Fault Analysis using Interaction Diagram

Dr K P Yadav
Director, KCC Institute of Technology and Management,
Greater Noida, Uttar Pardesh, India

Saroj Patel
Associate Professor, JNU,
Jodhpur, India

Tannu Arora
Associate Professor, JNU,
Jodhpur, India

Abstract- This paper mainly focuses on the interaction diagram test case generation techniques. We have considered the execution state of the system. Software testing phase of the software development life cycle generally find the faults or bugs during running state of the software. The case study of Railway Reservation System is discussed in this paper. The test cases are generated using sequencing or interactions in the system from which fault analysis will be computed.

Keywords – Fault Analysis, Interaction Diagram, Railway Reservation, Software Testing

I. INTRODUCTION

This is blunder to give assumption that automated testing is only imprisoned and play again of testing process manually. Actually, automation is basically different from testing manually: there are entirely different concepts and chance. Even the automation never completely replace manual testing, because automation is about inevitability and users are inherently erratic. Test Automation methods involves the following major factors for generating test case such as Unified Modeling Language (UML) diagrams, Testing types like Black Box Testing, White Box Testing, Testing techniques like model based, search based and symbolic execution, coverage criteria includes code based, fault based and function based, UML and Automation Testing tools and algorithms.[1]

So, use automation to verify what we expect, and use manual testing for what we don't. There is a good potential for reducing overall effort for test case generation from object oriented applications automatically from UML diagrams as the system features are broadly classified during design phase. Test case generation from design specifications has the added advantage of allowing test cases to be available early in the software development cycle, thereby making test planning more effective.[2]

This paper includes the case study of real time application which is Railway Reservation System. Test cases are generated from model based optimization techniques. This will be done with the help of UML behavior diagrams which are dynamic in nature. The behavior diagram used here is sequence or interaction diagram. The UML diagrams are created using ArgoUML tool. Firstly, class diagrams are generated using this tool. After that, interaction diagrams are drawn which is converted into IDG i.e., Interaction Diagram Graph. Then, this graph is traversed from which test cases are generated for each possible combination showing inputs and expected outputs. From the test cases, mutation testing is to be done to analyze the faults from where mutation score is computed. The overall steps are shown with the help of flow chart.

The figure 1 shows the steps of the analysis from the interaction diagram.

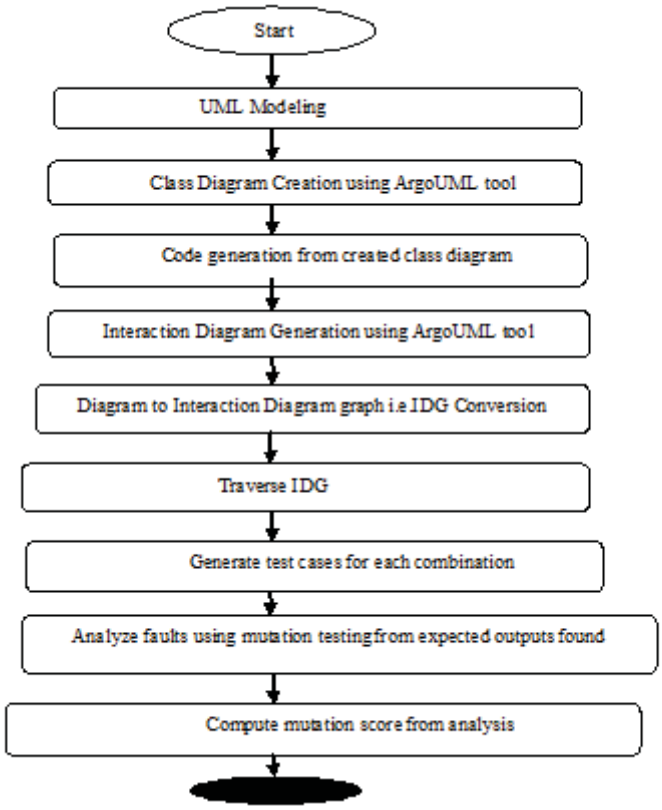

FIG 1 Flow Chart of Test Case Generation

II. CASE STUDY: RAILWAY RESERVATION SYSTEM

This case study includes the generation of test cases of Railway Reservation System. The steps performed will be according to the flow chart.

