COURSE NAME – ADAPTIVE SOFTWARE ENGINEERING

COURSE CODE – 23CI2001

SESSION-20

TOPIC:

- A Strategics Approach to Software Testing
- Strategic Issues

AIM OF THE SESSION

To familiarize students with the basic concept of software testing strategies and Issues

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate **software testing strategies**
2. Describe **Verification and Validation**
3. List out the **Strategies issues**
4. Describe the Various t**esting methodologies**

LEARNING OUTCOMES

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

1. Define **Software Verification and Validation**
2. Describe **Importance of software Testing Strategies**
3. Summarize **Unit, smoke, Regression and Integration testing**

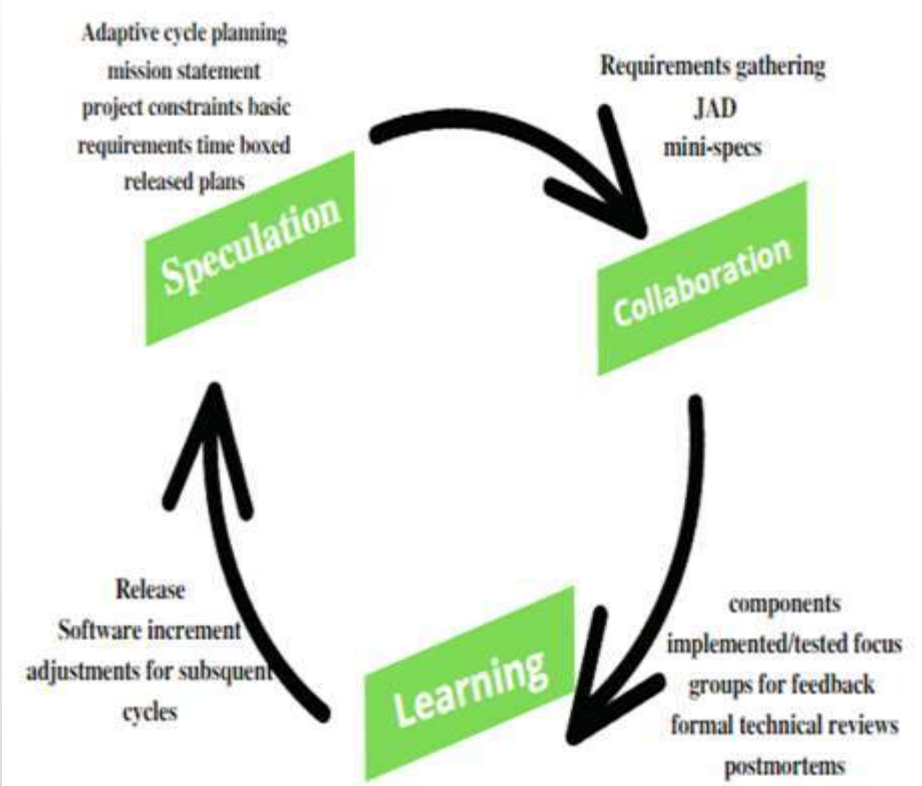
OUTLINE OF SESSION

- Introduction
- Adaptive Software Development Process
- Software Testing Strategies
- A Strategic Approach To Software Testing
- Verification and Validation
- Testing Strategy
- Strategic Issues
- Test Strategies for Conventional Software

❖ INTRODUCTION

Definition:

- The definition given by ASD's creators, Jim Highsmith and Sam Bayer, is that ASD "embodies the principle that continuous adaptation of the process to the work at hand is the normal state of affairs."
- **Adaptive Software Development** is a method to build complex software and system. ASD focuses on human collaboration and self-organization. ASD **"life cycle"** incorporates three phases namely:
  1. Speculation **2.** Collaboration **3.** Learning



THE ADAPTIVE SOFTWARE ENGINEERING DEFINITION:

- "Adaptive Software Engineering is an engineering Approach ,which includes the principles, methods and process to support customers changes at any phases of software life cycles."

