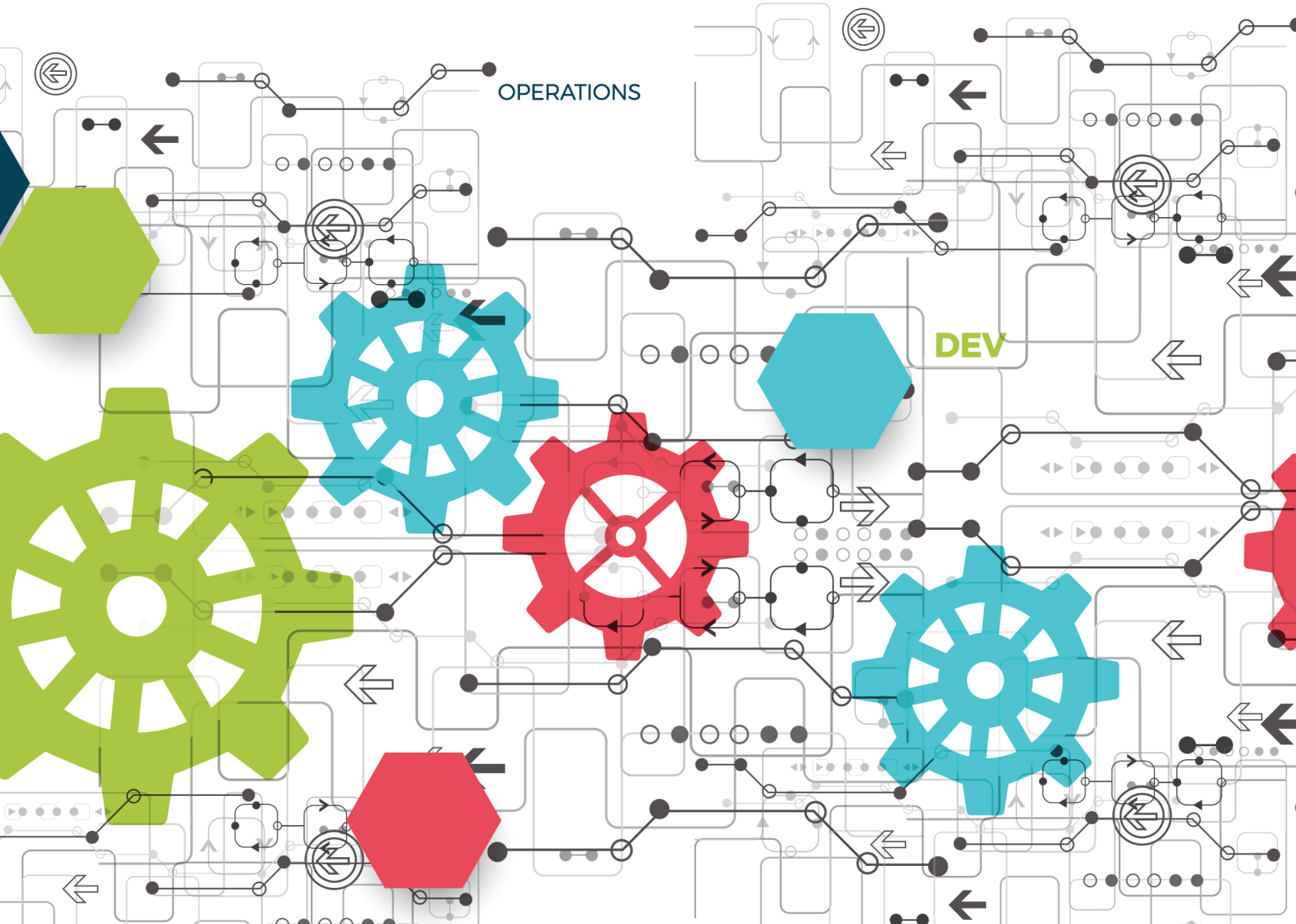
B.Tech Computer Science and Engineering in DevOps

Continuous Integration and Continuous Delivery

MODULE 1

Overview of Continuous Integration and Continuous Delivery



Contents

MODULE 1

Overview of Continuous Integration and Continuous Delivery

Module Objectives

At the end of this module, you will be able to:

- Provide an overview of Continuous Integration (CI)
- Define practices associated with CI and the working mechanism
- Discuss the various benefits of CI
- Explain Continuous Delivery (CD) and the CD pipeline
- List prerequisites and business benefits of CD
- Describe continuous deployment and its business drivers
- Identify several benefits of continuous deployment

Facilitator Notes:

Explain the module objectives to the participants.

Module Topics

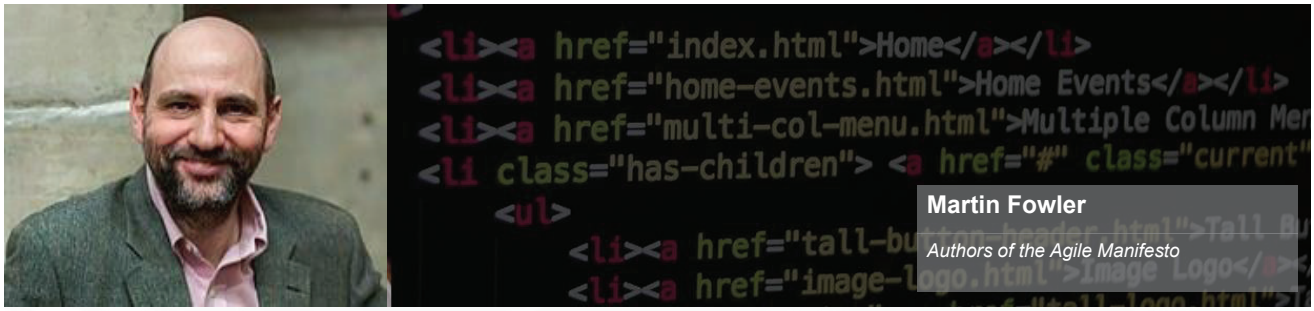
Let us take a quick look at the topics that we will cover in this module:

- Introduction to CI (Continuous Integration)
  - Practices of CI
  - How does CI work?
  - CI Workflow
  - Benefits of CI
- Capitalizing on CI to Establish CD (Continuous Delivery)
  - Introduction to CD
  - CD pipeline
  - Prerequisites for CD
  - The checklist for CD
  - Business benefits of CD
- Continuous Deployment
  - Business drivers for implementing continuous deployment
  - Benefits of continuous deployment
- DevOPS CI Case study

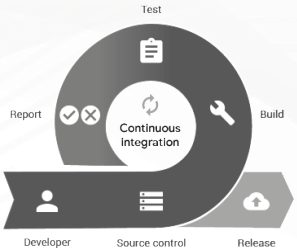
Facilitator Notes:

Inform the participants about the topics that they will be learning in this module.

1.1 Introduction to Continuous Integration



"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day."



Facilitator Notes:

Give the participants a brief about continuous integration.

We've learnt the basics of continuous integration and delivery in one of the previous courses. In this module, we'll dive deep into CI/CD and the tools and practices associated with them. Let's look at a quick recap of continuous integration.

Continuous integration is the first step towards building an automated pipeline. CI is a practice where software developers integrate the code very frequently into a shared repository, preferably multiple times a day. Each integration is then verified by automated build and tests, which help in detecting integration errors as early as possible. The CI process makes use of a centralized server which continuously monitors and pulls the changes done to the source code. This approach helps in reducing the integration problems significantly and teams can develop software in a cohesive and rapid manner.

CI is about the build or integration stage of the software release process. It is the process by which builds and extensive suites of automated test cases are run at the point that software is committed to the version control system. The CI process encompasses both an automation component (e.g., a CI or build service) and a cultural component (e.g., learning to integrate frequently).

