ML OPS/DEV OPS PRACTICES AND BASIC CONCEPTS AND TERMS

Best Practices for Implementing DevOps

1. Agile project management

[Agile](https://www.atlassian.com/agile) is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Agile teams focus on delivering work in smaller increments, instead of waiting for a single massive release date. Requirements, plans, and results are evaluated continuously, allowing teams to respond to feedback and pivot as necessary.

The following are key concepts for agile project management:

* Start with a workflow that includes four phases: to do, in progress, code review, and done. [Read more about workflows](https://www.atlassian.com/agile/project-management/workflow).
* Teams need to break large-scale projects into smaller tasks and respond to changes in needs or scope as they make progress. [Read more about how to use epics, stories, and themes](https://www.atlassian.com/agile/project-management/epics-stories-themes) to scope and structure work.
* How do you plan, track, and measure the incremental work? [Scrum](https://www.atlassian.com/agile/scrum) and [kanban](https://www.atlassian.com/agile/kanban) are core frameworks for teams practicing agile methodology.

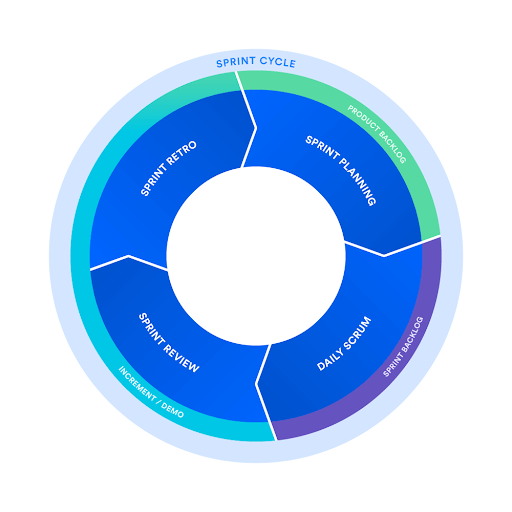
**Agile vs. scrum**

People often think scrum and agile are the same thing because scrum is centered around continuous improvement, which is a core principle of agile. **However, scrum is a framework for getting work done, whereas**[**agile**](https://www.atlassian.com/agile)**is a philosophy.** The agile philosophy centers around continuous incremental improvement through small and frequent releases. You can’t really “go agile”, as it takes dedication from the whole team to change the way they think about delivering value to your customers. **But you can use a framework like scrum to help you start thinking that way and to practice building agile principles into your everyday** communication and work.

The Agile manifesto outlines four values:

* Individuals and interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan

Lean thinking reduces waste and focuses on essentials. The scrum framework is heuristic; it’s based on continuous learning and adjustment to fluctuating factors. It acknowledges that the team doesn’t know everything at the start of a project and will evolve through experience. Scrum has **re-prioritization built into the process and short release cycles so your team can constantly learn and improve.**



**What is kanban?**

Kanban is a popular framework used to implement [agile](https://www.atlassian.com/agile) and [DevOps](https://www.atlassian.com/devops/what-is-devops) software development. It requires real-time communication of capacity and full transparency of work. **Work items are represented visually on a kanban board, allowing team members to see the state of every piece of work at any time.**

