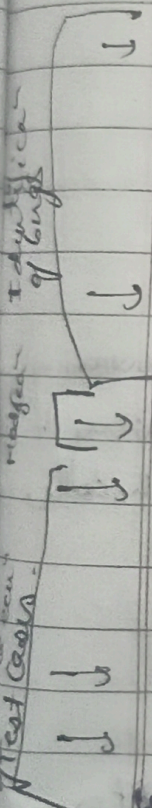
Unit 4

① Regression Testing

→ It is the process of testing the modified parts of the code & the parts that might get affected due to the modifica^n's to ensure that no new errors have been introduced in the s/w after the modifica^n's have been made.

→ Regression means return of sth., & here it means return of bug.

② Process of Regression Testing



→ Firstly, when we make some changes to the source code like adding new f^ality, optimiza^n etc, then our prog. fails for the previously designed test suit for obvious reasons.

→ After the failure, the source code is debugged to identify the bugs in the prog.

→ After that, appropriate modifica^n's are made.

→ Then appropriate test cases are selected from the already existing test suite which covers all the modified & affected parts of the source code.

→ We can add new test cases if reqd.

→ In the end regression testing is performed using the selected test cases.

③ Selec^n of test cases for regression testing

→ Select all test cases :- Simple & safe but not efficient

→ Select test cases randomly :- useful only if all test cases are equally good in their fault detec^n capability which is rare

→ Select modifica^n traversing test cases :- Only those test cases which cover & test the modified por^n's of the code.

→ Select higher priority test cases :- Priority codes are assigned to each test case based upon their bug detect^n capability, customer requirements, etc.
•) After assigning the priority codes, test cases with highest priorities are selected for the process of regression testing.
•) Priority code 2 has less priority that priority code 1.

(4) Tools for Regression Testing
→ Selenium
→ WATIR (Web App^n Testing in Ruby)
→ QTP (Quick Test Professional)
→ RFT (Rational F^n tester)
→ Winrunner
→ SilkTest.

Adv.	Disadv.
→ Ensures no new bug is introduced after adding new fabilities.	→ Time & Resource consuming
→ Can be easily automated.	→ Reqd. even after very small changes in the code.
→ maintains quality of source code.	

(5) Adhoc - Testing :-
→ Performed informally & randomly after the formal testing is completed to find out any loophole in the system.
→ Also kla Random / Monkey Testing.
→ Unstructured S/w Testing.
→ It has → No documentat^n
→ No test cases
→ No test design

Types of Adhoc Testing

→ Buddy Testing :- 2 bodies are involved one is from developer team & one from tester team.

→ Pair Testing :- 2 bodies from the tester team can be involved to test the same module.

→ Monkey Testing :- System is tested based on random i/ps without any test cases.

→ One tester can perform the random test & another can maintain the record of findings. So both of them pair up to exchange their ideas, opinions & knowledge, so good testing is performed on the module.

When to conduct Adhoc

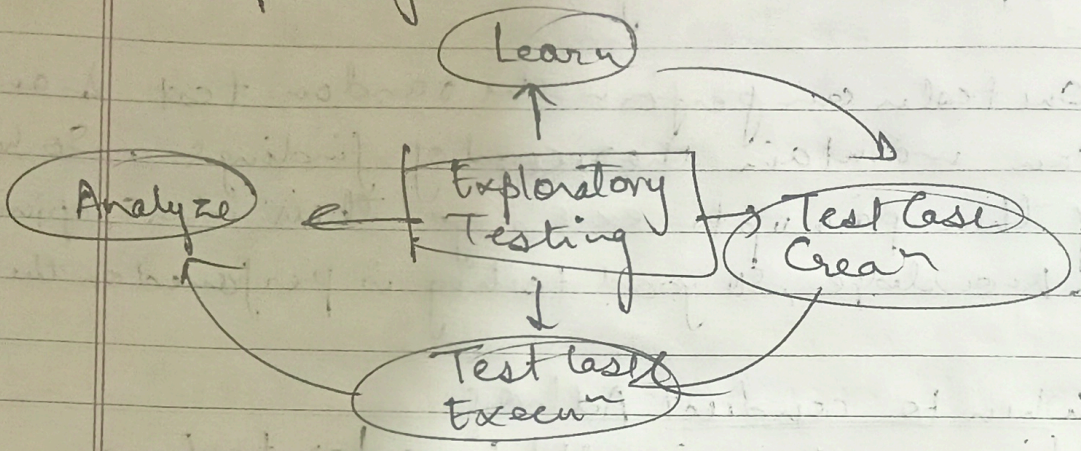
- when there is ltd time for testing.
- when there is no clear test cases.
- when formal testing is complete
- when development is mostly complete

