**Explore the following:**

**1: Extreme Programming Framework.**

* **Extreme programming** (**XP**) is an agile software development **framework** that aims to produce higher quality software. **XP** is the most specific of the agile frameworks regarding appropriate engineering practices for software development. XP is usually used in when there is a Dynamic change in software requirements, Risks caused by fixed time projects by using new technology, XP can be divided into small chunks to handle the tasks, the technology you are using allows for automated unit and functional tests. The prime examples of XP are **Test-Driven Development, The Planning Game, On-site Customer, Pair Programming, Code Refactoring, Continuous Integration, Small Releases, Simple Design.**

**XP** has 4 basic frameworks:

* Coding
* Testing
* Listening
* Designing

**2: Pair Programming and its Benefits.**

Pair programming is basically an Agile software development technique originating from **XP**. The pair Programming is the communication between teams to share information/knowledge among themselves and working together on one Computer which benefits the whole team in achieving the shared Goal and achieving collecting results of collective Code ownership and the benefits are:

* It Makes for Better Design. Pairing improves the quality of the design ideas.
* It Makes for Better Designers and Better Design Organizations. Pairing improves the culture of design for teams and organizations.
* Pair Design Makes for a More Effective Process.

**3: Git Repository Guidelines in Agile Development.**

Git Repository is a de facto version control system. Git Repo is a open source project`s that tracks any changes made in any Computer files and usually used for coordinating work among Programmer`s collaboratively developing source code during software development. Git repo is an essential part of Devops toolchain while using Agile and supports a wide range of workflows in any software team. Its distributed nature – rather than centralized – gives it superior performance characteristics and allows developers the freedom to experiment locally and publish their changes only when they're ready for distribution to the team.

Besides the benefits of flexibility and distribution, there are key functions of Git repo that supports and enhances the agile and DevOps development teams. Think of Git as a component of agile and DevOps development: changes can get pushed down the deployment pipeline faster than working with monolithic releases and centralized version control systems. Git works the way your agile and DevOps teams work (and should strive to work).

Git starts fitting into your agile workflow at this point. At Atlassian, we create a new branch for every single issue. Whether it's a new feature, a bug fix, or a small improvement to some existing code, every code change gets its own branch.

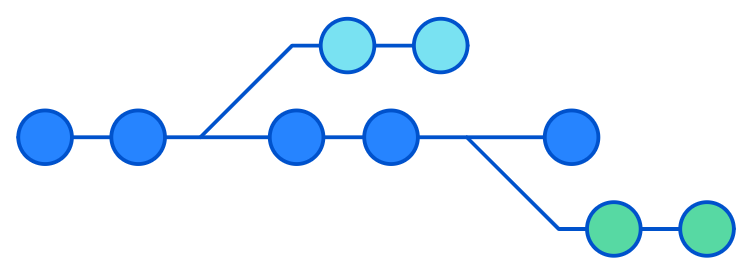
Branching is straight forwards and allows teams to easily collaborate inside one central codebase. When a developer creates a branch, they effectively have their own isolated version of the codebase in which to make changes. For an agile team this means that by breaking features into user stories and then branches, the development team has the ability to tackle tasks individually and work more efficiently on the same code but in different repositories; there is no doubling up of work and since individuals are able to focus on small chunks of work in repositories separate from the main repository there are not as many dependencies slowing down the development process.

Once branches are considered done and ready for code reviews, Git plays another key role in an agile development workflow: testing. Successful agile and DevOps teams practice code reviews and setup automated tests in CI/CD To help with code reviews and testing, developers can easily notify the rest of their team that the branch work is ready for review and that it needs to be reviewed through a pull request. More simply put, a pull request is a way to ask another developer to merge one of your branches into the main branch and that it is ready for testing.