**What is JIT ?**

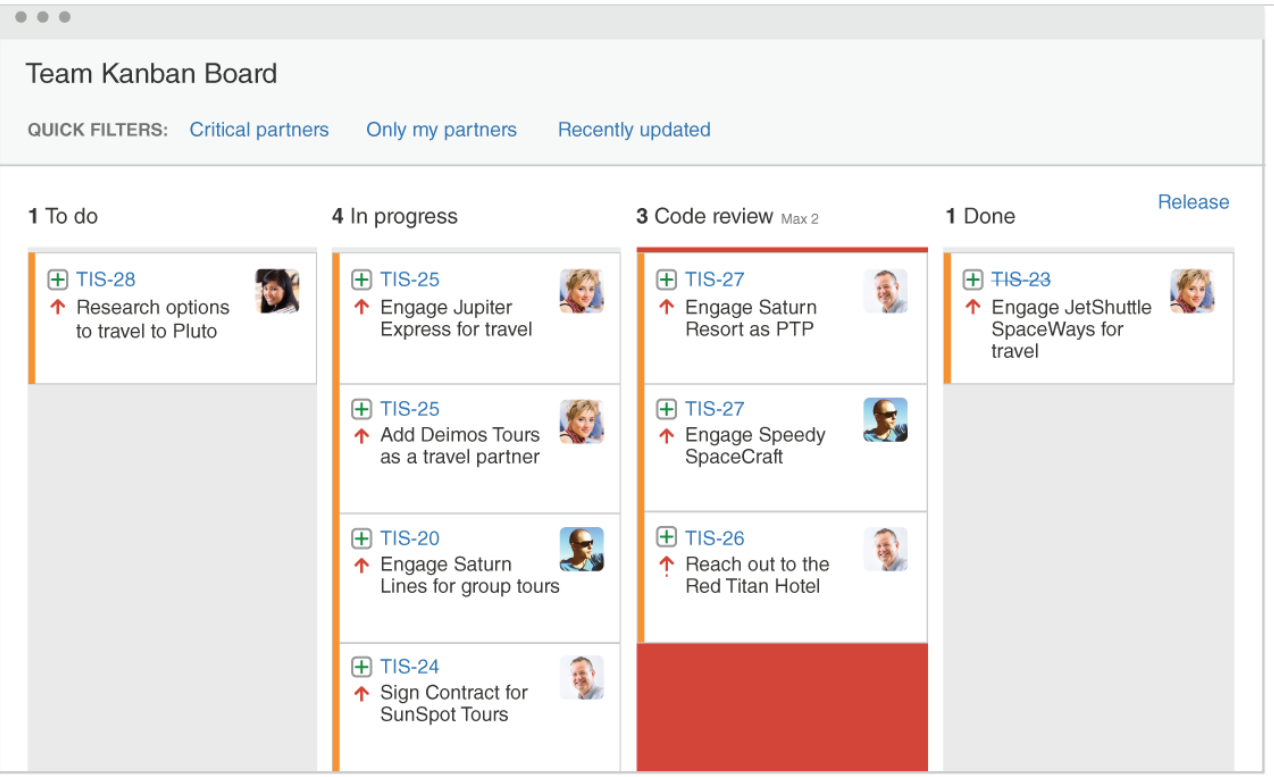
To communicate capacity levels in real-time on the factory floor (and to suppliers), workers would pass a card, or "kanban", between teams. When a bin of materials being used on the production line was emptied, a kanban was passed to the warehouse describing what material was needed, the exact amount of this material, and so on. The warehouse would have a new bin of this material waiting, which they would then send to the factory floor, and in turn send their own kanban to the supplier. The supplier would also have a bin of this particular material waiting, which it would ship to the warehouse. While the signaling technology of this process has evolved since the 1940s, this same "just in time" (or JIT) manufacturing process is still at the heart of it.

Kanban for software teams

Agile software development teams today are able to leverage these same JIT principles by matching the amount of work in progress (WIP) to the team's capacity. This gives teams more flexible planning options, faster output, clearer focus, and transparency throughout the development cycle.

The kanban methodology relies upon full transparency of work and real-time communication of capacity. Therefore, the kanban board should be seen as the single source of truth for the team's work.

Example:



2. ​​​​​​​ CI/CD

CI/CD falls under DevOps (the joining of development and operations teams) and combines the practices of continuous integration and continuous delivery. CI/CD **automates much or all of the manual human intervention** traditionally needed to get new code from a commit into production. **It encompasses** **build, test (including integration tests, unit tests, and regression tests), and deploy phases,** as well as infrastructure provisioning. With a CI/CD pipeline, development teams can make changes to code that are then automatically tested and pushed out for delivery and deployment.

In continuous testing, various types of tests are performed within the CI/CD pipeline. These can include:

* **Unit testing**, which checks that individual units of code work as expected
* **Integration testing**, which verifies how different modules or services within an application work together
* **Regression testing**, which is performed after a bug is fixed to ensure that specific bug won't occur again

 A purpose-built CI/CD platform **decreases tedious and time-consuming manual development work** and legacy approval processes, freeing DevOps teams to be more innovative in their software development. Automation makes processes predictable and repeatable so that there is less opportunity for error from human intervention.