Dr. A. Abdul Rahman-2023

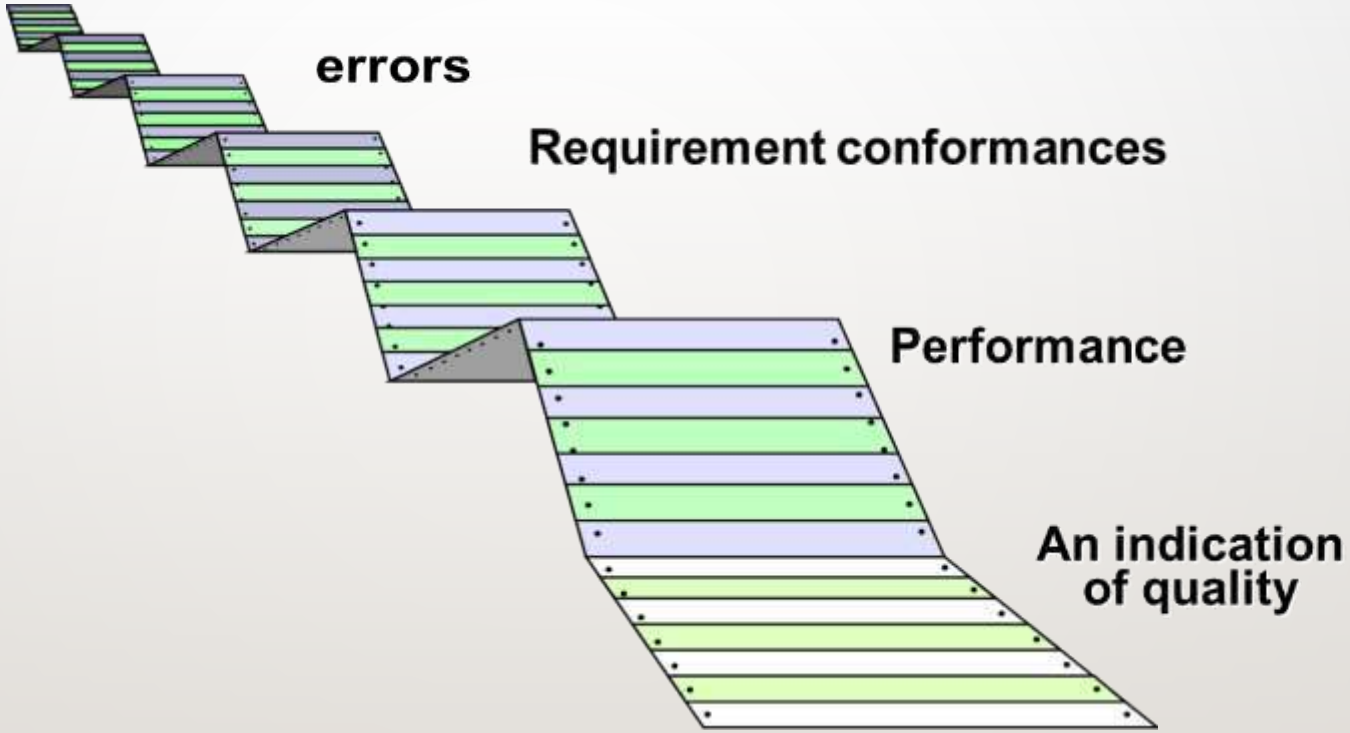
Software Testing Strategies

Testing is the process of exercising a **program with the specific intent of finding errors** prior to delivery to the end user.

A strategy for software testing provides a **road map** that describes the steps to be conducted as part of testing, when these steps are planned and then undertaken, and how much **effort, time, and resources will be required**. Therefore, any testing strategy must incorporate test planning, test case design, test execution, and resultant data collection and evaluation.

A software testing strategy should **be flexible** enough to promote a **customized testing** approach. At the same time, it must be **rigid** enough to **encourage reasonable planning and management tracking** as the project progresses.