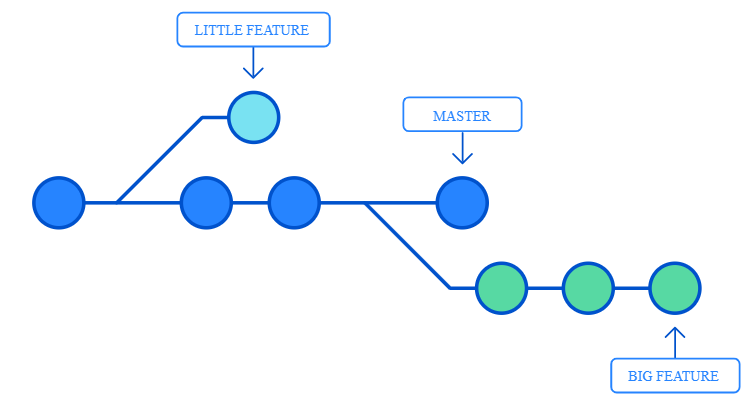
With the right tooling, your continuous integration server can build and test your pull requests before you merge them. This gives you confidence that your merge won't cause problems. This confidence makes it easier to retarget bug fixes and conflicts in general, because Git knows the difference between the branch and main code base since the branches have diverged.

Once you’ve merged a branch to the main branch, your agile workflow is done. Likewise, merging code through pull/Push requests means that when code is done, you have the documentation to confidently know that your work is green, that other team members have signed off on the code, and that it is ready to release. This keeps agile teams moving at a speed and with confidence to release often: a sign of a great agile team.

**Fig 1: Single Branch**



**Fig 2 of Multiple Branches**



**4: Benefits of working in Batches in Agile.**

The Working benefits of Batches in Agile is it reduces the time on feedbacks of cycle time and when changes are made it is easy to triage and remediate the problems that arise.

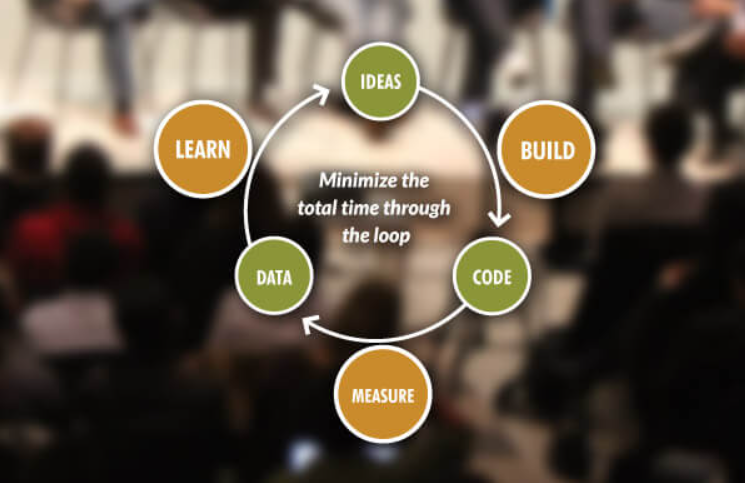
**Example**

**Decreased risk** and variability. Since the batch contains less content, it’s by essence easier to control and validate. Delays are anticipated with more ease, quality issues are easier to spot, integration is done sooner.

**Better prioritization**. Since batches are smaller, it’s easier to prioritize high-value low-cost items first rather than having to prioritize batches bundling low-value items with high value items.

**5: What Exactly is MVP? (Minimum Viable Product).**

The MVP is a version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort.” A minimum viable product helps product managers begin the process of learning quickly.



As you know the Main purpose of MVP is to always minimize time and effort wasted by testing how the market reacts to your idea before building the complete product.

The prime Examples of MVP are:

* Dropbox
* Facebook
* Uber

**6: Get a Good Basic Understanding of Test-Driven Development (TDD) and Behavior-Driven Development (BDD); and difference between them.**

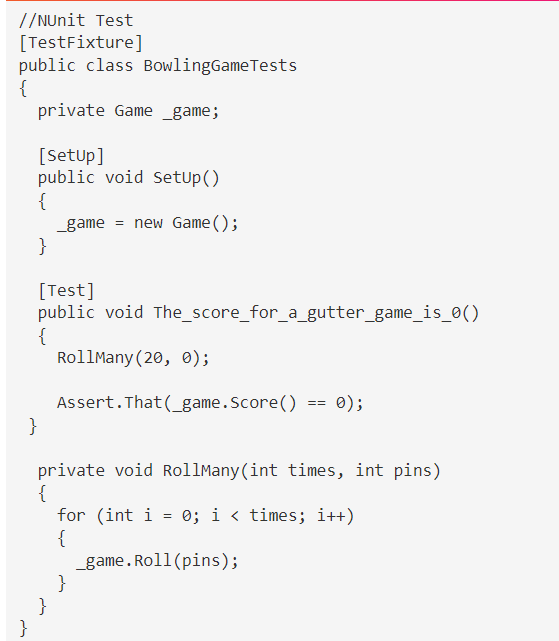
TDD is a development practice while BDD is a team methodology. In TDD, the developers write the tests while in BDD the automated specifications are created by users or testers (with developers wiring them to the code under test.)