1. Continuous integration is Continuous integration (CI) is a software development practice in which developers merge their changes to the main branch many times per day. Each merge triggers an automated [code build](https://semaphoreci.com/blog/build-stage) and [test sequence](https://semaphoreci.com/blog/automated-testing-cicd), which ideally runs in less than 10 minutes. A successful CI build may lead to further stages of continuous delivery (CD, explained later).

If a build fails, the CI system blocks it from progressing to further stages. The team receives a report and repairs the build quickly, typically within minutes. CI processes should have a version control system that tracks changes so you know the version of the code used.

Advantages of doing it Automatically :

* By merging changes frequently and triggering automatic testing and validation processes, you minimize the possibility of code conflict, even with multiple developers working on the same application.
* A secondary advantage is that you don't have to wait long for answers and can, if necessary, fix bugs and security issues while the topic is still fresh in your mind.

**TERMS:**

**1.** **What happens in a build stage**

In [continuous integration](https://semaphoreci.com/continuous-integration) (CI), this is where we build the application for the first time. The build stage is the first stretch of a [CI/CD pipeline](https://semaphoreci.com/blog/cicd-pipeline), and it automates steps like downloading dependencies, installing tools, and compiling.

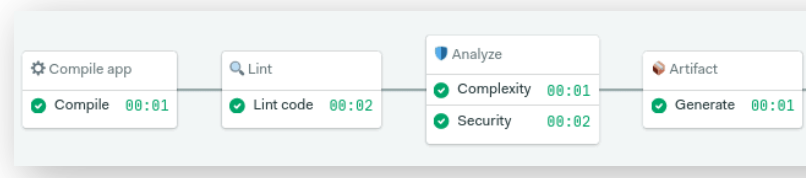
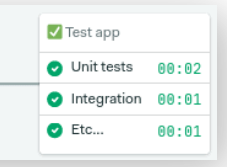
 

Fig a. Build Stage of CI Fig b. Test Stage of CI

Besides building code, build automation includes using tools to check that the code is safe and follows best practices. The build stage usually ends in the [artifact generation](https://semaphoreci.com/blog/semaphore-artifacts) step, where we create a production-ready package. Once this is done, the [testing stage](https://semaphoreci.com/blog/automated-testing-cicd) can begin ( i.e. Unit, Integration and Regression, as explained above. These are next step in CI step).

‘Build automation’ verifies that the application, at a given code commit, can qualify for further testing. We can divide it into three parts:

1. **Compilation**: the first step that compiles our Raw code into a result that can be a binary file, a code tarball, an installable package, a website, a container image — **the main thing is that we have something that we can run and test**.
2. **Linting**: checks the code for programmatic and stylistic errors.  Linters use a set of rules to enforce a unified standard that improves readability for the team
3. **Code analysis**: using automated source-checking tools, we control the code’s quality.
4. **Artifact generation**: the last step packages the application for release or deployment.

Q: What is commit vs push?

**git commit saves repository changes on local but not remote repository.** **Contrarily, Git push then updates your git commit changes and sends it to remote repository**. That is where where working developers will access them

2. Continuous deployment enables organizations to deploy their applications automatically, eliminating the need for human intervention. Once code has been tested and built as part of the CI process, CD takes over during the final stages to **ensure it’s packaged** with everything it needs to deploy to any environment at any time (**ex: If we want to deploy in android environment, the code file is packaged into ‘.apk’ file; Similarly, for deploying in Java environment, the code is packaged into ‘.jar’ file and so on**). With continuous deployment, DevOps teams set the criteria for code releases ahead of time and when those criteria are met and validated, the code is deployed into the production environment automatically.