The practices continuous integration, automated testing, CD and continuous deployment relate to each other, they differ by the way the code gets deployed to production. Let's look at the practices of continuous integration.

1.2 Practices of Continuous Integration

Given below are the important practices of CI	
1	Maintain a single source repository
2	Automate the build
3	Make your build self-testing
4	Everyone commits to the mainline everyday
5	Every commit should build the mainline on an integration machine
6	Fix broken builds immediately
7	Keep the build fast
8	Test in a clone of the production environment
9	Make it easy for anyone to get the latest executable
10	Everyone can see what's happening automate deployment

Facilitator Notes:

Explain the participants about the important practices of CI.

CI is a practice, not a tool, and it depends on the discipline that makes it effective. Keeping a CI system operating, especially large and complex CI systems, requires a significant degree of discipline from the development team as a whole. Let's now look at the important practices of CI which make the system effective.

- **Maintain a single source repository:** Teams should put in everything that is needed for a build, in a source control system. With version control systems like Git, we can have multiple branches to manage different streams of development. Though we can have

multiple branches, it is important to limit it to a minimum number, particularly it is essential to have a mainline, i.e., a single branch of a project that is currently under development. In short, the source control should have everything needed for a build, but not the actual build.

- **Automate the build:** A CI system should have automated environments for builds. For example, Unix world has had Make for a long time now, the Java community developed Ant, the .NET community has had Nant and now has MSBuild. With a CI system, we should be able to build and launch the system using these scripts using a single command. Everything should be included in an automated build.

- **Make your build self-testing:** Including automated tests in the build process is one of the quickest and most efficient ways to identify bugs. Testing can identify a lot of bugs. Methods like Test-driven development (TDD) and Extreme programming have contributed a lot to the popularity of self-testing code and there is a lot value of having self-testing code. Both TDD and XP insist on writing tests before writing the code that makes them pass. For self-testing code, a suite of automated tests is needed that can check the large part of the code base for bugs. We should be able to kick off the testing process with a simple command.

- **Everyone commits to the mainline everyday:** Developer's ability to build their code correctly is important in order to commit to the mainline. A developer first updates the working copy to match the mainline, and does conflict resolution, and then builds on their local machine. If the build passes, then they are free to commit to the mainline. This process is important to find out if there is any conflict between two developers. If the developers commit every few hours, conflicts can be detected quickly and they can be resolved quickly and efficiently.

- **Every commit should build the mainline on an integration machine:** Daily commits ensure that the team gets tested builds at frequent intervals. Teams should make sure that the regular builds happen on an integration machine and only if the integration build succeeds, the commit is considered to be done. It is the responsibility of the developer to monitor the mainline build, so they can fix if the build breaks. A CI server acts as a monitor to the repository. Every time a commit against the repository finishes the server automatically checks out the sources onto the integration machine, initiates a build, and notifies the committer of the result of the build.

- **Fix broken builds immediately:** If a build breaks, the developers responsible have to fix it immediately. They should identify the cause of the breakage as soon as possible and fix it. If we adopt this strategy, we will always be in the best position to work out what caused the breakage and fix it immediately. If this is not followed, it will take much longer for the build to be fixed.

- **Keep the build fast:** The most important concept of CI is rapid feedback. XP demands that a build is not more than 10 minutes. Any build that takes more time is considered unreasonable. Since CI demands frequent commits, reducing the build time is very important from a developer's perspective.

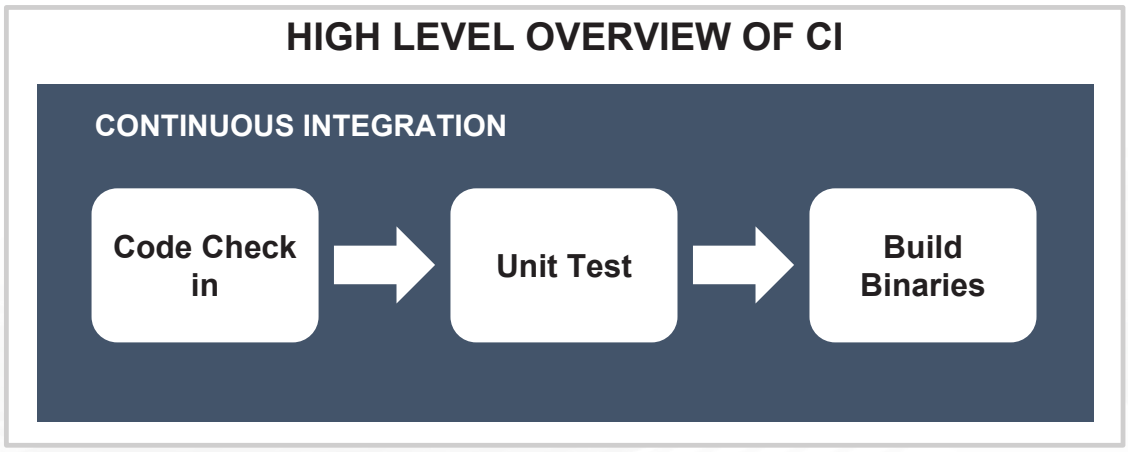
- **Test in a clone of the production environment:** The idea of testing is to get rid of any issue that the system might have in production. If the test environment is different from production, there is a high possibility that things that work under test environment might fail in a production environment. It is essential that the test environment mimics the production environment. Use the same database software, with the same versions, uses the same version of the operating system. Put all the appropriate libraries that are in the production environment into the test environment, even if the system doesn't actually use them.

- **Make it easy for anyone to get the latest executable:** It is important that the developers build the right software. Everyone involved in the development process should be able to get the latest executable and be able to run it. The team should make sure that people know the exact location of the latest executable.

- **Everyone can see what's happening automate deployment:** CI is all about effective communication and everyone sees the exact state of the system and the changes that have been made to it. For a manual CI process the visibility is essential.

We'll dive deep into core CI processes in the upcoming modules. Let's now look at in short, how CI works.

1.3 How does CI Work?



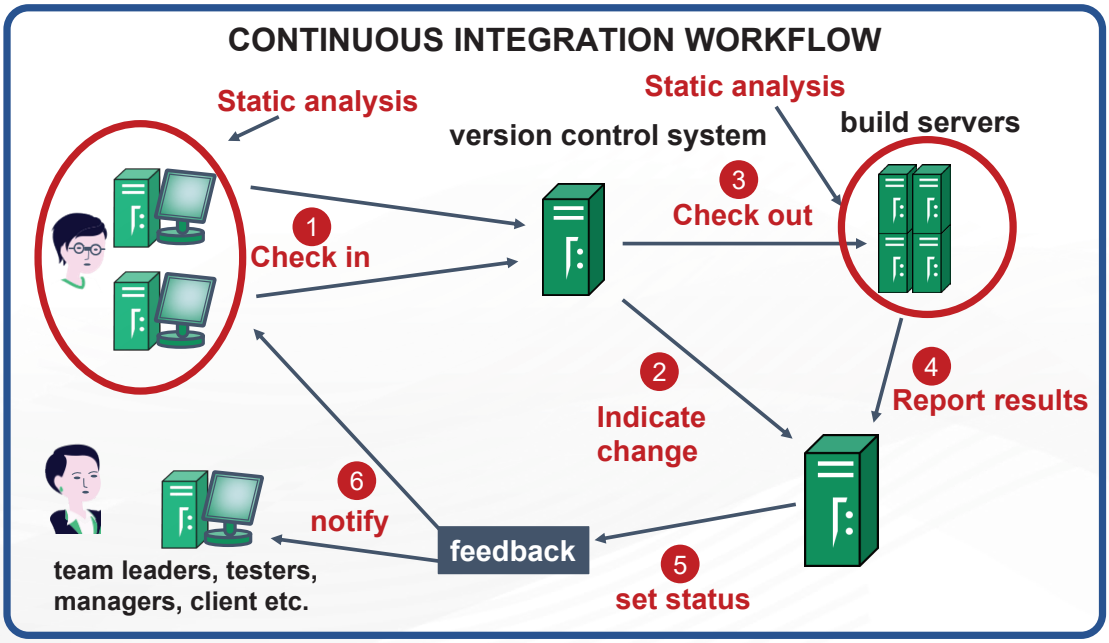
Facilitator Notes:

Give the participants a brief about the CI process.

