International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 18 (2015) pp 39091-39100

Graph Based Test Case Generation

V.Vani
Department of Information Technology, Easwari Engineering College Chennai 600089, Tamil Nadu, India v.vani465@gmail.com

G. S. Mahalakshmi
Department of Computer Science and Engineering College of Engineering Guindy, Anna University Chennai - 600025, Tamil Nadu, India gsmaha@annauniv.edu

Betina Antony J
Department of Computer Science and Engineering College of Engineering Guindy, Anna University Chennai - 600025, Tamil Nadu, India betinaantony@gmail.com

Abstract
Test case generation is among the most labour-intensive tasks in software testing. It has a strong impact on the effectiveness and efficiency of software testing. For these reasons, it has also been one of the most active topics in the research on software testing for several decades, resulting in many different approaches and tools. We present a comprehensive test case generation technique from UML models. We use the features in UML 2.0 use case diagram including conditions, iterations, asynchronous messages and concurrent components. In our approach, test cases are derived from analysis artifacts such as use cases and constraints specified across all these artifacts. We construct Use case Dependency Graph (UDG) from use case diagram. The test cases thus generated are suitable for detecting synchronization and dependency of use cases and messages, object interaction and operational faults.

Keywords: Use case dependency graph, test case generation, use case diagram, UML 2.0 and use case.

Introduction

Software testing is indispensable for all software development. It is an integral part of software engineering discipline. However, testing is labour-intensive and expensive. It is often accounted for more than 50% of total development costs. Thus, it is imperative to reduce the cost and improve the effectiveness of software testing by automating the testing process. In fact, there has been a rapid growth of practices in using automated software testing tools.

Currently, a large number of software test automation tools have been developed and become available on the market. It is no surprise that a great amount of research effort in the past decades has been spent on automatic test case generation. As a result, a good number of different techniques of test case generation has been advanced and investigated intensively. On the other hand, software systems have become more and more complicated, for example, with components developed by different vendor, using different techniques in different on different platforms.

Model-driven software development is a relatively recent software development paradigm. Its advantages are the increased productivity with support for visualizing domains like business and problem domains, solution domain and generation of implementation artifacts. In the model driven software development, practitioners also use the design model for testing software- especially object-oriented programs. Three main reasons for using design model in object-oriented software testing are: (1) traditional software testing techniques consider only static view of code, which is not sufficient for testing dynamic behavior of object-oriented software, (2) use of code to test an object-oriented software is a complex and tedious task,. in contrast, models help software testers to understand systems in better way and find test information only after simple processing of models compared to code, (3) model-based test case generation can be planned at an early stage of the software development life cycle, allowing software developer to carry out coding and testing in parallel. For these three major reasons, model-based test case generation methodology becomes an obvious choice in software industries and is the focus of this paper.

Related Work

Automated test case generation started as early as 1980's. These test sequences were carried out for sequential circuits were the input power is digital signals (0's and 1's) [1]. The generation of test cases from software artifacts is however a recent innovation. Test case generation from Software UML artifacts has started a new field of research around the globe.

The need for test cases from usecases, though recognized widely in theory, is not really experimented due to various reasons. One of the main reasons is the unformulated text of usecases when compared to other UML views. Thus fuzzy methods of automated extraction of test cases are highly complicated. The main TCG (Test Case Generation) works for software started from requirement specifications [2] [3]. In this case the main operation is the conformance test of formal specification of requirements and validating the behavioural automata build from them. The main drawback of this method is however its cost. Hence it is restricted to critical systems.

Among all the works done on generating tests from UML artifacts, only a few concentrate on usecases. [4] describe an approach to generate system level test cases from an accurate description of the use cases, including preconditions and postconditions. Each use case is transformed into a state machine, and test objectives must be defined by the tester into

the underlying formalism. The main limitation identified by the authors concerns the partial automation of the approach, e.g., the transformation from a use case description to a state machine is done manually, and the natural language is used for pre and postconditions.