What Testing Shows



A Strategic Approach To Software Testing

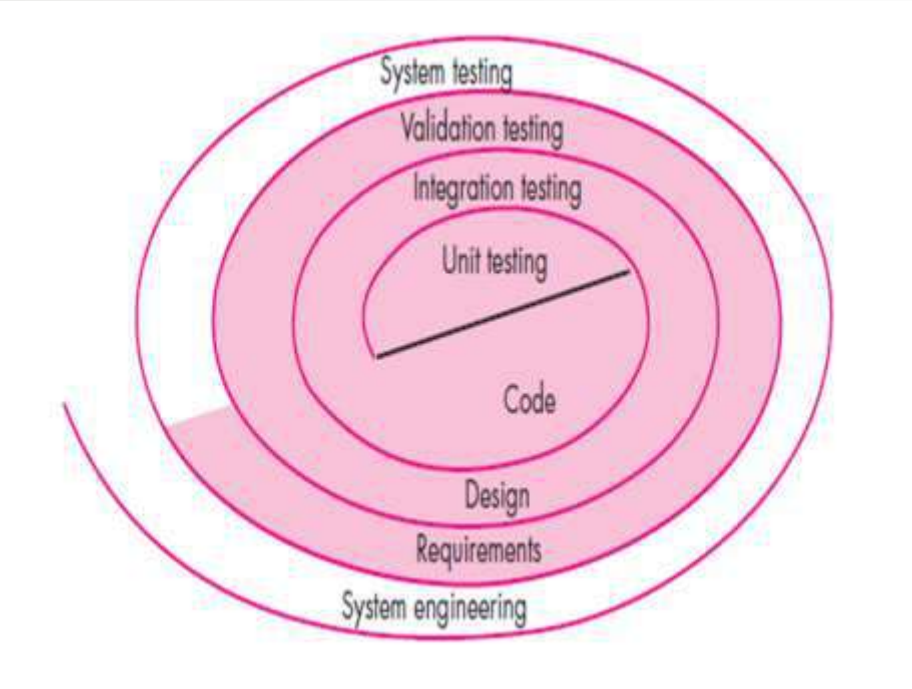
- To perform effective testing, you should **conduct effective technical reviews**. By doing this, many errors will be eliminated before testing commences.
- Testing begins at **the component level and works** "outward" toward the integration of the entire computer-based system.
- **Different testing** techniques are appropriate for **different software** engineering approaches and at different points in time.
- Testing is **conducted by the developer of the** software and (for large projects) **an independent test group**.
- **Testing and debugging** are different activities, but debugging must be **accommodated** in any testing strategy.