For small, co-located, developer-centric teams, TDD and BDD are effectively the same.

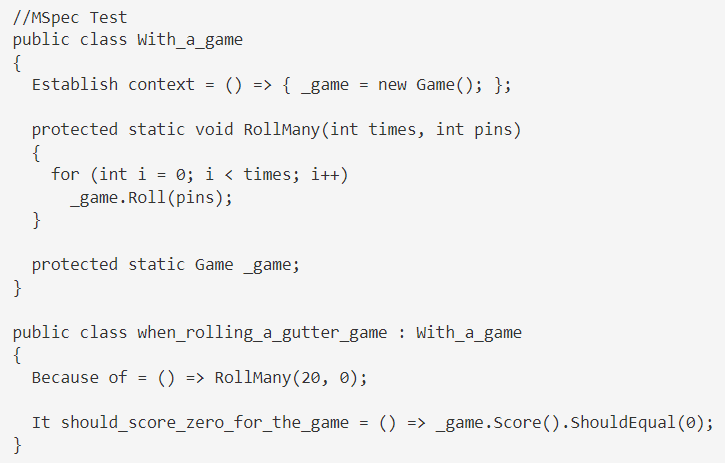
Comparison between TDD and BDD

NUnit and MSpec are 2 tools that provide for different styles of developer testing. NUnit promotes the Arrange-Act-Assert style of testing while MSpec requires the Given-When-Then (or Establish context-Because of-It should) style of testing.

**NUnit Testing**



### MSpec Testing



### How do BDD tests work?

The key to BDD is to get the specifications from the user. In other words, create tests that are not written by developers. This means tests that are not written in a programming language. These tests should be written in a language close to English.

### BDD Testing Tools and Advantages

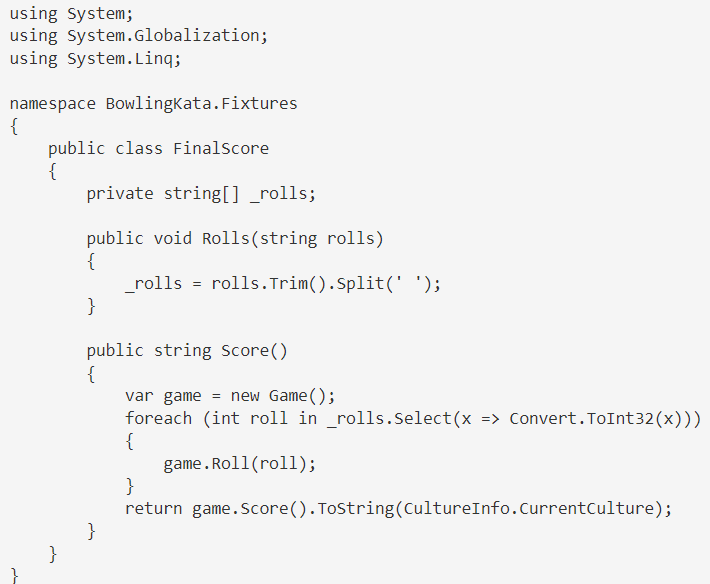
Some BDD testing tools include Zephyr scale, Concordian, and FitNesse. As an example, let’s look at a how FitNesse works.  The advantages of using FitNesse include.

* it facilitates thinking about features and problems in the language of business rather than the language of code.
* it requires you to focus on data in your tests.
* it can be easily included in a continuous integration environment.
* it includes a wiki for sharing information about the project.
* it requires the creation of fixtures that will help define, and refine, the API.
* it is easily shared with non-developer users.

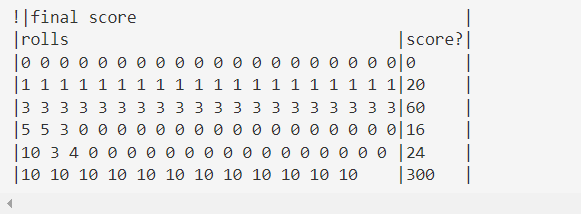
### FitNesse for BDD Testing?