The authors of [5] propose test approaches based on the dependencies between the use cases and modeling those dependencies using graphical notations. In [6], a scenario-based approach is detailed in order to systematically derive test cases for system testing. Scenarios are formalized using state charts and a graphical notation called dependency charts, which is used to model the dependencies between scenarios. The objective of the dependency charts is larger than ours since they allow modeling several kinds of dependencies such as sequential dependencies but also abstraction dependencies, time dependencies and data resource dependencies. On the other hand, test generation from such diagrams is not automated, but based on a manual application of heuristics.

Modules and Methodology

Our idea of test case generation from use case involves the following modules depicted in the flow diagram (figure 1) are

1. Pre-Processing of Use case.
2. Use Case Dependency Graph Generation.
3. Test case generation.

The following section describes the methodology used in each module of test case generation. The illustration is done with ATM use case scenario.

A. Pre-Processing

Pre-Processing involves generation of keywords from the use case. Keywords are generated by tagging the parts of speech using a POS tagger. POS tagger tags nouns using the NN suffix using which nouns related to the use case are separated. This is followed by frequency calculation of the nouns from which frequently occurring nouns are identified which results in yielding few named-entities related to the Use case.

B. Tokenization of text

In this phase candidate keys generated in the previous phase, list of named entities for the given scenario and a sequence list of use case descriptions are considered. This input is in the form of text which is incorporated into a suitable data structure to facilitate word by word text processing. For this purpose, array data structure is chosen. Hence the document (Use case input sequence) is separated into tokens of words. Each word is in the tokenized form is identified for duplicates and mapped to suitable values for adjacency matrix construction. Each of the fields in the use case sequence is mapped to value ranging from 0 to n using a suitable hash table. And the valid and invalid cases of the field are mapped to X and Y (table 1).

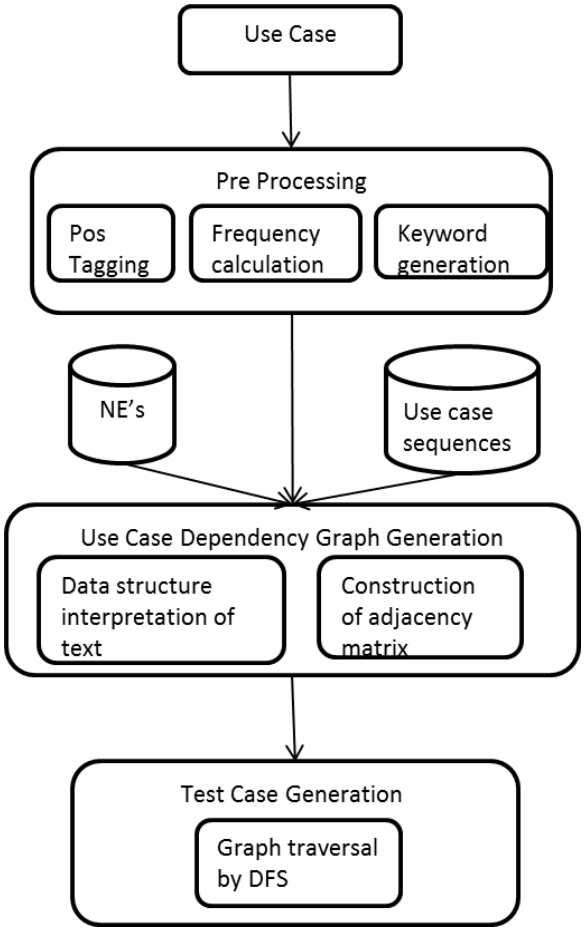

Fig.1. Overall Test case Generation System

TABLE.1. Code for Usecase Entities

Code	Function
0	Card Insertion
1	Pin
2	Amount
3	Transaction
4	Re-Insert Card
5	Success
6	Failure
X	Valid
Y	Invalid

C. Construction of adjacency matrix

This module constructs a graph from the tokens created in the previous stage. In order to represent the use case sequences in graphical form an adjacency matrix is constructed from the given use case sequence.

The construction of adjacency matrix uses the tokens created in the previous step. Each token mapped to a value is used as the index of an array. Hence a two dimensional array with row index ranging from 0 to n-1 (where n is the number of fields) and column index 0 to n-1 is constructed. Each entry in the adjacency matrix corresponds to the edge of the graph which is the relation between corresponding steps in the test case sequence. This entry is either 0 which represents absence of

edge, X represents a valid transition and Y represents and invalid transition in the use case sequence.