A.Class Diagram of Railway Reservation System

There are generally six classes in Railway Reservation System as shown in figure 2. These are basically train, customer, ticket, railway administrator, database and bank. These classes have their own attributes and operations according to their functionalities. There are also some relationships and roles which exists among these classes. The relationship between train and customer classes is aggregation. The multiplicity between them is one to many means that there are more than one passenger in the train but passenger belong to one train at a time. There is also aggregation between train and database classes. All the other classes have association relationship among them. The role of customer is to book the ticket. The customer information is stored in the database class. The role of railway administrator towards the train is monitoring and towards the customer is to authenticate, also administrator controls the database. The role between bank and ticket classes is to pay for the reservation.

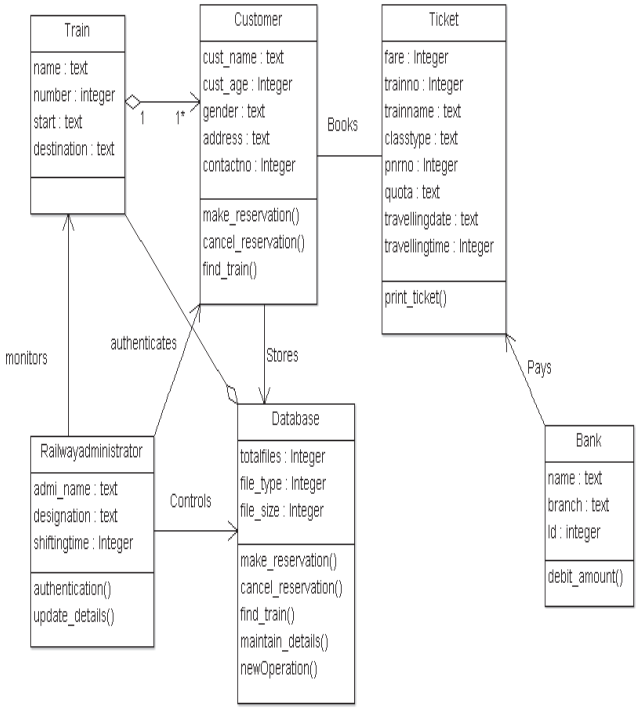

FIG. 2 Class diagram of Railway Reservation

B. Interaction Diagram of Railway Reservation System

In this Interaction diagram, there are seven objects i.e., customer, train, railway administrator, printer, database, ticket and bank which are connected among themselves through messages or interactions. These are seventeen sequences for complete reservation system as shown in figure 3.

