Journal of Universal Computer Science, vol. 24, no. 12 (2018), 1800-1820
submitted: 25/1/18, accepted: 15/11/18, appeared: 28/12/18 © J.UCS

Statistical Usage Testing at Different Levels of Testing

Kamaldeep Kaur
(New Delhi Institution of Management, New Delhi, India
kamaldeepkaurkalsi@yahoo.co.in)

Sunil Kumar Khatri
(Amity Institute of Information Technology, Amity University Uttar Pradesh, Noida, India
skkhatri@amity.edu, sunilkkhatri@gmail.com)

Alok Mishra
(Atilim University, Ankara, Turkey
alok.mishra@atilim.edu.tr)

Rattan Datta
(Mohyal Educational and Research Institute of Technology, Delhi, India
Rkdatta_in@yahoo.com)

Abstract: Statistical Usage Testing (SUT) is the testing technique defined in Cleanroom Software Engineering model [Runeson, 93]. Cleanroom Software Engineering model is a theory based and team oriented model that is based on development and certification of software in increments using statistical quality control [Linger 96]. SUT is a black box testing technique and concentrates on how the software completes its required function from the user's perspective [Runeson, 93]. SUT is carried out by developing usage models and assigning usage probabilities. Testing is carried out on usage models by performing statistical tests which are random sequences [Trammel 95]. Statistical testing can be viewed as a statistical experiment where random test cases are selected from all the usage models [Trammel 95].

This paper demonstrates the process and benefits of applying SUT at different levels of testing. Levels of testing include Unit level, Integration level, System level and Acceptance level. SUT is generally performed at System level and Unit testing is not the part of SUT. Unit testing makes it easier to access code and debug human errors. Detecting errors at an early stage helps reducing cost and effort. The paper proposes to allow Unit testing in Cleanroom Software Engineering Model, thus making it more flexible and suitable for varied applications. Unit testing is essentially performed to ensure that the code is working correctly and meets the user specifications [istqb, 15]. Errors may also exist when modules are integrated because of interchange of data and control information between various modules. Integration testing is performed when the modules are combined together to check their behaviour and functionality after integration. Once the Integration testing phase gets successfully completed, System testing is performed on the whole system [test-institute, 15]. The paper makes use of Student record software to demonstrate the process of performing SUT at different levels. In addition to performing SUT at System level, this paper helps in understanding the advantages of applying SUT at Unit level and Integration level.

Keywords: Statistical Usage Testing (SUT), Unit Testing, Integration Testing, System Testing, Markov Chains.
Categories: D.2.5

1 Introduction

1.1 Cleanroom Software Engineering

Cleanroom Software Engineering model is a theory – based team-oriented procedure for the development of high-quality software [Prowell, 99]. The Cleanroom approach focuses on the development of software that has correct design and high software quality [Prowell, 99]. Cleanroom Software Engineering has two major goals: a manageable development process and no failures in use [Prowell, 99]. The combined usage of conventional software modeling, verification and statistical quality assurance in this model leads to high-quality software [Pressmen, 00] [leansoftware, 15]. Cleanroom Software Engineering is the practical application of mathematical and statistical science [Prowell, 99]. The process begins with requirement gathering, followed by box structure specification and formal design. Once the design is complete correctness verification and code inspection are performed. In the later phase Statistical Usage Testing is carried out. SUT works by developing the usage models and assigning usage probabilities. In the next step statistical tests are performed on the usage models [Trammel 95].

The paper is divided into different sections. Section 1 gives the introduction of Cleanroom Software Engineering, Statistical Usage Testing (SUT) and Software testing levels. Section 2 makes use of Student record software to demonstrate the process of performing SUT at different levels. Section 3 deals with the findings and conclusions.

1.2 Statistical Usage Testing (SUT)

Statistical Usage Testing is the reliability certification method explained in the Cleanroom software development approach [Runeson, 93]. The foremost purpose of SUT is to certify the software reliability and to locate the faults which have high impact on the software reliability [Runeson, 93]. SUT provides statistically based stopping criteria of when to stop testing [Runeson, 93] .The intent of Statistical Usage Testing is not to eliminate faults like traditional testing, but to certify a definite predetermined reliability level [Runeson, 93].]. SUT is a black box testing technique. Black box testing does not consider or test the internal mechanism of a software [Kaur 14] .Various types of software can be tested using SUT like openoffice writer [Khatri 14] etc.

When performing Statistical Usage Testing it is essential to remove failures that are most serious and highly affect the reliability [Runeson 95]. SUT is based on usage models by producing statistically valid inferences about anticipated operational performance of a given version of the software [Prowell, 99]. In addition usage models offer a scientific foundation for model coverage testing, random testing, partition testing, and other forms of testing [Prowell, 99]. Statistical testing can be viewed as a statistical experiment where random test cases are selected from usage model of all uses [Trammel 95].

1.3 Software Testing Levels

Software testing levels are fundamentally used to spot the missing areas and avoid overlap and repetition between the SDLC (Software development life cycle) phases [istqb, 15]. In the entire SDLC there are many phases with a number of work products [istqb, 15]. Hence there are various levels of testing to test individual modules, integrated modules etc. Various levels of testing are Unit testing, Integration testing, System testing and User Acceptance testing .The overview of various levels is shown in figure 1.

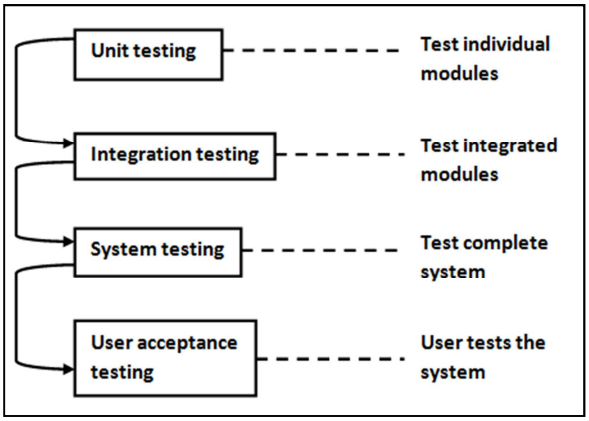

Figure 1: Overview of Software Testing Levels [test-institute, 15]

1.3.1 Unit Testing

Unit testing is performed on the smallest testable constituent of the entire software so that the number of test cases and test data are less [test-institute, 15]. It is essentially carried out by the developers to ensure that their code is working fine and it meets the user specifications [istqb, 15]. The smallest independent and testable part of the source code is called a unit [test-institute, 15]. The process of unit testing begins with testing individual units and collecting test results. If errors are found then the code is debugged and tested again. This continues till the code is error free.