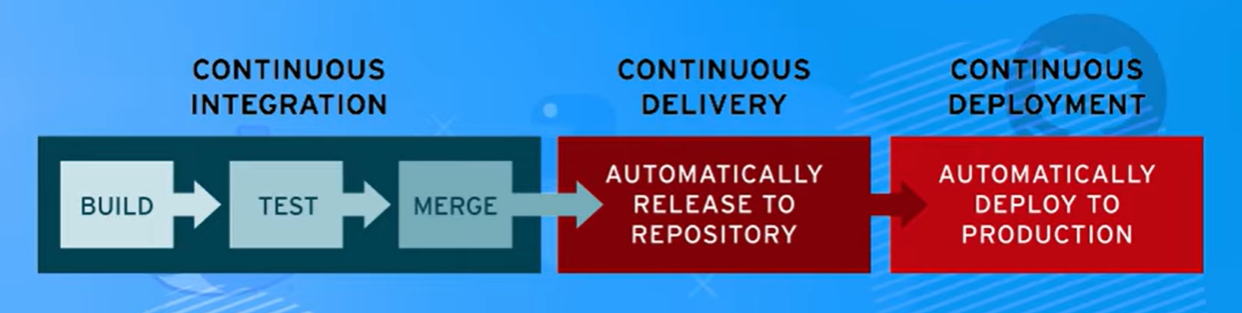
Difference between Continuous Delivery and Continuous Deployment?

[Continuous delivery](https://www.synopsys.com/glossary/what-is-continuous-delivery.html) (CD) is the automated delivery of completed code **to environments like testing and development**. CD provides an automated and consistent way for code to be delivered to these environments.

[Continuous deployment](https://www.synopsys.com/glossary/what-is-continuous-development.html) is the next step of continuous delivery. Every change that passes the automated tests is automatically **placed in production (or production environment**), resulting in many production deployments.

However, both are a part of CI/CD pipeline and of DevOps and MLOps.

**TOOLS for CI/CD : Jenkins , GitHub Actions etc.**



Basic Concept 1.

What is the difference between a repository and a server?

A repository store stuff (files) and a server provides services. You could say GitHub is a server that stores repositories.

You can think of a repository as a store of files alongside it's metadata in some consumable format. Most of the time when we speak of repositories in programming we mean a storage for stuff like code and **the advantage is that we don't just have one file of each filename but we can see different versions of that file** so we can, for example, go back to using an old one if a change was bad. We can **also organize the code files into branches and call them different versions of the program** we make.

A server is a computer/software set up to "serve" content. The term is pretty loosely defined and often ambiguously used. A file server serves files, a web server is typically a files server that serves web pages or other files via a web protocol, ex. http/https. A server is a generic term referring to a computer that is primarily accessed remotely, it could do anything from storing files to mining bitcoin.

A server can mean a few things. It can mean a central computer which we in some way connect to. When it comes to how we connect to the server people use both the word **server** and the word **service**. Sometimes people say: On that machine we run a web ‘server’, but they could just as well say on that server we run a ‘web service’. Personally, **I usually call the machine a server and the services a web server**. The context of the discussion makes people know what you refer to.

**A web server has to store files so it can serve them, but it’s not it’s purpose.**

Basic Concept 2.

**What is Git? Git v/s GitHub**

 Git is an extremely popular version control system that is at the heart of a wide variety of high-profile projects. Git is installed and maintained on your local system (rather than in the cloud) and gives you a self-contained record of your ongoing programming versions. One thing that really sets Git apart is its *branching* model. Branching allows you to create independent local branches in your code. This means you can try out new ideas, set aside branches for production work, jump back to earlier branches, and easily delete, merge, and recall branches at the click of a button.

GitHub is a Git repository hosting service.  It’s an online database that allows you to keep track of and share your Git version control projects outside of your local computer/server. Unlike Git, GitHub is exclusively cloud-based. GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code.

To understand exactly what GitHub is, you need to know two connected principles:

* **Version control** ---- version control lets developers safely work through **branching** and **merging**.

With **branching**, a developer duplicates part of the source code (in the **repository**). The developer can then safely make changes to that part of the code without affecting the rest of the project. Then, once the developer gets his or her part of the code working properly, he or she can **merge** that code back into the main source code to make it official.

All of these changes are then tracked and can be reverted if need be.

