chapter 3.

* Regression Testing

* What is Regression Testing

- When we modify software, we typically re test it. This process of Retesting is called regression testing.
- Regression testing is the process of retesting the modified parts of the software and ensuring that no new errors have been introduced into previously tested source code due to these modification.
- It serves several purposes like :
  - Increases confidence in the correctness of the modified program.
  - Locates errors in the modified program
  - preserves the quality & reliability of the s/w
  - Ensures the software's continued operation

1) Regression Testing process

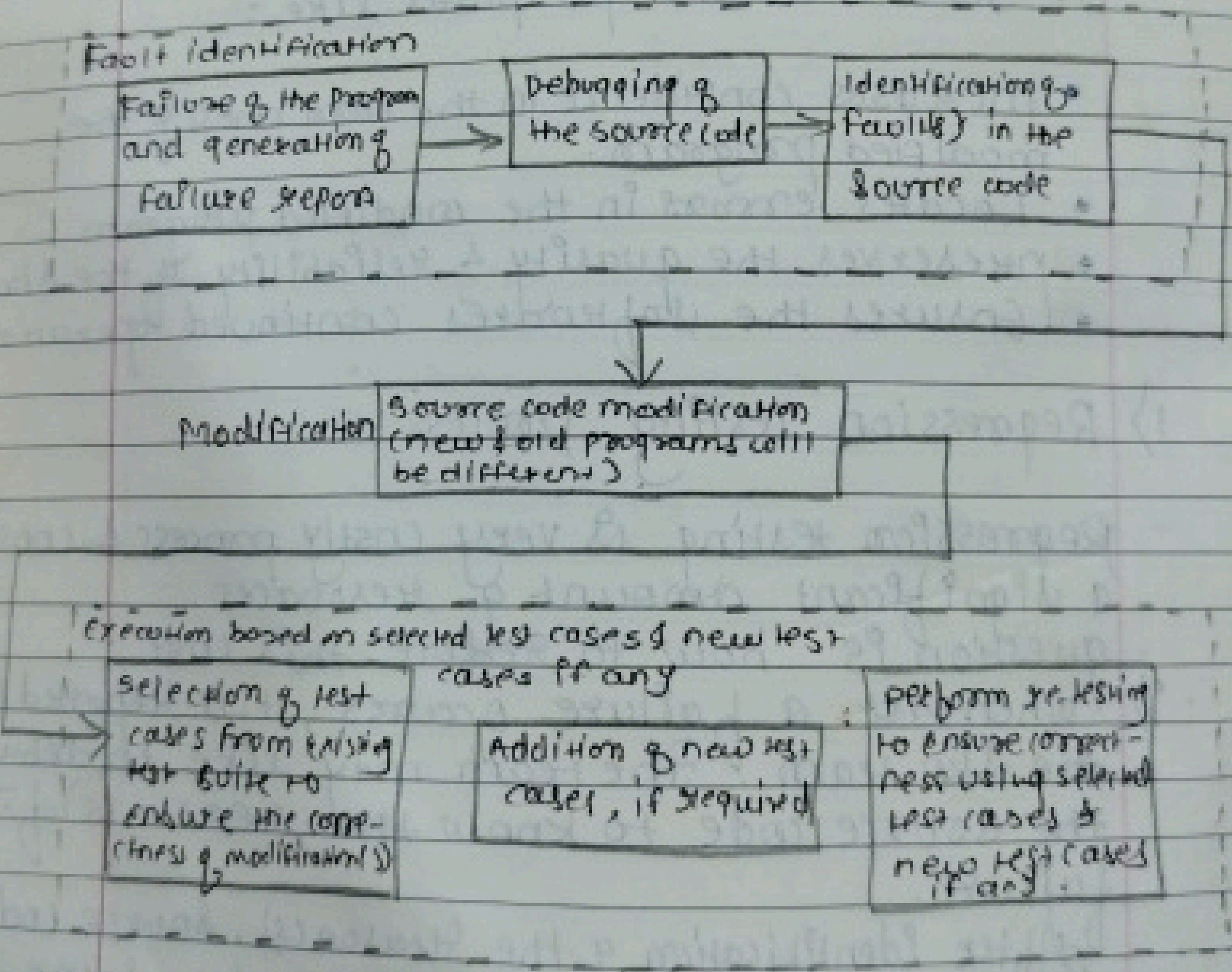
- Regression testing is very costly process & consumes a significant amount of resources.
- question is "how to reduce this cost ?