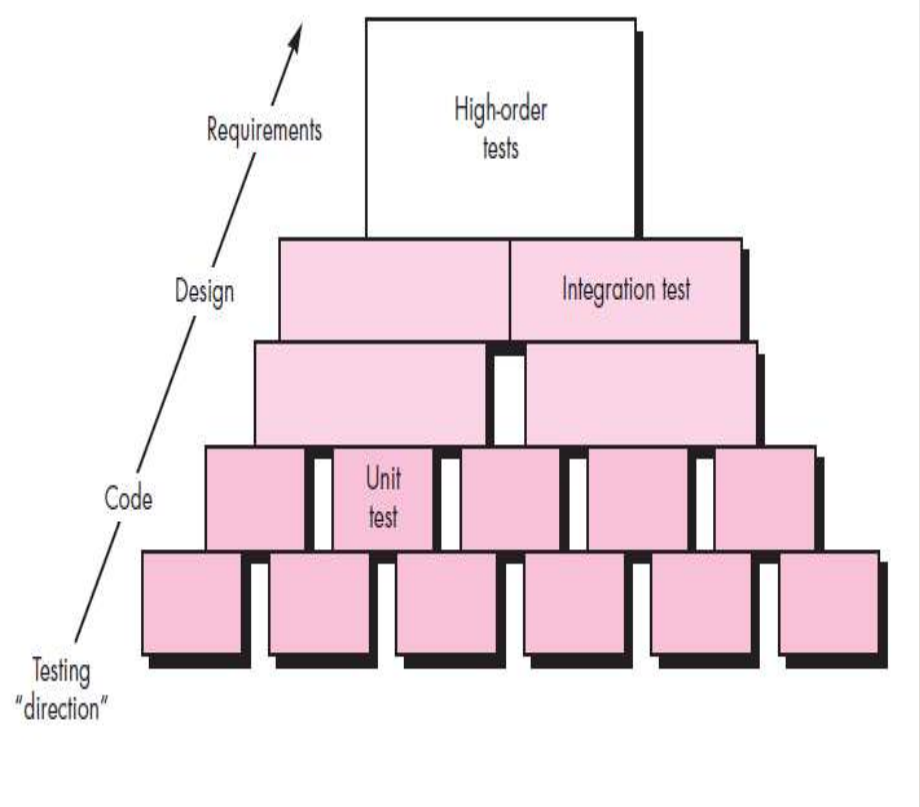
❖ Verification and Validation

- *Verification* refers to the set of tasks which ensure that software correctly implements a specific function.

- *Validation* refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. Boehm [Boe81] states this another way:
  - *Verification:* "Are we building the product right?"
  - *Validation:* "Are we building the right product?"

❖ SOFTWARE TESTING STRATEGY—THE BIG PICTURE





❖ Testing Strategy

- We begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’
- For conventional software
  - The module (component) is our initial focus
  - Integration of modules follows
- For Object Oriented software
  - Our focus when “testing in the small” changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

❖ STRATEGIC ISSUES

- **Specify product requirements** in a quantifiable manner long before testing commences.
- State **testing objectives** explicitly.
- **Understand the users** of the software and develop **a profile** for each user category.
- Develop a testing plan that emphasizes "**rapid cycle testing**."
- Build "robust" (healthy) software that is **designed to test** itself.
- Use effective **technical reviews** as a filter prior to testing.
- **Conduct technical reviews** to assess the **test strategy** and **test cases** themselves.
- Develop a **continuous improvement approach** for the testing process.

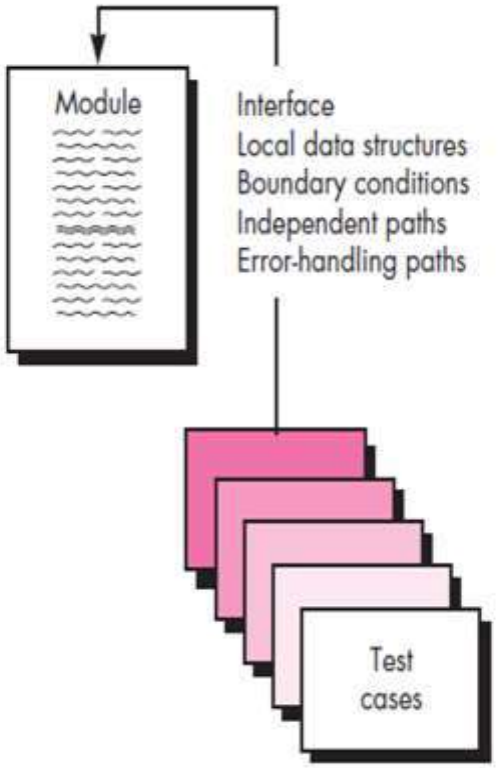
❖ Test Strategies for Conventional Software

- A testing strategy that is chosen by most software teams falls between the two extremes. It takes an incremental view of testing, beginning with the testing of individual program units, moving to tests designed to facilitate the integration of the units, and culminating with tests that exercise the constructed system.