The adjacency matrix created can be interpreted as a graph with fields of the use case as nodes and the relation between the fields and edges. The edges do not only represent valid and invalid cases. Certain constraints which represent the validity of the field can also be interpreted as valid and invalid cases in the adjacency matrix as edges in the constructed graph. Table 2 is the adjacency matrix generated for the illustrative ATM use case sequence considered:

The corresponding graph resulting in this phase is given in figure 2.

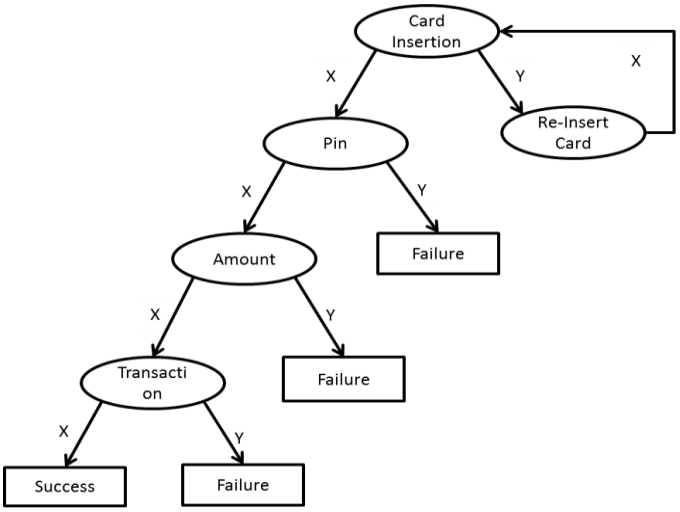

Fig.2. Graph for ATM Usecase

D. Test Case Generation

Test Cases are generated from the use case dependency graph by using Depth First Search method of graph traversal. This method of graph traversal generates all possible use cases denoted in the use case sequences. The illustration is of the graph traversal is given in figure 3.

TABLE 2. Adjacency matrix for ATM Usecase

	Card Insertion	Pin	Amount	Transaction	Re-Insert Card	Success	Failure
Card Insertion	0	X	0	0	Y	0	0
Pin	0	0	X	0	0	0	Y
Amount	0	0	0	X	0	0	Y
Transaction	0	0	0	0	0	X	Y
Re-Insert Card	X	0	0	0	0	0	0
Success	0	0	0	0	0	0	0
Failure	0	0	0	0	0	0	0

The test cases generated by Depth First Searching the graph in Fig.3 is shown in table 3.

TABLE 3. Test cases for ATM Usecases

S. No	Card insert (0)	Re insert card (4)	Pin (1)	Amount (2)	Transaction (3)	Expected Status (5)
1	valid	-	valid	>100	valid	success
2	valid	-	invalid	-	-	failure
3	valid	-	valid	<100	-	failure
4	valid	-	valid	>100	invalid	failure
5	invalid	valid	valid	>100	valid	success
6	invalid	valid	invalid	-	-	failure
7	invalid	valid	valid	<100	-	failure
8	invalid	valid	valid	>100	invalid	failure

Result and Evaluation

Graph based test case generation is done for the following scenarios and coverage is measured using a suitable coverage metric.

A. Coverage Metric

Test case cases have to be measured for completeness to ensure every requirement specified in the use-case description has been used appropriately.

In order to measure coverage we can use Sneed's equation for coverage.

$$Test\ case\ coverage = \frac{Tested\ attributes}{Specified\ attributes} \quad (1)$$

Every test case generated can be measured for coverage using equation 1.

A use case scenario specification handles requirements at all levels. Basic Test cases cover very less number of requirements compared to alternative flows. Hence coverage can vary between 0 and 1.To ensure the generated test cases cover the requirements at all levels, average of the coverage measure for all the test case for the given sequence is calculated (equation 2).

$$Mean\ Coverage = \frac{\sum coverage\ for\ each\ test}{no.of\ test\ cases} \quad (2)$$

The mean coverage has to lie between 0.4 and 0.6 which will denote the use-case description covers the requirements at all levels. The value of mean coverage can be neither 0 nor 1 since a value close to 0 specifies not all attributes have been used and a value close to 1 specifies alternate flows have not been handled. In table 4 TC1 covers only 1 attribute but TC5 covers all the 5 attributes. Hence a median value of 0.5 is selected to test coverage for each of the set of test cases generated. The following section shows the four stages of test case generation for ATM Usecase and its corresponding coverage value calculated.

