International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 12, No.1 (Aug-2022)

https://dx.doi.org/10.12785/ijcds/120157

The Automation of Java Smart Card Using Negative Testing

Amena Bataineh[1], Mohammad Alshraideh[2], Amjad Hudaib[3] and Fadi Wedyan[4]

[1]*Department of Computer Science, King Abdullah II for Information Technology, The University of Jordan, Amman 11942, JORDAN*
[2]*Department of Computer Science, King Abdullah II for Information Technology, The University of Jordan, Amman 11942, JORDAN*
[3]*Department of Computer Information System, King Abdullah II for Information Technology, The University of Jordan, Amman 11942, JORDAN*
[4]*Department of Software Engineering The Hashemite University Zarqa, 13315 Jordan*

Received 22 Jan. 2021, Revised 15 Jul. 2022, Accepted 23 Jul. 2022, Published 6 Aug. 2022

Abstract: Negative Testing verifies the behavior of the system under test by providing invalid inputs. In this research, we applied the boundary values analysis technique on Java Smart Card Applications. This type of application is deployed in highly essential areas around us to support access, identity, payment and other services. We applied the Negative testing approach which does not only aim to show any potential defect that could cause a risky impact on the exhaustion of the application on the whole but can be instrumental in determining the conditions under which the application can crash. Negative Testing is evaluated on Six Smart Java Programs, and the results show that the presented approach can reveal faults and unexpected behavior in these programs.

Keywords: Software Testing, Negative Testing, Java Card, Applets.

1. INTRODUCTION

Software testing is a vital, yet costly phase in software development that aims at producing more reliable systems [1], [2]. Studies show that testing cost, which is typically measured by time and project budget, might exceed half of the developed project cost. Testing approaches aim at maximizing the number of revealed faults while minimizing the cost [3]. Many software testing approaches have been proposed. Testing approaches can be classified according to various criteria. The first, Positive Testing (PT) approach, works when inserting valid data. The second is Negative Testing (NT), which works when inserting invalid data for parameters. Negative Testing is a fundamental approach in our daily lives like Automated Teller Machine (ATM) card, and in space. The end-users will face disastrous problems, especially when the users insert the wrong password in ATM more than 3 times. Maybe an unauthorized user or authorized user, but the authorized user (he or she) forgot the password. In that case, we need to apply the NT approach to avoid system corruption and measure the behavior of the system under testing when we insert invalid inputs for parameters that are defined in each applet or application. Negative Testing (NT) is performed to ensure that the product under test does not fail when an unexpected input is given. The purpose of negative testing is to verify the application response during unintentional inputs. There are characteristics of negative testing like determining the errors which cause big failures, finding the weakness of the application and exploiting it, and showing data corruption or security breaches [4].

Java Smart Card (JSC) is a card that has a main application as a platform and one applet or more, which is a small application to introduce a specific service. Platform and applets are burned on the chip of the card to become embedded within a system. The embedded applets can communicate with each other. There exists some constraint in java applications to keep security to prevent any unauthorized user from accessing, so we need to test the applications before deploying them, so there is a native code that is in the libraries of the applets that provide cryptography services. The goal of the testing is to check if the system works as expected or not.

We noticed, there is no testing for JSC programs using NT. So as mentioned earlier, software testing aims to design test cases that reveal as many faults as possible to improve the quality of the software and to increase the reliability of the software product. Also, manually testing for software system requires a huge effort, consumes time, and is costly [2], [5], [6], [7], [8]. A solution to these requirements was to automate the process of testing by a proposed heuristic tool. Automatic negative software testing

