IMPORTANT QUESTIONS **NV-AST**

1. **What is LCSAJ Testing ?**

LCSAJ stands for Linear Code Sequence and Jump, a white box testing technique to identify the code coverage, which begins at the start of the program or branch and ends at the end of the program or the branch.

LCSAJ consists of testing and is equivalent to statement coverage.

LCSAJ Characteristics:

- 100% LCSAJ means 100% Statement Coverage
- 100% LCSAJ means 100% Branch Coverage
- 100% procedure or Function call Coverage
- 100% Multiple condition Coverage
- Linear code sequence and jump, in the broad sense, is a software analysis method used to identify structural units in code under test. Its primary use is with dynamic software analysis to help answer the question "How much testing is enough?".
- Dynamic software analysis is used to measure the quality and efficacy of software test data, where the quantification is performed in terms of structural units of the code under test. When used to quantify the structural units exercised by a given set of test data, dynamic analysis is also referred to as structural coverage analysis.
-
- In a narrower sense, an LCSAJ is a well-defined linear region of a program's code. When used in this sense, LCSAJ is also called **JJ-path**, standing for jump-to-jump path.
-
- The formal definition of a LCSAJ can be given in terms of basic blocks as follows:[8]
- a sequence of one or more consecutively numbered basic blocks, *p*, (*p*+1), ..., *q*, of a code unit, followed by a control flow jump either out of the code [unit] or to a basic block numbered *r*, where *r*≠(*q*+1), and either *p*=1 or there exists a control flow jump to block *p* from some other block in the unit. (A basic block to which such a control flow jump can be made is referred to as a target of the [LCSAJ] jump.)

- have been mandated for many other projects, including aerospace, telephony and banking.[citation needed] One practical problem of using TER3 is that many LCSAJs can never be executed due to the conflicting conditions they contain.

- ## Example

- Consider the following C code:

```
1 #include<stdlib.h>
2 #include<string.h>
3 #include<math.h>
4
5 #define MAXCOLUMNS  26
```

```
 6 #define MAXROW      20
 7 #define MAXCOUNT    90
 8 #define ITERATIONS  750
 9
10 intmain(void)
11 {
12 intcount=0,totals[MAXCOLUMNS],val=0;
13
14 memset(totals,0,MAXCOLUMNS*sizeof(int));
15
16 count=0;
17 while(count<ITERATIONS)
18 {
19 val=abs(rand())%MAXCOLUMNS;
20 totals[val]+=1;
21 if(totals[val]>MAXCOUNT)
22 {
23 totals[val]=MAXCOUNT;
24 }
25 count++;
26 }
27
28 return(0);
29
30 }
```

- From this code, the following is a complete list of the LCSAJ triples for this code

LCSAJ Number	Start Line	Finish Line	Jump To Line
1	10	17	28
2	10	21	25
3	10	26	17
4	17	17	28
5	17	21	25

6	17	26	17
7	25	26	17
8	28	28	−1

- From this example it can be seen that the basic block identified by an LCSAJ triple may span a decision point, reflecting the conditions that must be in place in order for the LCSAJ to be executed. For instance, LCSAJ 2 for the above example includes the `while` statement where the condition `(count < ITERATIONS)` evaluates to true.
- Each line of code has an LCSAJ 'density' associated with it; line 17, for instance, appears within 6 unique LCSAJs - i.e. it has an LCSAJ density of 6. This is helpful when evaluating the maintainability of the code; If a line of code is to be changed then the density is indicative of how many LCSAJs will be affected by that change.
- A coverage level of TER3 = 100% would be achieved when the test data used causes the execution of each of these LCSAJs at least once.

- **What is Orthogonal Array Testing?**
- ORTHOGONAL ARRAY TESTING (OAT) is a testing technique that uses Orthogonal Arrays to create test cases. It especially useful when the system to be tested has huge data inputs.
- For example, when a train ticket has to be verified, factors such as - the number of passengers, ticket number, seat numbers, and train numbers have to be tested. One by one testing of each factor/input is cumbersome. It is more efficient when the QA engineer combines more inputs together and does testing. In such cases, we can use the Orthogonal Array testing method.
- This type of pairing or combining of inputs and testing the system to save time is called Pairwise testing. OATS technique is used for pairwise testing.