With continuous integration, developers frequently commit to a shared repository using a version control system such as Git. Prior to each commit, developers may choose to run local unit tests on their code as an extra verification layer before integrating. A continuous integration service automatically builds and runs unit tests on the new code changes to immediately surface any errors.

Continuous integration refers to the build and unit testing stages of the software release process. Every revision that is committed triggers an automated build and test. From continuous integration, emerges CD and deployment.

1.4 Continuous Integration Workflow



Facilitator Notes:

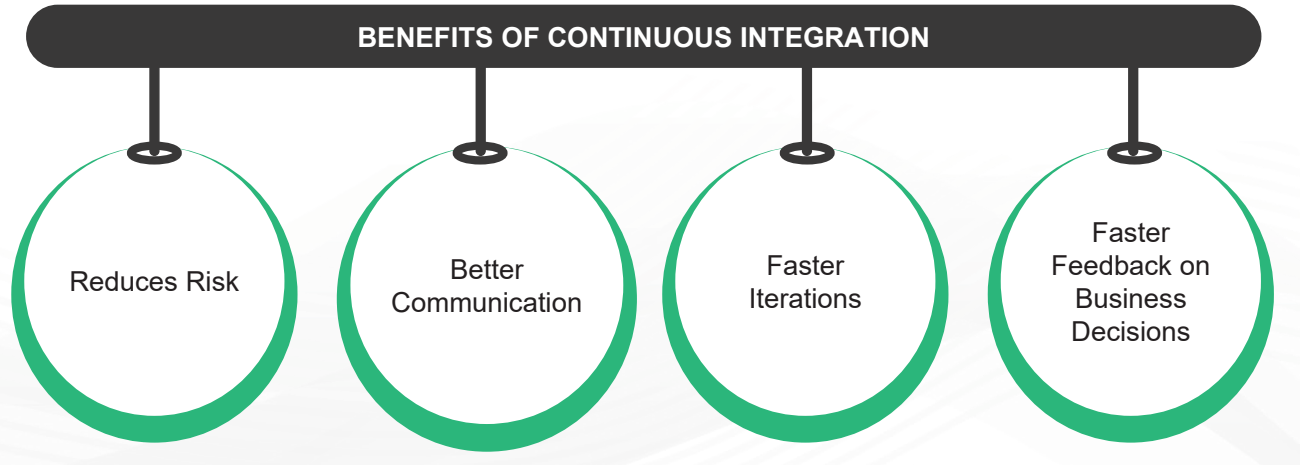
Give the participants a brief overview about CI workflow.

Refer the above image, Continuous Integration workflow is explained in 6 steps.

In current deployments it's Automation, Automation and Automation

- It should hopefully be clear by now that a successful CI implementation is largely dependent upon other processes that can be automated. For example:
  - checkout or update of source code by a version control tool
  - compilation of project source code by a build tool
  - execution of source code checking by a static analysis tool
  - unit test execution by a unit test harness or runner

1.5 Benefits of Continuous Integration



Facilitator Notes:

Explain the participants about the benefits of CI.

CI has a lot of benefits. A good CI system speeds up the workflow and makes the team commit frequently without the fear of breaking anything. The most important benefits of CI are shown in the slide above.

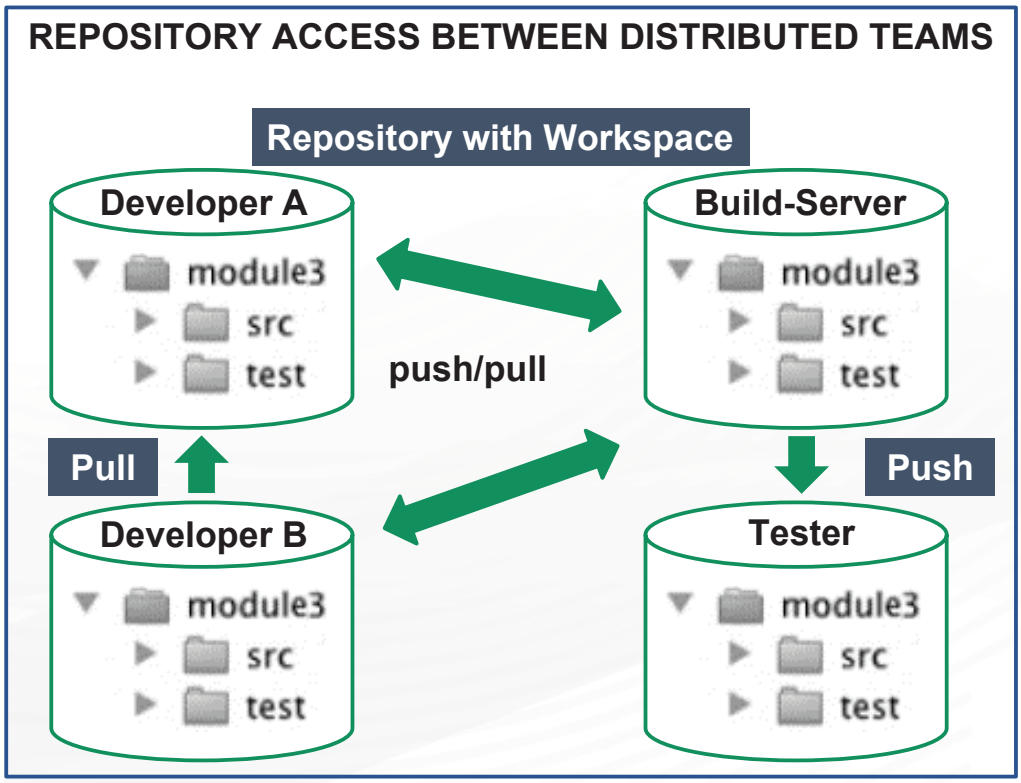
- **Reduces Risk:** If the code is deployed more frequently, it will reduce the risks than deploying code less frequently. So that bugs can be detected very quickly and easier to fix in small portions of code rather than large chunks of code. So that the feedback mechanism is very efficient.

- **Better Communication:** With Continuous Integration process in place it is easy to deploy the code regularly. As soon as code is checked in hooks will run the build, integration tests and achieve good collaboration between teams. So that all the teams working on a product will be on same page.

- **Faster iterations**: Since the code is released regularly, the difference between application in production and the one in developer environment will be negligible. Because every code change will be built automatically and entire team can track all the changes. If the build fails all the developers will be notified immediately and the issue can be fixed in less time frame. This enables to deploy and test the applications automatically and the end users can see the changes as soon as possible.

- **Faster feedback on business decisions**: Having a proper CI in place is not only beneficial to software developers, but for the management team as well. Since the code is pushed more often all parties can gather continuous feedback and gain insights much faster and it can help to check if the product is heading in a good direction. Metrics like Unit tests, build reports, code smell etc can also reflect the progress of project more frequently thus enabling faster technological and business decisions.