i) whenever a failure occures, it is reported to the s/w team. The team may like to debug the source code to know the reason(s) of failure.

ii) After identification of the reason(s), source code is modified & do not Expect the same failure again

In order to ensure this correctness, we re-test source code with a focus on modified option of the source code & also on affected portion we need test cases that target the modified & affected portions of the source code.

ii) Another option is to use the existing test cases which were designed for development testing & some of them might have been used during development testing.



Fault identification
Failure of the program and generation of failure report
Debugging of the source code
Identification of fault(s) in the source code
Modification
Source code modification (new & old programs will be different)
Execution based on selected test cases & new test cases if any
Selection of test cases from existing test suite to ensure the correctness of modification(s)
Addition of new test cases, if required
Perform re-testing to ensure correctness using selected test cases & new test cases if any.

Fig: Steps of regression testing process.

2) Selection of test cases.

We want to use the existing test suite for regression testing. How should we select an appropriate number of test cases for a failure?
The range is from "one test case" to "all test cases".
A 'Regression test cases' selection technique may help us to do this selection process.
The effectiveness of the selection technique may decide the selection of the most appropriate test cases from the test suite.

```
1. main()                     1.
2. {                          2.
3. int a, b, x, y, z;         3
4. scanf("%d, %d", &a, &b);   4
5. x=a+b;                     5
6. y=a*b;                     6. y=a-b;
7. if (x>=y) {                7.
8. z=x/y;                     8
9. }                          9
10 else {                     10
11. z=x*y;                    11
12. }                         12
13. printf("z=%d \n", z);     13
14. }                         14
```

(a) original pgm with fault in line 6

(b) modified program with modification in line 6.

Fig: prog. with printing value of z.

set of test cases S.No.	inputs a	b	Execution History
1	2	1	1 to 9, 13, 14
2	1	1	1 to 9, 13, 14
3	3	2	1 to 7, 10 to 14
4	3	3	1 to 7, 10 to 14

table : test suite for program given.

* Regression Test cases Selection

Many techniques are available for the selection of test cases for the purpose of regression testing.

1) Select All Test Cases.

- This is the simplest tech. where we do not want to take any risk, we want to run all test cases for any change in the program.
- A prog. may fail many times & every time we will execute the entire test suite.
- Practical only when the size of test suite is small.

2) Select Test cases Randomly
we may select test cases randomly to reduce the size of the test suite.

- we decide how many test cases are required to be selected depending upon time & available resources.
- when we decide the number, the same number of test cases is selected randomly
- if the no. is large, we may get good no. of test cases for execution & testing may be of some use
- if the no. is small, testing may not be useful at all.
- in this technique, our assumption is that all test cases are equally good in their fault detection ability.

3) Select Modification Traversing Test Cases.

- We select only those test cases that execute the modified portion of the program & the portion which is affected by the modification(s) other test cases of the test suite are discarded
- Actually, we want to select all those test cases that reveal faults in the modified program. These test cases are known as fault revealing test cases.
- It is impractical to apply these techniques to large commercial systems unless a tool is available that incorporates at least one safe test minimization tech.

* Reducing the number of test cases.

1) Minimization of Test cases

we select all those test cases that traverse the modified portion of the program & the portion that is affected by the modification. If we ~~we~~ find the selected number very large we may still reduce this using any test case minimization techniques. These test case minimization techniques attempt to find redundant test cases.

2) prioritization of Test cases

- We may indicate the order with which a test case may be addressed. This process is known as prioritization of test cases.
- A test case with the highest rank has the highest priority and the test case with the second highest rank has the second highest priority and so on.
- A prioritization does not discard any test case.
- There are two varieties of test case prioritization i.e. i) general test case prioritization
  ii) version specific test case prioritization
- prioritization guidelines should address two fundamental issues like :

i) what functions of the software must be tested?
ii) What are the consequentes if some functions are not tested?

