Unit 7&8

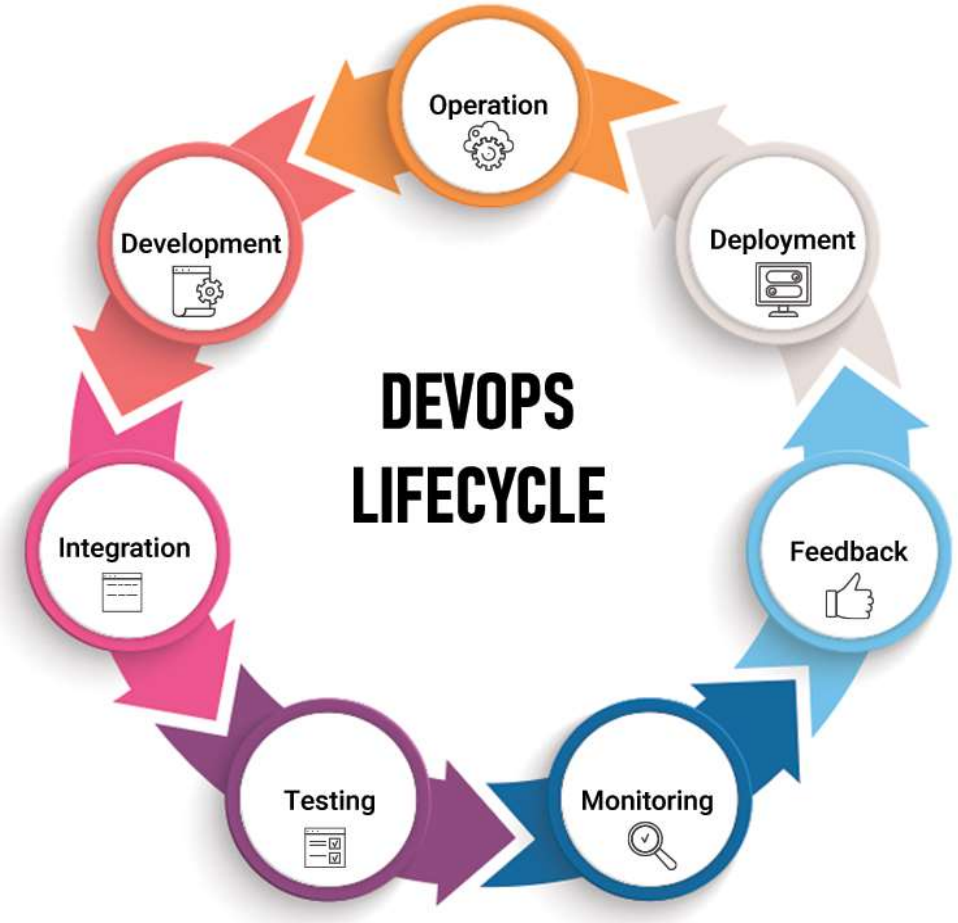
ITPM CSBS Sem VIII

Devops & its components

- **DevOps lifecycle is a combination of different phases of continuous software development, integration, testing, deployment, and monitoring. A competent DevOps lifecycle is necessary to leverage the full benefits of the DevOps methodology.**
- The DevOps approach embraces continuous innovation, agility, and scalability to build, test, consume, and evolve software products. It promotes a culture of experimentation, feedback, and constant learning to reinvent products, services, and processes. However, to implement DevOps, a proper understanding of different phases of the DevOps lifecycle is crucial.

Devops & its components

- **DevOps Lifecycle: Key Components**



TOOLBOX™

Devops & its components

- 1. **Continuous development**
- **2. Continuous integration**
- **3. Continuous testing**
- **4. Continuous deployment**
- **5. Continuous monitoring**
- **6. Continuous feedback**
- **7. Continuous operations**

Containerization using Docker

- **Containerization is a technology that allows a developer to package an application and its dependencies into a single container.**
- To give you an analogy, containerization is kind of like packing all the stuff you need for a road trip into a single suitcase. You can put all your clothes, personal care items, and other essentials into the suitcase. Then you just grab it and go. It doesn't matter where you're going or what kind of car you're taking. As long as you have your suitcase, you have everything you need.

Analogous wrt putting all the all the application related packages in one container

Containerization using Docker

- Benefits:
- **Portability**
- **Isolation**
- **Resource efficiency**
- **Easy to package, ship, and deploy**
- **Ease of scaling**
- **Enhanced security**

Containerization using Docker

- **Docker is a popular containerization platform that allows developers to easily build, deploy, and run applications using containers.** Note that Docker is both the name of the company and the name of the technology.
- he Docker Ecosystem
- One of the key strengths of Docker is the ecosystem of tools and resources that have been built up around it. There are a wide variety of tools available that can be used to build, deploy, and manage Docker containers. Some of the key tools include:
- **Docker Engine:** The core component of the Docker platform, responsible for building and running Docker containers.
- **Docker Hub:** A cloud-based registry (a storage location) for Docker images, allowing developers to share and discover Docker images.
- **Docker Compose:** A tool for building and running applications that are *composed* of multiple containers.
- **Docker Swarm:** A tool for orchestrating Docker containers across a cluster of servers.

