I.J. Education and Management Engineering, **2017, 4, 21-28**
Published Online July 2017 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijeme.2017.04.03

Available online at http://www.mecs-press.net/ijeme

Detection of Infeasible Paths in Software Testing using UML Application to Gold Vending Machine

Gufran Ahmad Ansari

Department of Information Technology, College of Computer, Qassim University Al-Qassim Saudi Arabia (KSA)

Abstract

Software testing is an integral part of the software development cycle. Software testing involves designing a set of test cases. In white box testing, test cases are usually designed based using path testing. The basis path testing approach involves generation of test cases from a set of independent paths. Each test case is forced to execute a certain test path of the control flow graph. Some cases might arise paths of the control flow graph have no test data to force execution. These paths are infeasible paths. Identification and removal of infeasible paths earlier will reduce testing efforts and cost. In the present work, we used Unified Modeling Language (UML) for detecting of these infeasible paths. For detection of these infeasible paths, the author builds the control flow graph from sequence diagram and then generated independent paths from it. Each path is converted into a set of a linear equation and solved. If there is an inconsistent solution, then the corresponding path is infeasible. The presented approach is evaluated on a case study of an automatic gold vending machine.

Index Terms: UML, Design, Testing, Infeasible paths, Test cases

1. Introduction

Unified Modeling Language (UML) is well-known modeling language used by software engineers and researchers for designing artifacts of the software intensive systems. UML is a commanding visual modeling language which is used to solve software research problems [1, 2]. In recent years, an Object-oriented paradigm of the software development is heavily used in the software industry for development of manageable and reusable software. Despite the several benefits of the Object-oriented paradigm in software the industry, there are still several obstacles for both the developers and testers [3]. Software testing is done to find errors in the software. For various kinds of testing several kinds of UML diagram are used [4, 5].Traditional unit testing in Objected-oriented technology has shifted to class level testing and integration testing changes to cluster level

* Corresponding author.
E-mail address: 3632@qu.edu.sa , ansariga1972@gmail.com

testing where interaction between the object of different classes is tested. Software engineers are heavily dependent on software testing to confirm the bug-free software. In the software industry, the development cost usually grows too high due to software testing. Software testing is done to find the error in the software. Test engineers generate test cases to be executed on the software. After that, a result of the test cases is evaluated [6]. Authors proposed test suite design methodology and develop a tool that can be extended as needed by the particular project [7-8].The design of test cases is a creative activity. Usually, code or design specification is used by the testers for test cases generation. The test cases generated from code considers the internal structure of the programs and different paths are forced to be executed using a test data. There are some paths of programs control flow graph which are not feasible to execute using a test data. Some paths in the program's control flow graph are not feasible; they cannot be executed by any data. These paths are called infeasible paths. Identification of these paths earlier will reduce the testing efforts and cost. In our study author used UML sequence diagram for identification of infeasible path. Different techniques of identification of infeasible paths are proposed by various researchers. Kundu et. al [9] used UML sequence diagram technique for detection of infeasible paths from sequence diagrams. The authors found two interaction pattern namely Null Reference Check (NRC) and Mutual exclusive (MUX) which caused a large number of infeasible test paths. Ngoc and Tan [10] presented an approach of infeasible path detection based on the heuristic for dynamic test data generation. Authors identify some common properties of infeasible paths by execution traces of the program. Gong and Yao [11] presented a code-based approach of infeasible path detection. BMinh Ngoc Ngo and Hee Beng Kuan Tan proposed automatic detection of branch correlation of different conditional statements. The proposed approach is applied to some programs [12]. Bueno and Jeno [13] used a genetic algorithm for test data generation. A test input is iterated to find the value that satisfies a given path of the software testing. If no such path is obtained through repetition, then the path is infeasible path. This method of identification of an infeasible path is costly and use heavy computation. The paper is structured as follows: Section 1 provide a brief description about work, section 2 discuss about the infeasible path, section 3 describes the proposed approach, section 4 discuss about the design and detection of infeasible path from gold vending machine, section 5 gives a conclusion and discussion.

2. Infeasible Path

Using the control flow of the program that shows the control in a different statement of in the program. From beginning to the end there can be many paths in the program. Infeasible paths are those paths of the control flow graph of the program for which no data can traverse that path. An infeasible path is a path that is not implemented by any set of given values. Infeasible paths are controlled by approximately each genuine program and infeasible pathfinding is the task to identify potentially infeasible paths in the program. Since detection of infeasible paths is an important issue, the task may be exhausted during the test generation. As a result, inaccurate results may be produced. These can be handled through the use of by bug finding tools like Find Bugs [14]. Test data are generally obtained by the solving system of linear equations drawn from the path which is to be traversed. A case of an infeasible path, no solution is provided by the resulting system of linear equations. Due to the presence of infeasible paths different software engineering activities are affected in the form of extra cost.

