International Journal of Engineering & Technology, 7 (1.1) (2018) 219-222

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

An effective method for improving software quality by automated test generation technique

Srinivasa Rao S [1] *, Karthik Reddy T [2], Prasanna Sai N [2], Vamsi Krishna Ch [2]

[1] Assoc.Professor, Koneru Lakshmiah Educational Foundation
[2] Student, Koneru Lakshmiah Educational Foundation
**Corresponding author E-mail: srinu1479cse@kluniversity.in*

Abstract

The general undertaking of the product building is to guarantee conveyance of superb programming to the end client. To ensure great programming, it is required to test programming. Testing is a critical constituent of programming building. In programming testing there are number of fundamental issues like compelling age of experiments, prioritization of experiments which should be handled. This mechanized test structure predominantly relies upon these four angles: test system, experiment age, test execution and test assessment. Test methodology is an accumulation of systems that decides the testing way to deal with be trailed by the testing group. The experiment age alludes to the age of experiments in light of the given application. The test execution briefs about the execution of those tests at that point contrasting the normal outcome and genuine outcome. The test assessment explores the experiments and causes us to produce test rundown report and programming quality confirmation report consequently. The aim of delivering this device is to produce test cases naturally and to diminish the cost of testing notwithstanding collect the season of determining experiments physically. Subsequently this framework enhances general nature of the product.

Keywords: *Software Quality Assurance Report; Software Testing; Test Cases; Test Evaluation; Test Execution; Test Strategy; Test Summary Report.*

1. Introduction

Programming testing is the noteworthy part in programming advancement life cycle and also it has an unequivocal influence in guaranteeing programming quality. Test automation makes utilization of specific programming to control the execution of tests and to contrast the real outcome and anticipated outcome. Choosing perfect time to go for automation, characterizing extension for computerization and choosing the correct device for mechanization are the vital choices in which the testing group must detail in the test design. Indicating the correct subtle elements of the item for mechanization very decides the triumph of the automation. The viability of this confirmation and approval strategy relies on the quantity of bugs recognized and settled before discharging the framework [1]. It relies on the nature of experiments created. The most huge issue in the field of programming testing research is the age of the experiments in view of automation. To cut down the cost of manual testing and to expand consistency of the testing, it is basic to mechanize the experiment age [1]. Contingent on these experiments, the test outline report and programming quality affirmation report will likewise be robotized. Test Summary Report is a huge deliverable which is set up toward the finish of a testing, or rather in the wake of testing is finished. The primary goal of this test outline report is to elucidate diverse subtle elements and exercises about the testing performed. Programming quality affirmation report assesses the nature of an item and finish adherence to programming item norms and techniques. It is a sunshade movement that guarantees agreement to measures and systems all through the Software advancement life cycle of a product item.

In existing method, MBT strategy concentrates just on utilitarian testing. It will apply specifically to useful testing not for security testing. The MBT strategy doesn't totally create the programmed test execution due to two reasons: I) the model produces test are not finished on the grounds that the parameters can't be determined specifically. ii) Doesn't quickly executes the test got from test demonstrate in light of the fact that the test show utilize diverse programming dialects.

In our proposed procedure, we utilize another apparatus strategy called Model based Integration and System Test Automation (MISTA) for producing test code from a Model Implementation Description (MID). It will coordinate the utilitarian and security testing. MID contains the Model Implementation Mapping (MIM) and test show. The test models are practical model, get to control model and security display. It utilizes the abnormal state petrinet display for confirming the product framework. Test models planned by the petrinet can incorporate the two information and control stream of test models. MISTA can create programmed display based experiments, including the test inputs and expected test outcomes. MISTA demonstrates the relations of test models and usage level for the test condition. It will consequently create test code from test demonstrate.