B. Sample Scenario: ATM Use case

i. Use case specification

The use cases for ATM scenario were obtained in written from 50 BE (CSE) students from Anna University. One such usecase written in Jacobson format is given below. For our processing, we consider only the scenario sections of the usecases.

ii. Use case sequence

The usecases can be reduced to simple sequence by NLP processing. The sequences representation has only two parts: An entity or operation and status of it. The above usecase was condensed to 8 scenarios comprising of both basic and alternate flows. It is to be noted that the sequences are not restricted and depends purely on the type of usecases considered.

iii. Adjacency matrix construction

The usecase sequences thus obtained contains a list of operations or entity terms like "*card-insertion*", "*pin*" etc. These represent the nodes of the graph to be constructed. The graph is represented in the form of an adjacency matrix as shown in table 2.

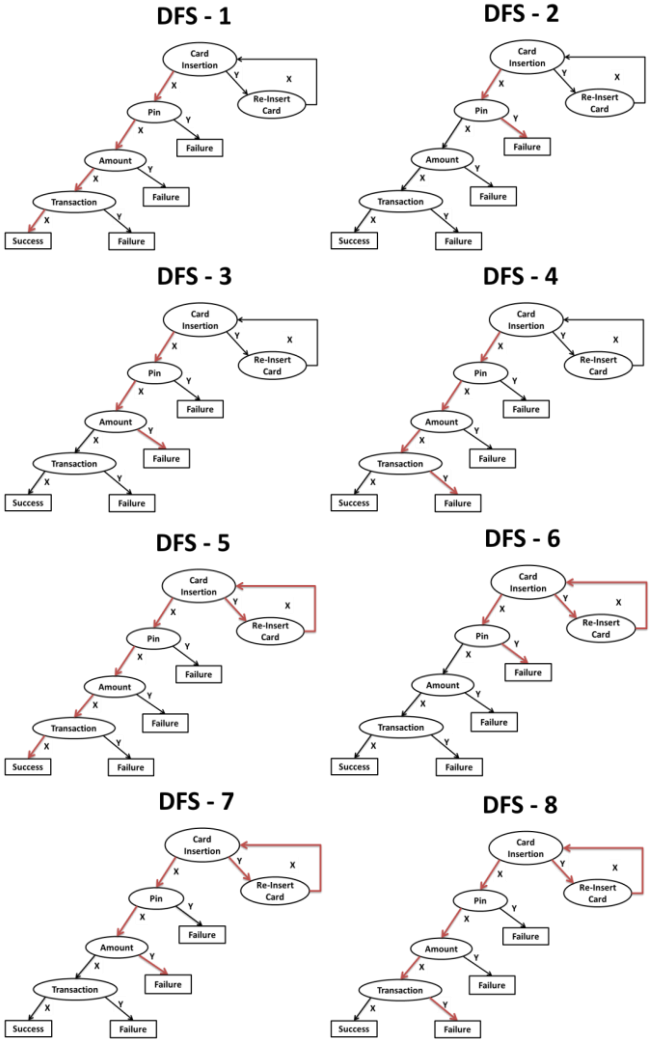

Fig.3. DFS of Scenario Graph

Terms: card-insert, card-reinsert, pin-enter, pin-reenter, pin-enter-count, enter-amount, amount-reenter, confirm, complete-transaction.

Basic flow: Successful Transaction

Step1: The user inserts the card into the ATM. If card-insert is VALID, the screen navigates to pin insertion screen.

Step2: The user enters his/her pin. The pin-enter is VALID and the screen navigates to amount insertion screen.

Step3: In the next screen, the amount to be withdrawn is entered. If the amount-enter is VALID, the screen navigates to the confirmation screen.

Step4: If confirm is VALID, complete-transaction SUCCESSFULLY.