Code A:

```
int sum(int a) {
/*1*/ int sum=0;
/*2*/ if(a>=0)
/*3*/ sum=sum+a;
/*4*/ if(a<0)
/*5*/ sum=sum-a;
/*6*/ return sum ;}
```

Path 1-2-3-4-5-6 is Infeasible because a>=0 and a<0 cannot be simultaneously satisfied by any value of a.

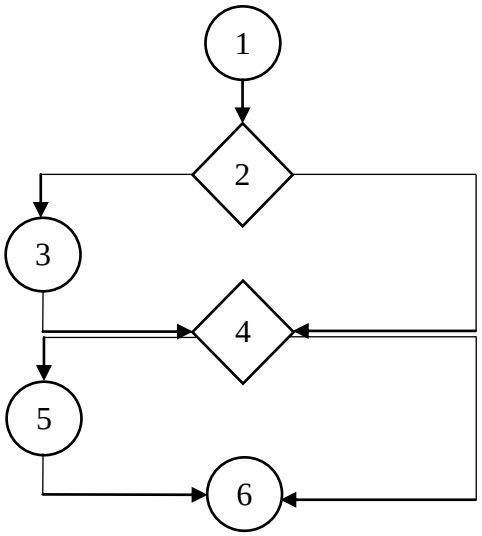

Fig.1.Control flow graph uses Code A

3. Proposed Approach

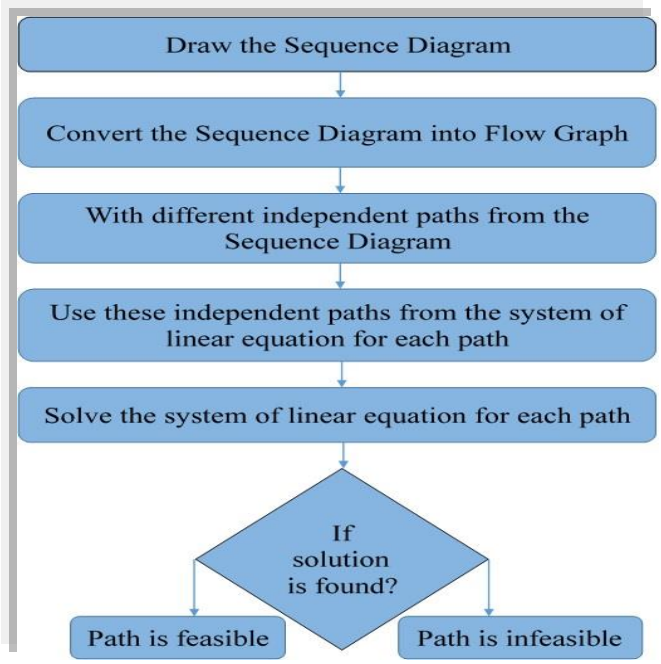

Fig.2. Proposed Technique for Detection of Infeasible Test Paths

The proposed approach is briefly defined as given points below:

1. Generate UML Sequence Diagram;
2. Transform the Sequence Diagram into Sequence Control Flow Graph;
3. From Sequence Control Flow Graph generate independent paths;
4. For each independent path generate set of linear equations, the decision node of each independent path is used to form the equation.
5. If decision nodes contain no input data and two outcomes with true and false, then the equation for that decision node is derived from Boolean logic.

   The true outcome for decision node is represented by 1, and 0 is used for false. Variables in alt condition can take 0 or 1 value. 0 means that alt condition is not satisfied and 1 means that alt condition is satisfied.

   Equation a+b=1 is taken for true a-b=0 for false where a and b are variables in alt condition of the sequence diagram.
6. Solve the decision nodes with no input value. If no solution is found then the path is infeasible otherwise path feasible.