1.5.1 How CI Benefits Distributed Teams?



Facilitator Notes:

Explain the participants how Continuous Integration benefits for distributed teams

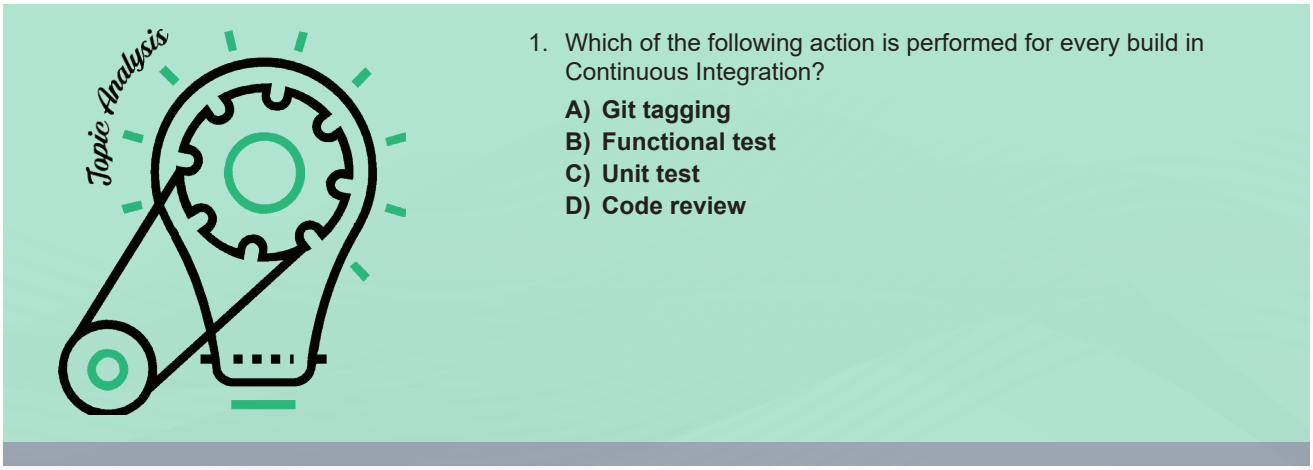
The above image represents the repositories that is shared with distributed teams and build server.

WIth Continuous Integration the code can be merged frequently by developers within the same team. CI is the practice of keeping changes small and frequent. Multiple developers can work on the same feature. While merging the code into Version control system, developers need to take care of the code conflicts. In the above picture two developers are working on Module 3. They need to Sync the workspaces frequently in order to update the changes of other developers from the distributed repository.

With CI, small, frequent and simple changes are merged into the repository.

- Keeping changes small means the impact of the change should also be small, the risks reduced, and the opportunities for change increased. It sounds overly simplistic but it is also very true. The following diagram gives some indications into how this could look:
- Developing in small and frequent changes can also help with reducing complexity and maintaining quality. Small changes also make merging of code easier to manage as you only have a small amount of changed code to bring together; this can also help you if you are moving towards the engineering utopia of developing from trunk all of the time.

What did You Grasp?



1. Which of the following action is performed for every build in Continuous Integration?

   A) Git tagging
   B) Functional test
   C) Unit test
   D) Code review

2. Code commit is successful only if _____ is passed.
   - **A) Functional test**
   - **B) Integration test**
   - **C) Regression test**
   - **D) Code review**

Facilitator Notes:

Answer :

1. c) Unit test

For every new build the first step is to run Unit tests. The build will run other tests after the Unit tests are executed. Developers are responsible for Unit test failures.

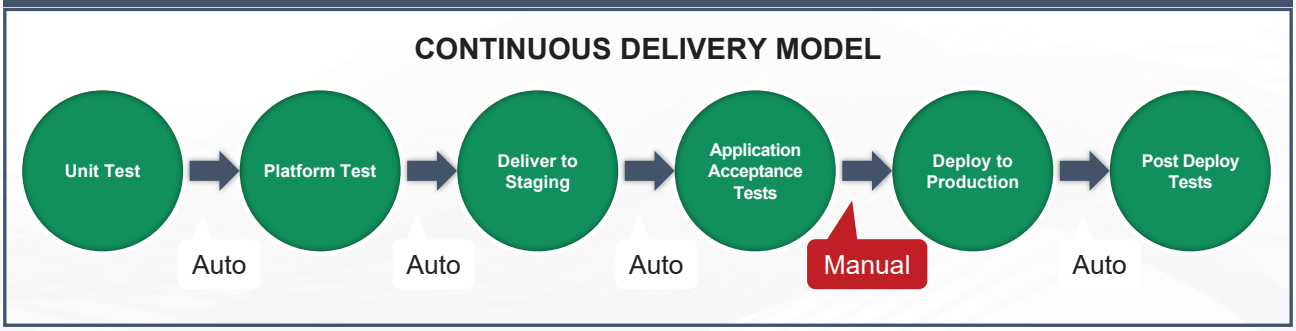
2. b) Integration test

For every new code commit, Integration tests are performed automatically in test environment. The build is finally released to other teams only if all the integration tests are passed. It is developers responsibility to monitor the results of Integration tests for every build.

2.1 CD

CD takes the idea of continuous integration one step further and advances automation along the pipeline.

With CD, code is not only integrated with changes on a regular basis, but as a second stage it is also deployed to a given environment, such as staging or production.



Facilitator Notes:

Give a brief about CD.

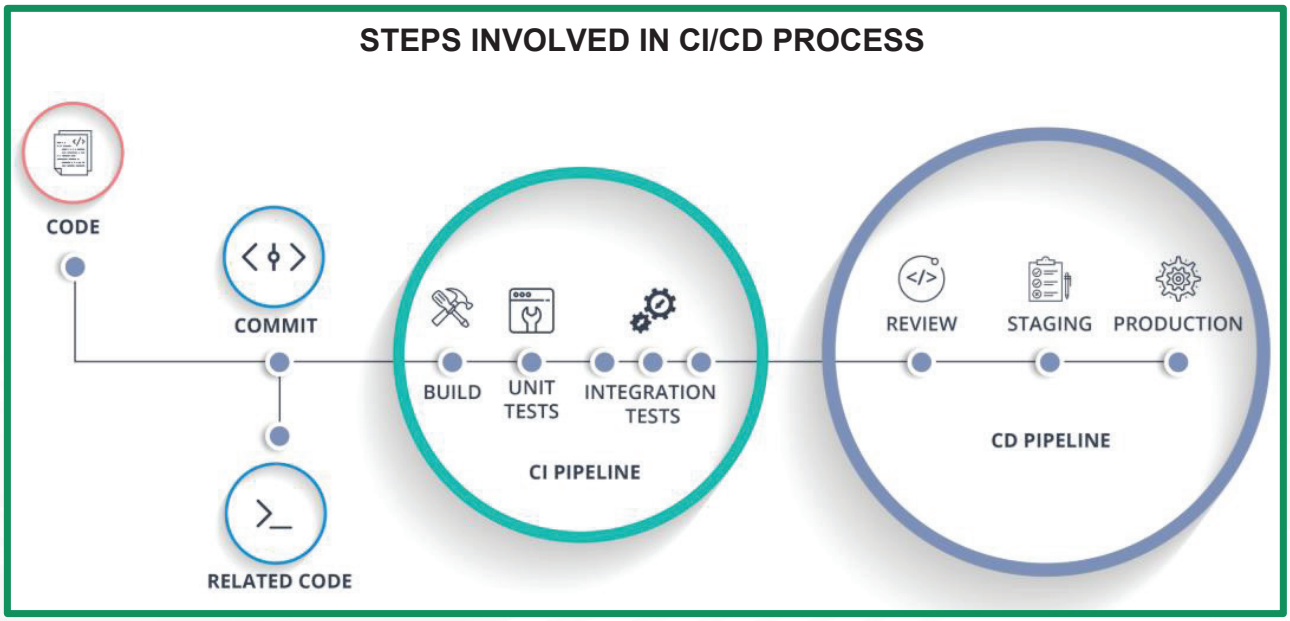
CD takes the idea of continuous integration one step further and advances automation along the pipeline. With CD, code is not only integrated with changes on a regular basis, but as a second stage it is also deployed to a given environment, such as staging or production. Some teams use the term ‘CD’ interchangeably with the similar DevOps term ‘continuous deployment’. The difference between the two terms is subtle but important. CD means that the updated code base is available to move on to the next development stage, whether that be staging, user review, or production.