To carry out integration testing it is essential to perform the unit testing for all the units. For unit testing one need to have a clearly defined test plan and test cases [test-institute, 15] .There are many benefits of unit testing. Only once all the units of the source code are working correctly one can proceed to integration testing. When unit testing is carried out, the code is refined and defects begin to lessen. So, the base of the software is strong and in the later stages the software development becomes faster [test-institute, 15].

1.3.2 Integration Testing

Integration testing is performed when the modules are combined together to check their behavior and functionality after integration [istqb, 15]. During this the testing team tests the interaction among different units and their output for various scenarios

[test-institute, 15]. Integration testing is performed to verify whether different units are able to execute as per expectations when combined [test-institute, 15].

Various types of integration testing include Big bang, Top down, Bottom up and Functional incremental integration testing [istqb, 15].

In big bang form of testing all the modules are integrated together to make up an entire system and then tested for errors [test-institute, 15].

Top down Integration testing is organized approach where the modules at the top level are tested first and then the lower modules are added step by step and tested. In Top down approach dummy modules called stubs can be used if modules are not available or not ready [test-institute, 15].

The Bottom up Integration testing is the contrary approach of top down. In this approach the bottom most modules are tested first and one by one the top level modules are added and tested [test-institute, 15]. In this method if the top level module is not available then a dummy module called a driver can used as the calling program [test-institute, 15].

In Functional incremental form of testing integration is done in order to uncover defects related to functional, requirement and performance levels [test-institute, 15].

1.3.3 System Testing

When the Integration testing is complete the testing team progresses to System testing where the entire system along with all components is ready for further testing [test-institute, 15]. In System testing the testers principally check the compatibility of the application with the system [istqb, 15]. The system is tested in its entirety to see if it is in conformity with the functional and technical specifications and the quality standards defined by the organization. It is also imperative that Integration testing is carried out by a very skilled testing team [test-institute, 15].

System testing is entirely a black box testing. The system is tested as per the requirement specifications. The testing is carried out from the user's perspective. It is performed to test the behavior of the application, design and anticipation of the end user. This testing authenticates and confirms the architecture of application and the requirements of the end user [test-institute, 15].

1.3.4 Acceptance Testing

Acceptance testing is essentially performed to ensure that the requirements of the specification are met [istqb, 15] .Once the system has been scrupulously tested using Unit, Integration and System testing, and then user acceptance testing is performed. The Acceptance testing tests the external interfaces as well as the internal functioning of the system. This testing is very critical as there are legal and contract requirements associated with the software for it to be accepted by the client [test-institute, 15].

Acceptance testing can be of two types: Alpha testing and Beta testing [test-institute, 15]. Alpha testing is performed to make sure that the product is of high quality. Alpha testing is carried out at the end of software development where the system can be tested completely. It is performed by testing team to test the software from the point of view of a customer [test-institute, 15]. Alpha testing is done at the developer's site [istqb, 15].

Once the Alpha testing is complete, Beta testing is performed to improve the software quality and to check if the software is in conformance to the requirement of the customer. It is performed in the real world scenario by the end users who actually use the software [test-institute, 15]. Beta testing is done at the customer's site. It is just carried out before the launch of the product [istqb, 15]. Acceptance testing also includes Contract Acceptance Testing, Regulation Acceptance Testing, and Operational acceptance testing [usersnap, 15].

2 SUT at Different Levels of Testing

2.1 Proposed Approach

The paper demonstrates the process and benefits of applying SUT at different levels of testing. Levels of testing include Unit Level, Integration Level, System Level and Acceptance level testing. SUT is generally performed at System level and Unit testing is not the part of SUT. The inclusion of Unit testing can improve this aspect of Cleanroom Software Engineering thus making it easier to access code and debug human errors. Unit testing is essentially performed to ensure that the code is working correctly and meets the user specifications. Detecting errors at an early stage helps reducing cost and effort. The paper proposes to allow Unit testing in CSE, thus making it more flexible and suitable for varied applications. Thus, allowing Unit testing would improve the software quality. SUT can also be combined with other White box and Black box testing techniques at Unit level for code scrutiny and checking the external interface. Detecting errors at an early stage prevents them from becoming very grave, thus saving effort, budget and time. After carrying out SUT at Unit level, SUT is performed at the Integration level. Errors may also exist when modules are integrated because of interchange of data and control information between various modules. Integration testing is performed when the modules are combined together to check their behavior and functionality after integration. After completing the Integration testing phase successfully, System testing is performed on the whole system. In System testing the system is tested in its entirety to see if it is in conformity with the functions, quality standards and requirements of an organization. The system is tested as whole and it is checked for compliance with requirement specifications. Once the entire system is tested user verifies the system (approach illustrated in figure 2).

2.2 Case: Student Record Software

The paper makes use of open source student record software developed in Visual Basic taken from a repository, for illustrating the proposed approach. The software is a simple open source student record keeping application taken from a repository, with 3 menus viz. forms (admission form, marks record, exit), edit (add class, edit structure) and reports (student report, final report, class report) [Kaur, 12]. The sample screen shot of the software under consideration is shown in figure 3 respectively.