4. Design and Detection of Infeasible Paths from Gold Vending Machine

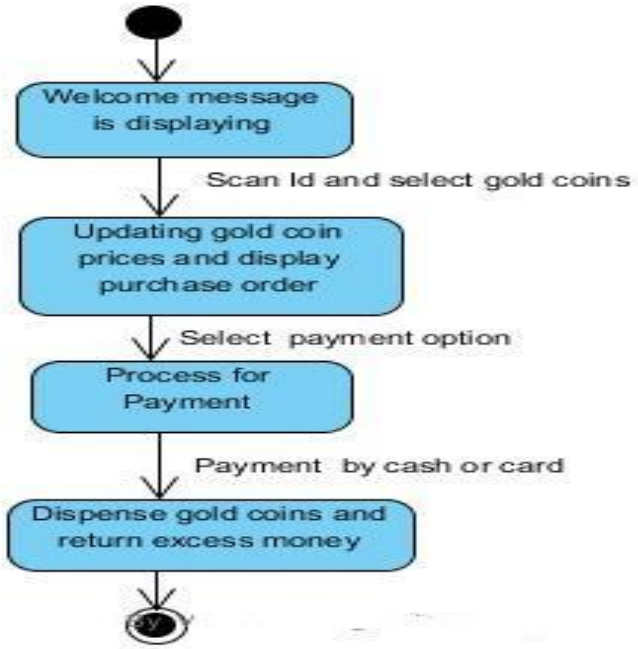

Fig.3. Represents the UML state diagram of Gold Vending Machine

Fig.3 represents the process of purchasing gold bar from Gold Vending Machine (Gold ATM) which is available in several high-security cities in the world [15]. These machines can accept either local currency of the country in which a machine is installed or credit card. In the initial state, the Gold vending machine is in the state which shows the welcome message. Then customer scans their ID and selects gold coins. After selection of gold coins machine updates the prices of gold coins online and display total purchase order.