- **Unit Testing**
- **Integration Testing**

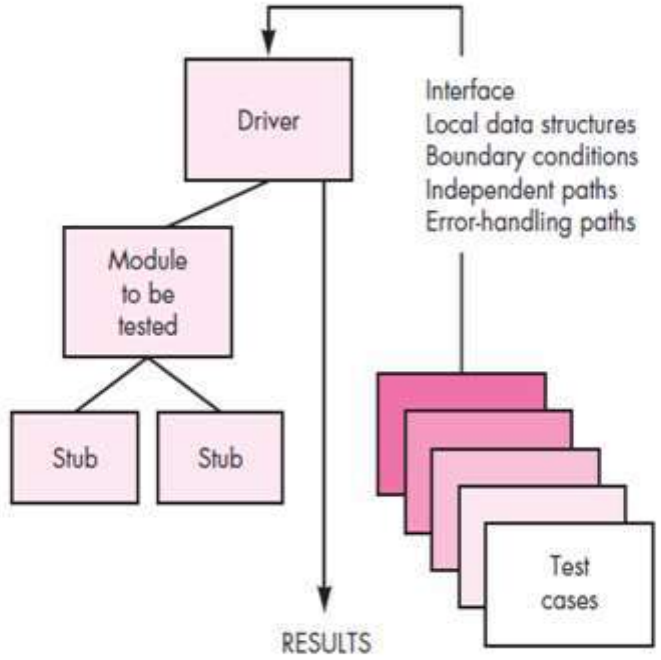
❖ Unit Testing

- Unit testing focuses **verification** effort on the **smallest** unit of software design—the software component or module.
- **Unit-test considerations:** The module **interface** is tested to ensure that information **properly flows into and out** of the program unit under test. All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once. **Boundary conditions** are tested to ensure that the module operates properly at boundaries established **to limit or restrict processing**. And finally, all error-handling paths are tested.



UNIT-TEST PROCEDURES

Unit testing is normally considered as an **adjunct to the coding step**. The design of unit tests can occur before coding begins or after source code has been generated. A review of design information provides guidance for establishing test cases that are likely to uncover errors in each of the categories discussed earlier. Each test case should be coupled with a set of expected results.

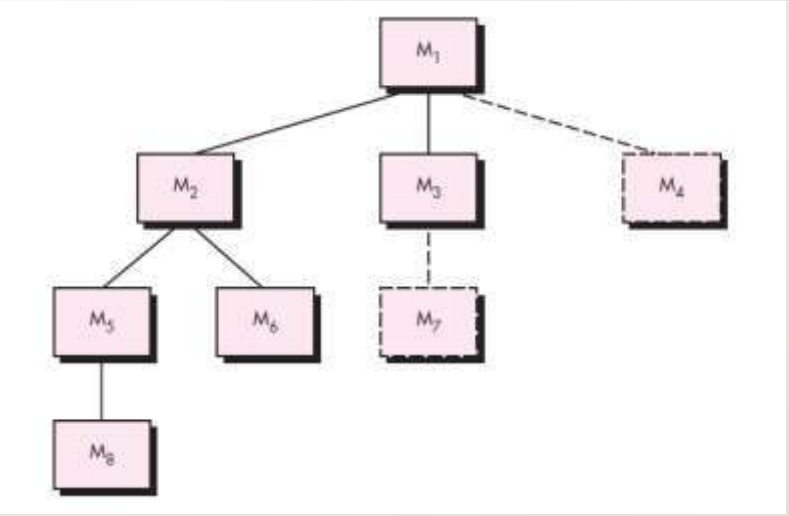


Stubs and drivers both are dummy modules and are only created for **test** purposes. ... On the other hand, **Drivers** are the ones, which are the "calling" programs. **Drivers** are used in bottom-up **testing** approach. **Drivers** are dummy code, which is used when **the sub modules are ready, but the main module** is **still not** ready.

A **stub** is a **small piece of code** that takes the place of another component during **testing**. The benefit of using a **stub** is that it returns consistent results, making the **test** easier to write. And you can run **tests** even if the other components are not working yet.

INTEGRATION TESTING

Integration testing is a systematic technique for constructing the **software architecture** while at the same time conducting tests to uncover **errors associated with interfacing**. The objective is to take unit-tested components and build a program structure that has been dictated by design.