Every reduction activity has an associated risk. All prioritization guidlines should be designed on the basis of risk analysis.
The simplest priority category scheme is to assign a priority code to every test case. The priority code may be based on the assumption that "test case of priority code 1 is more important than test case of priority code 2". we may have priority codes as follows:

priority code 1	:	essential test case
priority code 2	:	important test case
— 11 — 3	:	execute. if time permits
— 11 — 4	:	Not important test case
— 11 — 5	:	Redundant test case.

There may be other ways for assigning priorities based on customer requirements or market conditions like :-

priority code 1	:	important for the customer
— 11 — 2	:	Required to increase cust. satisfact
— 11 — 3	:	Help to increase market shar of the product.

Lucky Page No.
Date / /

* Risk Analysis :

1) What is RISK ?

Tomorrow's problems are today's risks. Therefore, a simple defⁿ of risk is a problem that may cause some loss or threaten the Success of the project, but which has not happed yet.
Risk is defined as the "probability of occurrence of an undesirable Event and the impact of occurrence of that Event".
Risky projects may also not meet specified quality levels. Hence, there are two things associated with risk as given below:

i) Probability of occurrence of a problem
ii) Impact of that problem

Risk analysis is a process of identifying the potential problems & then designing a probability of occurance of the problem value & impact of that problem value for each identified problem.
The risks may be ranked on the basis of its risk exposure.
A risk analysis table may be prepared as given table.

Risk Analysis table.

SR NO	potential problem	probability of occurrence of pro	impact of that problem	Risk Expo
1				
2				
3				

2) Risk Matrix.

Risk matrix is used to capture Identified problems, Estimate their probability of occurre-nce with impact and rank the risks based on this information. we may use the risk matrix to assign thresholds that group the potential problems into priority categories.
The risk matrix is shown in following fig. with 4 quadrants.

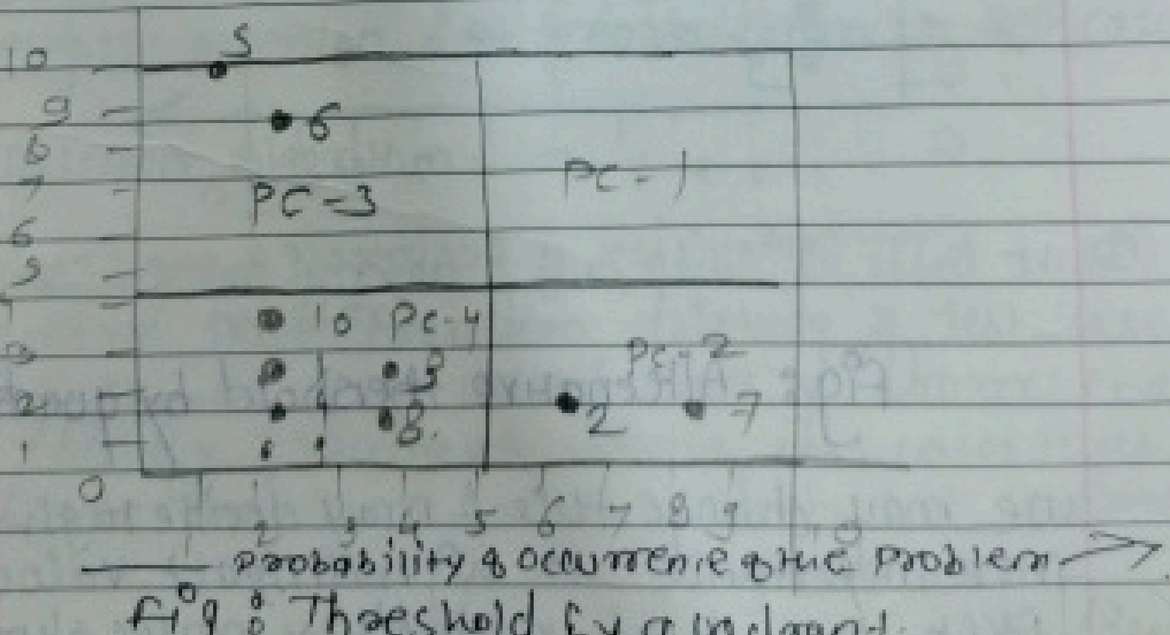


fig : Threshold by quadrant

Lucky Page No.:
Date: / /

The priority category in defined as:

priority category 1 (PC-1) = High prob. value & high impact val.
—11— 2 (PC-2) = High prob. value & low impact val.
—11— 3 (PC-3) = Low prob. val. & high impact val.
—11— 4 (PC-4) = Low prob. val. & low impact val.