Managing Source Code and Automating Builds

- **Build automation** is the process of automating the retrieval of source code, compiling it into binary code, executing automated tests, and publishing it into a shared, centralized repository. Build automation is critical to successful **DevOps** processes.

Managing Source Code and Automating Builds

- **5 Benefits of Build Automation**
- There are five main benefits of build automation.
- **Increases Productivity**
- Build automation ensures fast feedback. This means your developers increase productivity. They'll spend less time dealing with tools and processes — and more time delivering value.
- **Accelerates Delivery**
- Build automation helps you accelerate delivery. That's because it eliminates redundant tasks and ensures you find issues faster, so you can release faster.
- **Improves Quality**
- Build automation helps your team move faster. That means you'll be able to find issues faster and resolve them to improve the overall quality of your product — and avoid bad builds.
- **Maintains a Complete History**
- Build automation maintains a complete history of files and changes. That means you'll be able to track issues back to their source.
- **Saves Time and Money**
- Build automation saves time and money. That's because build automation sets you up for CI/CD, increases productivity, accelerates delivery, and improves quality.

Managing Source Code and Automating Builds

How to Automate the Build Process

- **Write the code.**
- **Commit code to a shared, centralized repository — such as Perforce Helix Code.**
- **Scan the code using tools such as static analysis.**
- **Start a code review.**
- **Compile code and files.**
- **Run automated testing.**
- **Notify contributors to resolve issues.**

Managing Source Code and Automating Builds

- **Automated Build Tools**
- Automated build tools will help you ensure build automation.
- **Build runners** in particular, are critical for automation. These tools help you automate the process of building, testing, and deploying code.
- Example : Jenkins

Automated Testing & Test Driven Development

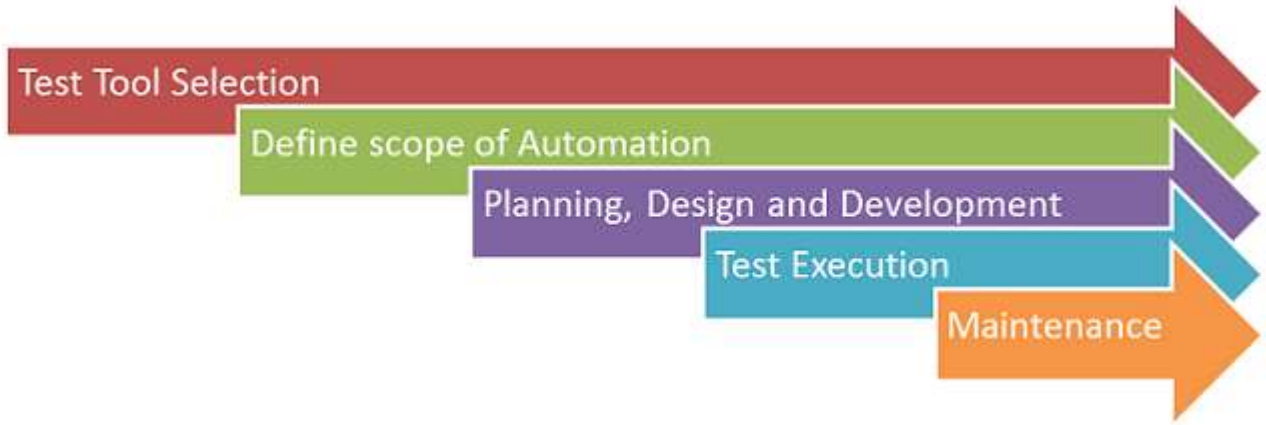
- **What is Automation Testing?**
- **Automation Testing** is a software testing technique that performs using special automated testing software tools to execute a test case suite. On the contrary, Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps.
- The automation testing software can also enter test data into the System Under Test, compare expected and actual results and generate detailed test reports. Software Test Automation demands considerable investments of money and resources.

Automated Testing & Test Driven Development

- **Test Automation** is the best way to increase the effectiveness, test coverage, and execution speed in software testing. Automated software testing is important due to the following reasons:
- Manual Testing of all workflows, all fields, all negative scenarios is time and money consuming
- It is difficult to test for multilingual sites manually
- Test Automation in software testing does not require Human intervention. You can run automated test unattended (overnight)
- Test Automation increases the speed of test execution
- Automation helps increase Test Coverage
- Manual Testing can become boring and hence error-prone.