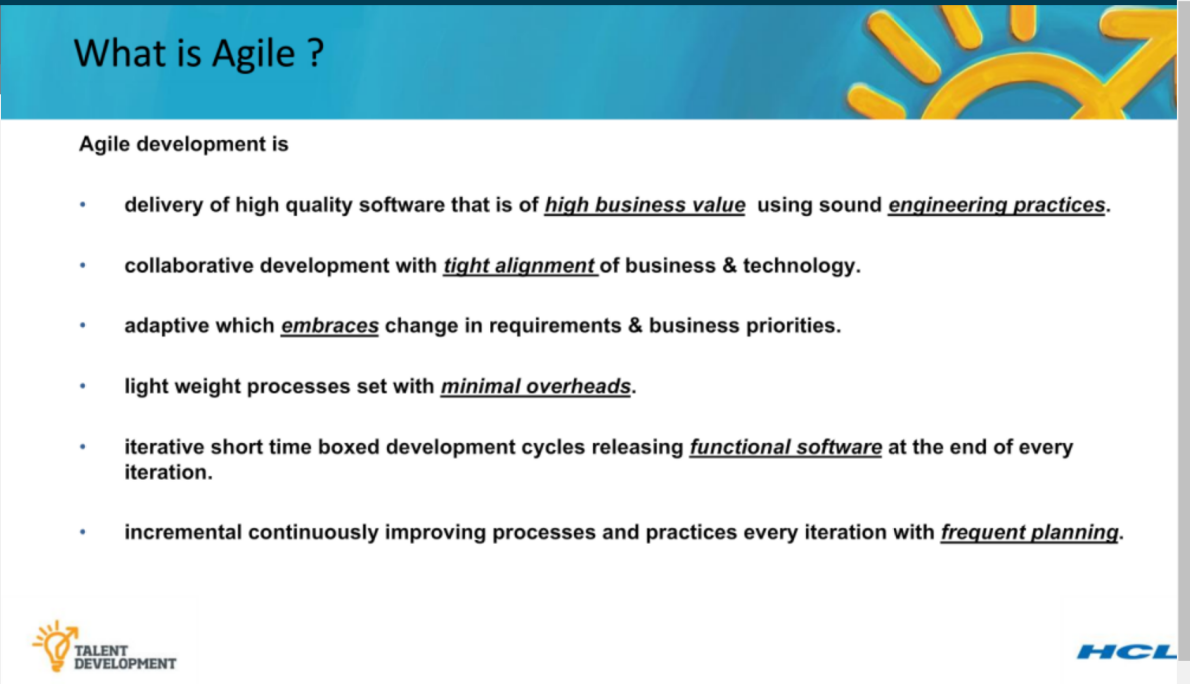
* **Git** ----- Git is a distributed version control system, which means that the entire codebase and history is available on every developer’s computer, which allows for easy branching and merging. (See Above for more)