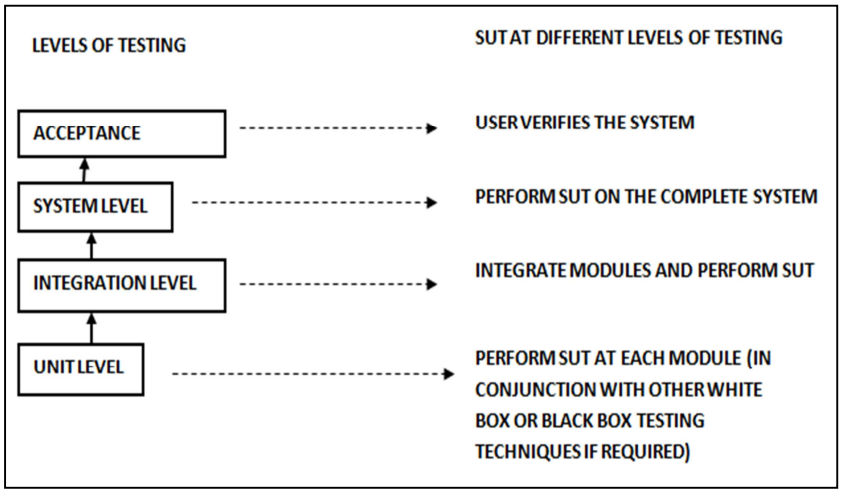

Figure 2: SUT at different levels of testing

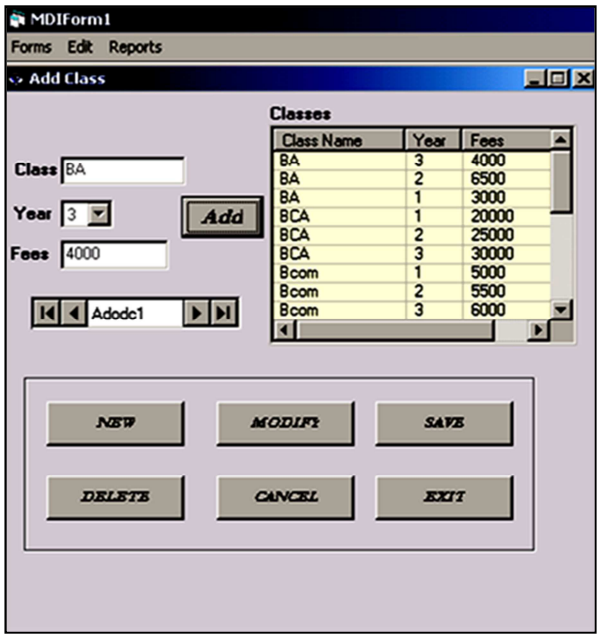

Figure 3: Add class form of Student Record Software

2.3 SUT at Unit Level

Unit testing is essentially carried out by the developers to ensure that their code is working fine and meets the user specifications [istqb, 15] .SUT can be performed at

Unit level by testing each module separately. SUT works by developing the usage models and assigning usage probabilities. In the next step statistical tests are performed on the usage models [Trammel 95]. A usage Markov chain for software has states that include all inputs and state transitions that are labeled with system inputs and transition probabilities [Whittaker, 94]. When all the states have been identified including the start state and a terminate state, a state transition diagram is drawn by considering the outcome of each input from each of the recognized states. The preliminary Markov chain for the student record software is shown in figure 4. A transition matrix is used to illustrate the transitions from various states of a Markov chain. The entries in the transition matrix are non-negative real numbers representing a probability.

For the problem under consideration the input domain consists of the up-arrow key, the down-arrow key, left arrow key and right arrow key which move the cursor to the desired menu item, and the "Enter" key, which selects the item [Whittaker, 94]. The software has 3 menus viz. forms (admission form marks record, exit), edit (add class, edit structure) and reports (student *report*, final report, class report) and the cursor can be placed on any of the menu item. Pressing "enter" on any of the menu item leads to the opening of the desired form.

Usage variable included is cursor location abbreviated as CL [Whittaker, 94] and takes on values "forms", "admission form", "marks record" , "exit", "edit", "add class, edit structure" , "reports" ," student report", "final report" and "class report" for each respective menu item.

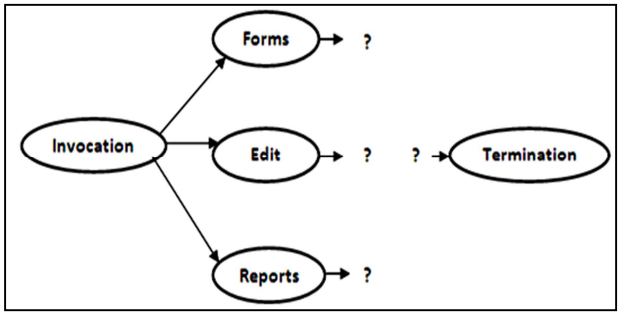

Figure 4: Initial states of Markov chain

Once the usage Markov chain is complete, the testing Markov chain is constructed. The test cases are a series of input sequences generated randomly and applied to software [Whittaker, 92]. Initially, the testing Markov chain has the identical states and arcs as that of the usage Markov chain, with every arc marked with a count of zero. The arc frequency counts are updated as the test cases are generated and executed [Whittaker, 92]. As failures are discovered and the software's internal faults repaired, the software evolves, becoming more or less reliable, depending on the success of the fixes [Whittaker, 94].

SUT is performed at Unit level by testing various modules separately. The Markov chain for module 1 is shown in figure 5 and its transition matrix in table 1. The Markov chain for module 2 is shown in figure 6 and its transition matrix in table 2

and for module 3 Markov chain is shown in figure 7 and its transition matrix in table 3.At this stage, SUT can also be used with other testing techniques [Khatri, 15].