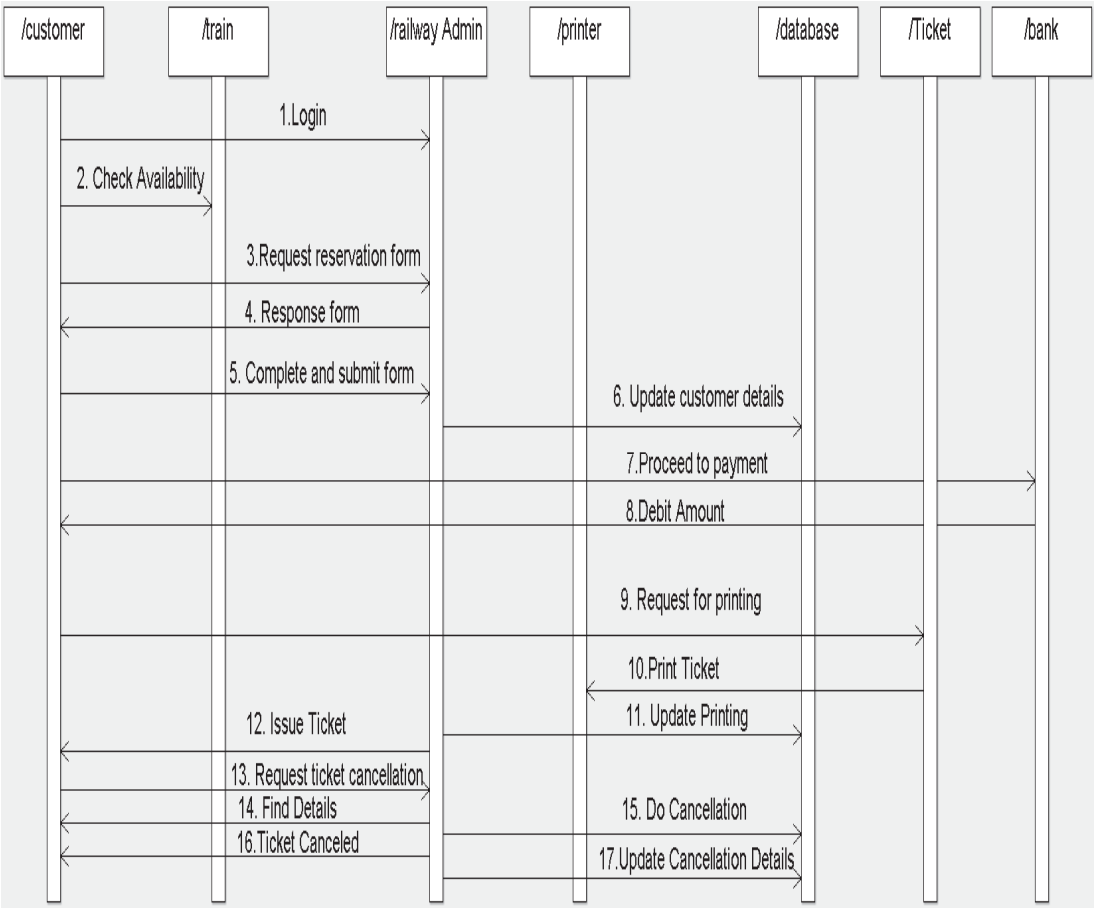

FIG 3 Sequence Diagram of Railway Reservation

C. Interaction Diagram Graph(IDG)

The created interaction diagram is converted into graph known as IDG i.e., Interaction Diagram Graph. This graph consists of nodes and edges. The nodes indicate the states and edges indicate the sequences or interactions among these states. State A is the initial state whereas, State B, C and D are final states. All others are intermediate states numbered from s1 to s17 showing executing stages as shown in figure 4.

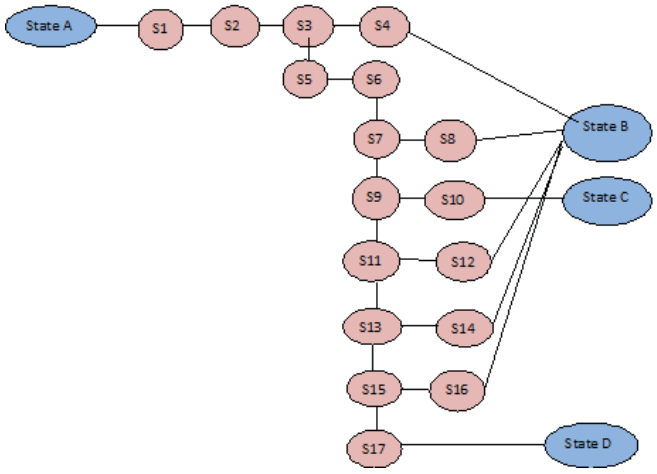

FIG. 4 Interaction Diagram Graph

D.Test Case Generation

After the interaction diagram conversion to IDG, the next step is the case generation of our study. These cases show input including present state and the expected output which consists of next state and the result or action performed. The action can be executing or exit stage depending on the intermediate or final states. These cases are also helps to analyze the faults in the system. Generated test cases are tabulated in table 1.

TABLE 1 Test Cases from IDG

Test Case	Input	Expected Output	
	Present State	Next State	Result
tc1	State A	S1	Executing
tc2	S1	S2	Executing
tc3	S2	S3	Executing
tc4	S3	S4	Executing
tc5	S3	S5	Executing
tc6	S4	State B	Exit
tc7	S5	S6	Executing
tc8	S6	S7	Executing
tc9	S7	S8	Executing
tc10	S7	S9	Executing
tc11	S8	State B	Exit
tc12	S9	S10	Executing
tc13	S9	S11	Executing
tc14	S10	State C	Exit
tc15	S11	S12	Executing
tc16	S11	S13	Executing
tc17	S12	State B	Exit
tc18	S13	S14	Executing
tc19	S13	S15	Executing
tc20	S14	State B	Exit
tc21	S15	S16	Executing
tc22	S15	S17	Executing
tc23	S16	State B	Exit
tc24	S17	State D	Exit

III. Result and Analysis

For any other program, faults may occur in any development phase of a software. A fault is a structural weakness in a software system that may lead to the systems eventually failing. To eradicate those faults in software system, an efficient test case is needed. The more efficient the test cases are, the more testing can be performed in a given time and therefore the more confidence, can be kept in the software. To solve this problem, mutation testing is performed. Fault testing technique, is used to achieve correctness of the software. There are various mutation operators. We have considered four of these(see Table 2).

