MODULE -5

- Software Configuration Management: Using version control, Managing dependencies, Managing software configuration, Managing build and deployment environments.
- Continuous Integration: Prerequisites for continuous integration, Essential practices.
- Continuous Delivery: Principles of Software delivery, Introduction and concepts.
- Build and deployment automation, Learn to use Ansible for configuration management.
- Test automation (as part of continuous integration), Learn to set up test automation cases using Robot Framework.

1. Glenford J. Myers, et. al., *The Art of Software Testing*, Wiley.
2. Lee Copeland, *A Practitioner's Guide to Software Test Design*, Artech House Publishers.
3. **Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Pearson Education.**

Software Configuration Management

- Configuration management is a term that is widely used, often as a synonym for version control.
- **Configuration management** refers to the process by which all artifacts relevant to your project, and the relationships between them, are stored, retrieved, uniquely identified, and modified.
- Your configuration management strategy will determine how you manage all of the changes that happen within your project.

- In Software Engineering, **Software Configuration Management**(SCM) is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.
- The primary goal is to increase productivity with minimal mistakes.
- The objective is to maintain software integrity and traceability throughout the software life cycle.

1. Getting the prerequisites in place to manage your application's build, deploy, test, and release process- We tackle this in two parts: getting everything into **version control** and **managing dependencies**.
2. **Managing** an application's **configuration**.
3. **Configuration management of whole environments**—the software, hardware, and infrastructure that an application depends upon; the principles behind environment management, from operating systems to application servers, databases, and other commercial off-the-shelf (COTS) software.

- **Version control** is a component of software configuration management
- Everything you need to build, deploy, test, and release your application should be kept in some form of **versioned storage**.
- This includes requirement documents, test scripts, automated test cases, network configuration scripts, deployment scripts, database creation, upgrade, downgrade, and initialization scripts, application stack configuration scripts, libraries, toolchains, technical documentation, and so on.
- All of this stuff should be version-controlled, and the relevant version should be identifiable for any given build.
- That is, these *change sets* should have a single identifier, such as a build number or a version control change set number, that references every piece.

Using Version Control

- **Version control systems**, also known as **source control**, **source code management systems**, or **revision control systems**, are a mechanism for keeping multiple versions of your files, so that when you modify a file you can still access the previous revisions.
- The aim of a version control system is twofold: **First**, it retains, and provides access to, every version of every file that has ever been stored in it. **Second**, it allows teams that may be distributed across space and time to collaborate.

Keep Absolutely Everything in Version Control :-

- The objective is to have everything that can possibly change at any point in the life of the project stored in a controlled manner.
- This allows you to recover an exact snapshot of the state of the entire system, from development environment to production environment, at any point in the project's history.
- It is even helpful to keep the configuration files for the development team's development environments in version control since it makes it easy for everyone on the team to use the same settings.

- Analysts should store requirements documents.
- Testers should keep their test scripts and procedures in version control.
- Project managers should save their release plans, progress charts, and risk logs here.
- In short, every member of the team should store any document or file related to the project in version control.

Check In Regularly to Trunk:-

- it is important to be sure that your work, whatever it may be, is ready for the level of publicity that a check-in implies.
- When a new features is developed incrementally, commit them to the trunk in version control on a regular and frequent basis.
- This keeps the software working and integrated at all times.
- It means that your software is always tested because your automated tests are run on trunk by the continuous integration (CI) server every time you check in.
- It reduces the possibility of large merge conflicts caused by refactoring, ensures that integration problems are caught immediately when they are cheap to fix, and results in higher-quality software.

Use Meaningful Commit Messages:-

- Every version control system has the facility to add a description to your commit.
- It is easy to omit these messages, and many people get into the bad habit of doing so.
- The most important reason to write descriptive commit messages is so that, when the build breaks, you know who broke the build and why.
- A commit message explaining what the person was doing when they committed that change can save you hours of debugging.

Managing Dependencies

- The most common external dependencies within your application are the third party libraries it uses and the relationships between components or modules under development by other teams within your organization.
- ***Managing External Libraries:-*** External libraries usually come in binary form, unless you're using an interpreted language. Even with interpreted languages, external libraries are normally installed globally on your system by a package management system.

- It is better to keep copies of your external libraries somewhere locally.
- This is essential if you have to follow compliance regulations, and it also makes getting started on a project faster.
- we emphasize that your build system should always specify the exact version of the external libraries that you use.
- If you don’t do this, you can’t reproduce your build.
- Whether you keep external libraries in version control or not involves some trade-offs. It makes it much easier to correlate versions of your software with the versions of the libraries that were used to build them.

Managing Software Configuration

- Configuration is one of the three key parts that comprise an application, along with its binaries and its data.
- Configuration information can be used to change the behavior of software at build time, deploy time, and run time.
- Delivery teams need to consider carefully what configuration options should be available, how to manage them throughout the application's life, and how to ensure that configuration is managed consistently across components, applications, and technologies.
- you should treat the configuration of your system in the same way you treat your code: Make it subject to proper management and testing.

- **Configuration and Flexibility**
- Everyone wants flexible software. But flexibility usually comes at a cost.
- Most applications they are designed for a specific purpose, but within the bounds of that purpose they will usually have some ways in which their behavior can be modified.
- The desire to achieve flexibility may lead to the common antipattern of “ultimate configurability” which is, all too frequently, stated as a requirement for software projects.
- It is at best unhelpful, and at worst, this one requirement can kill a project.