E-mail address:amoon$_b$at@hotmail.com, mshridah@ ju.edu.jo, ahudaib@ ju.edu.jo, fadi.wedyan@hu.edu.jo

significantly reduces cost, time, and increases confidence in the products. There are some benefits to automatic NT taken from [9] discover variant methods to make the application crash and solve it easily; how can the system deal with bad data? prevent the system from crashing, improve the quality of the application, improve test coverage, find the weak points in applications. For the previous reasons, we applied an automated NT approach on six (JSC) applets using boundary analysis technique and branch coverage for all parameters, which included predicates, and different data types. The applets are MinMax, MidValue, AllTrue, QuadEq, NumDay, and Calculator. The results show that the presented approach can reveal faults and unexpected behavior in these programs, and consume time and effort. Also, we noticed in nested IF statements, the tool needed more time. The rest of this paper is organized as follows. Section 2 presents related work. Section 3 describes the details of the proposed approach. Section 4 presents the experiment setup. In section 5, the results of the experiment are given. Finally, conclusions and future work are outlined in Section 6.

2. RELATED WORK

According to Weelden etl. in [10] tested a Java Card applet, in a black-box setting, based on a formal specification of its required behavior. They demonstrate their testing methodology by applying it to a simple electronic purse application as a case study. They have presented an approach to automate the testing of Java Card applets using the test tool GAST. The test case derivation is based on a State Chart specification of the applet under test. As we know, most of the testing for any application is done by one approach of testing called Positive Testing (PT), but Semwezi [4] described Negative Testing (NT) definitions and its techniques. The Author propose a unit testing extension framework for negative testing called NegTest, which is used to test different open-source libraries of different complexities and the type and number of discovered failures documented. NegTest is developed in the Ruby programming language as an expansion to the MiniTest unit testing framework, which is the standard testing framework included in the Ruby system. The expansion is developed based on the Negation Testing principle. Eight JSC programs were tested using two methodologies, the first one was introduced by Manaseer et al [11], [12] when they used Genetic Algorithm (GA) to generate test data automatically to test JSC application. They applied GA to get the least possible test data to achieve the branch coverage criterion. The experimental results are measured by three parameters: the number of test data generation, execution time, and percentage of branch coverage depending on population size, which is considered in this study as follows (30, 50, 70, 90, 110). They applied GA to eight JSC programs, and the size of the programs ranges from (59) to (4,277) lines. The conclusion from their experiment was it had the monotone decreasing between the average number of generations and population size, but the relationship between execution time and the average number of generations had monotone increasing. The experiment covers 99% of branch coverage. The second one was by Allawi et al. [13] when they proposed a new technique to test the JSC application called Greedy Particle Swarm Optimization (GPSO), which is a combination of Greedy algorithm and Particle Swarm Optimization (PSO) algorithm. GPSO guarantees effectiveness and closes to get an optimal solution for generating the least possible number of the test data. They applied GPSO on eight applets and observed that it is better than GA in terms of the average number of iterations, execution time, and coverage percentage. To satisfy the goal of the experiment, covering all branches in the program under test, they implemented the program using Netbeans IDE 8.0.2. Kübler et al. [**?**] presented an approach to achieve negative test cases for test automation framework, and they used the framework-based CAST tool approach as a test environment to integrate the selection and generation of negative test cases. They consider the generation of negative test cases using model-implemented fault injection for the virtual model of the Production System (PS) as a suitable approach for the automated verification and validation of the reliability of a PS. Martin and Bousquet [14] described the method used in the verification process for Java Card applets and its application on a case study. This methodology is based on automatic test generation. They apply their methodology to the well-known type of smart card application called Purse. The Purse's applet provides the most common functions to the end-user such as debit, credit, and balance. They provide only a very few parts of their case study. These parts consist of the diagrams concerned by the Purse credit function. The experimental result obtained with their case study indicates that their methodology could fulfil industrial requirements for applet behaviour verification. Karhu, et al. [7] presented a survey by interviewing the employees who are working in product-oriented software development from different types of companies, customized systems development, and testing service providers to identify factors that affect the state of testing automation from their positions. They have observed the benefits of testing automation include quality improvement through better test coverage, and that more testing can be done in less time. a framework was proposed by Petrova-Antonova, et al. [15], they called it Testing as a Service Software Architecture(TASSA) to support the testing, validation and verification of both functional and non-functional behaviour of web service compositions at design time. It consists of a set of tools that can be used together with existing development environments of service-based applications. They focus on two of TASSA tools, namely Fault Injection tool and Isolation tool that respectively provide functionality for negative testing and unit testing of web service compositions described with Business Process Execution Language (BPEL). Villalobos-Arias, etal. [16] presented the Model-Based Testing Process for Java Applications (MBT4J), which can automatically generate and execute test cases for Java applications. The platform automates model building, test case generation, and test execution stages and allows to generate up to 2,438 test cases, detects up to 289 defects, and achieves up