One easy way to get started is to clone this repository and follow the instructions in the README.md.

FitNesse tests consist of 2 parts: the test pages in the wiki and the fixtures that connect the pages to the code under test. In the bdd tdd fitnesse repo, you will find a file FinalScore.cs in the Fixtures folder. This is the fixture used by the tests.



Result



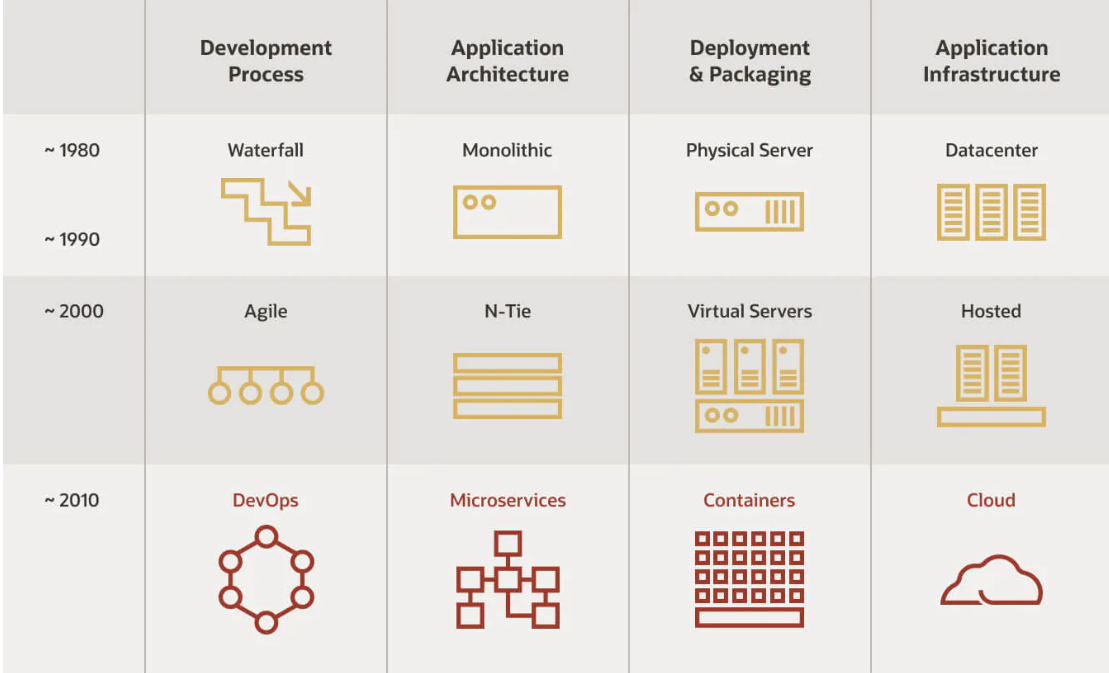
So the question arise which is better TDD or BDD and the Answer is Both because developer tests for the fast feedback and you want user tests to ensure that the features are built to the user specs.

**7: Why is TDD Important for DevOps?**

TDD is important for Devops because it increases speed as less time is spent debugging by the developers. As a result, it could take longer to create tests and production code in the beginning stages. However, as the project develops, adding and testing new functionality will go faster and with less rework and if more examples are required, we can refer Q6.

**8: What are Cloud Native Microservices?**

Microservices are the core of cloud native application architecture, and they have become a key tool for companies that are making the move to the cloud. Microservices arrange an application into multiple, independent services, each of which serves a specific function. Many software companies take advantage of microservices because they support DevOps, enable flexibility, and improve scalability, while also reducing costs. Cloud native microservices communicate with each other via APIs and use event-driven architecture, which serves to enhance the overall performance of each application. Oracle Cloud Native services follow the CNCF trail map to help simplify the journey and make it easier for companies to start building, deploying, and managing modern cloud native applications.