The adoption of CD is a journey, and it is one that should not be undertaken lightly. The organizational impact and political implications of CD practices are almost as significant as the benefits that they deliver. Having this knowledge from the outset allows individuals to set appropriate expectations and maintain focus on a well-defined objective.

Jez Humble, a Silicon Valley technologist and consultant, is widely credited with solidifying the goals and principles for CD in his 2010 book, *CD: Reliable Software Releases through Build, Test, and Deployment Automation* (Humble, Farley 2010). The principal objective of CD, Humble writes, is to deliver high-quality, valuable software in an efficient, fast, and reliable

manner. This can be traced back to the objectives of the Agile Manifesto published in 2000, which stated: “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”

2.2 Steps involved in CI/CD



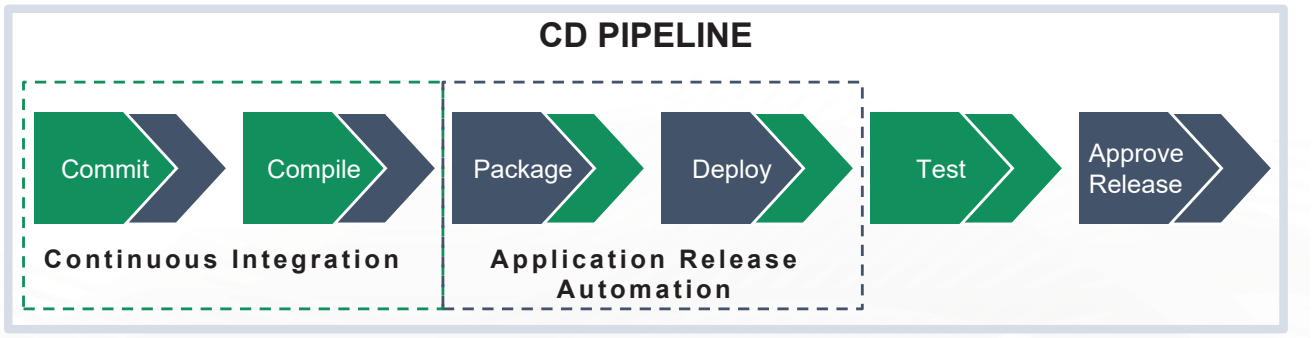
Facilitator Notes:

Give the participants a brief about the steps involved in CI/CD.

Go through the picture in the above slide. As part of Continuous Integration - Build, Unit tests and Integrations tests are executed through a pipeline.

As part of CD pipeline the build is reviewed, deployed to staging and production environments.

2.3 CD Pipeline



Facilitator Notes:

Give the participants a brief about the CD pipeline.

CD is a mechanism designed to increase both agility and velocity through the deployment process. Its ultimate success depends on breaking down the dividing lines between development, QA, and operations, so that the entire software lifecycle runs in a continuous and business-aligned fashion.

A CD pipeline involves a number of stages before the product is deployed into production. Between each of these stages, code typically goes through many different suites of automated tests before the new feature is released into production.

What is a deployment Pipeline:

A deployment pipeline is an end-to-end automation of build,test, deploy and releases. A deployment pipeline is nothing but releasing the software to the users that is written by developers.

The entire deployment flow can be scripted through a pipeline. For example in Jenkins we can define pipeline through Jenkinsfile which is based on Groovy DSL. Continuous Delivery pipeline stages can be broken down as follows:

- Source Code Control
- Build Automation

- Unit Test Automation
- Deployment Automation
- Monitoring

Example of a Jenkinsfile:

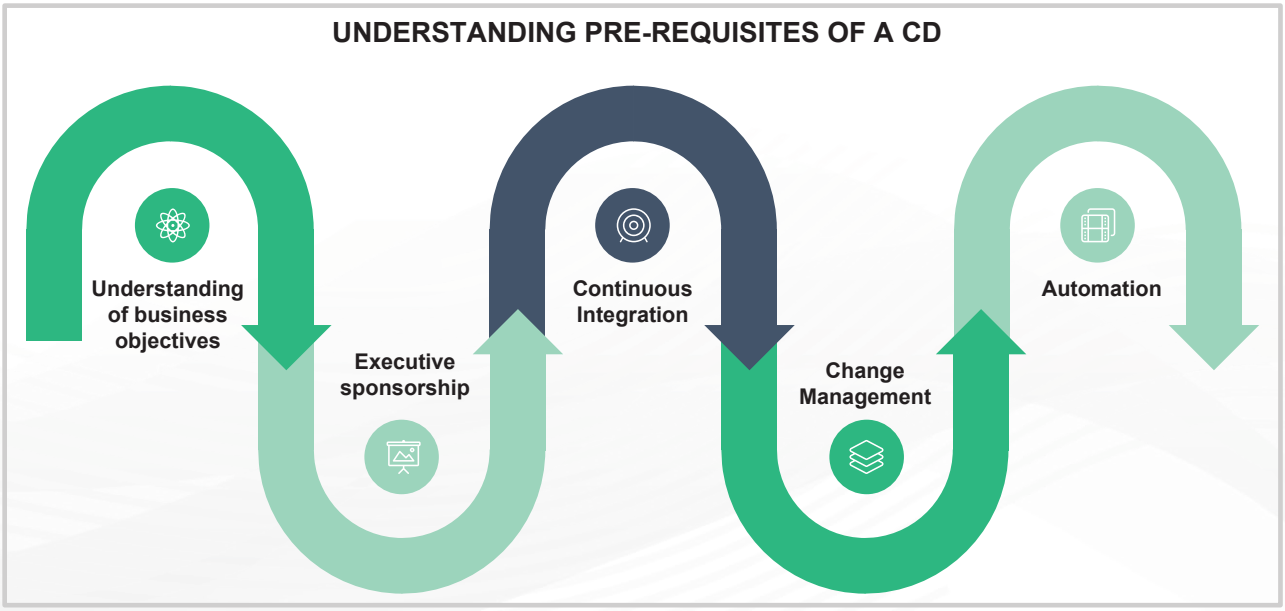
Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Build step..'
            }
        }
        stage('Test') {
            steps {
                echo 'Test stage.'
            }
        }
        stage('Deploy') {
            steps {
```