to 83There is a comparison between traditional and agile testing models where Dhir and Kumar [17] proposed a model for automated agile testing. The experimental work has also been represented in testing a web application. The proposed agile testing model worked with the production team in a planned and organized manner to deliver the products in the sprint. The authors found out that the results through agile testing are better than traditional testing.

3. THE PROPOSED RESEARCH METHODOLOGY

This section explains the detailed steps of how applying negative automation testing by automatically generating test data for Java Card applications to achieve branch coverage. By using the NT approach, we verify the system behavior with inputs values to the conditions, indices, and indicators outside the specified or valid range of the predicates in addition to input values outside the domain of the predicates data types (e.g., int, double, string, …) and input values with different data types of the predicated data type.

To apply the NT approach using our proposed tool, we used branch coverage criteria. Because our experiments depend on a false branch for every predicate. So the research is introducing a tool to verify this approach. To the best of our knowledge, this approach is considered the first to use NT to test Java Card applets. Java Card applets are used in vital areas in our lives, observing the execution of the applets. The goal is to validate whether they behave as intended and identify malfunctioning or not, which is an essential process that must be considered. There are two main steps in the test data generation process. In the first step, test data is generated randomly through the tool, and redundant data is removed. Also, we added the domain boundaries for each variable manually. Test data is generated on the predicates data type for each variable of the program associated. Here, we want to draw attention, we increased the number of iterations to investigate all cases in a random generation. The proposed algorithm is shown in Figure 1

Where, overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of digits – either higher than the maximum or lower than the minimum re-presentable value.

False branch c3 is any value that makes the condition of the branch False.

The algorithm starts by creating a pool of random test cases, these test cases have to cover all false branches. The tester provides the program under test (applet) to the tool, an applet is classified depending on the false branch for every conditional statement. Then it analyzes each parameter to get the parameter data type. After that, the tool goes to the library to determine the maximum (max overflow) and minimum (minimum overflow) value for the branch data type parameter(s). Then the tool generates invalid data types (e.g., String data type instead of integer or float). Also, the

```
Input:  goal-oriented branch Ti, test case t i, time m1
Output: the Solution  pool P
while times of running < m1 do
        if a solution of Ti is not in P exists then
                wi = getDataType (Ti)
                for each wi do
                        generate Overflow Max c1i,
                                validating the application
                        against  c1i
                        generate Overflow Min c2i,
                                validating the application
                        against  c2i
                        generate False branch value c3,
                                validating the application
                        against  c3
                        generate Invalid data c4i,
                                validating the application
                        against  c4i
                Endfor
        Endif
```