****

**Micro services Functions:**

The term **serverless functions** describes an architecture style that focuses on increasing developers’ productivity. A serverless application lets you write code on a platform that functions as a service (FaaS) using event-driven architectures and various backend-as-a-service (BaaS) models. This eliminates the need to worry about provisioning, patching, scaling, security, high availability, and so forth. With FaaS platforms, such as Oracle Functions, applications are broken up into small pieces of code (nanoservices), which are dynamically scheduled and run on demand when triggered by an event. The advantage of this approach is that code is called and executed only when needed, and you pay only for the resources used during the execution duration. This differs from a classic server approach, in which applications are loaded into a server and spend most of their time idle, waiting for requests. Thus, in serverless computing, you pay only for computing resources you actually use, rather than paying for idle resources.

## **What are cloud native services?**

Cloud-native services are at the core of digital innovations and are key to advanced analytics, mobile apps, and chatbots. DevOps practices remove most of the management tasks associated with building, operating, and maintaining a complex software platform. Software development, deployment, and testing activities reside in the cloud and can be expanded or contracted at will. Shifting applications, DevOps, and workloads to a cloud native architecture is integral to keeping your business competitive.

### What is Kubernetes?

Oracle’s cloud native services drive modern cloud native application development by using standards-based technologies such as Kubernetes, Docker, serverless functions, APIs, and Kafka. Often described as the “operating system for the cloud,” Kubernetes is an open source platform for managing clusters of containerized applications and services. The key components of Kubernetes are clusters, nodes, and the control plane. Clusters contain nodes. Each node comprises a set of at least one worker machine. The nodes host pods that contain elements of the deployed application. The control plane manages nodes and pods in the cluster, often across many computers, for resiliency and high availability.

Oracle provides the cloud tooling and automation necessary for these services so that development teams can reduce operational tasks and quickly build applications. Cloud native services run on Oracle Cloud Infrastructure (OCI), which offers a standards-based platform with higher performance and lower cost compared to other cloud providers. By taking advantage of services based on open source and open standards, OCI makes it possible for developers to run applications on any cloud or on-premises environment without refactoring—which gives them more freedom to focus on building and innovating.

## Cloud native services

### Container Registry

OCI Container Registry is an open standard based, Oracle-managed Docker registry service for securely storing and sharing container images. Engineers can easily push and pull Docker images with the familiar Docker command line interface (CLI) and API. To support container lifecycles, Container Registry works with Oracle’s Container Engine for Kubernetes, OCI Identity and Access Management, Oracle Visual Builder Studio, and third-party developer and DevOps tools.

### Notification

### Streaming

### Container Engine

### Functions

### 

**9: Designing for Failure in DevOps.**

Designing for failure in Devops means that **your team has automated processes in place for when your system fails, in addition to having as much control as possible over how this failure occurs**. A system designed for failure is more capable of self-healing, restarting, and maintaining service when the worst happens.

For example, **a vehicle that automatically limits speed when its engine is overheating or experiencing mechanical problems**. This may allow the occupants of the vehicle to get to a safe place before the engine completely fails.

**10: What is Taylorism?**

The word Taylorism is a practice/principles or scientific management which analyzes workflows. The main objective is improve economic efficiency, especially labor productivity. It was one of the earliest attempts to apply science to the engineering of processes to management.

Get a solid understanding of these concepts and connect the dots to understand how things go in agile development model.

Write two different articles on any two topics listed above.

Prepare notes for all the topics in your GitHub repo.

**Article 1 Microservices**

Microservices are the core of cloud-native application architecture, and they have become a key tool for companies making a move to the cloud. Microservices arrange an application into multiple, independent services, each serving a specific function. Many software companies take advantage of microservices because they support DevOps, enable flexibility, and improve scalability, while also reducing costs. Microservices are a product improvement procedure — a variation of the help-situated engineering (SOA) primary style — that organizes an application as an assortment of inexactly coupled administrations. In a microservices design, administrations are fine-grained, and the conventions are lightweight. The benefits of decomposing an application into different, more minor services include developing, deploying, and scaling the application independent of other services. Other benefits include the easy integration of new technologies into an existing system and the ability to scale subcomponents selectively. microservices communicate with each other via APIs and use event-driven architecture, which serves to enhance the overall performance of each application. To assess the security of your microservices architecture, it is essential to understand the potential vulnerabilities that can be introduced. These vulnerabilities can be broadly categorized into data leakage, distributed denial of service (DDoS), and service hijacking. Each of these categories presents its challenges that must be carefully managed. Data leakage, for example, can occur when microservices communicate with each other using insecure protocols or when sensitive data is stored in insecure locations. Microservices are a product improvement procedure — a variation of the help-situated engineering (SOA) primary style — that organizes an application as an assortment of inexactly coupled administrations. In a microservices design, administrations are fine-grained, and the conventions are lightweight. The benefits of decomposing an application into different, more minor services are numerous but often include a higher degree of developer productivity and the ability to evolve services independently. This can enable an organization to align its engineering and business strategies.