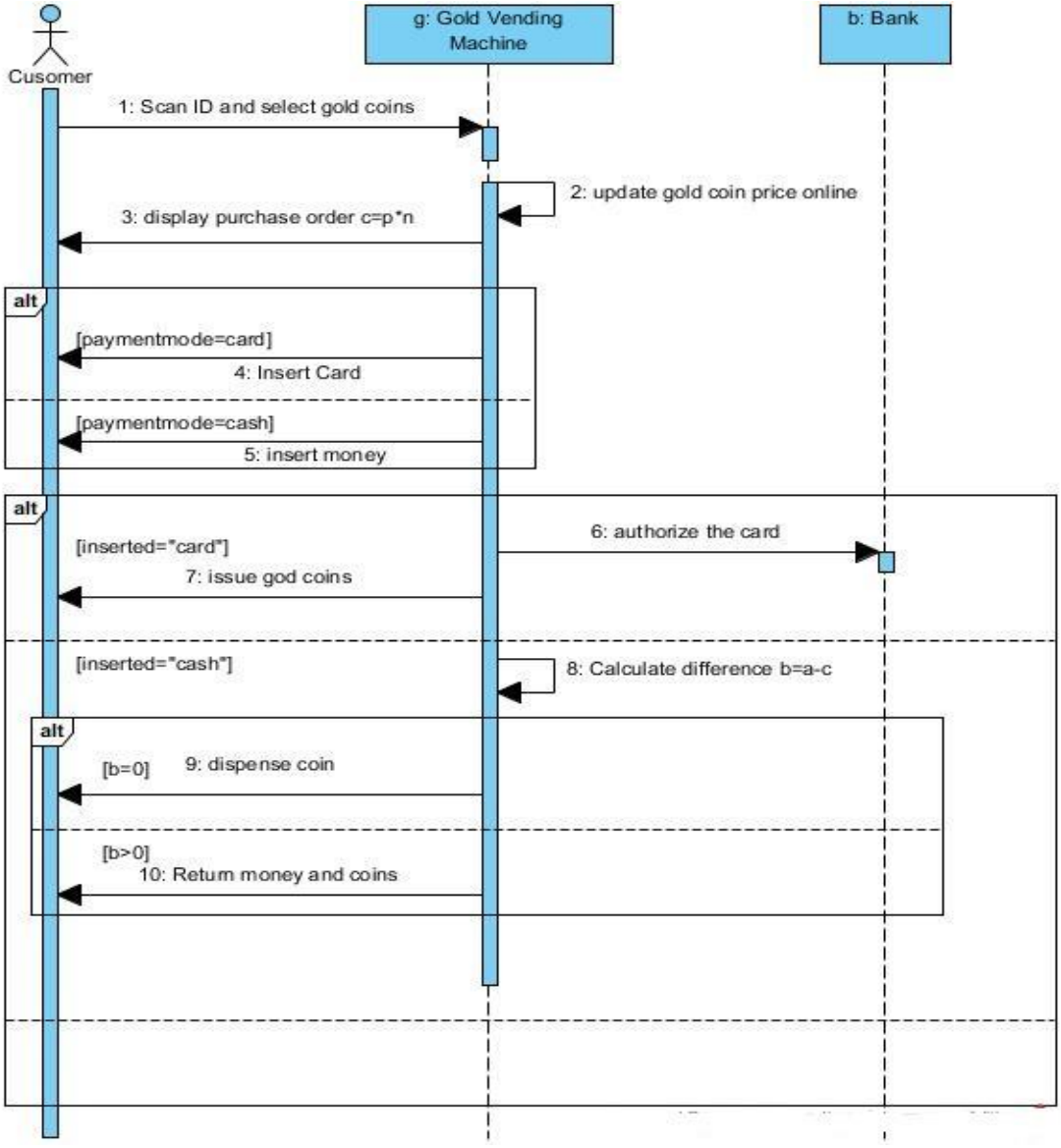

Fig.4. Sequence Diagram for Gold Coins Vending Machine

If the customer inserts card then vending machine check the authorization of card by respective bank. On successful validation of card, the machine dispenses the gold coins successfully. If payment is made with cash then extra money is dispensed by the machine along with the gold bars.

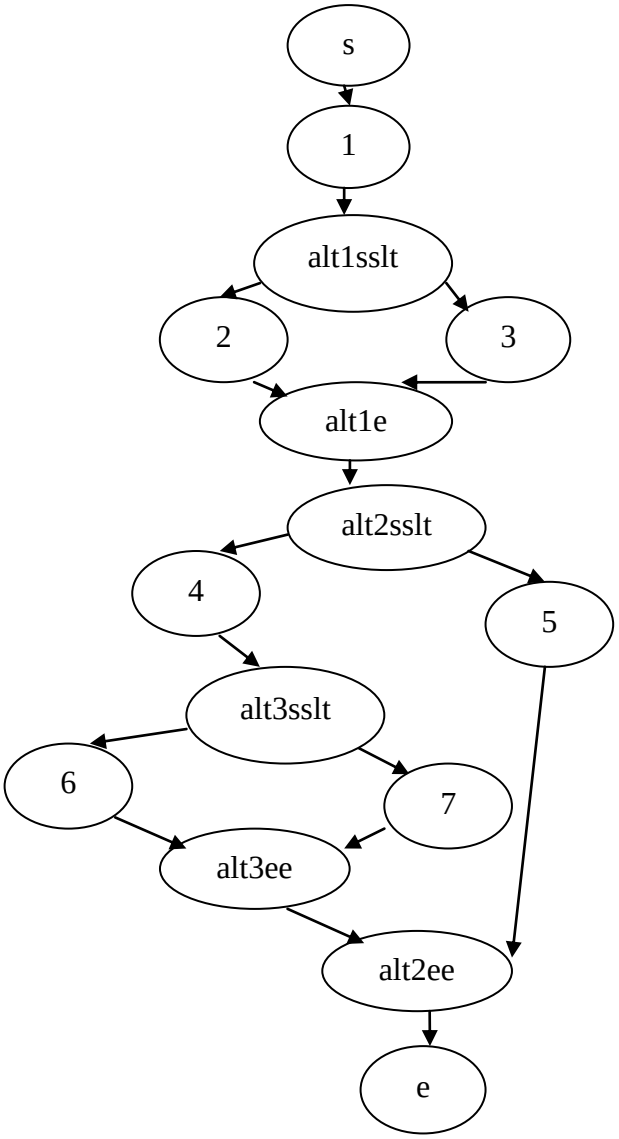

Fig.5. Sequence Control Flow Graph of Figure 4

Table 1. Independent Paths (For Simplicity Decision Nodes Are Discarded)

Path1	s-1-2-4-6-e
Path2	s-1- 2-5-e
Path3	s-1-3-5-e
Path4	s-1- 2 -4-7- e
Path5	s-1- 3- 4 -6-e
Path6	s-1-3-4-7-8-e

Consider the path s-1-2-5-e

In the above paths, there are three decision nodes. There will be are three equations for each decision node:

E1: a+b=1, where a=card, b=cash can take value 0 or 1
E2: a-b=0, where a, b are 0 or 1
F3: $a_3A+b_3P+c_3n+d_3$

On solving equation 1 and 2 we found a=1/2, b=1/2 which is not possible as a and b can take only 0 or 1 values. Therefore this path is infeasible.