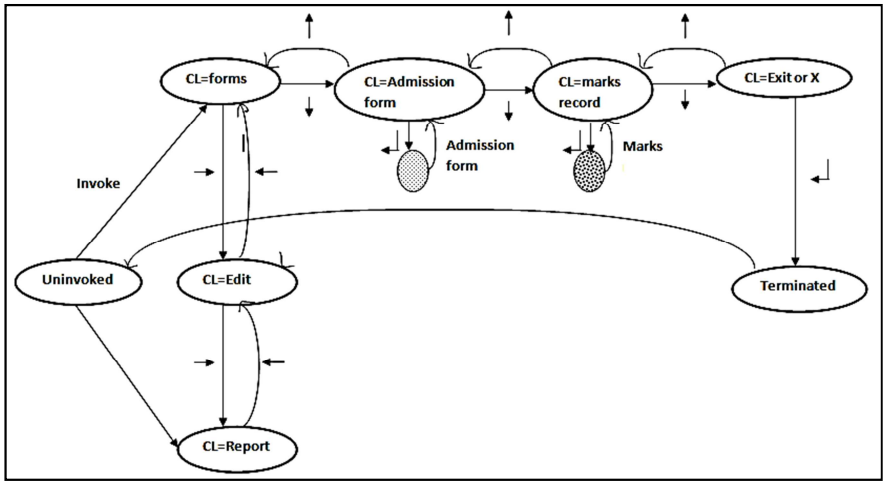

Figure 5: Markov chain for module 1

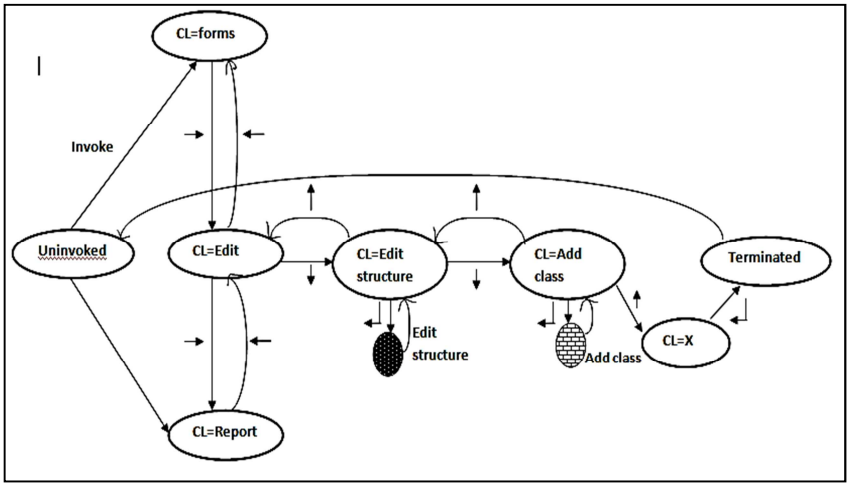

Figure 6: Markov chain for module 2

From state	**Transition stimuli**	**To state**	**Unif. Prob.**
Uninvoked	Invoke	{CL=forms}	1/3
Uninvoked	Invoke	{CL=edit}	1/3
Uninvoked	Invoke	{CL=reports}	1/3
CL=forms menu	↓ →	{CL=Admission forms} {CL=Edit menu}	1/ 2 1/ 2
CL=Admission form	↓ ↑ ↵	{CL=marks record } {CL= forms menu} Open admission form	1/3 1/3 1/3
CL=marks record	↓ ↑ ↵	{CL=exit } {CL= admission form} View marks record	1/3 1/3 1/3
CL=exit	↑ ↵	{CL=marks record } Exit application	1/ 2 1/ 2
Open admission form	New Open First Last Next Delete Save Previous Update Exit cancel	Add new student record Open existing record Goto first record Goto last record Goto next record Delete record Save record View previous record Update existing record Exit application Close admission form	1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11
Open marks record form	New Open First Last Next Delete Save Previous Update Exit Cancel	Add new student record Open existing record Goto first record Goto last record Goto next record Delete record Save record View previous record Update existing record Exit application Close admission form	1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11

Table 1: Transition matrix for module 1

From state	Transition stimuli	To state	Unif. Prob.
Uninvoked	Invoke	{CL=forms}	1/3
Uninvoked	Invoke	{CL=edit}	1/3
Uninvoked	Invoke	{CL=reports}	1/3
CL=edit menu	↓ → ←	{CL=edit structure } {CL= reports menu} {CL=forms menu}	1/3 1/3 1/3
CL=edit structure	↓ ↑ ↵	{CL= add class} {CL=edit menu } Open edit structure form	1/3 1/3 1/3
CL=add class	↑ ↵	{CL= edit structure} Open add class form	1/ 2 1/ 2
Open edit structure form	Save Modify Exit Cancel	save record Update existing record Exit application Close admission form	1/ 4 1/ 4 1/ 4 1/ 4
Open add class form	New Save Delete Modify Exit Cancel	Add new class Save new class Delete class Update existing record Exit application Close admission form	1/6 1/6 1/6 1/6 1/6 1/6

Table 2: Transition matrix for module 2

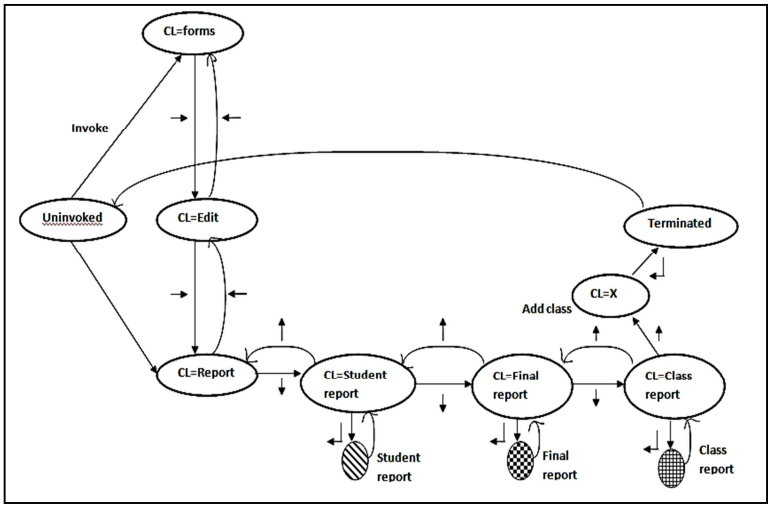

Figure 7: Markov chain for module 3

From state	Transition stimuli	To state	Unif. Prob.
Uninvoked	Invoke	{CL=forms}	1/3
Uninvoked	Invoke	{CL=edit}	1/3
Uninvoked	Invoke	{CL=reports}	1/3
CL=reports menu	↓ ←	{CL= student report} {CL= edit menu}	1/ 2 1/ 2
CL=student report	↑ ↓ ↵	{CL=reports menu} {CL= final report } View student report	1/3 1/3 1/3
CL=final report	↑ ↓ ↵	{CL=student report} {CL= class report} View final report	1/3 1/3 1/3
CL=class report	↑ ↵	{CL= final report} View class report	1/ 2 1/ 2
Open student report	Print Save Close Exit	Print report Save report Close report Exit application	1/ 4 1/ 4 1/ 4 1/ 4
Open final report	Print Save Close Exit	Print report Save report Close report Exit application	1/ 4 1/ 4 1/ 4 1/ 4
Open student report	Print Save Close Exit	Print report Save report Close report Exit application	1/ 4 1/ 4 1/ 4 1/ 4