Automated Testing & Test Driven Development

- **Automated Testing Process:**

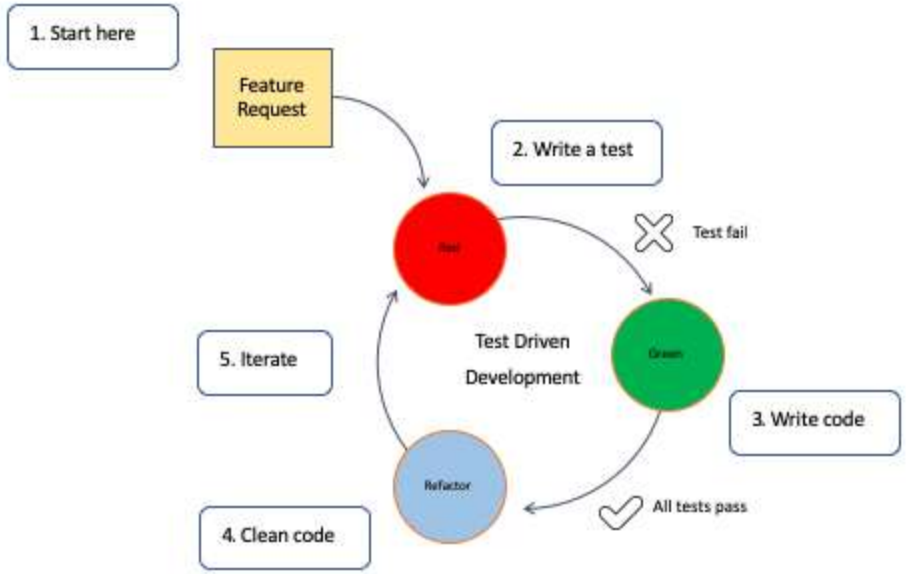


Automated Testing & Test Driven Development

- What is test-driven development?
- Test-driven development reverses traditional development and testing. So, instead of writing your code first and then retroactively fitting a test to validate the piece of code you just wrote, test-driven development dictates that you write the test first and then implement code changes until your code passes the test you already wrote.

Automated Testing & Test Driven Development

- Steps of test-driven development



Continuous integration in DevOps

- A developer develops or writes some form of code, often called as patches representing a change to the project's codebase [for example, a fix or bug].
- Merges the change to the centralized repository of that project like git, SVN, and bitbucket vary with the project.
- If the codes present in the centralized repos are needed later or while composing the application, i.e. building can be referred later at any point in time. Then these builds are deployed and can be manipulated or called at any point in time, often termed as packages or artifacts.

Continuous integration in DevOps

- Continuous Delivery and Continuous Deployment are part of Continuous Integration, which has helped to come automation so far by dividing and distributing all commits and patches into all new versions of different software.
- Continuous Delivery means the simultaneous commits being made into the repository, and then it is the responsibility of the human to decide whether to take it further for deployment or not, which means human effort is being involved somehow because of which companies don’t prefer and thinks twice before deployment.
- On the other hand, many companies put more emphasis on continuous deployment in a sense they will directly put the commits and changes as it is a direct and simple approach. But again, it involves a lot of risks and can hamper or create bugs when a product goes for production. Thus, some new approaches must be introduced to mitigate the bug risk and make the Continuous deployment process with continuous delivery and continuous integration more enhanced and powerful.

Continuous integration in DevOps

- How does continuous integration work?
- Continuous integration is generally achieved and completed in six steps. These steps are:
- Manual process identification
- Frequency and duration definition
- Sub-process selection for automation
- Automation script creation
- Artifact trigger scheduling
- Iterate, improve, and repeat

Continuous integration in DevOps

- Importance of continuous integration in software development
- **Reduction in code changes**
- **Issues isolation**
- **Faster mean time to resolution**
- **Smaller Backlogs**

- Example Jenkins, Azure DevOps

Continuous integration in DevOps

Continuous Integration	Continuous Development
Continuous integration is an automated approach to test each change to the codebase.	Continuous development is an approach to develop software in shorter cycles.
The process of continuous integration refers to the versioning of source code.	Continuous development refers to automated source code implementations.
Continuous integration focuses on automation testing to determine that the software has no errors or bugs.	Continuous development emphasizes the change in all stages of your production pipeline.
Continuous integration is performed immediately after the developer check-in.	In continuous development, developers deploy the code directly to the production stage when it is developed.
The development team sends continuous code merging requests in continuous integration even when the testing process is running.	The development team deploys the codes using an automated process in continuous development.

Configuration Management in DevOps

- Configuration management occurs when a configuration platform is used to automate, monitor, design and manage otherwise manual configuration processes.
- An important function of configuration management is defining the state of each system. By orchestrating these processes with a platform, organizations can ensure consistency across integrated systems and increase efficiency.

