. The defuzzification process consists of the output variables and the output labels.

According to [12], "a linguistic variable is characterized by a quintuple (x, T (x), U, G, M) where x is the variable name, T (x) is the end of set of x, that is, the set of names of linguistic values of x with each value starting a fuzzy number defined on U, G is a syntactic rule to generalize the name of the values of x, and M is a semantic rule associated with each value and its meaning ".

The following results are considered as linguistic variables and assume the label of RO. Another set of linguistic variables, the object of fuzzification are the expected results from tests that assume the label of RE. The linguistic variables are associated with a label called Input Label (set of names of values of the terms of entry). To apply the tests in the context of fuzzy classifier, we have:

x = Outcomes (RO)

U [0, n], considering that the return crisp values of the tests are the numbers represented by the interval from zero to n.

T(RO) = {Success, Fault, Error };

T(RE) = { Covered, Not Covered, Not Applied };

G = Test case Adherent or Shortly Adherent, for example.

M = A rule which combines a linguistic variable to a fuzzy set.

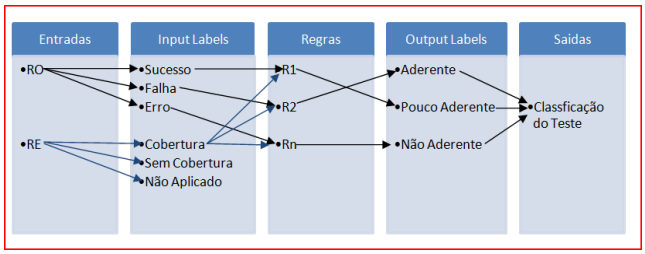


**Figure 6: Fuzzification and defuzzification process.**

Figure 6 illustrates the fuzzification and defuzzification process, in which the fuzzy production rules, represented by R1, R2 to Rn are related to the input linguistic variables.

A fuzzy production rule can be described as follows:

**IF** (Set of conditions are met) **THEN** **(1)**

(Set consequences that can be inferred)

The set of conditions to be fulfilled or the foregoing terms, are represented as a conditional variable between linguistic input and the label associated with it. The resulting set of accounts that can be inferred, or consequential terms, the result of processing of the foregoing terms and assume the value of a specific output label.

For the assembly of fuzzy production rules, it is necessary to combine input variables (linguistic variables) RO and RE and the names of linguistic values (or input label). For the set of consequents, the output values combined with the output labels were used, and the consequent were properly met. The application of fuzzy inference rule in the classifier of test cases is demonstrated in (2).

**IF** RO **IS** Success **AND** RE **IS** Covered **THEN** **(2)**

Classification **=** Adherent

The last step in the classification process of test cases is the defuzzification, or conversion process from fuzzy to scalar. The defuzzification process accounts for calculating the classification degree of a given test case. The degree classification degree is determined by the output variables, represented by output Adherent labels, Little or Not Adherent.

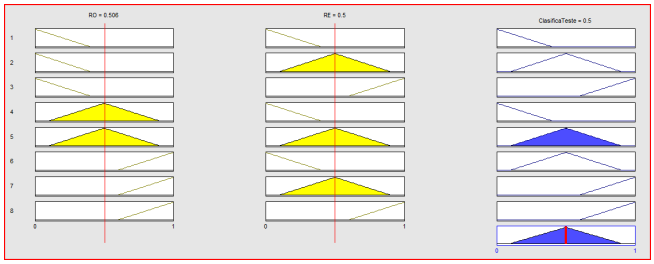

**Figura 7: Demonstrating the use of Fuzzy Inference Rules for using the Mathlab**

Figure 7 shows a test of the application of fuzzy inference rules using the Mamdani method of Max and Min. The analysis of the chart is made starting from the point that so far as it increases the values of linguistic variable "Success" and decrease that the values of the linguistic variable "Coverage" is the lowest degree of relevance and therefore “Little adherent" is the Test Case.

# 6 Results

The results obtained with the test conducted using the Mathlab tool, Figure 7, demonstrate the use of fuzzy logic to classify the test cases generated and tested by the proposed model of automatic generation of test cases for functional software. The application of fuzzy logic in decision support systems based its use as results of test cases generated classifier and tested for this model.