Table 3: Transition matrix for module 3

The transitions in the usage Markov chain are static and they do not change during testing. On the contrary, the transitions in the testing Markov chain are dynamic and the probabilities in testing chain are updated [Whittaker, 94]. The initial testing chain is same as the usage chain, with all arc probabilities set to 0 [Whittaker, 92]. If there are no software failures then the next testing chain is attained by incrementing arc frequencies from "Uninvoked" to "Terminating" state. Thus, the frequency counts on arcs in testing chain are every time attained from particular sequences applied to software [Whittaker, 94]. Once the fixes have been applied, the testing chain's arc counts are reset [Whittaker, 94].

To include failure into the testing Markov chain, a new state labeled *f,* is added into the Markov chain .The arcs to the new state *f* and from the new state *f* have the count of 1. In case the failure is extremely critical, then the execution of software is stopped, and the arc from *f,* goes to "Terminating" state [Whittaker, 94]. But on the other hand, if the failure is not so fatal then the arc from *f,* goes to the

next state and the test sequence is permitted to proceed [Whittaker, 94]. For the student record software under consideration it is seen that if user tries to perform some operations before entering any data then an error is encountered. Therefore, a new state labeled *f,* is placed in Markov chain .The failure state is shown in Figure 8. The arcs to and from the *f,* have frequency 1. For example a failure state *f* 's placed after CL=forms when the user tries to perform any operations before entering the data. Also another condition RP i.e. Records Present is added to all the states. Similarly figure 6 and 7 were modified to implement failure states.

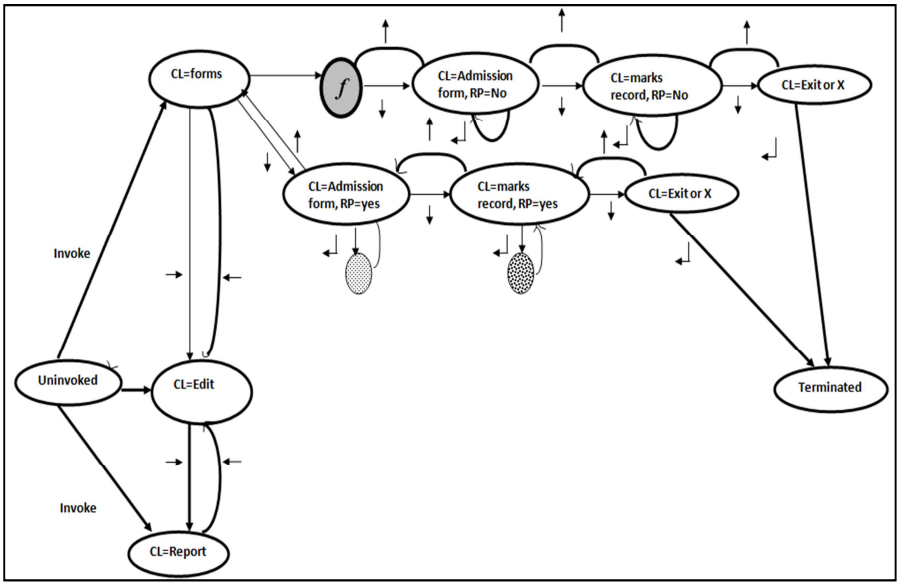

Figure 8: Markov chain with failure state for module 1

When no failures take place in the test history, convergence is eventually attained. The comparison of the actual development of Testing chain (including failures) with its expected evolution (without failures) assists statistical estimation of the software characteristics based on the software's actual performance. Any time in the testing process, the most recent test history is available for analysis [Whittaker, 94]. Stopping criteria for SUT is choosing some target reliability [Whittaker, 94].

2.4 SUT at Integration Level

Integration testing is performed when the modules are combined together to check their behavior and functionality after integration [istqb, 15]. Once the integration testing phase gets successfully completed, system testing is performed on the whole system [test-institute, 15]. This form of testing is carried out by a software testing engineer [test-institute, 15]. SUT can be performed at integration level by integrating various modules and then testing them. Figure 9 shows the SUT performed by integrating module 1 and 2, and table 4 shows the transition matrix for the same. Similarly module 2 and 3 & 1 and 3 are integrated and then tested.

From state	Transition stimuli	To state	Unif. Prob
Uninvoked	Invoke	{CL=forms}	1/3
Uninvoked	Invoke	{CL=edit}	1/3
Uninvoked	Invoke	{CL=reports}	1/3
CL=forms menu	↓ →	{CL=Admission forms} {CL=Edit menu}	1/ 2 1/ 2
CL=Admission form	↓ ↑ ↵	{CL=marks record } {CL= forms menu} Open admission form	1/3 1/3 1/3
CL=marks record	↓ ↑ ↵	{CL=exit } {CL= admission form} View marks record	1/3 1/3 1/3
CL=exit	↑ ↵	{CL=marks record } Exit application	1/ 2 1/ 2
CL=edit menu	↓ → ←	{CL=edit structure } {CL= reports menu} {CL=forms menu}	1/3 1/3 1/3
CL=edit structure	↓ ↑ ↵	{CL= add class} {CL=edit menu } Open edit structure form	1/3 1/3 1/3
CL=add class	↑ ↵	{CL= edit structure} Open add class form	1/ 2 1/ 2

Table 4: Transition from one state to another

2.5 SUT at System Level

In System testing the testing team is primarily concerned with testing the compatibility of the application with the system [istqb, 15]. After completing Integration testing, the entire system needs to checked as whole to uncover any further defects [test-institute, 15]. Figure 10 shows the entire system being tested together and table 5 shows the complete transition matrix.