- Configurable software is not always the cheaper solution it appears to be. It’s almost always better to focus on delivering the high-value functionality with little configuration and then add configuration options later when necessary.
- Most configuration information is free-form and untested.
- Configuration is not inherently evil. But it needs to be managed carefully and consistently.
- Modern computer languages have evolved all sorts of characteristics and techniques to help them reduce errors.

- In most cases, these protections do not exist for configuration information, and more often than not there are not even any tests in place to verify that your software has been configured correctly in testing and production environments.
- Deployment smoke tests are one way to mitigate this problem and should always be used.
- **Smoke tests** are a subset of test cases that cover the most important functionality of a component or system, used to aid assessment of whether main functions of the software appear to work correctly. **Smoke testing** is also done by testers before accepting a build for further testing.

Types of Configuration

Configuration information can be injected into your application at several points in your build, deploy, test, and release process, and it's usual for it to be included at more than one point.

- Your build scripts can pull configuration in and incorporate it into your binaries at **build time.**
- Your packaging software can inject configuration at **packaging time,** such as when creating assemblies, ears, or gems.
- Your deployment scripts or installers can fetch the necessary information or ask the user for it and pass it to your application at **deployment time** as part of the installation process.
- Your application itself can fetch configuration at **startup time or run time.**

- Whatever mechanism you choose, it is recommended that, as far as practically possible, you should try and supply all configuration information for all the applications and environments in your organization through the same mechanism.
- This isn’t always possible, but when it is, it means that there is a single source of configuration to change, manage, version-control, and override (if necessary).
- In organizations where this practice isn’t followed, people regularly spend hours tracking down the source of some particular setting in one of their environments.

Managing Your Environments (build and deployment)

- No application is an island.
- Every application depends on hardware, software, infrastructure, and external systems in order to work – refered as application's environment.
- The principle to bear in mind when managing the environment that your application runs in is that the configuration of that environment is as important as the configuration of the application.
- For example, if your application depends on a messaging bus, the bus needs to be configured correctly or the application will not work.

- The problems can be summed up as follows:

1. One small change can break the whole application or severely degrade its performance.
2. Once it is broken, finding the problem and fixing it takes an indeterminate amount of time and requires senior personnel.
3. It is extremely difficult to precisely reproduce manually configured environments for testing purposes.
4. The collection of configuration information is very large.
5. It is difficult to maintain such environments without the configuration, and hence behavior, of different nodes drifting apart.

- In order to reduce the cost and risk of managing environments, it is essential to turn our environments into mass-produced objects whose creation is repeatable and takes a predictable amount of time.
- The key to managing environments is to make their creation a fully automated process.
- It should always be cheaper to create a new environment than to repair an old one.

- Being able to reproduce your environments is essential for several reasons:-

1. It removes the problem of having random pieces of infrastructure around whose configuration is only understood by somebody who has left the organization and cannot be reached. When such things stop working, you can usually assume a significant downtime. This is a large and unnecessary risk.
2. Fixing one of your environments can take many hours. It is always better to be able to rebuild it in a predictable amount of time so as to get back to a known good state.
3. It is essential to be able to create copies of production environments for testing purposes. In terms of software configuration, testing environments should be exact replicas of the production ones, so configuration problems can be found early.

- The kinds of environment configuration information you should be concerned about are:

1. The various operating systems in your environment, including their versions, patch levels, and configuration settings.
2. The additional software packages that need to be installed on each environment to support your application, including their versions and configuration.
3. The networking topology required for your application to work.
4. Any external services that your application depends upon, including their versions and configuration.
5. Any data or other state that is present in them (for example, production databases)

- There are **two principles** that form the basis of an effective configuration management strategy:

1. Keep binary files independent from configuration information, and
2. keep all configuration information in one place.

- Applying these fundamentals to every part of your system will pave the way to the point where creating new environments, upgrading parts of your system, and rolling out new configurations without making your system unavailable becomes a simple, automated process.

- When evaluating third-party products and services, start by asking the following questions:

1. Can we deploy it?

2. Can we version its configuration effectively?

3. How will it fit into our automated deployment strategy?

- An environment that is in a properly deployed state is known as a *baseline* in configuration management terminology.
- Your automated environment provisioning system should be able to establish, or re-establish, any given baseline that has existed in the recent history of your project.
- Any time you change any aspect of the host environment of your applications, you should store the change, creating a new version of the baseline and associating that version of the application with the new version of the baseline.
- This ensures that the next time that you deploy the application or create a new environment, it will include the change.

- You should treat your environment the same way you treat your code—changing it incrementally and checking the changes into version control.
- Every change should be tested to ensure that it doesn’t break any of the applications that run in the new version of the environment.
- Integration is often a very painful process.
- If this is true on your project, integrate every time somebody checks in, and do it from the start of the project.
- If testing is a painful process that occurs just before release, don’t do it at the end. Instead, do it continually from the beginning of the project.

Benefits of Configuration Management

Configuration Management provides the following benefits:

1. Allowing configuration to be version controlled
2. Detecting and correcting configuration drift
3. Treating infrastructure as flexible resource
4. Facilitating automation
5. Enabling automated scale-up and scale-out
6. Providing environment consistency