In the present scenario, delivering a quality software product to the customer has become challenging due to the complexity of the code.

In the conventional method, test suites include test cases that have been derived from all combination of input values and pre-conditions. As a result, n number of test cases has to be covered.

But in a real scenario, the testers won't have the leisure to execute all the test cases to uncover the defects as there are other processes such as documentation, suggestions, and feedback from the customer that has to be taken into account while in the testing phase.

Hence, the test managers wanted to optimize the number and quality of the test cases to ensure maximum Test coverage with minimum effort. This effort is called Test Case Optimization.

- Systematic and Statistical way to test pairwise interactions
- Interactions and Integration points are a major source of defects.
- Execute a well-defined, concise of test cases that are likely to uncover most (not all) bugs.
- Orthogonal approach guarantees the pairwise coverage of all variables.

Example 2:

A microprocessor's functionality has to be tested:

1. Temperature: 100C, 150C and 200C.
2. Pressure : 2 psi,5psi and 8psi
3. Doping Amount :4%,6% and 8%
4. Deposition Rate : 0.1mg/s , 0.2 mg/s and 0.3mg/s

By using the Conventional method we need = 81 test cases to cover all the inputs. Let's work with the OATS method:

No. of factors = 4 (temperature, pressure, doping amount and Deposition rate)

Levels = 3 levels per factor (temperature has 3 levels-100C, 150C, and 200C and likewise other factors too have levels)

Create an array as below:

1. Columns with the No. of factors

- **2. Enter the number of rows is equal to levels per factor. i.e temperature has 3 levels. Hence, insert 3 rows for each level for temperature,**

Test case #	Temperature	Pressure	Doping amount	Deposition rate
1	100C			
2	100C			
3	100C			

4	150C
5	150C
6	150C
7	200C
8	200C
9	200C

- **3. Now split up the pressure, doping amount and the deposition rates in the columns.**
- For eg: Enter 2 psi across temperatures 100C,150C and 200C likewise enter doping amount 4% for 100C,150C and 200C and so on.

Test case #	Temperature	Pressure	Doping amount	Deposition rate
1	100C	2 psi	4%	0.1 mg/s
2	100C	5 psi	6%	0.2 mg/s
3	100C	8 psi	8%	0.3 mg/s
4	150C	2 psi	4%	0.1 mg/s
5	150C	5 psi	6%	0.2 mg/s
6	150C	8 psi	8%	0.3 mg/s

7	200C	2 psi	4%	0.1 mg/s
8	200C	5 psi	6%	0.2 mg/s
9	200C	8 psi	8%	0.3 mg/s

- Hence, in OAs, we need 9 Test cases to cover.
- **OAT Advantages**
- Guarantees testing of the pair-wise combinations of all the selected variables.
- Reduces the number of test cases
- Creates fewer Test cases which cover the testing of all the combination of all variables.
- A complex combination of the variables can be done.
- Is simpler to generate and less error-prone than test sets created by hand.
- It is useful for Integration Testing.
- It improves productivity due to reduced test cycles and testing times.

What is FIA in software estimation technique?

3.3.4 Estimation Techniques

The following estimation techniques are the most used and an expression of the best practice within estimation.

- FIA (finger in the air) or best guess
- Experience-based estimation
- Analogies and experts
- Delphi technique
- Three-point estimation (successive calculation)
- Model-based estimation
- Function points
- Test points
- Percentage distribution

3.3.4.1 Estimation; Best Guess (FIA)

This technique is more or less pure guesswork, but it will always be based on some sort of experience and a number of (unconscious) assumptions. The technique is very widespread, but since it is based on your gut feeling it is bound to be inaccurate. It is often not repeatable, and it is not always trusted.

The uncertainty contingency is probably around 200%–400% for estimates based on best guess. We can do better than that.

What is a Risk in software testing ?

A **software risk** can be of two **types** (a) internal **risks** that are within the control of the project manager and (2) external **risks** that are beyond the control of project manager. **Risk** management is carried out to: Identify the **risk**. Reduce the impact of **risk**. Reduce the probability or likelihood of **risk**.