```
            echo 'Deploy stage''
        }
    }
  }
}
```

2.4 Prerequisites for CD



Facilitator Notes:

Explain the participants, the important requirements for adopting CD.

All organizations that create business or consumer-facing applications benefit from the adoption of practices, but there are a number of prerequisites that need to be implemented in order to be successful.

Understanding of business objectives: The digitization of products and services, and of the channels through which they are marketed and sold, makes it essential that IT understand what really matters to the business. What are the goals and objectives of the business? What is the plan to deliver against those goals? And how do development and IT operations fit within the context of delivery?

Many IT organizations have skilled individuals whose sole responsibility are to act as a liaison between engineering and the business. These individuals work with the business stakeholders to get their feedback and input. But why simply include a limited subset of resources in this process? The smaller the number of individuals involved, the more likely that a level of bias – intentional or not – will be included in the transformation of the business requirements. When members of the application development and triage teams, operations, production support, and incident and problem management teams are included in the process, a much greater level of understanding and clarity of business requirements can be achieved.

This does not mean that IT should spend all of its time in concept or planning meetings, but that an increased level of business understanding will lead to a better final deliverable. A single product manager who “knows everything the customer wants” is not going to lead to a successful product. With greater involvement, IT can understand the vision, direction, and objectives of the business and implement a culture that supports rapid and successful delivery of high-quality, objective meeting applications. This integrated approach will often lead to more complete and successful implementations, as the longer-term objectives are included in design, definition, and implementation.

Executive Sponsorship: DevOps, like any other significant initiative, thrives with an executive management champion who has the authority to change business processes. The sponsor does not need to be the CIO, but with the current level of hype surrounding DevOps, the CIO may be more receptive to it. Business pressures alone may mean that the executive sponsor is ready to put an initiative in place to speed up the velocity of releases and enable more frequent application deliveries – and may use the term CD or DevOps to describe it.

Continuous Integration: CI is essentially a development practice, but because it also involves a level of automated software testing (at least unit testing), it begins to stray into the realm of QA. Agile approaches commonly integrate development and testing resources, so teams should not view incorporating unit testing into the development cycle as a new concept. As CI models push to deliver build artifacts further to the right – towards operations teams – the impact upon QA teams becomes more noticeable. Advocates for CD tend to agree that there

is little point in attempting to promote DevOps in an organization that has not yet adopted CI and made at least this level of commitment to process automation.

Change Management: Many enterprises are committed to ITSM/ITIL processes for organizing IT delivery and tracking issues. Although adoption of ITIL does mean implementation of formal release management or service transition processes, effective software deployment is reliant upon well defined operational change management practices, something towards which ITIL provides good guidance. ITIL practices, and DevOps practices in general, often overlook the value of tracking development-level changes against operational change. Having both a well-defined, ITIL-compliant operation process and a well-defined, agile-based development process, with little to no interaction between them, immediately causes a disconnect between Dev and Ops.

At the enterprise level, there are frequently multiple tools in place; remember that the best-of-breed tools for Ops change and the tool of choice for Dev change may be two completely isolated products with little to no process overlap. A simple way to break down the walls between Dev and Ops is to develop a common change management process for the entire deployment pipeline. It should be noted that change management is a risk-reduction strategy, and many CD advocates view a formal change management process as unnecessary in a successful deployment pipeline. The question must be asked for your organization: Is it enough to get deployments into your environment quickly, or do you also want the ability to validate and explain which specific Dev and Ops changes were delivered as part of the deployment?

Automation: Automating code and configuration deployments with a single set of deployment processes across all environments, using parameterization, will help ensure that environment-specific constraints and requirements are met. Do not deploy code in different ways simply because you are moving from test to production. Repeatability and consistency are the key to successful, accurate, and speedy delivery. Ensure that all artifacts are deployed from the same source location or artifact repository.

Deploying in the same way across all environments is efficient in both time and cost. Using the same process ensures more consistent testing, and any environment-specific issues are far easier to identify. The more automated this process is, the more repeatable and reliable it will be. This translates into faster deployments, reduced outage time, and more confidence from the business that change is being delivered in a reliable manner

2.5 CD Checklist

Following are the important steps to be followed for achieving CD

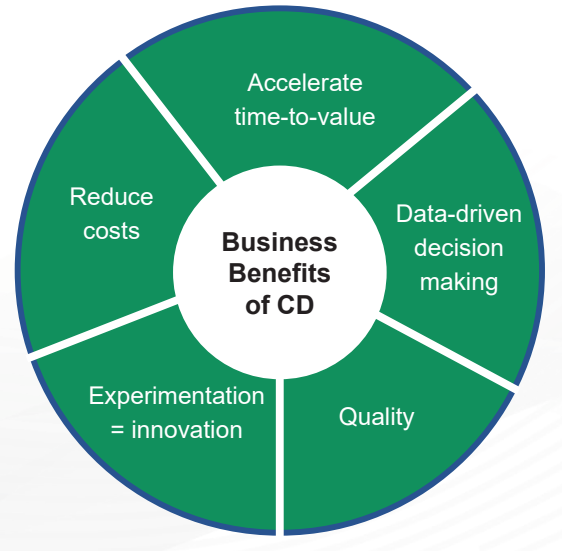