2. Proposed system

The automation test structure is an execution situation for electronic tests. It is a fused structure that sets the tradition of computerization for a specific item. A structure is a valuable mix of different techniques, programming principles, discernments, strategies, traditions, framework chains of importance, seclusion, scope component and test information infusions. These components go

about as minor auxiliary areas which should be accumulated to speak to an industry procedure .This structure furnishes the client with various advantages that encourages them to create, perform and declaration the computerization test contents productively. These robotized tests can run rapidly and intermittently, which is beneficial for programming items with an expanded administration life. This framework composes the test suites and thusly enhances the proficiency of testing. An organized test system helps in taking out the duplication of experiments which is computerized over the application.

A testing system is constantly autonomous of utilization and it can be utilized with any application all things considered of issues (like parts, stack, auxiliary plan and so on.) of use under test. Recently created test cases are continually added to existing computerization in relating to the advance of the product improvement. It is vital to be cognizant that general scope of all tests by methods for test automation isn't feasible. While choosing what tests should be computerized premier, thus cost and exertion are required to be considered. Experiments which contain high cost and low exertion ought to be robotized early. At that point test cases with normal utilize, changes, and past mistakes notwithstanding experiments with low to direct exertion will be additionally computerized. Test mechanization assuages analyzer's irritation and permits the test execution without client contact while ensuring repeatability and precision. Or maybe analyzers would now be able to concentrate more on muddled test situations. Discretionarily produced tests can discover abandons with high testability. This Test automation structure enables us to perform diverse sorts of testing proficiently and adequately.

This Testing structure is in charge of:

- Specifying the example in which to verbalize desires.
- Building a strategy to guide into or drive the application under test.
- Perform the tests.
- Testify the outcomes.

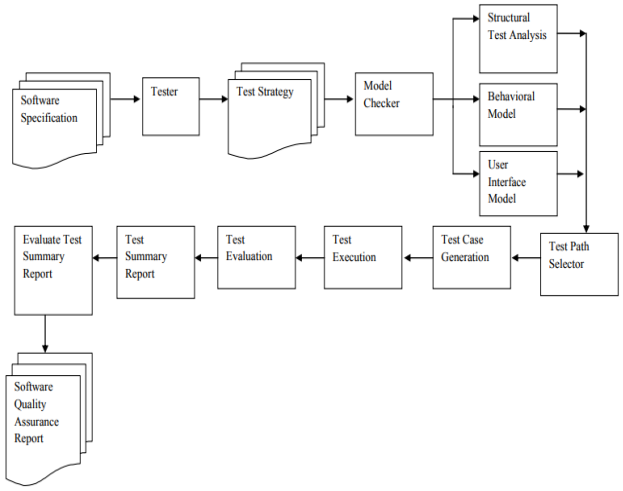

Fig. 1: Test Framework for Software Quality Assurance- Architecture.

3. Automated code generation

3.1. Architecture

Fig.2 demonstrates the engineering of MISTA. The contribution to the MISTA is MID determination, which incorporates the test show and the MIM particular. The test show signifies the petrinet display that contains utilitarian model, get to control demonstrate, and the risk show. The useful model determines the capacity in the framework, the entrance control demonstrate depicts the limitations on the framework and the risk display demonstrates the security arrangement in the framework. The MIM detail changes over the test model to usage requirements. MISTA utilizes distinctive dialects, for example, C, C++, HTML and so forth to create test code. It bolsters a different scope criteria for experiment age. MISTA is likewise extremely successful in the product blame discovery.

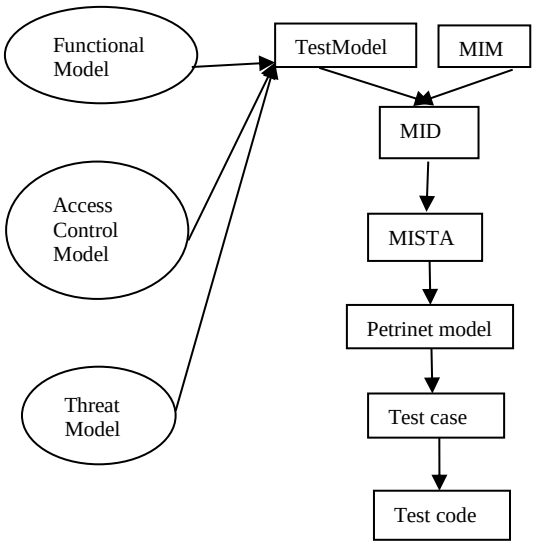

Fig. 2: Architecture Diagram of MISTA.