Figure 1. The proposed algorithm for negative testing

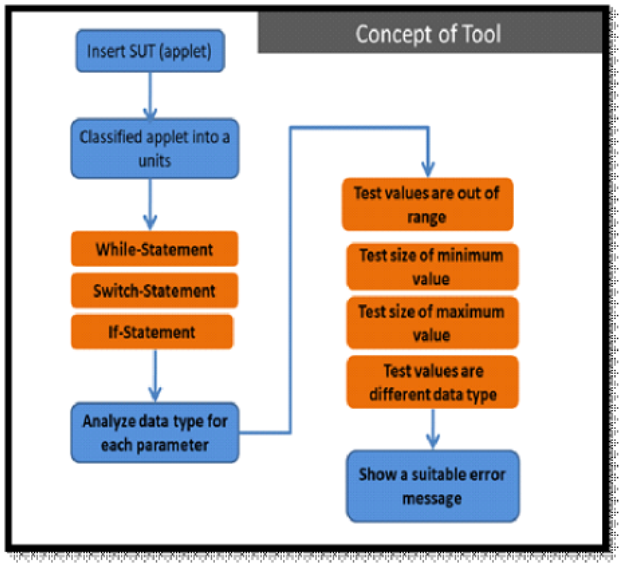

Figure 2. The concept of proposed tool

tool generates value to make the false branch. Finally, the tool validates the behavior of the program for each input. Figure 2 shows the concept of the tool. The following example explains the algorithm:

A. *Calculator:*

This program works to calculate the output by applying any one of the arithmetic operations between two numbers.

We have applied negative testing on this program before getting the results of the summation, the subtraction, the multiplication, and the division operations. The calculator program has three parameters: num1, num2, and op. We tested if the range of the two numbers is greater than the maximum size or less than the minimum size of data type. Then tested if the values are different data types like

```
//Java program to illustrate the nested-if statement

1.  classNestedIfDemo
2.  {
3.  Public static void main(String
    args[])
4.  {
5.       inti = 10;

6.    if(i == 10)
7.       {
        // First if statement
8.          if(i < 15)
9.        System.out.println("i is
    smaller than 15");

  // Nested - if statement
  // Will only be executed if statement above
  // it is true
10.           if(i < 12)
11.      System.out.println("i is
    smaller than 12 too");
12.            else
13.      System.out.println("i is
    greater than 15");
14.       }
15.     }
16.    }
```

Figure 3. Applet Java programs

String, Character, and Boolean for both numbers, and tested if the operation is any symbol except the four arithmetic operations (+, -, *, /). We executed the NT ten times. Each time has ten iterations. In each iteration, there is a random method to generate two integer numbers. Before applying any operation, the NT approach will work to decide if these numbers are Numeric data or not. If it is Numeric, then the tester should investigate whether the numbers are in the size range of data type or not. We generated numbers in the rang [-40000 40000] in two cases integer and floating-point data type, but the operation was inserted manually by the tester. We applied the NT when the operands are Numeric and in the range, but the operation is a division and the denominator is zero, then the error message will appear because this is not allowed in mathematics laws. We applied the negative testing on Numeric values and other data type like Character, String, Boolean for operands. Then we found the average execution time for different data types was less than Numeric data types. We also applied the NT approach to the operations. The tester inserts any symbol except the arithmetic operations like (?!,@,.. etc). Figure 3 below explain the results of negative testing. Here, the shown figures are the screenshot from the results which appear from different test cases.

In some cases like the following program not always we can generate all test cases to apply NT, so NT can be applied in a branch in line 6 where i=10, but to apply NT in the nested if statement in line 8 where i¡ 15 if statement in line 6 must be true, so for line 8 only we can generate test case to false the branch on line 8 but if we apply the choices of test cases then it not reach line 8, and the same with the branch