Configuration Management in DevOps

- Components:
- **Identification:**
  The process of finding and cataloging system-wide configuration needs.
- **Control:**
  During configuration control, we see the importance of change management at work. It's highly likely that configuration needs will change over time, and configuration control allows this to happen in a controlled way as to not destabilize integrations and existing infrastructure.
- **Audit:**
  Like most audit processes, a configuration audit is a review of the existing systems to ensure that it stands up to compliance regulation and validations.
- Like DevOps, configuration management is spread across both operational and development buckets within an organization. This is by design. There are primary components that go into the comprehensive configuration management required for DevOps:
- Artifact repository
- Source code repository
- Configuration management data architecture

Configuration Management in DevOps

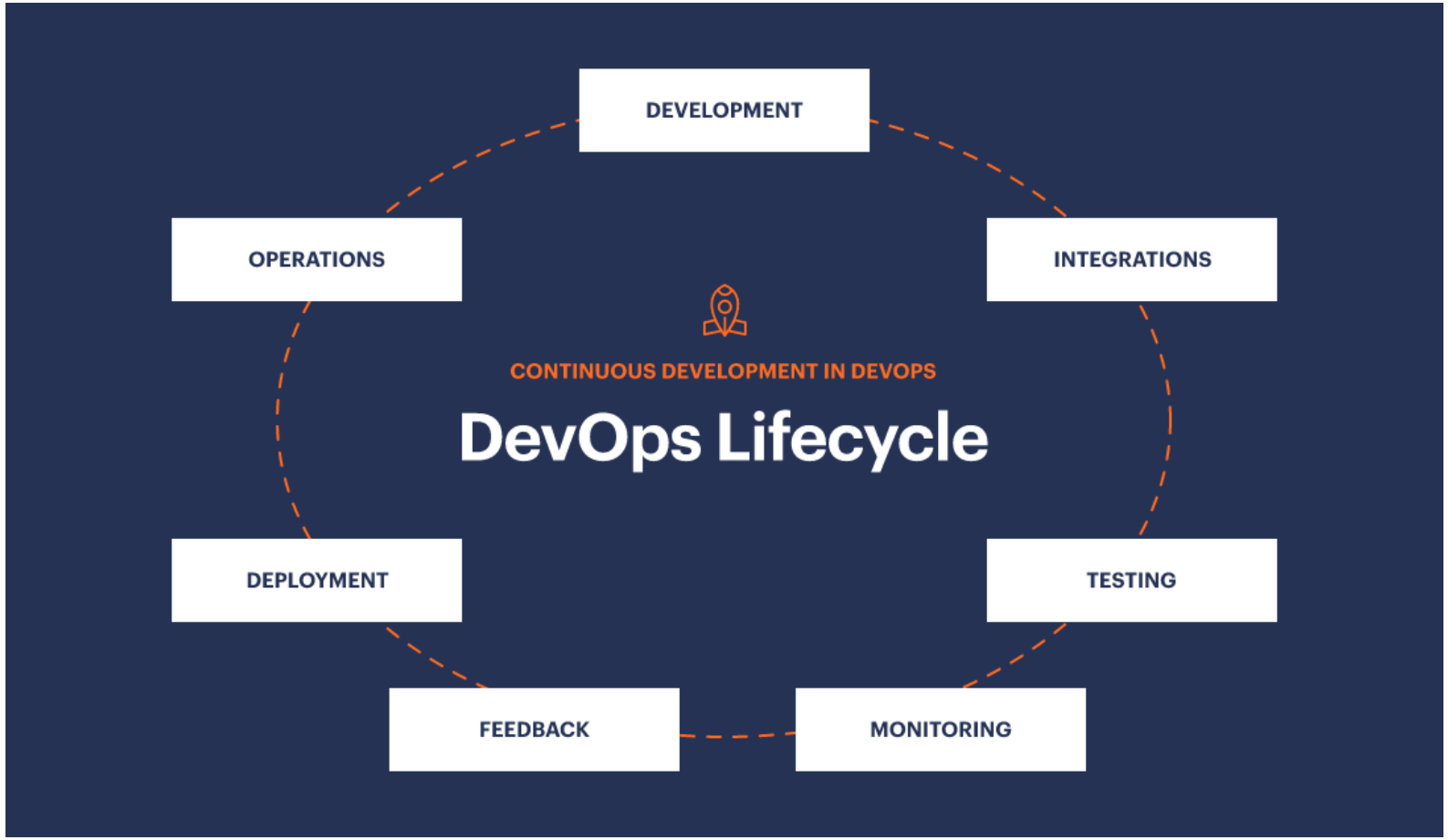
- Outcomes of Properly Managed Configurations:

- **Infrastructure-as-a-Code**
- **Configuration-as-a-Code**

Continuous Development in DevOps

- Various DevOps strategies are referred to as “Continuous Development.” Continuous Development’s sole purpose is to keep track of constant improvements, plan, test, and collect feedback to improve the product. As a result, understanding Continuous Development is crucial for ensuring that all your DevOps processes work correctly.