**Modularity is the Key to Microservices**

Modularity is the key in Microservices. The term Microservices has been used in many different ways. Still, it is generally understood to mean a software development technique - a variant of the service-oriented architecture (SOA) structural style - that arranges an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained, and the protocols are lightweight.

**Benefits of Microservices**

The main benefits of microservices are that they are easy to develop and maintain and can be deployed independently. They also have several disadvantages, including the potential for increased complexity and the need for a more sophisticated deployment infrastructure. One of the challenges of microservices is ensuring that the services work well together and that the overall system is scalable.

In a microservices architecture, services are fine-grained, and the protocols are lightweight. Join programming improvement (Dev) and data innovation tasks (Operations). DevOps aims to shorten the software development life cycle and provide continuous delivery with high software quality. DevOps is relevant to microservices because microservices need to be continuously delivered and deployed. Many DevOps tools can automate the delivery and deployment of microservices.

**SOA (Service-Oriented Architecture)** is a style of engineering scalable web applications as a suite of coarse-grained services, each running in its process and communicating using language-agnostic APIs. Each service has a limited scope and is highly independent. This modular approach lets you develop, deploy, and scale individual services independently. Services are well-suited for DevOps because of their small size and independent nature.

best practices is essential to build maintainable, scalable, and secure microservices. These practices will help ensure that your microservices can handle the demands of a distributed system while still functioning correctly. By following these best practices, you can avoid many of the common pitfalls that can occur when building microservices.

**Article 2 Failure in DevOps**

**What is DevOps?**

DevOps is the combination of Practices and tools used in IT industry to increases the Organization ability to deliver applications and services at a faster pace rather than the traditional process.

**What is DevOps culture?**

DevOps culture is all about a shared understanding between developers and operations, and sharing responsibility for the software they build by increasing transparency, communication, and collaboration across development, IT/operations, and "the business”.

**Brief history of DevOps.**

Devops was created with the collaboration of developers and IT operations when they felt the desperate need of the two operations together it was first started in between 2007 to 2008 and was given the term DevOps in 2009 by Patrick Debois.

**Failure in DevOps**

When DevOps was at its introductory stage, the team at IBM also was unable to get the essence of DevOps and could not bridge a gap between Development and Operation team. The team identified that the environment was not suitable for faster releases and strict deadlines and therefore there was a gap between the development and operations. Despite the infinite definitions and understating, DevOps fails at execution. Even, leading organizations and the best resources are failing to get DevOps implementation and there are some legitimate reasons stating below:

* Observe the before-DevOps situation, tools, and behaviour of the software development team.
* Identify ‘**What They Do**’ and ‘**What Should They Do and Why**’
* Train them not only with the tools, but make them understand **DevOps as A Philosophy**
* Make this transformation easier by establishing habits from top management.

There are many ways and practices which can help implement DevOps with guaranteed result. Most often, organizations are running after tools and not cultural shift making it the biggest cause of DevOps failure. In this digital era, every organization is a technology-driven organization regardless of the domains. The journey from ‘**Digital Transformation**’ to ‘**Continuous Digital Journey**’ demanded flexibility, agility, and quality as most-focused aspects.

DevOps has become a need for organizations that are associated with software delivery or often releasing an update or new features to serve their customers with quality and superiority.

No doubt, DevOps can make software development faster, but every organization has a different set of requirements.

## Think Of Powerful Infrastructure with DevOps Toolchain Or Fail Often

DevOps automates development, delivery, and deployment.

It is of utmost importance that the infrastructure set up needs a faultless strategy to plan, verify, sync, and monitor DevOps toolchain to avoid bigger loss.

What’s more important is continuous inspection and measurement to take necessary actions to get the expected outcome from DevOps efforts. DevOps requires faultless set up of tools and their links in order to plan, manage, configure, manage, and verify all the processes.