MSS Module 3 Important Topics

> ***Table of contents***

Design patterns

- The pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings.

Pattern Elements

- A name that is a meaningful reference to the pattern
- A description of the problem area that explains when the pattern may be applied.
- A solution description of the parts of the design solution, their relationships and their responsibilities
- A statement of the consequences—the results and trade-offs—of applying the pattern.

Open-source development

- Open-source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process
- Open-source software extended this idea by using the Internet to recruit a much larger population of volunteer developers
- Many of them are also users of the code. In principle at least, any contributor to an open-source project may report and fix bugs and propose new features and functionality

Open Source licensing

Most open-source licenses are variants of one of three General models

- The GNU General Public License (GPL)
  - If you use open-source software that is licensed under the GPL license, then you must make that software open source
- The GNU Lesser General Public License (LGPL)
  - Same as GPL, but it permits the software to be linked with proprietary software.
  - So, you can use an LGPL-licensed part in a closed-source project without making the whole project open-source.
- The Berkley Standard Distribution (BSD) License.
  - You can include the code in proprietary systems that are sold.
  - If you use open-source components, you must acknowledge the original creator of the code.
  - The MIT license is a variant of the BSD license with similar conditions.

Review Techniques

- The primary objective of technical reviews is to find errors during the process so that they do not become defects after release of the software.

Informal Review

- Informal reviews are like simple checks we do with a colleague while working on a computer program.
- Even though they're not super formal, just looking over things with a colleague can catch mistakes early.
- One way to make these checks better is to use simple checklists. These are like lists of questions that help guide us while looking at our work. They might be basic questions, but they make sure we don't miss important things.

Formal Technical Review

- A formal technical review (FTR) is a software quality control activity performed by software engineers (and others).
- The objectives are
  - To uncover errors in function, logic, or implementation for any representation of the software
  - To verify that the software under review meets its requirements
  - To ensure that the software has been represented according to predefined standards
  - To achieve software that is developed in a uniform manner
  - To make projects more manageable

Post Mortem Evaluations

- PME is like looking back at a computer project after it's done.
- PME examines the entire software project, focusing on both “excellences (that is, achievements and positive experiences) and challenges (problems and negative experiences)”
- The intent is to identify excellences and challenges and to extract lessons learned from both. The objective is to suggest improvements to both process and practice going forward

---

Software testing strategies

- Testing is a set of activities that can be planned in advance and conducted systematically

- Software is tested to uncover errors that were made inadvertently as it was designed and constructed
  Test strategies

1. Unit Testing

- Unit testing focuses verification effort on the smallest unit of software design—the software component or module
- This type of testing can be conducted in parallel for multiple components

2. 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
- In integration testing, the focus is on testing the interaction and cooperation between different components or modules of a software system.

3. Validation Testing

- Validation testing is like the final check to make sure the whole software package behaves the way it's supposed to, and if there are any problems, a plan is made to fix them.
- Testing focuses on user-visible actions and user-recognizable output from the system.

4. System Testing

- Recovery testing
  - Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.
- Security Testing
  - Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration.
- Stress Testing
  - Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume.
- Performance testing
  - Performance testing is designed to test the run-time performance of software within the context of an integrated system
- Deployment Testing
  - Deployment testing, sometimes called configuration testing, exercises the software in each environment in which it is to operate.

White box testing, Black box testing

White box testing

- White-box testing, sometimes called glass-box testing or structural testing, is a test-case design philosophy that uses the control structure described as part of component-level design to derive test cases.
    - White-box testing is like peeking inside a "box" to see how it's built. It's a way of testing a computer program by looking at its internal structure and logic.
    - It's also called "glass-box testing" or "structural testing" because it's like having a transparent box, and you can see what's going on inside.
    - The idea is to use the detailed plan (control structure) that was designed when creating each part of the program.
    - This plan helps figure out what tests to do. It's like knowing the blueprint of a building and testing each part to make sure it's put together correctly.

Black box testing

- Black-box testing, also called behavioral testing or functional testing, focuses on the functional requirements of the software.
- Unlike white-box testing, which is performed early in the testing process, black box testing tends to be applied during later stages of testing .
- Black-box testing techniques enable you to derive sets of input conditions that will fully exercise all functional requirements for a program
- Helps to find error in the following categories
    - incorrect or missing functions
    - interface errors
    - errors in data structures
    - behavior or performance errors
    - initialization and termination errors

Test-driven development

- Test-driven development (TDD) is an approach to program development that is based on the general idea that you should write an executable test or tests for code that you are writing before you write the code

- Test-driven development relies on automated testing.
  - Every time you add some functionality, you develop a new test and add it to the test suite.
  - All of the tests in the test suite must pass before you move on to developing the next increment.
- Benefits of test driven development
  - It is a systematic approach to testing in which tests are clearly linked to sections of the program code.
  - It should be possible to understand what the program does by reading the tests
  - TDD leads to simpler code, as programmers only write code that's necessary to pass tests.