Step5: Withdraw the amount from the machine.

Step6: Get Receipt and collect the ATM card from the machine.

Alternate flow 1: System failure/Invalid card

Step1: The user inserts the card into the ATM.

Step2: System encounters a failure in the machine or with the card. This results to card-insert INVALID.

Step3: User collects ATM card after UNSUCCESSFULLY complete-transaction.

Alternate flow 2: PIN invalid

Step1: The user inserts the card into the ATM.

Step2: There is a VALID card-insert. Screen navigates to PIN insertion screen.

Step3: User does an INVALID pin-enter.

Step4: The system prompts the user to enter the correct PIN within the next two attempts.

Step5: The user forgets the PIN and enters INVALID pin-enter.

Step6: The card gets blocked by the ATM after an UNSUCCESSFULLY complete-transaction.

Alternate flow 3: Insufficient balance

Step1: The user inserts the card into the ATM.

Step2: There is a VALID card-insert. Screen navigates to PIN insertion screen.

Step3: The user enters his/her pin. The pin-enter is VALID and the screen navigates to amount insertion screen.

Step4: The user enters the amount that exceeds the available balance in the account. The enter-amount value given is thus INVALID, and the system prompts to enter a valid amount.

Step5: Thus, the current transaction leads to an UNSUCCESSFULLY complete-transaction.

Alternate flow 4: Insufficient amount in the ATM

Step1: The user inserts the card into the ATM.

Step2: There is a VALID card-insert. Screen navigates to PIN insertion screen.

Step3: The user enters his/her pin. The pin-enter is VALID and the screen navigates to amount insertion screen.

Step4: The user enters the amount.

Step5: The entered amount cannot be dispensed by the system due to insufficient amount in the ATM. Thus, there is an INVALID confirm stage.

Step6: Thus, the current transaction leads to an UNSUCCESSFULLY complete-transaction.

Fig. 4.Usecases for ATM Scenario

TABLE 4. Reduced Usecase Sequence

1.card-insertion success pin success amount >100 transaction success successful	2. card-insertion success pin success amount <100 failed
3. card-insertion success pin failure failed	4. card-insertion success pin success amount >100 transaction failure failed
5. card-insertion failure reinsert-card success card-insertion success pin success amount >100 transaction success successful	6. card-insertion failure reinsert-card success card-insertion success pin success amount <100 failed
7. card-insertion failure reinsert-card success card-insertion success pin failure failed	8. card-insertion failure reinsert-card success card-insertion success pin success amount >100 transaction failure failed

iv. Result from adjacency matrix traversal

The adjacency matrix is then traversed starting with the first field as the flow of events is usually sequential. It is to be noted that last event of the sequence is always either "*Success*" or "*Failure*".

TABLE 5. Results of Traversal of Scenario Graph

Result of traversal	Explanation
0X1X2X3X5	This is a successful test case resulting in the first dfs done. In this every field is valid and results in a successful completion of transaction.
0X1Y6	This is a failure test case resulting in the second dfs done. The transaction fails due to an invalid pin.
0X1X2Y3	This is a failure test case resulting in the third dfs done. The transaction fails due to an invalid amount. The minimum amount to enter is 100.
0X1X2X3Y5	This is a failure test case resulting in the fourth dfs done. The transaction fails due to an invalid transaction
0Y4X0X1X2X3X5	This is a successful test case resulting in the fifth dfs done. The first card insertion results in a failure. The second card insertion succeeds which results in a successful transaction.
0Y4X 0X1Y6	This is a failure test case in the second insertion resulting in transaction failure due to an invalid pin entered.
0Y4X 0X1X2Y6	This is a failure test case in the second card insertion resulting in transaction failure due to an invalid amount entered.
0Y4X0X1X2X3Y6	This is a failure test case in the second card insertion resulting in transaction failure due to an invalid transaction.

v. Test cases Generated

The sequences thus obtained can now be represented in a table (table 6). Each DFS denotes one test case to be tested. The entire package gives a test suite for a given scenario.