The test performed with the data of the proposed model to generate test cases aims to classify the test cases generated and tested, shown in Figure 6 and Figure 7, does the inclusion of all possible inference rules and is applicable to test the handling of fuzzy inference machine (which would cause distortions in the analysis of a control system using fuzzy logic) in Mathlab they were either not relevant or did not apply to the set of values entry.

From the model, is possible to manage the results from the tests as well as make comparisons between expected outcomes and obtained results and the test engineer may act as an oracle. This point is important for the analysis and disclosure of test results.

# 7 Conclusions

The scope of the Automatic Generating Case software testing function is to generate test cases to test the API's designed to meet the particularities of an information system, such as API's focused on encapsulating command for connecting to databases, commands, database, communication devices with serial or parallel port communications. The proposed tool is not ready to test the API's operating system or web development.

A comparative study of methods for automatic generation using approaches such as Genetic Algorithms is important to better validate this model and this model was also a study aimed at checking the performance between the Mamdani and Takagi-Sugeno models with a view of the study by [9] show that in both models (Takagi-Sugeno and Mamdani), taking into account the number of entries, are comparable and that the systems "Takagi-Sugeno can be. More economical compared to the Mamdani model number of entries and rules when this model is no trapezoidal or no triangular".

# 8 References

[1] XUE, Ming e ZHU, Changjun (2009). A Study and Application on Machine Learning of Artificial Intelligence**;** International Joint Conference on Artificial Intelligence.

[2] LEATHER, Hugh e BONILLA, Edwin e O’BOYLE, Michael (2009)**.** Automatic Feature Generation for Machine

Learning Based Optimizing Compilation. International Symposium on Code Generation and Optimization.

[3] RAHMAN, A.M.J. Md. Zubair e BALASUBRAMANIE, P. (2008). An Efficient Algorithm for Mining Maximal Frequent Item Sets; Journal of Computer Science 4 (8): Páginas 638-645; Science Publications.

[4] MICHAEL, Christoph C. e MCGRAW, Gary e SCHATZ, Michael A. (2001). Generating Software Test Data by Evolution. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 27, NO. 12, DECEMBER. Página 1085.

[5] ANSARI, A. Q. e PATKI, Tapasya e PATKI, A. B. e KUMAR, V.(2007). Integrating Fuzzy Logic and Data Mining: Impact on Cyber Security. Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD).

[6] JAVED, A. Z. e STROOPER, P. A. e WATSON, G. N. (2007) Automated Generation of Test Cases Using Model-Driven Architecture. School of ITEE, The University of Queensland, Australia. Second International Workshop on Automation of Software Test (AST'07).

[7] BERNARDO, P. C. e KON, F. (2008). A Importância dos Testes Automatizados. Engenharia de Software Magazine , 1(3), Páginas. 54-57.

[8] BIANCHI, R. E. (2008). Extração de conhecimento simbólico em técnicas de aprendizado de máquina caixa-preta por similaridade de rankings. *Tese de Doutorado* São Carlos, São Paulo: ICMC/USP - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo.

[9] YING, Hao e DING, Yongsheng e LI, Shaokuan e SHAO, Shihuang (1999). Comparison of Necessary Conditions for Typical Takagi–Sugeno and Mamdani Fuzzy Systems as Universal Approximators. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS - PART A: SYSTEMS AND HUMANS, VOL. 29, NO. 5, SEPTEMBER.

[10] RECH, Jörg e ALTHOFF, Klaus-Dieter (2004). Artificial Intelligence and Software Engineering: Status and Future Trends. Künstliche Intelligenz : KI, Vol.18, No.3, Páginas 5-11 : Ill., Lit.

[11] LAST, Mark e FRIEDMAN, Menahem e KANDEL, Abraham (2003). The Data Mining Approach to Automated Software Testing. *SIGKDD '03*, August 24-27, Washington, DC, USA. ACM 1-58113-737-0/03/0008; Páginas. 388 - 396.