Overview of DevOps and Code Management

Devops

- DevOps (development + operations) integrates development, deployment, and support, with a single team responsible for all of these activities

Code Management

- Code management is a set of software-supported practices used to manage an evolving codebase
- You need code management to ensure that changes made by different developers do not interfere with each other and to create different product versions
- Code management tools make it easy to create an executable product from its source code files and to run automated tests on that product

Features

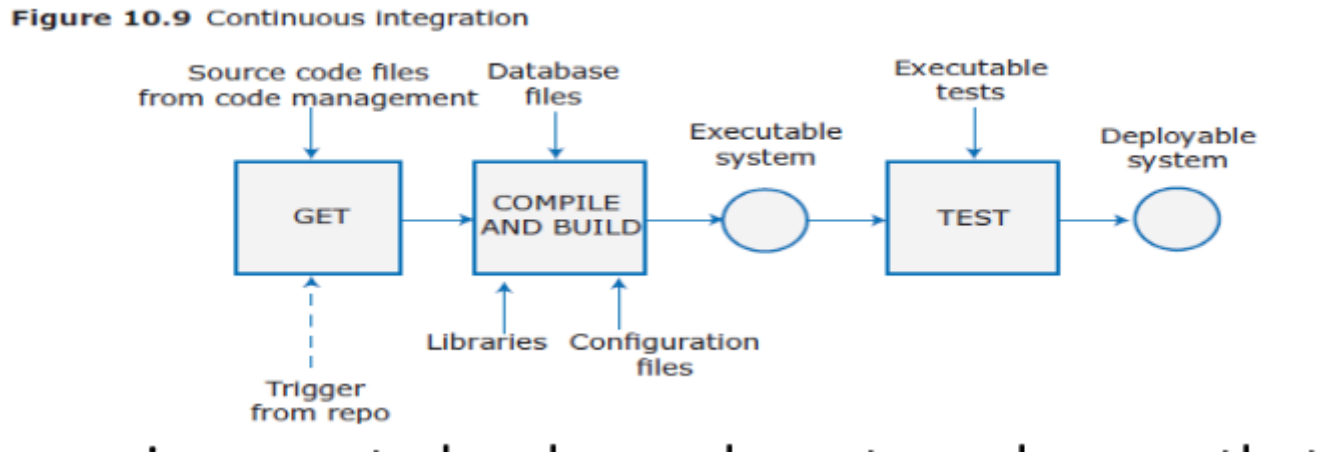
- Code transfer
  - Developers take code into their personal file store to work on it; then they return it to the shared code management system
- Version storage and retrieval
  - Files may be stored in several different versions, and specific versions of these files can be retrieved
- Merging and branching

    - Parallel development branches may be created for concurrent working. Changes made by developers in different branches may be merged.
- Version information
    - Version information: Information about the different versions maintained in the system may be stored and retrieved

---

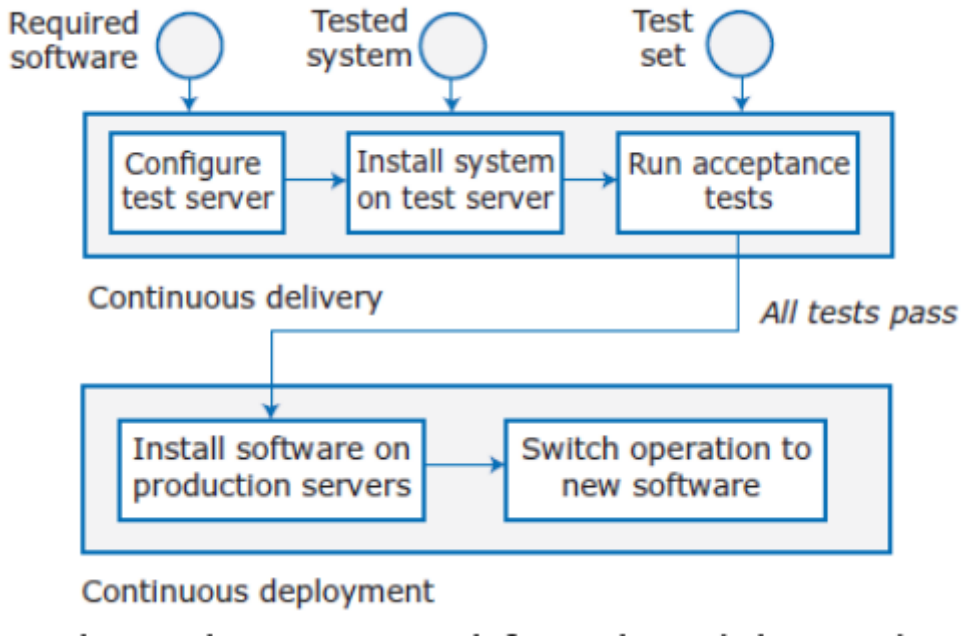
Continuous Integration, Delivery, and Deployment (CI/CD/CD)

Continuous Integration



- System integration (system building) is the process of gathering all of the elements required in a working system, moving them into the right directories, and putting them together to create an operational system
- Continuous integration simply means that an integrated version of the system is created and tested every time a change is pushed to the system's shared code repository.
- In a continuous integration environment, developers have to make sure that they don't "break the build." Breaking the build means pushing code to the project repository, which when integrated, causes some of the system tests to fail.
- The advantage of continuous integration compared to less frequent integration is that it is faster to find and fix bugs in the system.

Continuous Delivery and deployment



- Continuous delivery means that, after making changes to a system, you ensure that the changed system is ready for delivery to customers.
- This means that you have to test it in a production environment to make sure that environmental factors do not cause system failures or slow down its performance.
- Continuous deployment is obviously only practical for cloud-based systems. If your product is sold through an app store or downloaded from your website, continuous integration and delivery make sense

---

Software Evolution

- Software evolution is the ongoing process of making changes to a software system as it continues to be used.
- The best way to evolve a software system depends on the kind of software it is, how the organization develops software, and the skills of the people working on it.
- Whenever a software system needs to change, it usually starts with someone proposing the change, either in a formal or informal way
- This could be a suggestion from developers, users, or management. These proposals are like the driving force behind the evolution of the software system in all organizations
- The process of identifying changes and evolving the system doesn't happen just once; it's a continuous cycle. As the software is used and as new needs or issues come up, changes are identified and made to improve the system. This cycle keeps going throughout the entire life of the software.

- In situations where development and evolution are integrated, change implementation is simply an iteration of the development process