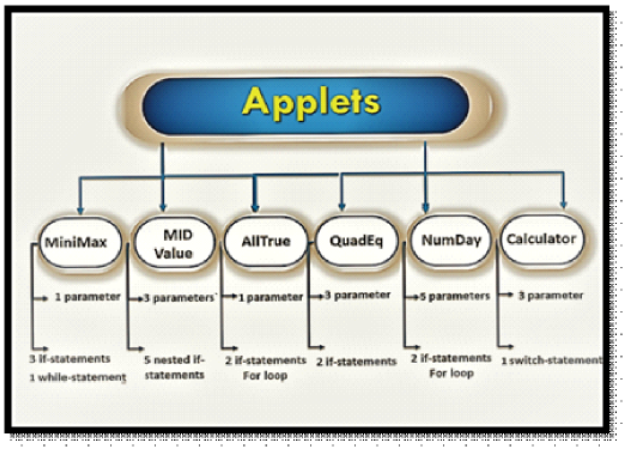

Figure 4. Information about each applet

on line 10, to reach line 10, if statements on line 6 and line 8 must be true, so we can generate test case only to false the branch on line 10. In this research, the NT applied to Six test Java smart programs that have been selected as benchmarks. Each program has unique characteristics to investigate the performance of NT as an approach of software testing to test Java Smart Card applets. These programs are briefly presented in Figure 4

Finally, each program has a set of parameters, which are tested in the NT approach. We will mention these parameters in detail to explain how to program works?

B. *Minimax*

This program works to find the minimum and maximum values from the array's values. In our experiment, the size of the array equals 5, the first element in the array is called ' password '. To apply for the program and get the target, there is one condition which is the value of the first element (password) should be equal to 100. When we applied the negative testing for this program, we take the first element which does not equal 100, but when we need to test 'idx' variables which are denoted to the address of the value in an array, the ' pass' variable should be equal to 100 because the condition statements are nested.

C. *Middle Value (Mid)*

This program works to find the middle value between three integer values. In this program, the parameters X, Y, and Z are defined as short integer data types, and it was generated randomly with range [-50000 50000]. We executed the program to apply NT ten times. Each time, the program generates three values in ten iterations randomly. When we applied NT for the Numeric data type, some iterations give the middle value, but others give an error message because some values are out of range. Applying the NT on Character data type, the middle value cannot be obtained because there is a different data type. We noticed that the elapsed time, [12], [5] to test a Numeric data type ten times is more than to test a Character data type.

D. AllTrue

This program works to determine if all values in a logical array are true [5]. Here we applied NT to test this program when the values are greater than the maximum or less than the minimum size of data type, when inserting different data types for parameters like a Character data type, and when the counter exceeds the length of the array. So, we generated values for 'X' parameter randomly. Then we checked if it is greater than zero or not. Depending on the answer, we filled the location of the array with true or false. Then when we set the values of 'X' for each address until we reached an address beyond the length of the array. In this case, we needed to apply the NT approach to prevent the program from any failure. After that, we calculated the number of the locations which have a true value and decided if the array is all true or not. We set the length of the array in our experiment to 4. We applied the negative testing ten times and all attempts gave an error message except four iterations which gave true values to all arrays. The execution time for Numeric data type testing is more than Character data type.

E. Quadratic Equation (QuadEq)

This program works to find the roots of a quadratic equation and determine if there is one root, two roots, or no root (imaginary number). The quadratic equation needs three variables (a, b, and c) to find a root. Then we should calculate the discrimination to distinguish between the equation which has one root, two roots, or no root (imaginary number). We applied NT by values out of data type range and with different data types such as Character data type. We noticed the execution time to find the solution for quadratic equation in Character data type is less than Numeric data type.

F. Number Of Days (NumDay)

This program works to calculate the number of days between two dates in the same year. NumDay program needs five variables to be able to determine the two dates at the same year: day1,month1,day2,month2, and year. There are some previous conditions for each parameter which we generated randomly like day1 and day2 must be between 1 and 31, month1 and month2 must be between 1 and 12, and year must be between 1 and 10000. In February, the number of the days is 28 and often 29. That depends on the year if it is a leap year or not. We can determine that by some rules like if the remainder from dividing the year on 4 is zero. Month1 must be less than or equal to month2, if they are equal, day2 must be greater than day1 to calculate the number of days. When we applied the NT in our program in the Numeric data type case, we generated random values between [-10 40] for days, [-5 20] for months, and [-50 20000] for the year, but in different data types like the Character, we generated a random capital letter from A to Z. We noticed the execution time of the Numeric data type was greater than the Character data type. After applying the NT, sometimes we got the number of days if the generated random number is in the range. Other times, we got an error message, which means some entered data was invalid.