Top-down integration: Top-down integration testing is an **incremental** approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module(main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner. Depth-first integration integrates all components on a major control path of the program structure. (In **breadth-first testing**, all the modules are refined at the **same level** of control). **Depth Testing**, the feature of a product is **tested** in **full detail.**

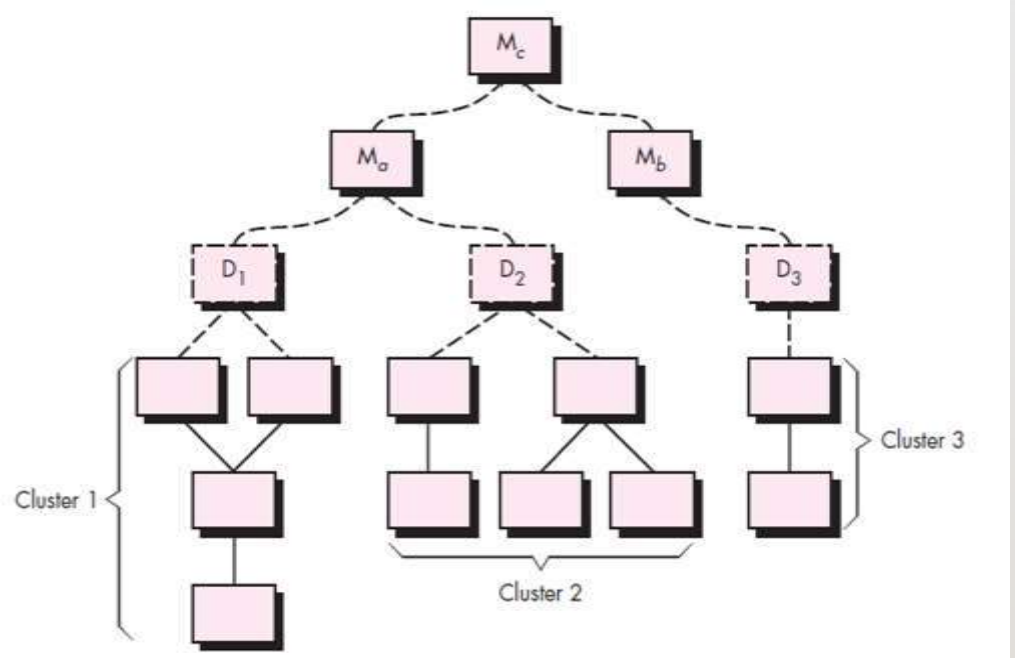
THE STEPS INVOLVED IN INTEGRATION PROCESS

1. The **main control module** is used as **a test driver** and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate **stubs are replaced one at a time** with actual components.
3. Tests are conducted as **each component is integrated**.
4. On completion of each set of tests**, another stub is** replaced with the real component.
5. **Regression testing** (discussed later in this section) may be conducted to ensure that **new errors have not been introduced**.

The process continues from step 2 until the entire program structure is built.

EXAMINE THE RESULT

- **Bottom-up integration:** Bottom-up integration testing, as its name implies, begins construction and testing with **atomic modules (i.e., components at the lowest levels in the program structure**).
  1. Low-level components are combined **into clusters** (sometimes called builds) that perform a specific software sub-function.
  2. A driver (a control program for testing) is written to coordinate test case input and output.
  3. The cluster is tested.
  4. **Drivers are removed, and clusters** are combined moving upward in the program structure.



Regression testing:

- Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

SMOKE TESTING

- Smoke testing 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.**

- **Smoke Testing** is a software testing process that determines whether the **deployed software build is stable or not**. Smoke testing is a confirmation for QA team to proceed with further software testing. It consists of a minimal set of tests run on each build to test software functionalities. Smoke testing is also known as "Build Verification Testing" or “Confidence Testing.”

- Smoke Testing is done whenever the new functionalities of software are developed and integrated with existing build that is deployed in QA/staging environment. It **ensures that all critical functionalities are working correctly or not.**

SELF-ASSESSMENT QUESTIONS

1. Define Software testing?
2. What is regression testing ?
3. List the various strategies issues.
4. Why Unit testing?.
5. How stub is different from driver.?
6. List the advantages of regression testing.?
7. Need of Bottom-up integration.
8. Define Object Oriented Software?
9. What are items will be covered by testing?
10. How Adaptive software Engineering is different from traditional Software Engineering Approaches?

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

(DEEMED TO BE UNIVERSITY)

Team – ADAPTIVE SOFTWARE ENGINEERING