Advs.	Dis advs.
→ Can be performed within ltd. time.	→ Resolving error based on identified issues is difficult as there is no documentation.
→ Helps to create unique test cases	→ Needs proper knowledge of prod. & testing.
→ Build strong prod. which is less prone towards future prods	→ No assurance that error will be definitely identified
→ Can be performed any time during SDLC.	→ Finding one error may take uncertain period of time.
→ Errors found by written test cases can also be found by Adhoc testing	

Date
Page

(6) Exploratory Testing

→ In this, the tester is free to select any possible methodology to test the s/w.
→ It is unscripted testing technique.
→ In this, s/w developers use their personal learning knowledge, skills & abilities to test the s/w developed by themselves.



Adv.

→ Requires no preparation.
→ finds critical defects quickly
→ testers use their knowledge, skills & experiences to test s/w

Disadv

→ once testing is done, its not reviewed
→ Keeping track of tests performed is difficult.
→ Not possible to repeat same test methodology.

(7) Iterative Testing

→ It is a process of updating the app^n with small changes & testing them to allow the app^n evolve over time with smaller changes rather than larger drastic changes.
→ In this, a product is tested on users repeatedly & by making use of the results after each test, the prod. is improved at diff. stages.
→ The main motive is to make the product foolproof & market ready before its launch.

(8) **Defect Seeding**

→ It is a practice in which defects are intentionally inserted into a prog by one group for detection by another group.

→ The ratio of the no. of seeded defects detected to the total no. of defects seeded provides a rough idea of the total no. of unseeded defects that have been detected.

(9)

Smoke Testing	Sanity Testing
→ Broad approach where all parts of the appⁿ are tested.	→ Narrow approach where specific parts are tested.
→ measures stability of system	→ measures rationality of system
→ Performed by both testers & developers.	→ Performed by only testers.
→ Tester has to go into depth of the appⁿ.	→ Doesn't require so.
→ Its the 1st testing performed on initial build.	→ Performed when build is comparatively stable.
→ Documented	→ Not documented like adhoc.
→ used to test end to end fⁿ of appⁿ.	→ used to test only modified or defect fixed fⁿs.
→ Subset of acceptance testing	→ Subset of regression testing

(10) **Slicing :-**

→ Its a technique which takes a slice/grp of prog. statements in the prog. for testing particular test condiⁿs/cases that may affect a value at a particula pt. of interest.

→ It can also be used for the purpose of debugging in order to find the bugs.

Types → Static
→ Dynamic.

Date
Page

Dynamic Slicing :-

→ A dynamic slice of a prog. contains all the statements that actually affect the value of a var. at any pt. for a particular execn of the prog.

→ Dynamic slices are generally smaller.

→ Considers only a particular execn of the prog.

Static

```
int sum;
if(sum>10)
  sum++;
else
  sum--;
```

Its for all possible test cases

Dynamic

```
int sum
if (sum>10)
  sum++;
```

Its for particular test cases like sum = 22.

Execution Trace :-

→ To analyze the failed test cases, the test case execn trace is captured.

→ It represents the order of the statements that were touched during the test case execn.

→ The execn logs present the o/p each test case is passed or failed.