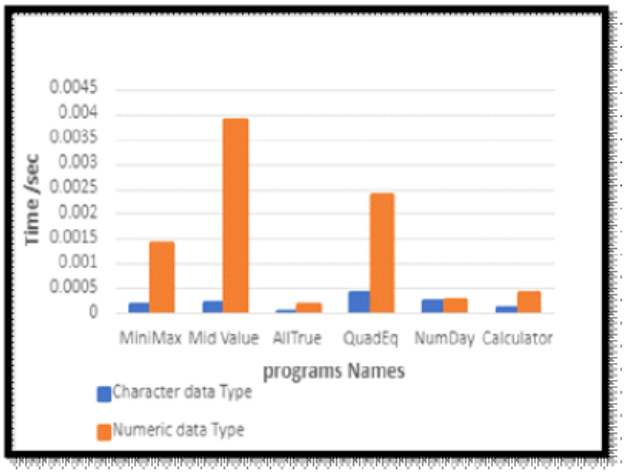

Figure 5. Average Execution Time for Numeric and Character Data Type for All Programs

4. EXPERIMENTAL RESULTS

The experimental results for each program under test were illustrated in this section. depending on some criteria to be measured like average required execution time when applying the negative testing was used on Numeric data type and Character data type, number of needed negative test cases to cover all negative situations, and average number of the required iterations to execute the target of the program. Table I below shows the results. The average execution time for the Numeric data type is higher than the Character data type in all programs because, in an invalid Numeric data type, there is a probability to get the target of the program. Forcing the application to execute more condition statements to test the parameters until ensuring whether the settings of the parameters to investigate all conditions to reach the mark or not. While in Character data type, from the first test, the behavior of the program will arrive to end the execution, because the parameter is not Numeric.

Figure 5 illustrates the difference between them. Also, in MiniMax and AllTrue programs, we have observed that the average number of iterations to get the target of the programs is higher than other programs because we are forced to execute the MiniMax plan more than 10 times to get the first iteration which investigates the target of the program. When we increase the range of random data generation, we noticed that the number of iterations to get the target increased. Figure 6 shows the average number of repetitions that are needed for each program.

In NumDays program, the number of needed negative test cases for applying NT is higher than residual programs because it has more parameters and more if-statements. Figure 7 illustrates the difference between several negative test cases needed to use negative testing for each application.

TABLE I. Experimental Results for All Programs

P. Name	A verage Execution Time(sec) Numeric data	Character data	Average Number of Iterations	Number of Negative Test Cases
MinMax	4	117	0.0001715	0.0014096
Mid Value	2	48	0.0002174	0.0038848
AllTrue	2	50	0.0000245	0.0001855
QuadEq	4	4	0.0004167	0.0023744
NumDay	8	10	0.0002508	0.0002758
Calculator	4	2	0.0001163	0.0004155