"**Risk** is future uncertain events with a probability of occurrence and potential for loss" **Risk** identification and management are the main concerns in every **software** project. Effective analysis of **software risks** will help to effective planning and assignments of work.

Types of Risk :

a. Project Risk b. Product Risk c. Process Risk d. Business Risk

The project risk that can endanger the project are:

- Risk such as the late delivery of the test items to the test team or availability issues with the test environment.
- There are also indirect risks such as excessive delays in repairing defects found in testing or problems with getting professional system administration support for the test environment.

For any risk, project risk or product risk we have four typical actions that we can take:

- **Mitigate:** Take steps in advance to reduce the possibility and impact of the risk.
- **Contingency:** Have a plan in place to reduce the possibility of the risk to become an outcome.
- **Transfer:** Convince some other member of the team or project stakeholder to reduce the probability or accept the impact of the risk.
- **Ignore:** Ignore the risk, which is usually a good option only when there is little that can be done or when the possibility and impact of that risk are low in the project.
-
- **Product risk** is the possibility that the system or **software** might fail to satisfy or fulfill some reasonable expectation of the customer, user, or stakeholder. (Some authors also called the '**Product risks**' as 'Quality **risks**' as they are **risks** to the quality of the **product**.)

What is classification of tree method ?

The **Classification Tree Method** is a method for test design,[1] as it is used in different areas of software development.[2] It was developed by Grimm and Grochtmann in 1993.[3] Classification Trees in terms of the Classification Tree Method must not be confused with decision trees.

The classification tree method consists of two major steps:[4][5]

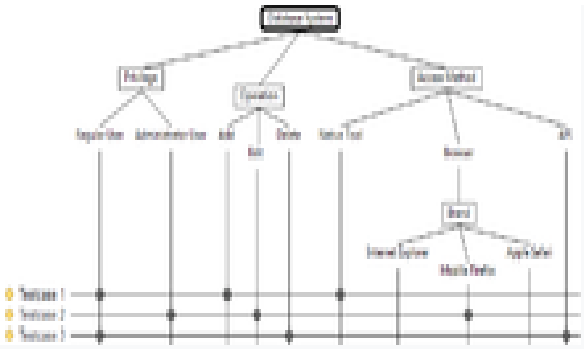
1. Identification of test relevant aspects (so called *classifications*) and their corresponding values (called *classes*) as well as
2. Combination of different classes from all classifications into test cases.

The identification of test relevant aspects usually follows the (functional) specification (e.g. requirements, use cases …) of the system under test. These aspects form the input and output data space of the test object.

The second step of test design then follows the principles of combinatorial test design.[4]

While the method can be applied using a pen and a paper, the usual way involves the usage of the Classification Tree Editor, a software tool implementing the classification tree method.

Example[edit]



Classification Tree for a Database System

For a database system, test design has to be performed. Applying the classification tree method, the identification of test relevant aspects gives the classifications: *User Privilege*, *Operation* and *Access Method*. For the *User Privileges*, two classes can be identified: *Regular User* and *Administrator User*. There are three *Operations*: *Add*, *Edit* and *Delete*. For the *Access Method*, again three classes are identified: *Native Tool*, *Web Browser*, *API*. The *Web Browser* class is further refined with the test aspect *Brand*, three possible classes are included here: *Internet Explorer*, *Mozilla Firefox*, and *Apple Safari*.

The first step of the classification tree method now is complete. Of course, there are further possible test aspects to include, e.g. access speed of the connection, number of database records present in the database, etc. Using the graphical representation in terms of a tree, the selected aspects and their corresponding values can quickly be reviewed.

For the statistics, there are 30 possible test cases in total (2 privileges * 3 operations * 5 access methods). For minimum coverage, 5 test cases are sufficient, as there are 5 access methods (and access method is the classification with the highest number of disjoint classes).

In the second step, three test cases have been manually selected:

1. A regular user adds a new data set to the database using the native tool.
2. An administrator user edits an existing data set using the Firefox browser.
3. A regular user deletes a data set from the database using the API.

234567