TABLE 6. Test case Sequence

	Card insert (0)	Reinsert card (4)	Pin (1)	Amount (2)	Transaction (3)	Status (5)
dfs 1	Valid	-	Valid	>100	Valid	Success
dfs 2	Valid	-	Invalid	-	-	Failure
dfs-3	Valid	-	Valid	<100	-	Failure
dfs-4	Valid	-	Valid	>100	Invalid	Failure
dfs-5	Invalid	Valid	Valid	>100	Valid	Success
dfs-6	Invalid	Valid	Invalid	-	-	Failure
dfs-7	Invalid	Valid	Valid	<100	-	Failure
dfs-8	Invalid	Valid	Valid	>100	Invalid	Failure

vi. Test for coverage

Each test case is now evaluated for the number of parameters it covers. The average coverage for a test suite is then calculated.

TABLE 7. Test case Coverage Calculation

Test Case	Coverage
TC1	4/5
TC2	2/5
TC3	3/5
TC4	4/5
TC5	2/5
TC6	3/5
TC7	4/5
TC8	5/5
Mean coverage	0.6

The above set of operations was carried out for two other scenarios such as login usecases and course selection usecase. The results obtained are similar leaving the conclusion that the system can be applied to usecases of any domain.

C. Empirical Analysis

50 samples of use case descriptions were considered for the ATM scenario. Test cases generated for these use cases fall under 20 different types. After the set of operations, the final set of mean coverage values are listed in table 8.

TABLE 8. Mean Coverage for Test Scenarios

Scenario Number	Mean Coverage
1	0.53
2	0.52
3	0.57
4	0.43
5	0.53
6	0.6
7	0.51
8	0.60
9	0.56
10	0.55
11	0.55
12	0.55
13	0.54
14	0.55
15	0.60
16	0.59
17	0.56
18	0.56
19	0.55
20	0.41

The consolidated list of mean coverage values shows that the values of coverage lie predominantly between 0.4 and 0.6 which is our expected result.

Conclusion

We have presented a strategy for integration testing which combines information from use case sequence. This methodology predominantly uses graph based test case generation. Our proposed technique uses only use case sequences as the input. It doesn't require input in non UML-formats. Our approach exercises object interactions in the context of use case dependencies to fulfil the requirements of the user.

Future Works

The whole process of this work concentrates more on test case generation which has now become a fairly explored area of work. Test case evaluation is done using pattern elimination comparison which is a comparison with another set of test cases. A self-evaluating test case generating methodology will serve the purpose better and still remains an unexplored research topic for which this work may serve as a basis.

References

[1] Larrabee. T. "Test pattern generation using Boolean satisfiability."Computer-Aided Design of Integrated Circuits and Systems", IEEE Transactions on vol. 11, no. 1, pp: 4-15, 1992.

[2] G. Bernot, M.-C. Gaudeland, B. Marre, "Software Testing Based on Formal Specifications: A Theory and a Tool," Software Eng. J., vol. 6, no. 6, pp. 387-405. 1991.

[3] Dick. J and Faivre. A, "Automating the Generation and Sequencing of Test Cases from Modelbased Specifications," Proc. Int'l Symp. Formal Methods Europe, pp. 268-284, 1993.

[4] Fröhlich, P., and Link, J., "Automated test case generation from dynamic models." ECOOP 2000—Object-Oriented Programming. Springer Berlin Heidelberg, pp. 472-491, 2000.

[5] Briand, L., and Labiche, Y., "A UML-based approach to system testing." Software and Systems Modeling vol. 1, no. 1, pp. 10-42, 2002.

[6] Ryser, J., and Glinz, M., "A scenario-based approach to validating and testing software systems using statecharts." Proc. 12th International Conference on Software and Systems Engineering and their Applications. 1999.

