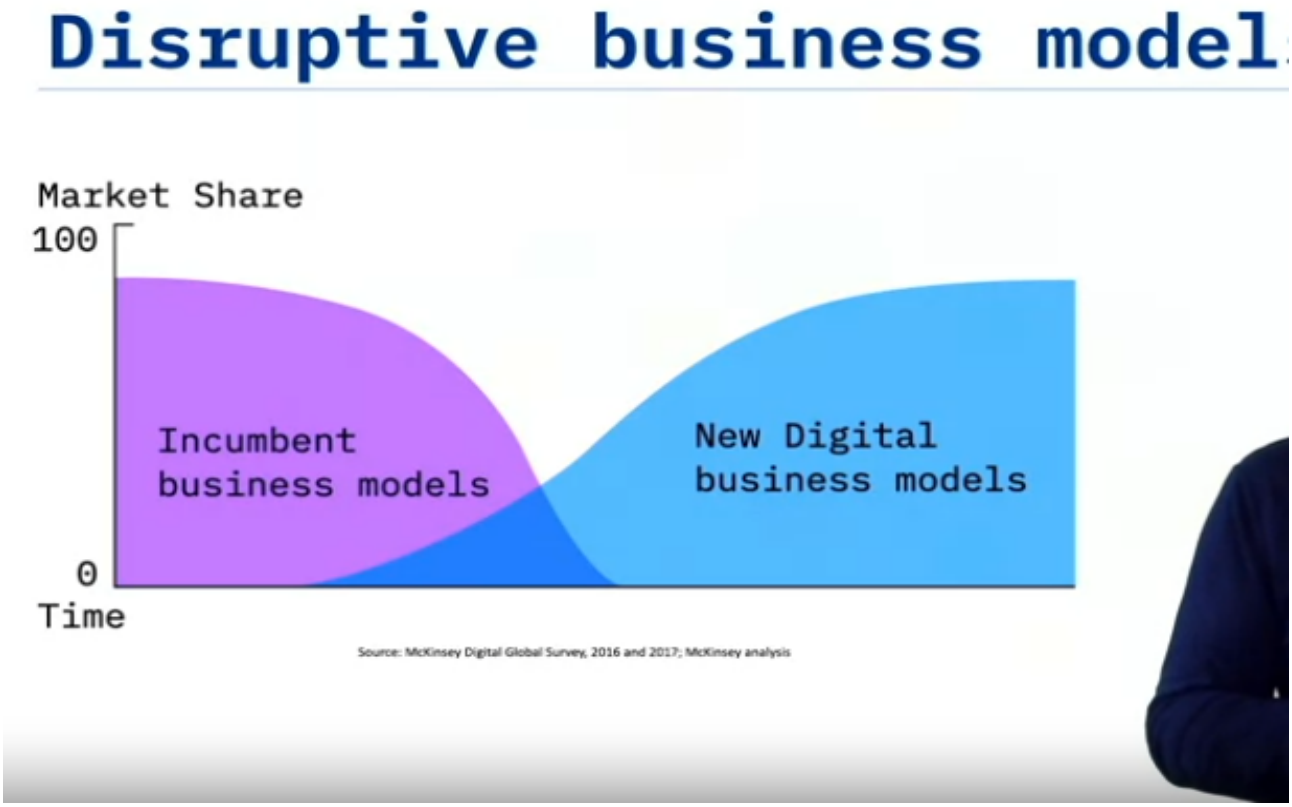
Feb 6, 2023

Dev Ops

Notes

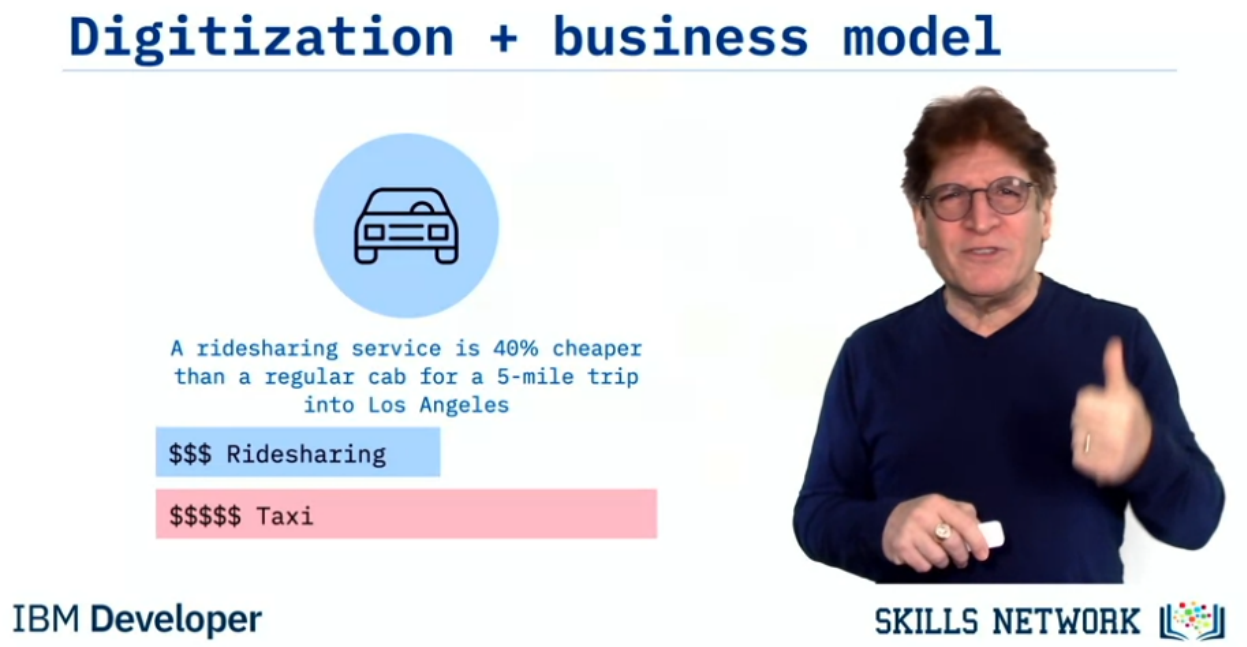
Business Case study for the company-

- **Describe how disruption facing companies today amplifies the need for DevOps.**

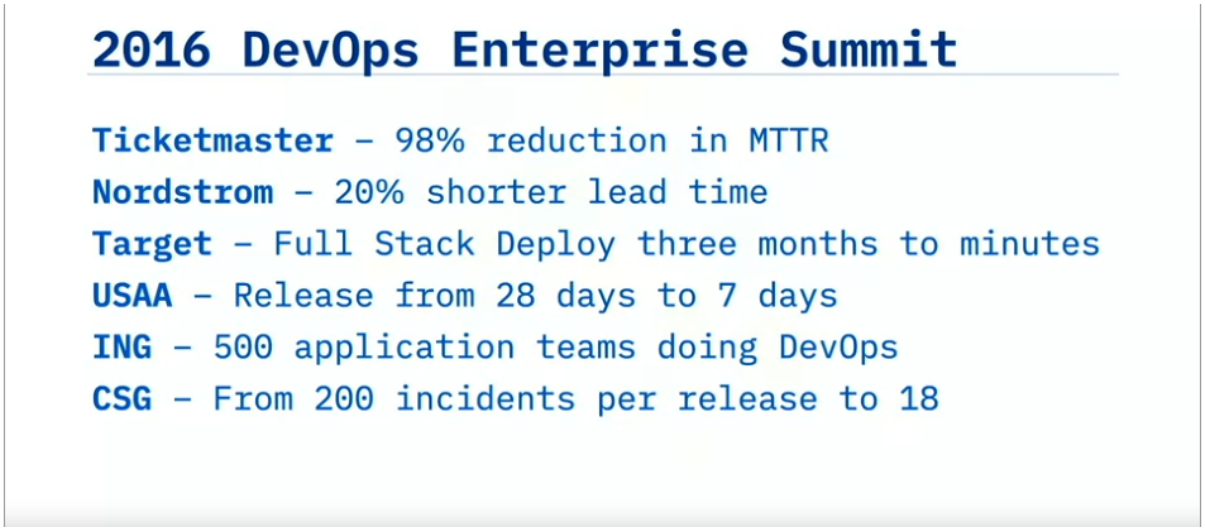


- **This is teaching us the innovations and implementation of the innovation of a industry.**

- **How things gone so far and improving all the sectors. This show how much development is important.**



- **but the message here is that technology is the enabler of innovation. It is not the driver of innovation. You must have a great business model idea and a great model and then leverage that technology. Technology alone is not going to drive innovation. It just enables it.**