Continuous Development in DevOps



Continuous Development in DevOps

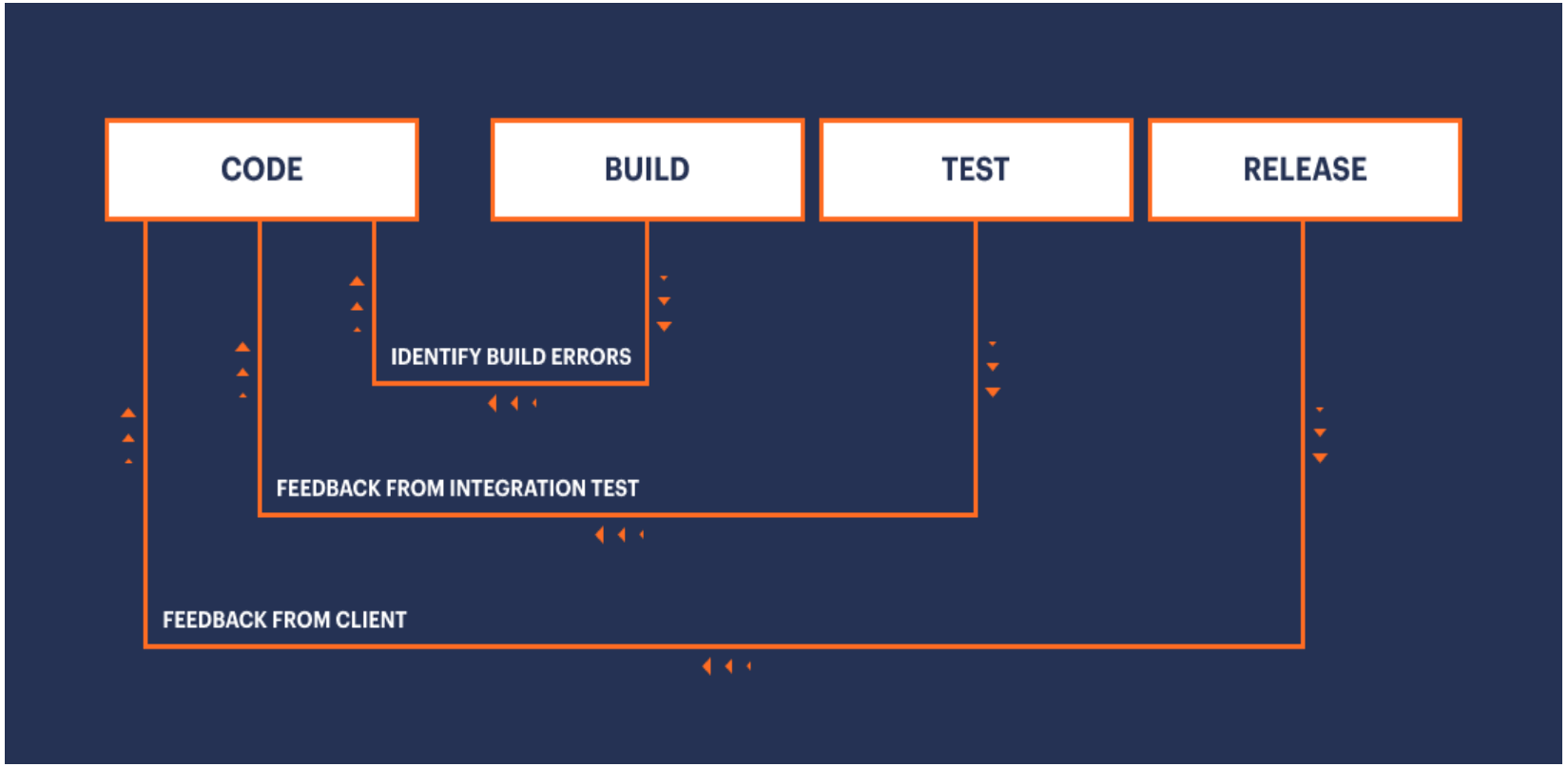
- Benefits of Continuous Development:
- **1. Good quality software**
- **2. Swift transitions**
- **3. Lesser Risks**
- **4. Lesser Resources**
- **5. More Productivity**

Continuous Development in DevOps

- Factors of Continuous Development:
- **Continuous integration**
- **Continuous delivery**
- **Continuous deployment**
- **Continuous testing**

Continuous Development in DevOps

- Process of Continuous Development



Continuous deployment in DevOps

- Continuous deployment vs. continuous delivery
- Continuous delivery is a software development practice where software is built in such a way that it can be released into production at any given time. To accomplish this, a continuous delivery model involves production-like test environments. New builds performed in a continuous delivery solution are automatically deployed into an automatic quality-assurance testing environment that tests for any number of errors and inconsistencies. After the code passes all tests, continuous delivery requires human intervention to approve deployments into production. The deployment itself is then performed by automation.
- Continuous deployment takes automation a step further and removes the need for manual intervention. The tests and developers are considered trustworthy enough that an approval for production release is not required. If the tests pass, the new code is considered to be approved, and the deployment to production just happens.