APPENDIX

Sample 1:

	Valid- card	Correct-pin	Withdrawal-amount	Retry-pin	Transaction	Result
Dfs1	Success	Success	Success		Success	Successful
Dfs2	Failure					Unsuccessful
Dfs3	Success	Failure		Failure		Unsuccessful
Dfs4	Success	Success	Failure			Unsuccessful
Dfs5	Success	Success	Success		Failure	Unsuccessful
Dfs6	Success	Failure	Failure	Success		Unsuccessful
Dfs7	Success	Failure	Success	Success	Failure	Unsuccessful

Sample 2:

	Machine status	Valid card	Enter amount	Reenter amount	Enter pin	Mode of transaction	withdrawal	Retry pin2	Retry pin 3	Result
1	Success	Success	Success		Success	Success	Success			Successful
2	Success	Success	Success		Failure	Success	Success	Success		Successful
3	Success	Success	Success		Failure	Success	Success	Failure	Success	Successful
4	Success	Success	Failure	Success						Successful
5	Failure									Unsuccessful
6	Success	Failure								Unsuccessful
7	Success	Success			Failure			Failure	Failure	Unsuccessful
8	Success	Success	Failure		Failure			Failure	Success	Unsuccessful
9	Success	Success	Failure		Failure			Success		Unsuccessful
10	Success	Success	Failure		success					Unsuccessful
11	Success	Success	Success		Failure		Failure	Failure	Success	Unsuccessful
12	Success	Success	Success		Failure		Failure	Success		Successful
13	Success	Success	Success		Success		Failure			Unsuccessful

Sample 3:

	card-insertion	PIN-number1	PIN-number2	PIN-number3	amount-entered-correct	amount-reentered-correct	transaction-confirmation	result
1	Success	Success			Success		Success	Successful
2	Success	Success	Success		Success		Success	Successful
3	Success	Failure	Failure	Success	Success		Success	Successful
4	Success	Failure	Failure	Failure				Unsuccessful
5	Success	Success			Failure	Failure		Unsuccessful
6	Success	Success			Failure	Success	Success	unsuccessful

Sample 4:

	Insert-card	Pin-verification	Enter-amount-	view-balance	print-mini-statement	pin-reenter-verification	transaction	Result
1	Success	Success	Success				Success	Successful
2	Success	Success		Success			Success	Successful
3	Success	Success			Success			Successful
4	Failure							Unsuccessful
5	Success	Failure				Failure		Unsuccessful
6	Success	Failure	Failure			Success		Unsuccessful
7	Success	Success	Failure				Failure	Unsuccessful

Sample 5:

	insert-card	Reinsert-card	enter-pin	reenter-pin-attempt2	reenter-pin-attempt3	enter-amount	Result
1	Success		correct			Valid	Successful
2	Failure	success	incorrect	incorrect	correct	valid	Successful
3	Success		Incorrect	Incorrect	Incorrect		Unsuccessful
4	Success		Correct			Invalid	unsuccessful

Sample 6:

	Insert-card	Enter-pin	Enter-amount	Result
1	Success	Success	Success	Successful
2	Failure			Unsuccessful
3	Success	Failure		Unsuccessful
4	Success	Success	Failure	unsuccessful

Sample 7:

	machine-status	insert-card	enter-valid-pin	reenter-valid-pin	enter-withdrawal-amount	reenter-withdrawal-amount	Withdrawal	Result
1	Success	Success	Success		Success		success	Successful
2	Success	Success	Failure	Failure				Unsuccessful
3	Failure							Unsuccessful
4	Success	Success	Success		Failure			Unsuccessful
5	Success	Success	Success		Failure	Failure		unsuccessful
6	Success	Success	Success		Failure	Success	Success	Successful
7	Success	Failure						Unsuccessful
8	Success	Success	Success		Success		Failure	unsuccessful

Sample 8:

	ATMcard	Pin	withdrawal-amount	Transaction	Result
1	Accepted	Accepted	Accepted	Success	Successful
2	Unaccepted				Unsuccessful
3	Accepted	Accepted	Accepted		Unsuccessful
4	Accepted	Accepted	Accepted	Failure	Unsuccessful

Sample 9:

	insert-card	enter-pin	enter-amount	Transaction	Result
1	Success	Success	Success	Successful	Successful
2	Failure				Unsuccessful
3	Success	Success	Success	Failed	unsuccessful

Sample 10:

	insert-the-card	enter-pin-number	enter-amount	Transaction	Result
1	Valid	Valid	Valid	Success	Successful
2	Invalid				Unsuccessful
3	Valid	Invalid			Unsuccessful
4	Valid	Valid	Valid	Failure	Unsuccessful