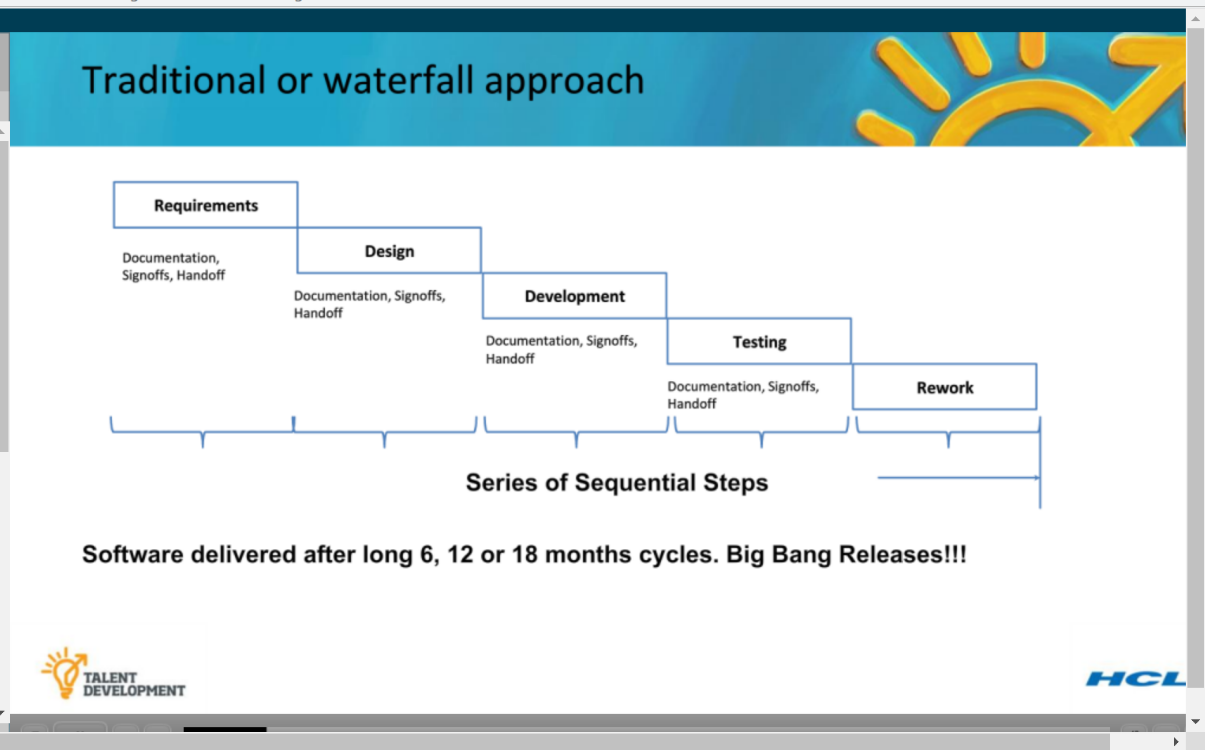
**MLOps v/s DevOps:**

The only difference is that the MLOps file/Application would need retraining again and again, while once a DevOps file/Application is created, it does not need any retraining again.

This is because the ML file is working on Big Data which is constantly being generated every second. Thus, if we don’t include this data into the ML file/app , it is bound to become obsolete after some time. Hence, we need to Retrain our model again and again. (Note: Retraining is DIFFERENT from software updates. Updates would happen in both MLOps and DevOps).

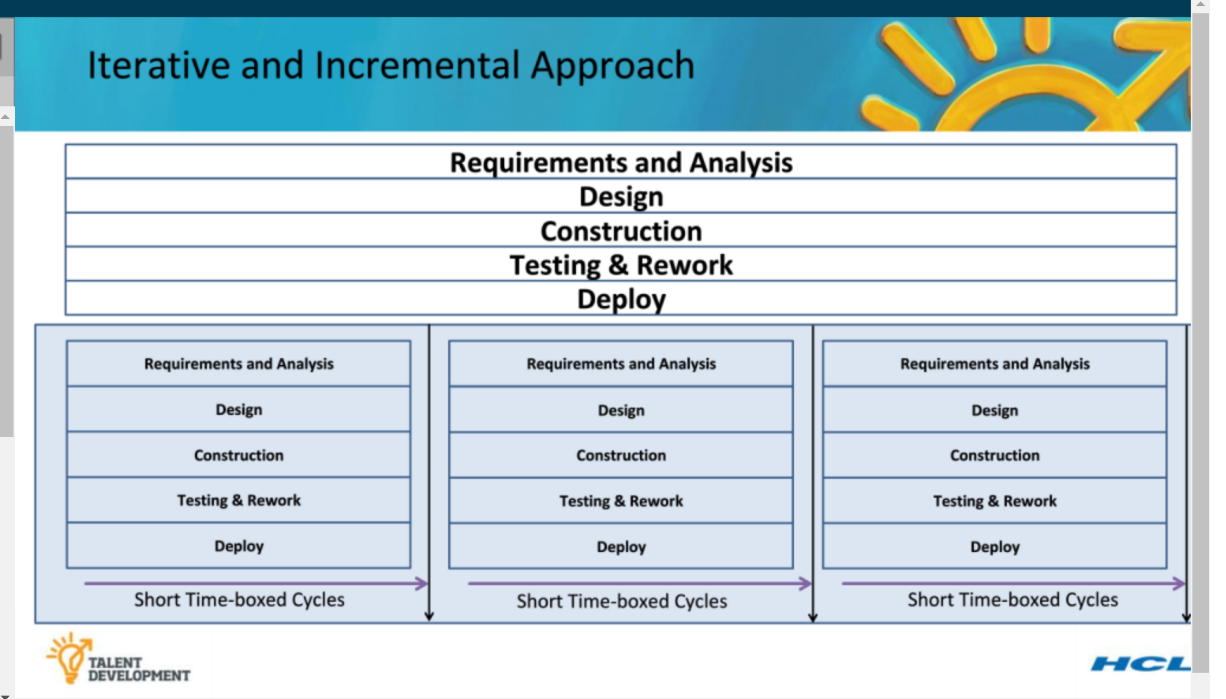
**AGILE :** (continued ….. from top)