In this case, a risk with high probability value is given more importance than a problem with high impact value.

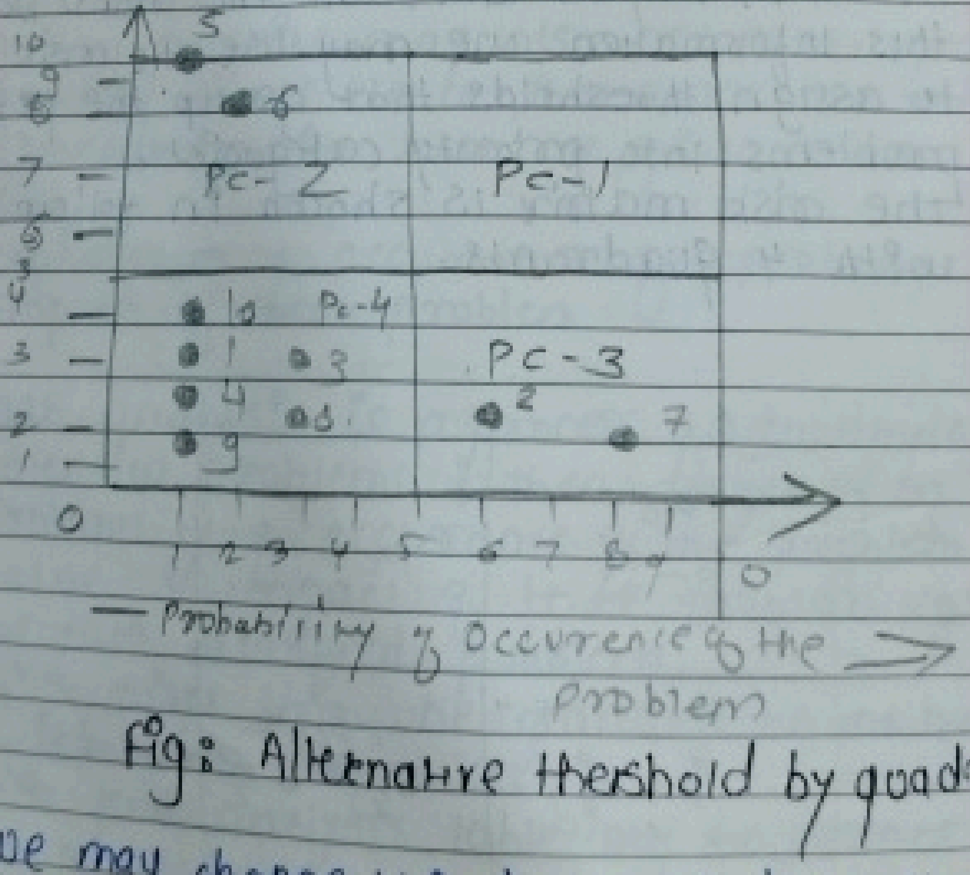


Fig: Alternative threshold by quadrant

We may change this & may decide to give more importance to high impact value over the high probability value shown in above figure.

28/12/20

* Code coverage prioritization Technique

i) Test cases Selection criteria
This technique identifies those test cases that:

- Execute the modified lines of source code at least once
- Execute the lines of source code after deletion of deleted lines from the execution history of the test case & are not redundant.

The techniques uses two algorithms - One for 'modification' & the other for 'deletion'.

ii) Modification Algorithm.

- Used to minimize and prioritize test cases based on the modified lines of source code.

Step I : Initialization of variables.
Step II : selection and prioritization of test cases

iii) Deletion Algorithm

- The deletion portion of the tech is used to
i) update the execution history of test cases by removing the deleted lines of source code
ii) Identify & remove those test cases that cover only those lines which are covered by other test cases of the program.

Step I: Initialization of variables
Step II: Identification of redundant test cases.

[illegible]