- ☑ Before submitting changes, check to see if a build is currently in the 'Successful' status. If not, you should assist in fixing a build before submitting new code.
- ☑ If the status is currently 'Successful', you should rebase your personal workspace to this configuration.
- ☑ Build and test locally to ensure the update doesn't break functionality.
- ☑ If Successful, check in new code.
- ☑ Allow CI to complete with new changes.
- ☑ If build fails, stop and fix on your machine. Return to step 3.
- ☑ If build passes, continue to work on the next item.

Facilitator Notes:

Explain the participants the important steps to be followed for achieving CD.

The steps for creating a simple and important checklist for CD is given on the slide. The steps given above are to be considered as a general outline. Different tools and solutions might not follow the exact steps as given above. However, it's important to be aware of these steps.

2.6 Business Benefits of CD



Facilitator Notes:

Explain the participants about the business benefits of CD.

- A big business that has mapped out its value stream and has complex investments and obligations across a large organization will find that CD helps accelerate time-to-value.
- Deploy, measure, adjust. You can still push large-scale releases, but your processes will be better suited to continuous data gathering. That will shorten the feedback loop with your customers. It sharpens your ability to respond, plan your next move and keep ahead of the competition.
- Behaving like you're releasing continuously forces you to raise your quality bar and fully automated test practices. Better quality means happier customers, lower costs, fewer fire-drills and less unplanned work.
- Developers and lines of business are free to try new things cheaply, unlocking innovative ideas that have been penned-in behind long, high-investment release cycles.
- Big releases have big costs and big consequences when things go wrong. Keeping deliverables in a release-ready state drives the cost of delivery downward.

What did You Grasp?

1. Identify the correct order of Continuous Integration.
   - **A) Commit code -> Run unit tests -> Deploy application -> Run Integration tests**
   - **B) Commit code -> Run unit tests -> Run Integration tests -> Deploy application**
   - **C) Commit code -> Deploy application -> Run unit tests -> Run Integration tests**
   - **D) Commit code -> Run Integration tests ->Run unit tests -> Deploy application**

2. State if below statement is True or False

   If a build fails, continue checking in new code and fix the problematic code parallelly.

   A) True
   B) False

Facilitator Notes:

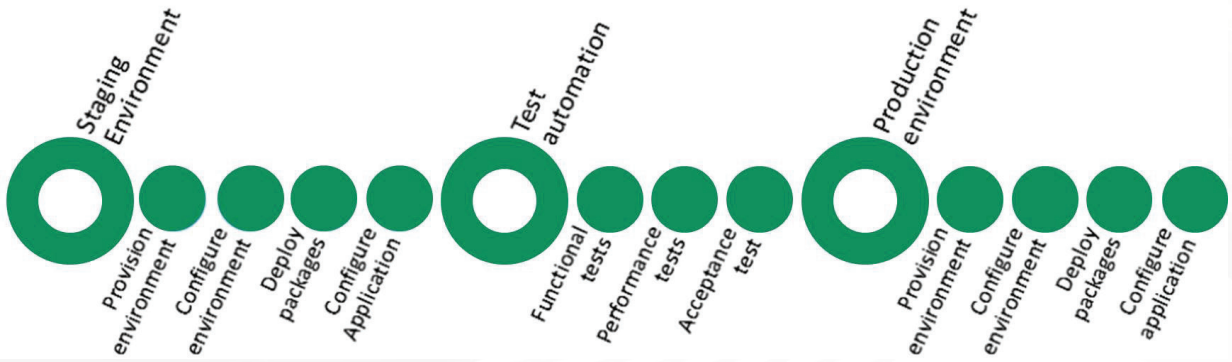
Answer:

1. b) Commit code -> Run unit tests -> Run Integration tests -> Deploy application

2. b) False

3.1 Continuous Deployment

- Continuous deployment is a natural progression of CD.
- Once all software changes pass automated and functional testing, as well as the build stages, they are automatically deployed to production without human intervention.
- Continuously deploying code without human intervention is the goal that most companies hope to achieve.



Facilitator Notes:

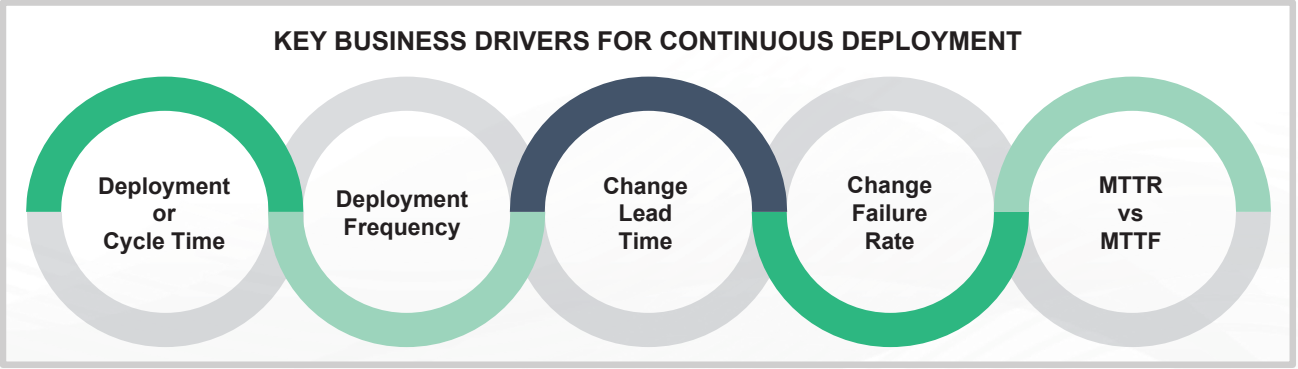
Give the participants an overview of the continuous deployment process.

- Continuous deployment is generally integrated with continuous integration. When continuous integration has done, its work by generating the final deployable packages, continuous deployment kicks in and starts its own pipeline. This pipeline is called the **release pipeline**. Release pipeline consists of multiple environments with each environment consisting of tasks responsible for provision of environment, configuration of environment, deploying applications, configuring applications, executing operational validation on environments and testing the application on multiple environments. We will look at release pipeline in greater details in the next chapter and also the chapter on continuous deployment.

- Employing continuous deployment provides immense benefits. There is a high level of confidence in the overall deployment process which helps in faster and risk-free releases on production. The chances of anything going wrong comes down drastically. The team would have lower stress levels and rollback to the previous working environment is possible if there are issues in the current release:

- Although every system demands its own configuration of release pipeline, a minimal sample of continuous deployment is shown in the preceding figure. It is important to note that generally provisioning and configuring of multiple environments is part of the release pipeline and approvals should be sought before moving to the next environment. The approval process might be manual or automated depending on the maturity of the organization.

3.2 Business Drivers for Continuous Deployment



Facilitator Notes:

Explain the participants about the key business drivers for continuous deployment.

Following are the key business drivers for continuous deployment:

- **Deployment or Cycle Time:** A term borrowed from manufacturing, 'cycle time' is the time it takes for a feature to go from build to production. By measuring the phases of your development process, for example, coding, review process, test and release to production, the average cycle time is obtained. This metric provides insight into the bottlenecks in your process and the overall speed or time to deployment.

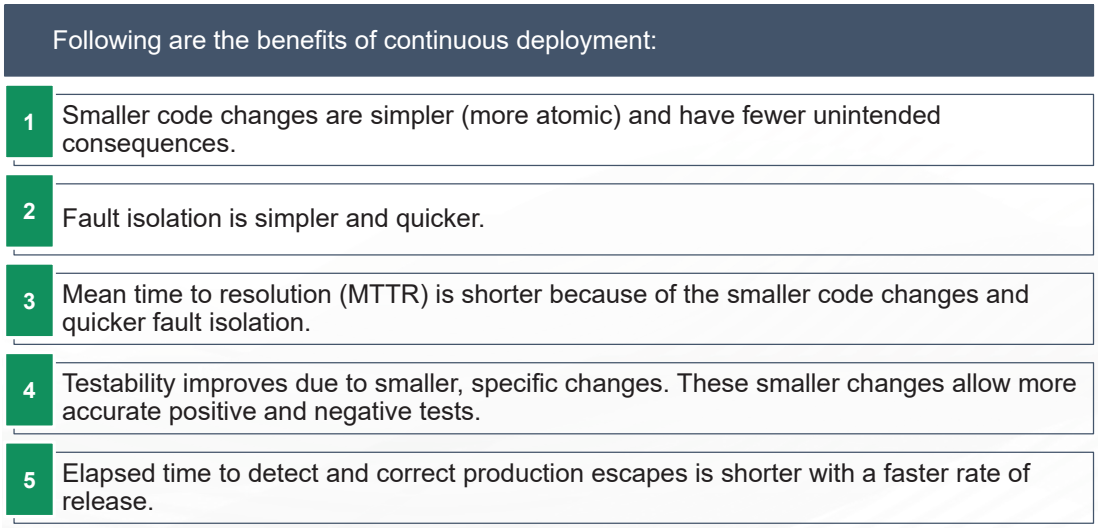
- **Deployment Frequency:** Not only do more frequent deployments give you an opportunity to improve upon your software, but by measuring frequency, it allows you to analyze any bottlenecks you may find during your automation journey. The general rule is that more frequent smaller releases reduce the risk of defects, and maybe more importantly, increases the ability to fix them when found. This metric is an overall measure of your team's efficiency and cohesiveness.