The usage Markov chain in Figure 10 describes all the feasible input sequences for the software in a succinct model. The path from the initial "Uninvoked" state to the final "Terminating" state represents a single execution of the software [Whittaker, 94]. Test cases for the chain are such random sequences from the initial state to the terminating state. Since there are loops and cycles in the model it is possible to generate an infinite number of sequences [Whittaker, 94]. Sequences are produced from the model by stepping through state transitions and recording the sequence of inputs on the path traversed [Whittaker, 94]. Table 5 lists each transition with

probabilities assigned by uniform distributions. For example when CL (Cursor Location) is forms menu, the transition stimuli of down arrow ↓ positions the cursor on admission form with the uniform probability of ½ (since total number of states is 2 i.e. Admission forms and Edit menu). Similarly, the transition stimulus of right arrow → positions the cursor (CL) on edit menu with the uniform probability of ½.

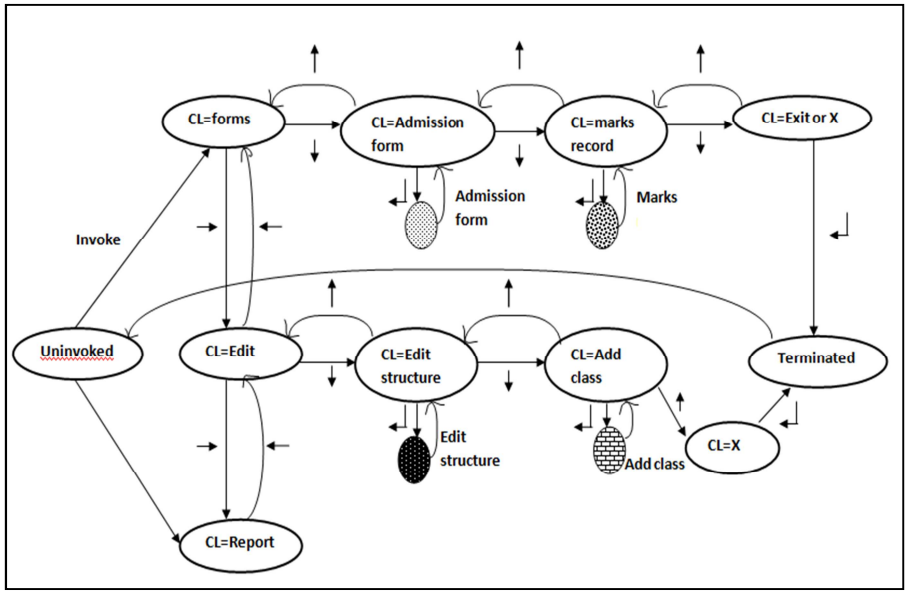

Figure 9: Integration Testing

The problem is also represented using the state chart diagram. State chart diagram is one of the UML diagrams used to present dynamic nature of a system. The diagram helps in illustrating various states an object during its lifetime where these states are altered by events [tutorials point, 15]. Figure 11 shows the state chart diagram for the problem under consideration.

State chart diagram depict the flow of control from one state to another state. States are nothing but conditions in which an object exists and the state changes whenever any event occurs [tutorials point, 15] .The process begins from the first state which is the idle state. For the problem under consideration the next states include events like open forms menu, open edit menu, and open report menu. These events are responsible for state changes of order object [tutorials point, 15] .Various events and states are shown in Figure 11.

