DEPARTMENT OF CSE , CSIT & AI&DS

COURSE NAME – ADAPTIVE SOFTWARE ENGINEERING

COURSE CODE – 23CI2001

SESSION-21
TOPIC: TEST STRATEGIES FOR CONVENTIONAL SOFTWARE

AIM OF THE SESSION

To familiarize students with the basic concept of Project planning and estimation

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

This Session is designed to:

1. Unit Testing
2 Integration Testing
3. Regression testing

LEARNING OUTCOMES

At the end of this session, you should be able to:

1 . What is testing
2. **How can test software.**

AGENDA

- Unit Testing
- Unit-test considerations
- Integration Testing
- Bottom-up integration
- Smoke testing
- Strategic options

Introduction

Following are the issues considered to implement software testing strategies.

❖Specify the requirement before testing starts in a quantifiable manner.

❖According to the categories of the user generate profiles for each category of user.

❖Produce a robust software and it's designed to test itself.

❖Should use the Formal Technical Reviews (FTR) for the effective testing.

❖To access the test strategy and test cases FTR should be conducted.

❖To improve the quality level of testing generate test plans from the users feedback

FOLLOWING ARE THE FOUR STRATEGIES FOR CONVENTIONAL SOFTWARE

1) Unit testing

2) Integration testing

3) Regression testing

4) Smoke testing

Unit testing

•Unit testing focus on the smallest unit of software design, i.e module or software component.

•Test strategy conducted on each module interface to access the flow of input and output.

•The local data structure is accessible to verify integrity during execution.

•Boundary conditions are tested.

•In which all error handling paths are tested.

•An Independent path is tested.

FOLLOWING FIGURE SHOWS THE UNIT TESTING

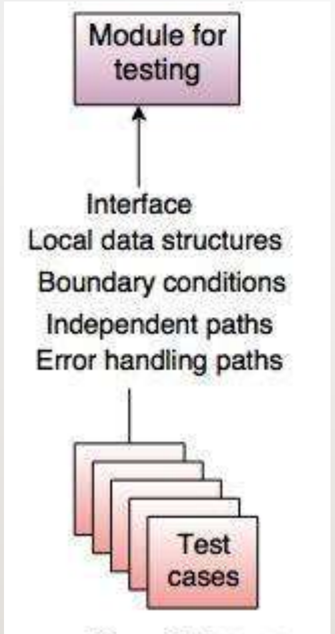

Fig. - Unit test

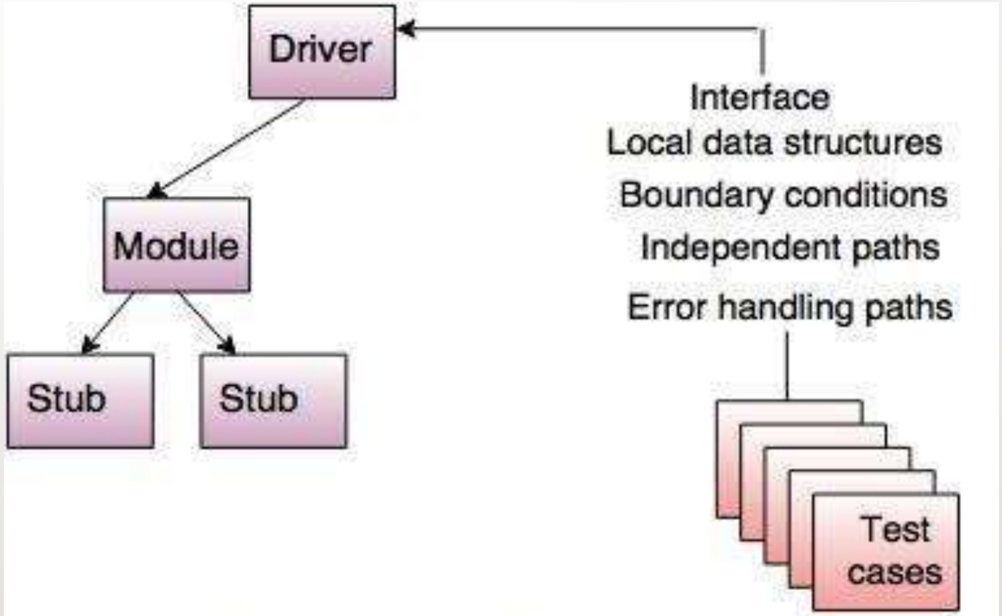

Fig. - Unit test environment

INTEGRATION TESTING

•Integration testing is used for the construction of software architecture

There are two approaches of incremental testing are:
i) Non incremental integration testing
ii) Incremental integration testing

i) Non incremental integration testing Combines all the components in advanced.
•A set of error is occurred then the correction is difficult because isolation cause is complex.
•**ii) Incremental integration testing** The programs are built and tested in small increments.
•The errors are easier to correct and isolate.
•Interfaces are fully tested and applied for a systematic test approach to it.