Continuous deployment in DevOps

- Continuous deployment vs. continuous integration
- In order for automation of deployment processes to work, all the developers working on a project need an efficient way of communicating the changes that take place. Continuous integration makes this possible.
- Typically, when working on the same software development project, developers work off of individual copies of a master branch of code. However, functionality issues and bugs can occur after developers merge their changes onto the main codebase, especially when developers work independently from each other. The longer they work independently, the higher the risk.
- With CI, everyone merges their code changes into a repository at least once per day. As updates occur, automated build tests run to ensure that any changes remain compatible with the master branch. This acts as a fail-safe to catch integration problems as quickly as possible.

Continuous deployment in DevOps

Continuous deployment tools:

- **Version control**
- **Code review**
- **Continuous integration (CI)**
- **Configuration management**
- **Release automation**
- **Infrastructure monitoring**

Automated Monitoring in DevOps

DevOps automation is the practice of automating repetitive and manual DevOps tasks to be carried out without any human interaction. Automation can be applied throughout the DevOps lifecycle, spanning:

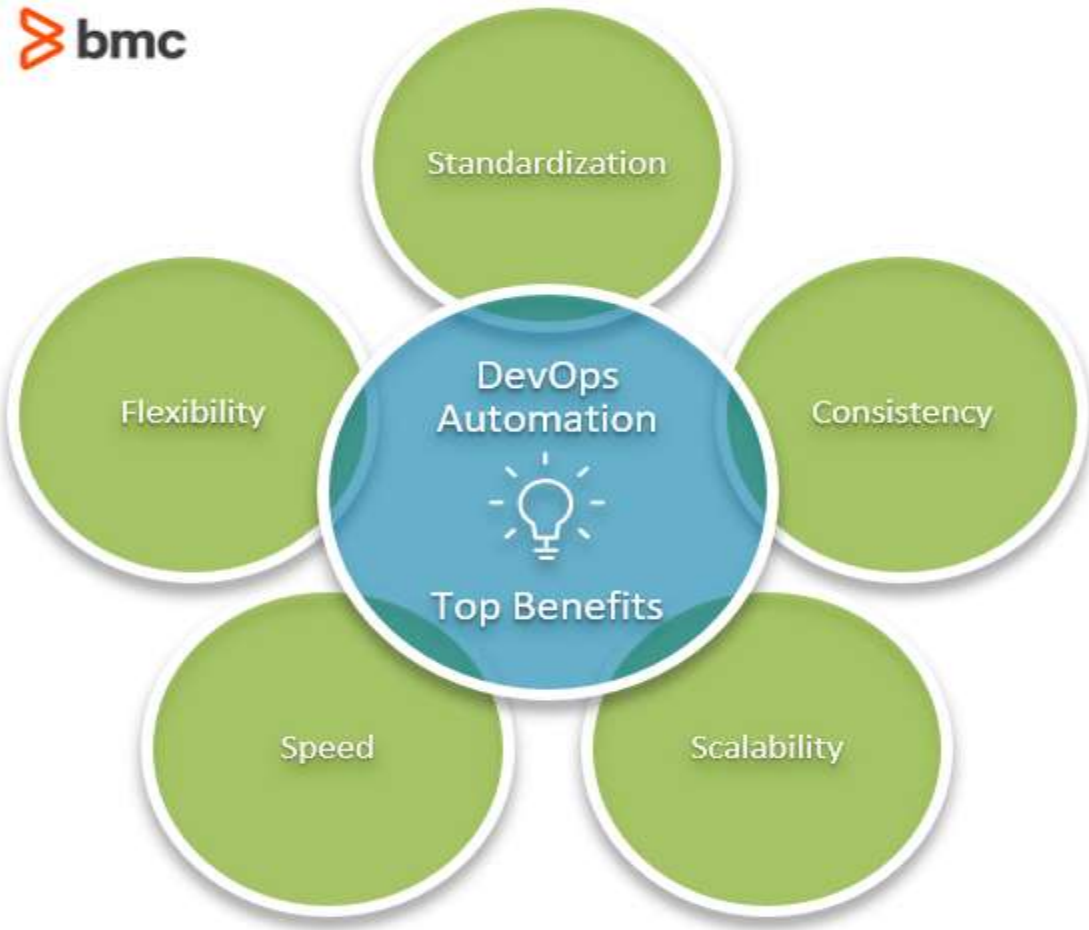
- Design and development
- Software deployment and release
- Monitoring

The goal of DevOps automation is to streamline the DevOps lifecycle by reducing manual workload. This automation results in several key improvements:

- Eliminates the need for large teams
- Drastically reduces human errors
- Increases team productivity
- Creates a fast-moving DevOps lifecycle

Automated Monitoring in DevOps

- Features :



Automated Monitoring in DevOps

Benefits:

- Consistency
- Scalability
- Speed
- Flexibility

Other Agile Methodologies

- Introduction to XP
- FDD
- DSDM
- Crystal

Introduction to XP

- Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

Introduction to XP

5 Values:

Communication

- Software development is inherently a team sport that relies on communication to transfer knowledge from one team member to everyone else on the team. XP stresses the importance of the appropriate kind of communication – face to face discussion with the aid of a white board or other drawing mechanism.