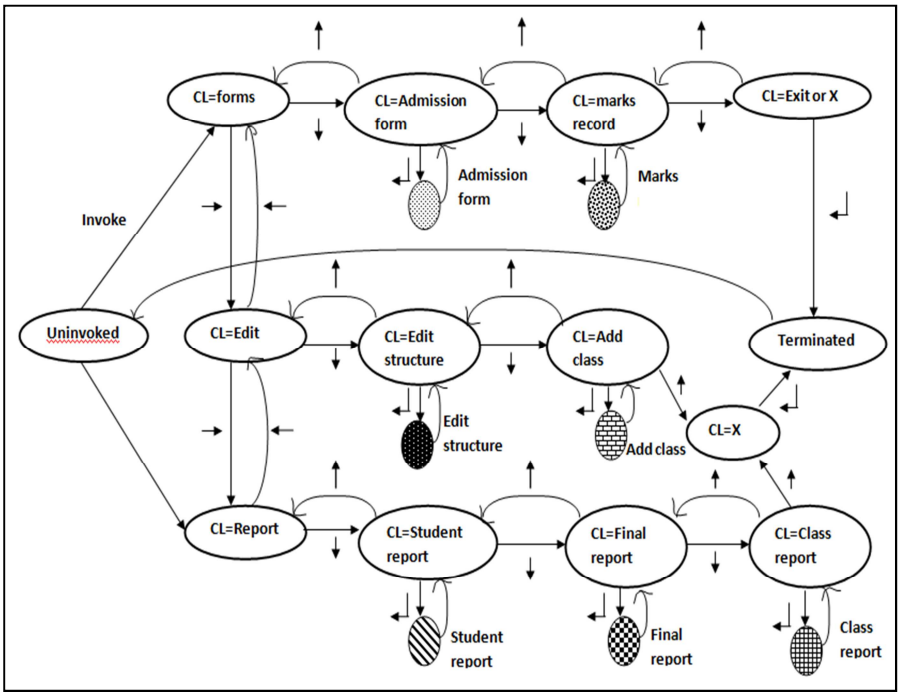

Figure 10: Usage Chain for entire Student Record Software

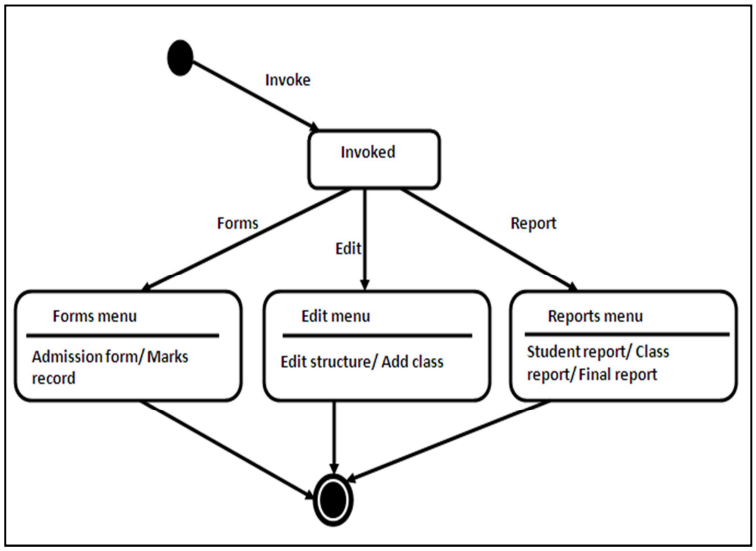

Figure 11: State chart diagram for Student Record Software

From state	Stimuli	To state	Unif Prob	Rel. prob
Uninvoked	Invoke	{CL=forms} {CL=edit} {CL=reports}	1/3 1/3 1/3	1/3 1/3 1/3
CL=forms menu	↓ →	{CL=Admission forms} {CL=Edit menu}	1/ 2 1/ 2	1/ 8 1/2
CL=Admission form	↓ ↑ ↵	{CL=marks record } {CL= forms menu} Open admission form	1/3 1/3 1/3	1/8 1/8 6/8
CL=marks record	↓ ↑ ↵	{CL=exit } {CL= admission form} View marks record	1/3 1/3 1/3	1/8 1/8 6/8
CL=exit	↑ ↵	{CL=marks record } Exit application	1/ 2 1/ 2	1/8 7/8
CL=edit menu	↓ → ←	{CL=edit structure } {CL= reports menu} {CL=forms menu}	1/3 1/3 1/3	1/8 1/2 1/2
CL=edit structure	↓ ↑ ↵	{CL= add class} {CL=edit menu } Open edit structure form	1/3 1/3 1/3	1/8 1/8 6/8
CL=add class	↑ ↵	{CL= edit structure} Open add class form	1/ 2 1/ 2	1/8 7/8
CL=reports menu	↓ ←	{CL= student report} {CL= edit menu}	1/ 2 1/ 2	1/8 1/2
CL=student report	↑ ↓ ↵	{CL=reports menu} {CL= final report } View student report	1/3 1/3 1/3	1/8 1/8 6/8
CL=final report	↑ ↓ ↵	{CL=student report} {CL= class report} View final report	1/3 1/3 1/3	1/8 1/8 6/8
CL=class report	↑ ↵	{CL= final report} View class report	1/ 2 1/ 2	1/8 7/8
Open admission form	New Open First Last Next Delete Save Previous Update Exit Cancel	Add new student record Open existing record Goto first record Goto last record Goto next record Delete record Save record View previous record Update existing record Exit application Close admission form	1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11	1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11 1/11

Open marks record form	New	Add new student record	1/11	1/11
	Open	Open existing record	1/11	1/11
	First	Goto first record	1/11	1/11
	Last	Goto last record	1/11	1/11
	Next	Goto next record	1/11	1/11
	Delete	Delete record	1/11	1/11
	Save	Save record	1/11	1/11
	Previous	View previous record	1/11	1/11
	Update	Update existing record	1/11	1/11
	Exit	Exit application	1/11	1/11
	Cancel	Close admission form	1/11	1/11
Open edit structure form	Save	save record	1/ 4	1/ 4
	Modify	Update existing record	1/ 4	1/ 4
	Exit	Exit application	1/ 4	1/ 4
	Cancel	Close admission form	1/ 4	1/ 4
Open add class form	New	Add new class	1/6	1/6
	Save	Save new class	1/6	1/6
	Delete	Delete class	1/6	1/6
	Modify	Update existing record	1/6	1/6
	Exit	Exit application	1/6	1/6
	Cancel	Close admission form	1/6	1/6
Open student report	Print	Print report	1/ 4	1/ 4
	Save	Save report	1/ 4	1/ 4
	Close	Close report	1/ 4	1/ 4
	Exit	Exit application	1/ 4	1/ 4
Open final report	Print	Print report	1/ 4	1/ 4
	Save	Save report	1/ 4	1/ 4
	Close	Close report	1/ 4	1/ 4
	Exit	Exit application	1/ 4	1/ 4
Open student report	Print	Print report	1/ 4	1/ 4
	Save	Save report	1/ 4	1/ 4
	Close	Close report	1/ 4	1/ 4
	Exit	Exit application	1/ 4	1/ 4

Table 5: Transition probabilities for usage model

The sequence below shows the updation of probabilities when some actions are performed

Uninvoked
Invoke transition change:
{CL=forms} from 0 to 1
{CL=edit} from 0 to 1
{CL=reports} from 0 to 1
CL=forms menu transition change:
Down arrow key {CL=Admission forms} from 0 to 1

CL=Admission form transition change:
Down arrow key {CL=marks record} from 0 to 1
Enter View marks record

View marks record transition change:
{CL=Close} from 0 to 1

CL=forms menu transition change:
Down arrow key {CL=Admission forms} from 1 to 2

CL=Admission form transition change:
Down arrow key {CL=marks record} from 1 to 2

CL=marks record transition change:
Down arrow key {CL=exit} from 0 to 1

CL=exit transition change:
Up arrow key CL=marks record from 0 to 1

CL=marks record transition change:
Down arrow key {CL=exit} from 1 to 2

Terminated transition change:
Exit application from 0 to 1

The sequence below shows the same execution of sequence when a failure state is encountered while viewing the records of the students [Whittaker, 94].

Uninvoked
Invoke transition change:
{CL=forms} from 0 to 1
{CL=edit} from 0 to 1
{CL=reports} from 0 to 1
CL=forms menu transition change:
Down arrow key {CL=Admission forms} from 0 to 1

CL=Admission form transition change:
Down arrow key {CL=marks record} from 0 to 1
Enter View marks record

View marks record add failure state
Transition change (View marks record)
From 0 to 1

Failure state i transition change: (Failure State, Terminated) from 0 to 1

CL=forms menu transition change:
Down arrow key {CL=Admission forms} from 1 to 2

CL=Admission form transition change:
Down arrow key {CL=marks record} from 1 to 2

CL=marks record transition change:
Down arrow key {CL=exit} from 0 to 1

CL=exit transition change:
Up arrow key CL=marks record from 0 to 1

CL=marks record transition change:
Down arrow key {CL=exit} from 1 to 2
Terminated transition change:
Exit application from 0 to 1

2.6 SUT at Acceptance Level Testing

Acceptance testing is principally carried out to make sure that the requirements of the specification are met [istqb, 15]. Once the entire system is tested using SUT at all the previous levels (unit, integration and system testing), user acceptance testing can be performed. If user has knowledge of SUT, only then it can be used for user acceptance testing .Otherwise other techniques can be used for user acceptance testing.