3.2. Petrinet model

PrT nets are otherwise called abnormal state Petri nets. The past work has likewise clarified that PrT nets can determining access control techniques and security dangers. Since the test models determined by PrT nets can coordinate the two information and control streams of test prerequisites. MISTA can create programmed demonstrate based experiments, including the test inputs and the normal test outcomes.

The age of test show from the PrT net diminishes the determination of substantial and invalid experiments. The petrinet is the coordinated bipartite diagram, in which the hubs speak to spots and changes. The coordinated circular segments depict in the petrinet show in which places portrays are pre or post conditions for the changes. It is otherwise called put/progress net. It is one of the created numerical displaying dialects for the detail of conveyed frameworks [1].

A petrinet has five tuples:

Where,

R - Limited arrangement of spots

S - Limited arrangement of changes

T - Limited arrangement of typical circular segment

U - Limited arrangement of inhibitor circular segment

L1 - Set of beginning markings

3.3. Model implementation mapping (MIM)

The MIM detail mapping the contributions of the test model to the usage level execution. The objective of model-based testing is to check whether a usage of a product framework identifies with the model of that framework. The prerequisites and the test models checks by the MIM determination in the execution organize. Via naturally create the experiments and expected outcomes from the detail of the framework, it requires a formal determination. At times the formal determination is likewise executable by the normal outcomes acquired by executing the predefined particular with the test inputs. The setting predicate indicated an information state of the part that ought to be designed accurately before called by the segment. For instance, the test age part of MISTA requires that the scope standard be set before it is summoned. This can happen by calling the mutator capacity of the predicate scope. The MIM detail get the components from the petrinet model< R, S, T, U, L1> utilized as a part of the objective dialect to the programming

dialects. MIM comprises of character of the framework, rundown of concealed predicates in the test display and the mapping components. The assistant code 'h' is utilized to create the test code. It contains the header, setup techniques and created code.

3.4. Model implementation description (MID)

The contribution to the MISTA is known as a Model Implementation Description (MID) and it comprises of a test show and a Model-Implementation Mapping (MIM). MID is the front end dialect for MISTA, and gives the nuts and bolts for the computerized test age approach. The MIM determination mapping the contribution of the test model to the execution level develops. For the MID strategy, the test code can be produced by MISTA for the objective dialects, for example, Java, C#, C, C++, HTML and so on in light of the different scope foundation of the test model, for example, reachability scope, state scope, change scope, profundity scope, and objective scope. Amid the improvement arrange, we have connected MISTA to the useful testing to discover numerous issues happen in the framework. This MISTA method has demonstrated that it is extremely compelling in blame recognition of the frameworks.

3.5. Test code

The objective dialect utilized as a part of the progress tree is utilized to produce the test code. MISTA produces the test code from the theoretical test from MIM determination [20]. The test code created in the Selenium IDE can be consequently executed. Partner code indicates to the test code that serves to the analyzer to create the executable test code. Test code age is to change over the progress tree to create the test code as per the MIM detail and the assistant code. The framework under test quickly produces and executes the test code. The created test code is as various target dialects from a given info change tree. Different dialects can goes about as a contribution to produce the test code. For instance, Jfcunit is an expansion for JUnit for GUI testing of the Java programs.