- **Change Lead Time:** Measures the start of development to deployment. Like deployment frequency, this metric is also an indicator of your entire development process and how well your team works together. If the lead time is too long, it may suggest bottlenecks in your process or that your code and development systems are overly complicated.

- **Change Failure Rate:** This metric focuses on the number of times your deployment was successful versus the number of times the deployment failed. This metric is one that should decrease over time. It is generally a useful measure of the success of your DevOps process.

- **MTTR vs MTTF:** The Mean Time to Recovery (MTTR) is the amount of time it takes for your team to recover from a failure; either a critical bug or a complete system failure. The Mean Time to Failure (MTTF) on the other hand measures the amount of time between fixes or outages. Both metrics are a reflection of your team's ability to respond and fix problems. Generally, you will want to see a downward trend for these metrics.

3.3 Benefits of Continuous Deployment



Facilitator Notes:

Explain about the benefits of continuous deployment.

Other than the benefits given on the slide, few more benefits are:

- The backlog of non-critical defects are lower because defects are often fixed before other feature pressures arise.
- The product improves rapidly through fast feature introduction and fast turnaround on feature changes.
- Upgrades introduce smaller units of change and are less disruptive.
- CI-CD product feature velocity is high. The high velocity improves the time spent investigating and patching defects.
- Feature toggles and blue-green deploys enable seamless, targeted introduction of new production features.
- You can introduce critical changes during non-critical (regional) hours. This non-critical hour change introduction limits the potential impact of a deployment problem.

- Release cycles are shorter with targeted release and this blocks fewer features that aren't ready for release.
- End-user involvement and feedback during continuous development leads to usability improvements. You can add new requirements based on customer's needs on a daily basis.

What did You Grasp?

1. Which of the following options are elements of release pipeline?
   A) Continuous Integration + Continuous Delivery
   B) Continuous Delivery + Continuous Deployment
   C) Continuous Integration + Continuous Deployment
   D) Continuous Testing + Continuous Deployment

2. Release cycles are shorted due to one of the following:
   A) Continuous Integration
   B) Continuous Delivery
   C) Continuous Deployment
   D) Continuous Testing

Facilitator Notes:

Answer :

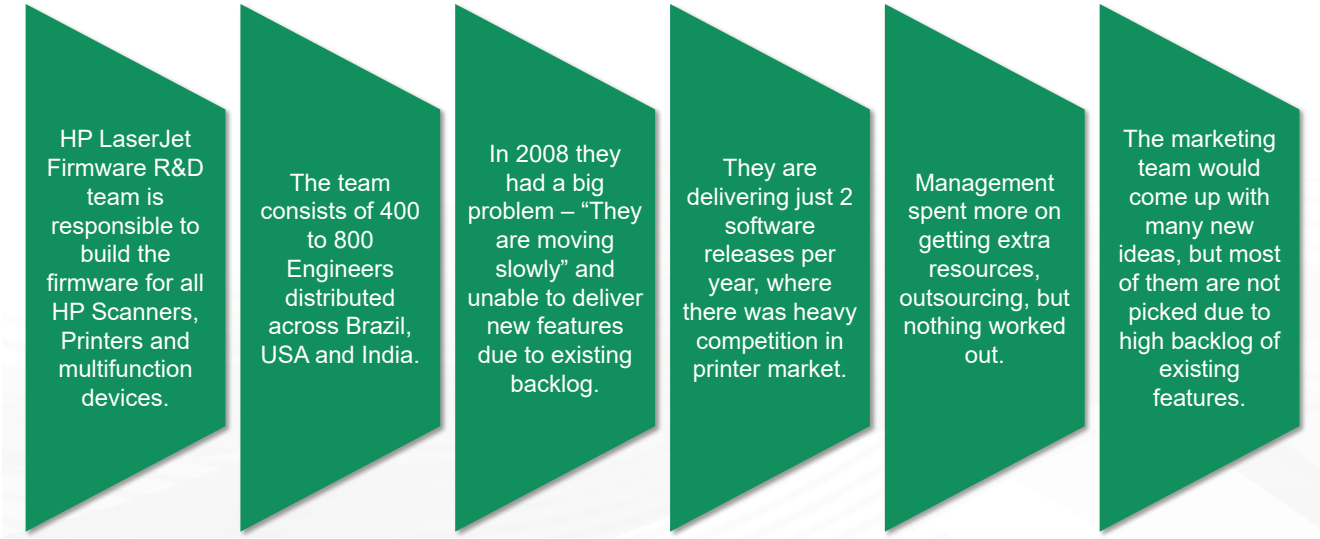
1. c Continuous Integration + Continuous Deployment

Once the latest code is delivered as part of continuous integration it can be deployed using continuous deployment method.

2. c) Continuous Deployment

Due to continuous deployment new features are deployed at rapid speed. It reduces the bugs and increases the quality of the product. So the difference between production and staging environment is very minimal.

4 CD – The HP Laserjet Case Study



Facilitator Notes:

Discuss Case study of HP Laserjet.

Let's look at the main points of case study:

- The problem was that the team was facing tremendous difficulty in keeping up the demand for new and innovative features in spite of having heavy competition in Printer market. The team maintains 10MM+ lines of code.
- Once a developer check-in the code, it would take 1 week to determine if the integration is successful or not. The build distribution to other teams is not automated. It would take another 1 day for the other team to get new build and run acceptance tests.
- And it takes an additional 6 weeks to run full integration testing on the provided build.

The Leadership team decided to transform their existing build and release infrastructure to DevOPS model through Agile transformation.

- The HP Laserjet team started using Continuous integration/CD
- Reduce the planning time spent on new features
- Significant investment in test automation
- Create a virtual platform that simulates all the hardware components to run tests

Facilitator Notes:

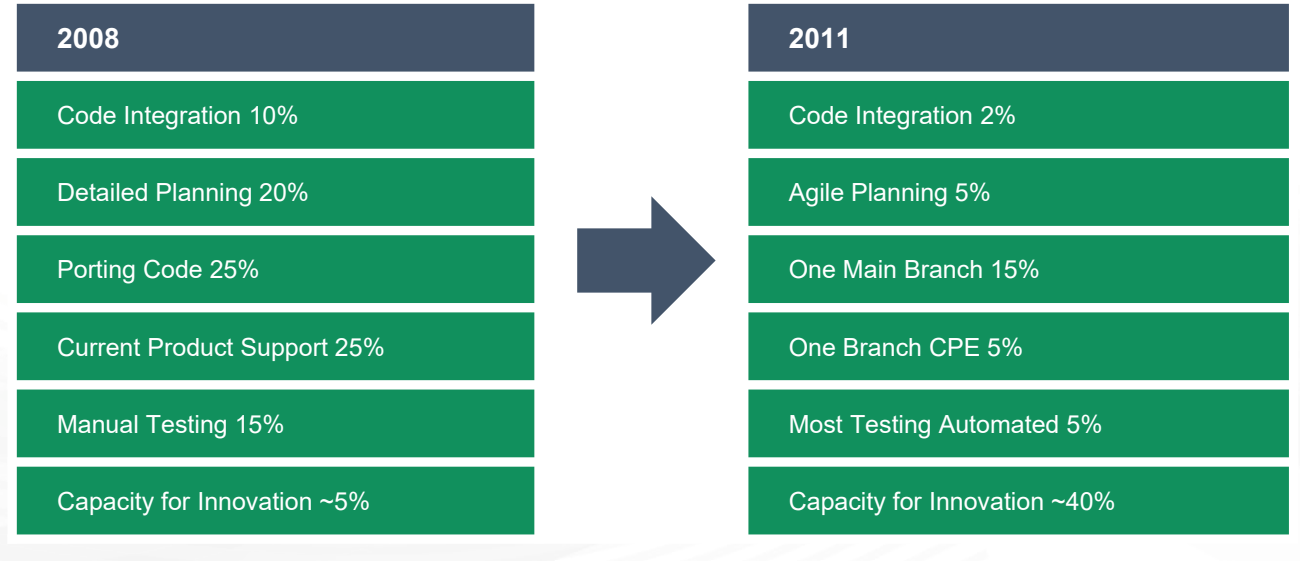
Discuss about the measures LaserJet team had taken to overcome the problem.

This Project was led by Gary Gruver, who decided to adopt the DevOPS methodologies to speed up their release cycles, include more features and to reduce the Test cycle.

The targets set by the management team are to improve the team productivity by a factor of 10, so as to reduce the operations costs and get firmware off the critical path. They had three high-level goals:

- Creation of a single platform to support all devices
- Migrate all the teams on to a common codebase

- Increase quality of the builds prior to release
- Reduce the amount of time spent on planning
- Automate the testing for every build, don't release to other teams if the build fails



Facilitator Notes:

Discuss about the transformation benefits of HP Laserjet to CI/CD model.

After transforming to CI/CD model Laserjet team productivity increased enormously:

- **Build:** Before vs. DevOPS cycle time: 1 week per build -> 3 hours (10-15 builds per day)
- **Commits:** 1 commit per day -> 100 commits/day
- **Testing:** Regression test cycle duration: 6 weeks -> 24 hours

Business benefits :

- Overall development costs were reduced by ~40%.
- Programs under development increased by ~140%.
- Development costs per program went down 78%.
- Resources driving innovation increased eightfold.

In a nutshell, we learnt

1. Overview of CI
2. Practices associated with CI and the working mechanism
3. Benefits of CI
4. Overview of CDand the CD pipeline
5. Prerequisites for CD and the business benefits of CD
6. Continuous deployment and the business drivers of continuous deployment
7. Benefits of continuous deployment
8. Case study of The HP LaserJet transformation to CI/CD model

Facilitator Notes:

Share the module summary with the audience.

Ask the participants if they have any questions. They can ask their queries by raising their hands.