TABLE 2. Mutation Operators

	Description
ABS	Absolute value insertion
Parameter	Change the letters of the parameter
DSA	DATA statement alterations
SDL	Statement deletion

A. Fault Analysis of Interaction Diagram

The faults were injected in the above mentioned operators and those were found shown in table 3 resulting in total fault found 28 while seeded fault was 32.

Table 3. Fault Analysis

Operator	Fault seeded	Fault Found
ABS	8	7
Parameter	7	7
DSA	10	8
SDL	7	6
	32	28

Now, based on the data obtained from the above table, the graph drawn in figure 5.

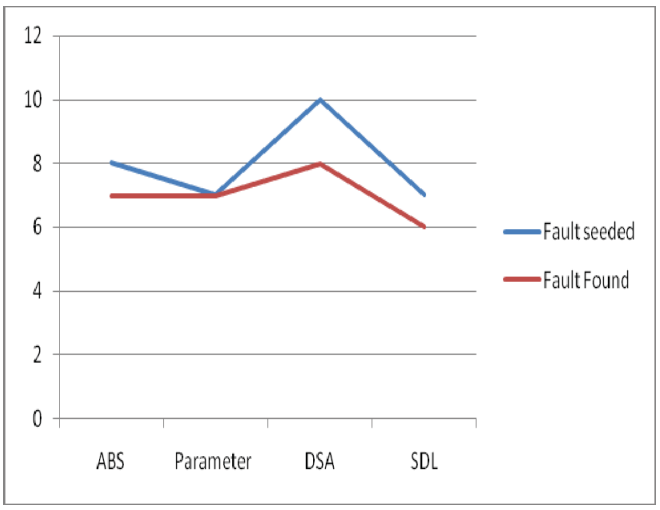

FIG. 5 Fault Analysis

After the analysis of faults, mutation score was computed. The formula of computing mutation score is given as
Mutation Score= (Fault found/Fault Injected)*100
Now, the values put in the above formula are
Mutation Score= (28/32)*100 = 87.5%

IV. CONCLUSION

Fault analysis is the process of seeding the faults in the software to improve the efficiency. After test case generation, mutation score is calculated by mutation or fault analysis. Experimental results have shown that in Interaction diagram, analysis of faults conducted is approximately 87.5%. In further researches, we will include model based testing that will generate test cases because MBT is beneficial than other techniques.
The technique can be expanded for more complex and bigger applications. In future, the approach can be employed to other UML diagrams like deployment diagrams etc.
However to improve our proposed system a combined approach is essential. We can also use collaboration diagram, since unlike interaction diagram, it has the capability to handle branch statements which are more complex in nature.

REFERENCES

[1] V. Maheshwari ,M. Prasanna. Generation of Test Case using Automation in Software Systems – A Review. Indian J of Sci & Tech 2015; 8:1-9.
[2] Ashish Verma,Maitrayee Dutta. Automated Test case generation using UML diagrams based on behavior. Int J of Innovations in Engg & Tech 2014; 4:31-39.
[3] G. Fraser, A. Zeller. Mutation-driven generation of unit tests and oracles. IEEE Trans on S/w Engg 2012; 38:278–92.
[4] M.H Moghadam, S.M Babamir. Mutation Score Evaluation in terms of Object- oriented Metrics. IEEE 4th Int Conf on Comp & Knowledge Engg 2014 ; 775-780 .
[5] J Lallchandani, R Mall. A Dynamic Slicing Technique for UML Architectural Models. IEEE Trans on S/w Engg 2011; 37:737-71.
[6] Vikas Panthi, Durga Prasad Mohapatra. Test Sequence Generation Using Sequence Diagram. Int Conf on Advances in Computing, Springer 2012; 174:277
[7] P R Mateo, M P Usaola, J Offutt. Mutation at the multi-class and system levels. Sci of Comp Programming, Elsevier 2013; 78:364–387.