4. Background

We utilize MISTA apparatus for practical and security testing. D. Xu et al. [4] proposed a Threat Model-Implementation (TMID) way to deal with robotized age of security tests by utilizing formal risk models that can be indicated as Predicate/Transition nets. This model creates assault ways. A formal risk driven approach of security dangers was portrayed by D. Xu et al. [5] that goes about as the middle person between security objectives and utilizations of the security highlights. Y. L. Traon et al. [6] gives a test driven strategy and arrangement choice point to dissect the adaptability of the framework. The security approach alteration can serves to changes in the test code for adaptability. The property is characterized as the level of coupling in the middle of access control rationale and business rationale in the framework. H. Zhu et al. [7] presents another strategy for test the product framework rely upon abnormal state petrinets. For that approach utilize four testing strategy called state arranged testing, stream situated testing, change situated testing and determination arranged testing. All systems are utilized an arrangement of compositions for examine and create a testing comes about and different scope criteria. J. Desel et al. [8] examined another idea called cause impact charting for produces the experiments and the test code. The abnormal state petrinet goes about as a middle of the road level. The approach utilized for creating test from the limited state show was proposed by A. masood et al. [9] and assessed the Role Based Access Control (RBAC) arrangement. The test suite produced from this model is reasonable for blame recognition. To maintain a strategic distance from blame and increment security arrangement W. Mallouli et al. [10] depicted a system called expanded limited state machine.

To create test cases Alexander et al. [11] characterized the model based approach. Jacques et al. [12] talked about the way to deal with include security testing with the practical testing by utilizing dialect articulation in display based approach. In a secluded innovation, H. Huang et al. [13] determined and confirmed the security strategies of the framework utilized the hued petrinet process. The security arrangement of the module is considered as extremely adaptable. Mortensen [14] indicated the hued petrinet model to dissect the entrance control framework to produce the test code. The principle qualities are that model is concentrate just on get to control show, yet not for the intermediation between the entrance control display and practical model. In our approach, the entrance control display incorporates with the practical model. The security testing utilized the assault trees requires generally the manual work for change over the assault tree into security test. In our work, to discover the blame happen in programming utilizing model based testing.

5. Test generation technique algorithms

Model Generation Algorithm
Input: Group of Model Segments S, Meta-Model MM
Output: Group of Models L matched to MM

1) If there are unmasked Model Segments in S do
2) { construct an vacant model M
3) If the model size boundary is not extended (1) and M still can
4) amplify do
5) { select an unmasked model segment MS in S
6) for all object segment OS in MS do
7) { search an object O which is occurrence of the class partly specified by OS (3)
8) for each condition CT defined in OF on the attribute A do
9) { if A is an attribute (value division case) then
10) select a value and place it to P in O (5)
11) else (multiplicity division case)
12) { select a cardinality N following to CT (5)
13) if the category of A is a class then
14) discover N objects with a P category and put them to A in O (3)
15) else uncover N values in the division of A and locate them to A in O (5)
16) }}
17) append O to M
18) end of M until it is conformed to MM (2, 4)
19) }}
20) spot MS as enclosed
21) append M to L}

Test Generation Algorithm
Input: Test Generation gathering of Test Case Segments TS, classifications C, Managed Meta-Model MMM
Output: Group of Models L coordinated to MM

1) In the event that there are unmasked Model Segments in S do
2) { develop an empty model M
3) In the event that the model size limit isn't broadened (1) M still can
4) open up do
5) { select an unmasked demonstrate portion MS in S
6) for all protest fragment OS in MS do
7) { look through a protest O which is event of the class halfway determined by OS (3)
8) for each condition CT characterized in OF on the property A do
9) { if A is an attribute (value division case) then
10) select a value and place it to P in O (5)
11) else (multiplicity division case)
12) { select a cardinality N following to CT (5)
13) if the category of A is a class then

14) discover N objects with a P category and put them to A in O (3)
15) else uncover N values in the division of A and locate them to A in O (5)
16) add test generation based on category C,
17) Add Test generation set items, TS
18) Generate report.
19) }}}}}

6. Conclusion

The frameworks have displayed a method for computerized age and execution of useful and security tests from a test show incorporating with the mapping from the components of the model to the usage builds. The mapping makes it possible to change over the model level tests into the executable type of test code. MISTA strategy is exceptionally proficient and successful for producing experiments and test code. The fundamental favorable position is that the method can incorporates framework capacities; get to control strategies and security show. This strategy can produce executable test code and identify the blame happen in the framework. Because of the specialized engineering, this strategy is anything but difficult to present another test generator, target dialect and test execution condition. The conceivable approach is to utilize PrT nets for partner the changes with time interims same as in Time petrinets. In future work, present documentations for true frameworks and contrast the blame discovery and different scope criteria.