Sample 11:

	card-insertion	Reinsert card	pin-number	reinsert-pin	amount-to-withdraw	Transaction	Result
1	Success		Valid		Valid	Complete	Success
2	Failure	Success	Valid		Valid	Complete	Success
3	Failure	Failure					Unsuccessful
4	Success		Invalid	Valid	Valid	Complete	unsuccessful
5	Success		Valid		Invalid		Unsuccessful
6	Success		Valid		Valid	Incomplete	Unsuccessful

Sample 12:

	card-validation	PIN-entry	enter-amount	ATM-check-sufficient-notes	transaction	Result
1	Success	Success	Success	Success	Success	Successful
2	Failure					Unsuccessful
3	Success	Failure				Unsuccessful
4	Success	Success	Failure			Unsuccessful
5	Success	Success	Success	Failure		unsuccessful
6	Success	Success	Success	Success	Failure	unsuccessful

Sample 13:

	card-insertion	enter-pin	enter-pin1	enter-pin2	amount-to-withdraw	transaction	Result
1	Proper	Unmatched	matched		Valid	Complete	successful
2	Proper	Unmatched	Unmatched	matched	Valid	Complete	Successful
3	Proper	Unmatched	Unmatched	Unmatched			Unsuccessful
4	Improper						Unsuccessful
5	Proper	matched			Invalid		unsuccessful
6	Proper	matched			Valid	Incomplete	unsuccessful

Sample 14:

	card-insert	Enter-pin-number	enter-withdrawal-amount	Money-dispensal	transaction	Result
1	Success	Success	Success	Success	Success	Successful
2	Failure					Unsuccessful
3	Success	Failure				Unsuccessful
4	Success	Success	Failure			Unsuccessful
5	Success	Success	Success	Failure		unsuccessful
6	Success	Success	Success	Success	Failure	unsuccessful

Sample 15:

	Card	PIN	withdrawal-amount	Transaction	Result
1	Accepted	Accepted	Accepted	Success	Successful
2	Unaccepted				Unsuccessful
3	Accepted	Accepted	Accepted		Unsuccessful
4	Accepted	Accepted	Accepted	Failure	Unsuccessful

Sample 16:

	Insert-card	Enter-PIN-attempt1	Enter-PIN-attempt2	Enter-PIN-attempt3	Valid-withdrawal-amount	amount-reentered-correct	result
1	Success	Success			Success		Successful
2	Success	Success	Success		Success		Successful
3	Success	Failure	Failure	Success	Success		Successful
4	Success	Failure	Failure	Failure			Unsuccessful
5	Success	Success			Failure	Failure	Unsuccessful
6	Success	Success			Failure	Success	unsuccessful

Sample 17:

	insert-card	Verify-PIN	Enter-amount	PIN-re-enter-verification	Transaction	Result
1	Success	Success	Success		Success	Successful
2	Failure					Unsuccessful
3	Success	Failure		Failure		Unsuccessful
4	Success	Success	Failure			Unsuccessful
5	Success	Success	success		Failure	Unsuccessful
6	Success	Failure	Failure	success		Unsuccessful
7	Success	Failure	success	Success	Failure	Unsuccessful

Sample 18:

	insert-valid-card	enter-valid-pin	enter-valid-amount	transaction	Result
1	Success	Success	Success	Successful	Successful
2	Failure				Unsuccessful
3	Success	Success	Success	Failed	unsuccessful

Sample 19:

	Card-insertion	PIN-authentication	withdrawal-amount	Transaction	Result
1	Valid	Valid	Valid	Success	Successful
2	Invalid				Unsuccessful
3	Valid	Invalid			Unsuccessful
4	Valid	Valid	Valid	Failure	Unsuccessful

Sample 20:

	insert-valid-card	Reinsert-valid-card	enter-valid-pin	reenter-valid-pin-attempt2	reenter-valid-pin-attempt3	enter-valid-amount	reenter-valid-amount	Transaction	Result
1	Success		Correct			Valid		Complete	Successful
2	Failure	Failure							Unsuccessful
3	Success		Incorrect	Incorrect	Incorrect				Unsuccessful
4	Success		Correct			Valid		Incomplete	Unsuccessful