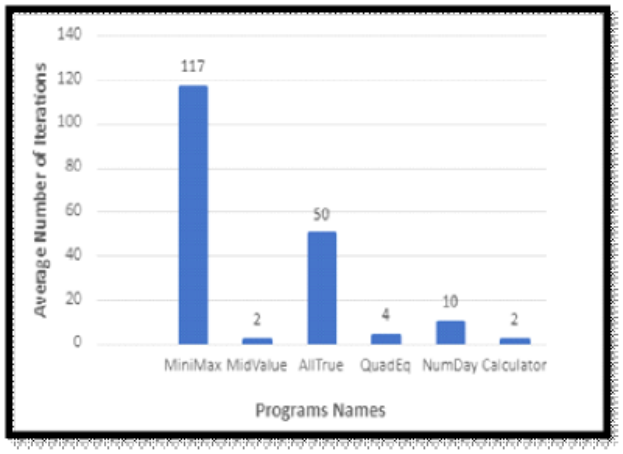

Figure 6. Average Number of Required Iteration to Get a Target for All Programs

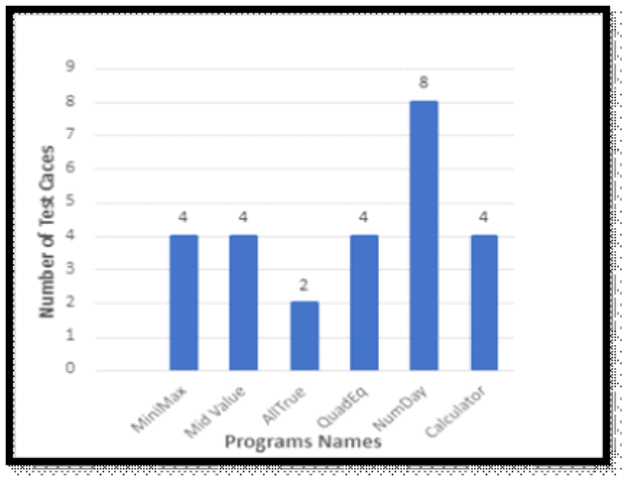

Figure 7. Number of Negative Test Cases for Each Applet

In summary, in Numeric data type, the behaviour of the program goes towards the target if all conditions were investigated, but in Character data type, from the first test, the error message will appear because it is a different data type, so there is no need for more time.

5. CONCLUSION AND FUTURE WORK

We applied the boundary values analysis technique to test Java Smart Card Applications in a negative testing approach. This type of application is deployed in highly essential areas around us to support access, identity, payment and other services

We have noticed the average execution time for the Numeric data type are higher than the Character data type in all programs. Because of the invalid Numeric data type, there is a probability to get the target of the program. We have also got the average number of iterations in the case of Numeric data only because the Character data type cannot investigate the objective of each original program. In the NumDays application, the number of needed negative test cases for applying negative testing is higher than residual programs because it has more parameters and more if-statements.

The central aspect of the future work in this paper is to make a real-world simulation of the automation NT approach on an actual Java Smart Card and other types of programs. Also, testing Java Card programs using different test coverage criteria such as path coverage to distinguish which is the most appropriate test coverage criterion for testing Java Smart Cards. In addition to Applied Negative Testing into Object-oriented and Compound data types.

REFERENCES

[1] B. D.J. and W. A., "High volume software testing using genetic algorithms," *In Proceedings of the 38th Annual Hawaii International Conference on System Sciences IEEE*, pp. 318–328, 2005.

[2] M. B. Alshraideh M, Bottaci L, "Using program data-state scarcity to guide automatic test data generation," *Software Quality Journal*, vol. 18, pp. 109–144, 2010.

[3] B. B., "Software testing technique," *Book 2nd ed.*, 1990.

[4] B. J., "https://dev.to/rfornal/negative-testing-4k55," 2022.

[5] A. M., "A complete automation of unit testing for javascript programs," *Journal of Computer Science*, 2008.

[6] Guru, "Automated vs. manual testing," *Accessed online [4.12.2021] available at https://www.guru99.com/difference-automated-vs-manual-testing.html*.

[7] T. O. Karhu K., Repo T. and S. K., "Empirical observations on software testing automation," *International Conference*, pp. 201–209, 2009.

[8] A. N. T. L. A. M., "Maintenance-oriented classifications of efsm transitions," *Journal of Software*, vol. 11, pp. 64–79, 2016.

[9] Guru, "Negative testing," *Accessed online [4.12.2019]available at,https://www.guru99.com/negative-testing.html*.

[10] F. L. K. P. Weelden A., Oostdijk M. and T. J., "On-the-fly formal testing of a smart card applet," *In IFIP International Information Security Conference*, pp. 565–576, 2005.

[11] A. M. H. N. Manaseer S., Manasir W. and A. O., "Automatic test data generation for java card applications using genetic algorithm," *Journal of Software Engineering and Applications*, vol. 8, 2015.