Path s-1-2-4-6-e
E1: a+b=1
E2: a+b=1
F3: $a_3A+b_3P+c_3n+d_3$

On solving equation 1 and 2 we get a=1 and b=0 or a=0 and b=1 path is feasible

5. Conclusion

Infeasible paths of control flow graph pose many problems for software engineers. In the present article, we have presented an approach that can be used in finding out the infeasible paths in testing. The presence of infeasible paths causes expensive and non-optimized software. Identification of infeasible paths from design models can save testing time and efforts. We propose a novel method in order to detect infeasible path earlier without the creation of actual code. We build the control flow graph from a sequence diagram and then generate independent paths from it. Specifically, in the presented approach, we use UML sequence diagram as a source of control flow graph as sequence diagram resemble code and are designed at low abstraction level which is close to implementation level detail. As an application, the design of gold vending machine is presented using UML state diagram and sequence diagram. UML state diagram shows the different states in which an object goes into its lifetime. Sequence diagram represents the exchange of messages between objects in increasing order of time.

References

[1] http://technav.ieee.org/tag/1933/unified-modeling-language,[Accessedon 03-10-2016]
[2] G., Booch, J.Rumbaugh, and I Jacobson. “The Unified Modelling Language User Guide”. Addison-Wesley, Reading, MA. 1999.
[3] Grady Booch "Object-Oriented Analysis and Design with Applications", second Edition, Addison-Wesley, 1994.
[4] Ashalatha Nayak and Debasis Samanta, “Automatic Test Data Synthesis using UML Sequence Diagrams", Journal of Object Technology, vol. 09, no. 2, March-April pp. 75(104), 2010
[5] Huaizhong Li and C. Peng Lam, “Software Test Data Generation using Ant Colony Optimization", World Academy of Science, Engineering and Technology 2005.
[6] Soma Sekhara Babu Lam et al. “Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony” Procedia Engineering, Elsevier pp. 191-200, 2012.
[7] Abhinandan H. Patil et.al “Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques” I.J. Education and Management Engineering, pp. 32-39, 2016
[8] Abhinandan H. Patil, Neena Goveas and Krishnan Rangarajan. “Test Suite Design Methodology using Combinatorial Approach for Internet of Things Operating Systems”, Scientific Research Publishing, Journal of Software Engineering and Application, 2015.

[9] DebasishKundu, Monalisa Sharma and DebasisSamanta, A UML model Based Approach to detect infeasible paths, The Journal of System and Software, Vol. 107, pp. 71-92, 2015
[10] Minh Ngoc Ngo and HeeBengKuan Tan, Heuristic Based Infeasible Path detection for dynamic test data generation, Information and Software Technology, Vol. 50, Issue 7-8, pp. 641-655,2008
[11] D. Gong and X. Yao, Automatic Detection of Infeasible Paths in Software Testing, IET Software, Vol. 4, No. 5, pp. 361-370, 2010.
[12] BMinh Ngoc Ngo and Hee Beng Kuan Tan, "Detecting Large Number of Infeasible Paths through Recognizing their Patterns", ESEC/FSE07, September 3-7, Cavtat near Dubrovnik, roatia, 2007.
[13] Paulo Marcos, Siqueira Bueno and Mario Jino, Automatic Test Data Generation for Programs Paths using Genetic Algorithms, International Journal of Software Engineering and Knowledge Engineering, Vol. 12, No. 6, pp. 691–709, 2002
[14] Hovemeyer, D. and W. Pugh "Finding bugs is easy", ACM SIGPLAN Notices, 39: 92-106, 2004
[15] http://www.goldbarsworldwide.com/PDF/RT_8_Gold_Vending_Machines.pdf

Authors' Profiles

Dr. Gufran Ahamd Ansari received his Bachelor degree (B.Sc. Computer Science) from Shia P.G. College, Lucknow in 1997, MCA from DR. B.R. Ambedkar University Agra in 2002 and Ph.D. (Computer Science) from Babasaheb Bhimrao Ambedkar (A central) University, Lucknow, U.P., India in 2009. He is currently working as an Assistant Professor in the department of Information Technology, College of Computer, Qassim University, Saudi Arabia. He has over 14 years of experience in teaching undergraduate as well as postgraduate students of Computer Science, Information Technology and Computer Applications. He has authored more than 30 research papers on Software Engineering in leading international journals and conferences. His current research interests are Software Engineering, UML, Modelling, Testing, Artificial Intelligence, Software Quality, and Software Security.

How to cite this paper: Gufran Ahmad Ansari,"Detection of Infeasible Paths in Software Testing using UML Application to Gold Vending Machine", International Journal of Education and Management Engineering(IJEME), Vol.7, No.4, pp. 21-28, 2017.DOI: 10.5815/ijeme.2017.04.03