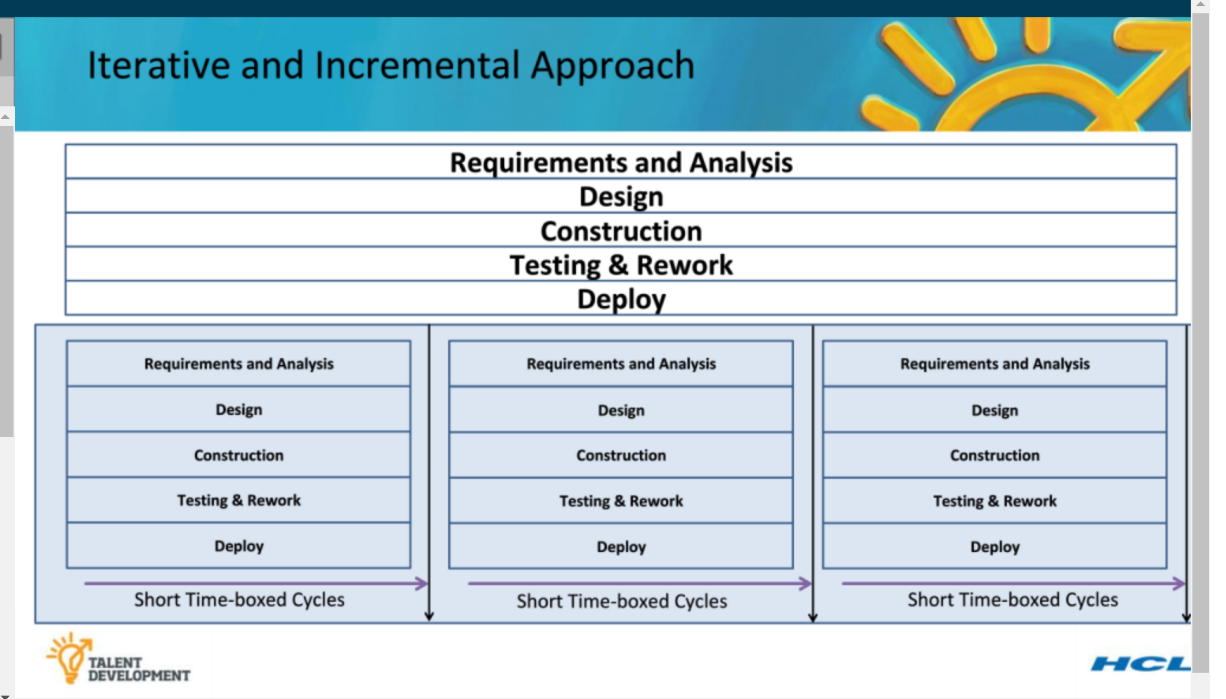




The Traditional Waterfall approach was Bad. This creates delays at every hand-off that leads to frustrated teams and dissatisfied customers. The product eventually goes live through a tedious and error-prone process that delays revenue generation.

Thus, Agile approach was widely adopted.





**WHAT IS AGILE,THEN ?**

**AGILE = ITERATIVE APPROACH + SOMETHING .**



The AGILE FRAMEWORK:

1. SCRUM, KANBAN, LEAN etc. are a part of AGILE or used in AGILE. (But, SCRUM does NOT equal AGILE)