References

[1] Mohammad Reza Keyvanpour, Hajar Homayouni and Hossein Shirazee, “Automatic Software Test Case Generation: An Analytical Classification Framework” at International Journal of Software Engineering and Its Applications Vol. 6, No. 4, October, 2012.

[2] Rosziati Ibrahim, Mohd Zainuri Saringat, Noraini Ibrahim and Noraida Ismail, “An Automatic Tool for Generating Test Cases from the System's Requirements” at Seventh International Conference on Computer and Information Technology.

[3] Mr. Navnath Shete, Mr. Avinash Jadhav “An Empirical Study of Test Cases in Software Testing” at ICICES2014 - S. A. Engineering College, Chennai, Tamil Nadu, India.

[4] Isabella and Emi Retna, “Study Paper On Test Case Generation For GUI Based Testing” at International Journal Of Software Engineering & Applications, Vol.3, No.1, January 2012.

[5] Nicha Kosindrdecha and Jirapun Daengdej, “A Test Case Generation Technique and Process”.

[6] Itti Hooda and Rajender Chhillar “A Review: Study of Test Case Generation Techniques” at International Journal of Computer Applications (0975 – 8887) Volume 107 – No. 16, December 2014.

[7] Karambir and Kuldeep Kaur, “Survey of Software Test Case Generation Techniques” at International Journal of Advanced Research in Computer Science and Software Engineering Volume 3, Issue 6, June 2013.

[8] Steven P. Reiss, “Towards Creating Test Cases Using Code Search” at IEEE International Conference on Software Maintenance and Evolution 2014.

[9] A. Masood, R. Bhatti, A. Ghafoor, and A. Mathur, “Scalable and effective test generation for role based access control systems,” IEEE Trans. Softw. Eng., vol. 35, no. 5, pp. 654–668, 2009. https://doi.org/10.1109/TSE.2009.35.

[10] W. Mallouli, J.M. Orset, A. Cavalli, N. Cuppens, and F. Cuppens, “A formal approach for testing security rules,” in Proc. 12th ACM. Symp. Access Control Models and Technologies, 2007, pp. 127–132. https://doi.org/10.1145/1266840.1266860.

[11] A. Pretschner, Y. L. Traon, and T. Mouelhi, “Model-based tests for access control policies,” in Proc. 1st Int. Conf. Software Testing Verification and Validation (ICST'08), Lillehamer, Norway, Apr. 2008. https://doi.org/10.1109/ICST.2008.44.

[12] J. Julliand, P. A. Masson, and R. Tissot, “Generating security tests in addition to functional tests,” in Proc. 3rd Int. Workshop Automation of Software Test, 2008, pp. 41–44. https://doi.org/10.1145/1370042.1370051.

[13] H. Huang and H. Kirchner, “Formal specification and verification of modular security policy based on colored Petri nets,” IEEE Trans. Depend. Secure Comput., vol. 8, no. 6, pp. 852–865, Nov./Dec. 2011. https://doi.org/10.1109/TDSC.2010.43.

[14] K. H. Mortensen, “Automatic code generation method based on coloured Petri net models applied on an access control system,” in Application and Theory of Petri Nets. New York, NY, USA: Springer-Verlag, 2000, pp. 367–386.

[15] A. Marback, H. Do, K. He, S. Kondamarri, and D. Xu, “A threat model-based approach to security testing,” in Software: Practice and Experience, Expanded Version of the AST'09 Workshop Paper, Feb. 2013, vol. 43, pp. 241–258.