3 Findings & Conclusion

In this paper SUT is used at different Levels. This improvement demonstrated the process and benefits of applying SUT at different levels of testing. SUT is generally performed at System level and Unit testing is not the part of SUT. The inclusion of Unit testing can improve this aspect of Cleanroom Software Engineering thus making it easier to access code and debug human errors. Detecting errors at an early stage helps reducing cost and effort. The study proposed to allow Unit testing in CSE, thus making it more flexible and suitable for varied applications. Thus, allowing Unit testing would improve the software quality. SUT can also be combined with other White box and Black box testing techniques at Unit level for code scrutiny and checking the external interface.

After carrying out SUT at Unit level, SUT is performed at the Integration level. Errors may also exist when modules are integrated because of interchange of data and control information between various modules. Integration testing is performed when the modules are combined together to check their behavior and functionality after integration.

After completing the Integration testing phase successfully, System testing is performed on the whole system. When SUT was applied at Unit level, errors were detected at an early stage. Errors were also uncovered at Integration stage. But very few errors were detected at the System level, as errors were already detected in the previous levels of testing. Detecting errors at an early stage prevents them from becoming very grave, thus saving effort, budget and time. Finally, in System testing

the system is tested in its entirety to see if it is in conformity with the functions, quality standards and requirements of an organization. The system is tested as whole and it is checked for compliance with requirement specifications.

The findings of the proposed approach are enumerated below:

- Statistical usage testing can be used to test smallest testable modules at Unit level. When individual modules were tested using SUT errors were uncovered (figure 8). Detecting errors at an early stage helps reducing cost and effort. SUT can also be combined with other white box and black box testing techniques at unit level for code scrutiny and checking the external interface [Khatri, 15].
- More errors were uncovered at integration stage when module 1 and 2 was integrated and module 2 and 3 was integrated and tested.
- Only 2 errors were found during System testing as errors were already uncovered during Unit testing and Integration testing.
- Once the entire system is tested using SUT, User Acceptance testing can be performed. If user has knowledge of SUT, only then it can be used for User Acceptance testing .Otherwise other techniques can be used for User Acceptance testing.

SUT can efficiently be employed at unit level and integration level to uncover more errors which help in reducing time, cost and testing effort.

4 Limitations & Future Work

The paper has used one software for testing. For more general results, the proposed approach can be applied to various other software's also.

Acknowledgment

Authors express their deep sense of gratitude to the founder president of Amity University Dr Ashok K. Chauhan for his keen interest in promoting research in Amity University and have always been an inspiration for achieving great heights.

References

[Kaur, 12] Kaur K., Khatri S. K., Datta R.: Analysis of Statistical Usage Testing Technique with Markov Chain Model, Proceedings 2nd International Conference on Reliability, Infocom Technologies and Optimization, December 2012

[Kaur 14] Kaur K., Khatri S. K., Datta R.: Analysis of Various Testing Techniques**,** published at International Journal of System Assurance Engineering and Management, Springer, Volume 5, Issue 3, 276–290, September 2014.

[Khatri, 15] Khatri S. K., Kaur K., Datta R.: Using Statistical Usage Testing in Conjunction with other black Box Testing Techniques, International Journal of Reliability, Quality and Safety Engineering, World Scientific Publishing Company Vol . 22, Number 1, World Scientific Publishing Company, DOI: 10.1142/S0218539315500047, 2015.

[Khatri 14] Khatri S. K., Kaur K., Datta R.: Testing Apache Open office Writer Using Statistical Usage Testing Technique**,** published in International Journal of System Assurance Engineering and Management, Springer, Vol. 6, Number 1, 3-17, DOI 10.1007/s13198-014-0237-2, Feb 2014.

[Linger 96] Linger R. C., Trammell C. J.: Cleanroom Software Engineering Reference Model", http://www.sei.cmu.edu/reports/96tr022.pdf, November 1996.

[Pressmen, 00] Pressmen R. S.: A Practitioner's approach to software engineering, TMH, 5th ed., ISBN-13: 978-0073655789, 2000.

[Prowell, 99] Prowell S. J., Trammell C. J., Linger R. C., Poore J. H.: Clean room Software Engineering Technology and Process, Addison-Wesley, An imprint of Addison Wesley Longman , Inc, 1999

[Runeson, 93] Runeson P., Wohlin C.: Statistical Usage Testing for Software Reliability Certification and Control, Proceedings 1st European International Conference on Software Testing, Analysis and Review, 309-323, London, UK, 1993.

[Runeson 95] Runeson P., Wohlin C.: Statistical Usage Testing for Software Reliability Control, Informatica, Vol. 19, Number 2, 195-207, 1995

[Trammel 95] Trammel C.: Quantifying the reliability of software: statistical usage testing based on usage models, Software Engineering Standards Symposium, 1995. (ISESS'95) 'Experience and Practice', Proceedings Second IEEE International, DOI: 10.1109/SESS.1995.525966, 208 – 218, 1995.

[Whittaker, 92] Whittaker J. A., Poore J. H.: Statistical Testing for Cleanroom Software Engineering, Proceedings 25th Annual Hawaii International Conference on System Sciences, pp. 428-436, Hawaii, USA, 1992

[Whittaker, 94] Whittaker J. A., Michael G. T.: "A Markov Chain Model for Statistical Software Testing", IEEE Transactions on Software Engineering. Vol. 20, No. 10, October 1994

[istqb, 15] http://istqbexamcertification.com/what-are-software-testing-levels/, 2015

[leansoftware, 15] http://leansoftwareengineering.com/2009/02/04/leanroom/, 2015

[usersnap, 15] http://usersnap.com/blog/types-user-acceptance-tests-frameworks/, 2015

[test-institute, 15] http://www.test-institute.org/Software_Testing_Levels.php, 2015

[tutorials point, 15] http://www.tutorialspoint.com/uml/uml_statechart_diagram.htm, 2015