[12] A.-S. S. Alshraideh M., Mahafzah BA., "A multiple-population genetic algorithm for branch coverage test data generation," *Software Quality Journal*, vol. 19, pp. 489–513, 2011.

[13] A. M. W. Allawi H.M. and A. M., "A greedy particle swarm optimization (gpso) for testing real-world smart card applications," *International Journal on Software Tools for Technology Transfer*, vol. 6, pp. 1–12, 2018.

[14] M. H. and B. L., "Automatic test generation for java card applets," *International Java Card Workshop*, pp. 121–136, 2000.

[15] M. I. Petrova-Antonova D., Ilieva S. and M. D, "Towards automation design time testing of web service compositions," *6*, pp. 61–70, 2012.

[16] M. A. Villalobos-Arias L., Quesada-López C. and J. M., *International Conference on Software Process Improvement*, pp. 165–174, 2018.

[17] D. S. and K. D., "Automation softwaretestingon web-basedapplication," *Software Engineering*, pp. 691–698, 2019.

Amena Bataineh is an instructor in Dar Al-arqam school since 2011, licensed ICDL examiner and trainer . She got her master degree in computer science from the university of Jordan in 2019. In addition to bachelor of computer science from the university of alyarmouk, Jordan in 2007. Here research interest is software testing fields especially in negative testing .

Mohammad Alshraideh is a Professor of Software Engineering in the Department of Computer Science at the University of Jordan, Jordan. He got his PhD in Software Engineering (Testing) from the University of Hull (UK) in 2007. He has 15 years of experience in the IT industry before moving to academics. His IT experiences include the design and development of software systems and management of software development projects. He worked in different administrative positions (Head Director Assistant for Computer Technology at the Hospital of the University Human Resource Director at the University of Jordan, Registrar General at the University of Jordan), and Dean of Graduate school. He was granted several academic projects. He has published more than thirty papers in his research areas. He participated in many workshops, seminars, and conferences in the field of software engineering, and computing. His research interests are Software Testing, Computational Intelligence, Data Mining, Digital Humanity.

Amjad Hudaib is a Prof. of software engineering at the Department of Computer Information Systems (CIS), University of Jordan. He received his Ph.D. in Computer Science from University of Pisa, Italy in 2003. He earned his MSc. and BSc. In Computer Science from University of Jordan in 2000 and Mutah University in 1991; respectively. Prof. Hudaib was the Dean of King Abdallah II School of Information Technology During (Sept. 2018- Sept 2020). He was the Chairman of the Department of Computer Information Systems during (2017-2018), (2004– 2008). He has acted as the Director of Accreditation and Quality Assurance Center at the University of Jordan (2009-2014). Also, he acted as an Assistant Dean for Labs Affairs (2003-2004). His teaching and research interests cover a broad range of computing and business subjects including software engineering, project management, software testing, architectural design, data mining, optimization algorithms, and artificial intelligence techniques. Prof. Hudaib published more than eighty numerous international peer reviewed articles in scientific journals and conferences including several number of case studies. Also, he co-authored the official text books used for Ministry of Education for grades seven to twelve (Tawjihi). As for scientific and graduate studies, he supervised (and currently supervising) more than 70 Master and PhD Students and more than 60 bachelor graduation projects at The University of Jordan and other universities. Also, he was editor and reviewer in several well reputed international conferences and journals.

Fadi Wedyan is an Associate Professor in the Department of Software Engineering at the Hashemite University, Jordan. He completed his PhD in computer science at Colorado State University (2011). He received his Master in computer science from Colorado State University (2008). He also holds a Masters in Computer Science from Al-albayt University, Jordan (1999). His research interests include Evolutionary software testing, search-based software engineering, software quality metrics, and software design. His interests also include AI applications mainly planning and scheduling, and classification. He is also interested in mobile computing and the design and development of smartphone applications for health care, educational, and social uses.