Simplicity

- Simplicity means “what is the simplest thing that will work?” The purpose of this is to avoid waste and do only absolutely necessary things such as keep the design of the system as simple as possible so that it is easier to maintain, support, and revise. Simplicity also means address only the requirements that you know about; don’t try to predict the future.

Feedback

- Through constant feedback about their previous efforts, teams can identify areas for improvement and revise their practices. Feedback also supports simple design. Your team builds something, gathers feedback on your design and implementation, and then adjust your product going forward.

Courage

- Kent Beck defined courage as “effective action in the face of fear” (Extreme Programming Explained P. 20). This definition shows a preference for action based on other principles so that the results aren’t harmful to the team. You need courage to raise organizational issues that reduce your team’s effectiveness. You need courage to stop doing something that doesn’t work and try something else. You need courage to accept and act on feedback, even when it’s difficult to accept.

Respect

- The members of your team need to respect each other in order to communicate with each other, provide and accept feedback that honors your relationship, and to work together to identify simple designs and solutions.

Introduction to XP

Practices

- The core of XP is the interconnected set of software development practices listed below.
- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-hour week
- On-site Customer
- Coding Standard

FDD (Feature Driven Development)

- Feature Driven Development (FDD) is an agile framework that, as its name suggests, organizes software development around making progress on fe
- FDD was designed to follow a five-step development process, built largely around discrete “feature” projects. That project lifecycle looks like this:
- Develop an overall model
- Build a features list
- Plan by feature
- Design by feature
- Build by feature
- atures.

FDD (Feature Driven Development)

FDD (Feature Driven Development)

FDD’s strengths include:

- Simple five-step process allows for more rapid development
- Allows larger teams to move products forward with continuous success
- Leverages pre-defined development standards, so teams are able to move quickly

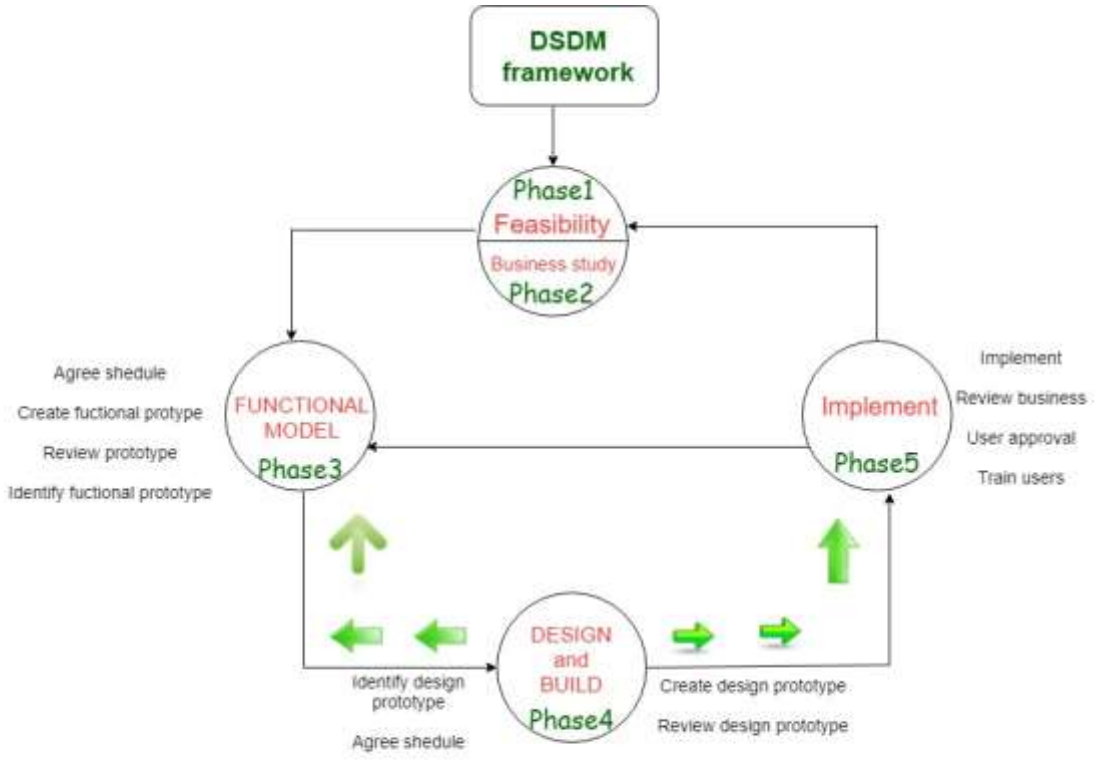
FDD’s weaknesses include:

- Does not work efficiently for smaller projects
- Less written documentation, which can lead to confusion
- Highly dependent on lead developers or programmers

DSDM (Dynamic System Development Method)

- The Dynamic Systems Development Method (DSDM) is an agile framework that addresses the entire project lifecycle and its impact on the business. Like the broader agile philosophy, DSDM is an iterative approach to software development, and this framework explicitly states “any project must be aligned to clearly defined strategic goals and focus upon early deliver of real benefits to the business.” The framework is built on four principles: feasibility and business study, functional model and prototype iteration, design and build iteration, and implementation.

DSDM (Dynamic System Development Method)



Dynamic Systems Development Method life cycle

DSDM (Dynamic System Development Method)

Like other agile methods, the DSDM method is based on an **incremental process**, with frequent releases and testing. It describes **7 project phases**:

- **Pre-project**: preparatory work to define a vision and set goals,
- **Feasibility**: checking whether the project is realistic and objectives are attainable,
- **Foundations**: defining the solutions and methods that will be used for the project,
- **Exploration**: prioritising and iterative definition and testing of features,
- **Engineering**: developing the project incrementally,
- **Deployment**: implementing each iteration of the project,
- **Post project**: assessing the benefits obtained from the project.

Crystal

- The crystal method is an agile framework that is considered a lightweight or agile methodology that focuses on individuals and their interactions. The methods are color-coded to significant risk to human life. It is mainly for short-term projects by a team of developers working out of a single workspace.

Crystal

- **Properties of Crystal Agile Framework :**
- **Frequent Delivery-** It allows you regularly deliver the products and test code to real users. Without this, you might build a product that nobody needs.
- **Reflective Improvement-** No matter how good you have done or how bad you have done. Since there are always areas where the product can be improved, so the teams can implement to improve their future practices.
- **Osmotic Communication-** Alistair stated that having the teams in the same physical phase is very much important as it allows information to flow in between members of a team as in osmosis.
- **Personal Safety-** There are no bad suggestions in a crystal team, team members should feel safe to discuss ideas openly without any fear.
- **Focus-** Each member of the team knows exactly what to do, which enables them to focus their attention. This boosts team interaction and works towards the same goal.
- **Easy access to expert users-** It enhances team communication with users and gets regular feedback from real users.
- **Technical tooling-** It contains very specific technical tools which to be used by the software development team during testing, management, and configuration. These tools make it enable the team to identify any error within less time.

Crystal

- How does it function?
- Crystal family consists of many variants like Crystal Clear, Crystal Yellow, Crystal Red, Crystal Sapphire, Crystal Red, Crystal Orange Web, and Crystal Diamond.
- **Crystal Clear-** The team consists of only 1-6 members that is suitable for short-term projects where members work out in a single workspace.
- **Crystal Yellow-** It has a small team size of 7-20 members, where feedback is taken from Real Users. This variant involves automated testing which resolves bugs faster and reduces the use of too much documentation.
- **Crystal Orange-** It has a team size of 21-40 members, where the team is split according to their functional skills. Here the project generally lasts for 1-2 years and the release is required every 3 to 4 months.
- **Crystal Orange Web-** It has also a team size of 21-40 members were the projects that have a continually evolving code base that is being used by the public. It is also similar to Crystal Orange but here they do not deal with a single project but a series of initiatives that required programming.
- **Crystal Red-** The software development is led by 40-80 members where the teams can be formed and divided according to requirements.
- **Crystal Maroon-** It involves large-sized projects where the team size is 80-200 members and where methods are different and as per the requirement of the software.
- **Crystal Diamond & Sapphire-** This variant is used in large projects where there is a potential risk to human life.

Crystal


CLEAR
6 people
YELLOW
20 people
ORANGE
40 people
RED
80 people
MAROON
More people
DIAMOND & SAPPHIRE
Large Size
CRYSTAL TEAM SIZE

Crystal

Benefits of using the Crystal Agile Framework :

- Facilitate and enhance team communication and accountability.
- The adaptive approach lets the team respond well to the demanding requirements.
- Allows team to work with the one they see as the most effective.
- Teams talk directly with each other, which reduces management overhead.

Drawbacks of using the Crystal Agile Framework :

- A lack of pre-defined plans may lead to confusion and loss of focus.
- Lack of structure may slow down inexperienced teams.
- Not clear on how a remote team can share knowledge informally.