TOP-DOWN INTEGRATION

- It is an incremental approach for building the software architecture.
- It starts with the main control module or program.
- Modules are merged by moving downward through the control hierarchy.

Following figure shows the top down integration.

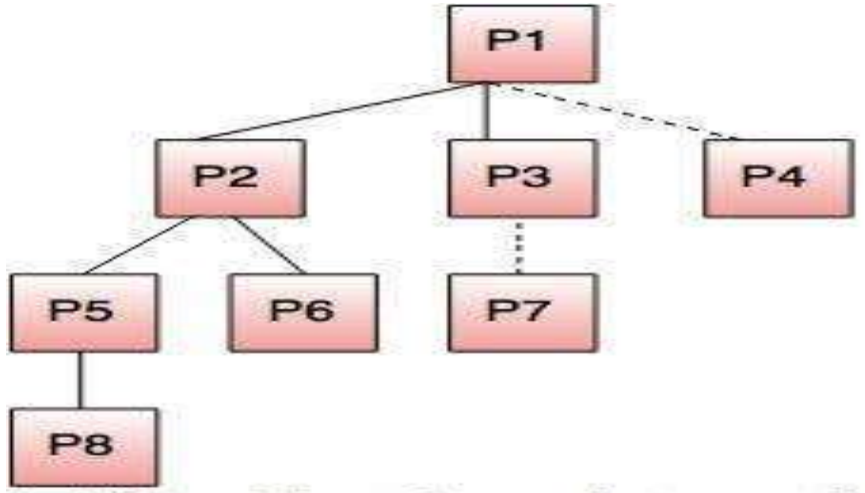

Fig. - Top-down integration

Bottom-up integration

In bottom up integration testing the components are combined from the lowest level in the program structure.

The bottom-up integration is implemented in following steps:

•The low level components are merged into clusters which perform a specific software sub function.
•A control program for testing(driver) coordinate test case input and output.
•After these steps are tested in cluster.
•The driver is removed and clusters are merged by moving upward on the program structure.

Following figure shows the bottom up integration:

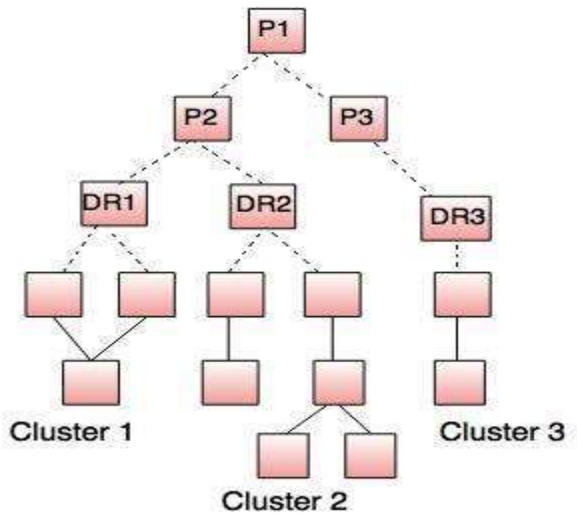

Fig. - Bottom-up integration

SELF-ASSESSMENT QUESTIONS

- **1. Describe the UNIT TESTING.**
- **2 Describe the Top-down integration**
- **3. Analyse how can do Smoke testing**
- **4. . Analyse how can do Regression testing**

SMOKE TESTING

- It is an integration testing approach that is commonly used when product software is developed. It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess the project on a frequent basis. In essence, the smoke-testing approach encompasses the following activities:
- 1. Software components that have been translated into code are integrated into a build. A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
- 2. A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover "showstopper" errors that have the highest likelihood of throwing the software project behind schedule.
- 3. The build is integrated with other builds, and the entire product is smoke tested daily. The integration approach may be top down or bottom up.

INTEGRATION TESTING

- As integration testing is conducted, the tester should identify critical modules. A critical module has one or more of the following characteristics:
- (1) Addresses several software requirements,
- (2) Has a high level of control,
- (3) Is complex or error prone?
- (4) Has definite performance requirements.

REFERENCES FOR FURTHER LEARNING OF THE SESSION

TEXTBOOKS:

1. Roger S.Pressman, "Software Engineering – A Practitioner's Approach" 7th Edition, Mc Graw Hill,(2014).
2. Ian Sommerville, "Software Engineering", Tenth Edition, Pearson Education, (2015).
3. Agile Software Development Ecosystems, Jim Highsmith, Addison Wesley; ISBN: 0201760436; 1st edition

Reference Book

Agile Modelling: Effective Practices for Extreme Programming and the Unified Process Scott Amber John Wiley & Sons; ISBN: 0471202827; 1st edition.

WEB REFERNCES/MOOCS:

https://www.digite.com/kanban/what-is-kanban/
http://www.scaledagileframework.com
https://www.guru99.com/test-driven-development.html
https://junit.org/junit5/

THANK YOU

Team – ADAPTIVE SOFTWARE ENGINEERING