[12] ZADEH, L.A. (1973) Outline of a New Approach to the analysis of complex systems and Decision Process. IEEE Transactional on Systems, MAN, and Cybernetics, Vol. SMC-3, Nº 1, Janeiro, Páginas 28-44.

[13] CHARETTE, R. N. (2007). "Learning from Software Failure". *IEEE Spectrum*,. http://www.spectrum.ieee.org/computing/software/learning-from-software-failure, September.

[14] DELAMARO, M. e MALDONADO, J. C. e JINO, M. (2007**).** Introdução ao teste de software. São Paulo, Elsevier, São Paulo.

[15] XUE, Ming e ZHU, Changjun (2009). A Study and Application on Machine Learning of Artificial Intelligence. International Joint Conference on Artificial Intelligence, páginas 272-274.

[16] HAMBLING, B. e MORGAN, P. e SAMAROO, A. e THOMPSON, G. e WILLIAMS, P. (2007). Software Testing. United Kingdown, Inglaterra: BCS - The British Computer Society.

[17] HARMAN, M. (2007). Automated Test Data Generation using search based software engineering. *Second International Workshop on Automation of Software Test* .

[18] PAPO, J. (2009). *http://josepaulopapo.blogspot.com/*. Acesso em 28 de Outubro de 2009, disponível em http://josepaulopapo.blogspot.com/2009/10/testes-unitarios-beneficios-economicos.html.

[19] PRATI, R. C. (2006). Novas abordagens em aprendizado de máquina para a geração de regras, classes, desbalanceamentos e ordenação de casos. *ICMC/USP - Instituto de Ciências Matemáticas e Computação da Universidade de São Paulo*, Julho.

[20] REZENDE, S. O. e PRATI, R. (2005). *Sistemas Inteligentes - Fundamentos e Aplicações.* Barueri, SP: Manole.

[21] TALON, B. e LECLET, D. e LEWANDOWSKI, A. e BOURGUIN, G. (2009). Learning Software Testing using a Collaborative Activities Oriented Platform. *Ninth IEEE International Conference on Advanced Learning Technologies,* Maio.


# ICAI'10 - The 2010 International Conference on Artificial Intelligence

**ICAI'10 is the 12th annual conference**

CALL FOR PAPERS

You are invited to submit a full paper for consideration. All accepted papers will be published in the ICAI conference proceedings (in printed book form; later, the proceedings will also be accessible online). Those interested in proposing workshops/sessions, should refer to the relevant sections that appear below.

Topics of interest include, but are not limited to, the following:

- Brain models / cognitive science
- Natural language processing
- Fuzzy logic and soft computing
- Software tools for AI
- Expert systems
- Decision support systems
- Automated problem solving
- Knowledge discovery
- Knowledge representation
- Knowledge acquisition
- Knowledge-intensive problem solving techniques
- Knowledge networks and management
- Intelligent information systems
- Intelligent data mining and farming
- Intelligent web-based business
- Intelligent agents
- Intelligent networks
- Intelligent databases
- Intelligent user interface
- AI and evolutionary algorithms
- Intelligent tutoring systems
- Reasoning strategies
- Distributed AI algorithms and techniques
- Distributed AI systems and architectures
- Neural networks and applications
- Heuristic searching methods
- Languages and programming techniques for AI
- Constraint-based reasoning and constraint programming
- Intelligent information fusion
- Learning and adaptive sensor fusion
- Search and meta-heuristics
- Multisensor data fusion using neural and fuzzy techniques
- Integration of AI with other technologies
- Evaluation of AI tools

- Induction of document grammars
- Supervised and unsupervised classification of web data
- General Structure-based approaches in information retrieval, web authoring, information extraction, and web content mining
- Latent semantic analysis
- Aspects of natural language processing
- Intelligent linguistic
- Aspects of text technology
- Computational vision
- Bioinformatics and computational biology
- Biostatistics
- High-throughput data analysis
- Biological network analysis: protein-protein networks, signaling networks, metabolic networks, transcriptional regulatory networks
- Graph-based models in biostatistics
- Computational Neuroscience
- Computational Chemistry
- Computational Statistics
- Systems Biology
- Algebraic Biology

**Click Here** for more details