DevOps is a recognition that development and operations must stop working in solos. They have to start working together. I like to say DevOps is the practice of development and operation engineers working together during the entire development lifecycle, following lean and Agile principles that allow them to deliver software in a rapid and continuous manner. I want to do it fast. I want to do it continuous. And I want to do it with Dev and Ops both together through the entire software lifecycle. To do that, we need to change to a culture of collaboration that values openness, trust, and transparency. We must adopt a new application design that does not require entire systems to be redeployed just to add a single function. You are not going to be able to deploy these large, monolith applications 10 times a day. We need automation that accelerates and improves the consistency of application delivery so that we can deploy and deliver our software with speed and stability.

- the Ops team and you're done.

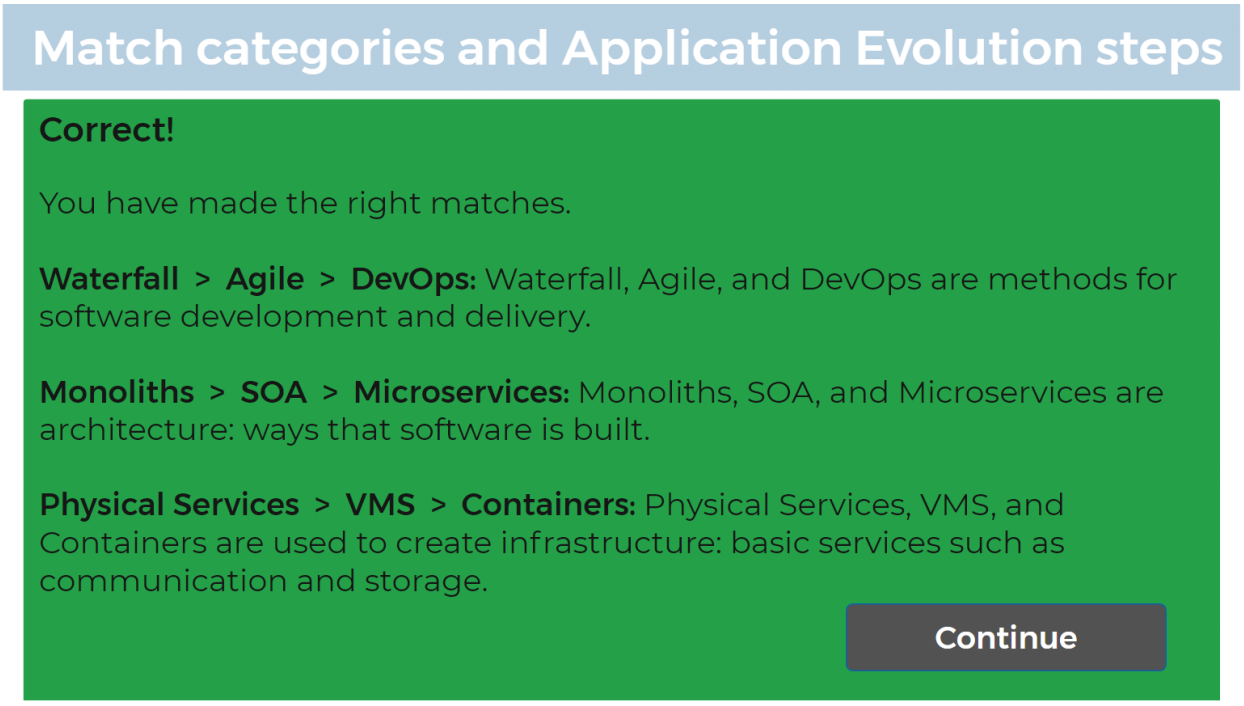
- DevOps is not a separate team. I have seen companies make DevOps teams. It is not a separate team. If you know anything about Agile, you do not make an Agile team. You don't say, "Oh, we have that team over there. They are the agile team. They make us Agile." No. Instead, the company becomes Agile. DevOps is not a team. It's just like Agile.

- And DevOps is not a tool. There are many tools that support DevOps. These tools can reinforce your DevOps culture but they will not change your culture. You cannot become DevOps by simply buying a set of tools.

- There is no one-size-fits-all strategy. This is what makes it difficult. You must find out what works for your business. Are you shipping shrink wrap software? Do you provide software as a service? Is your product a software that people download and install themselves? It will be different based on what you are delivering, and you might not be delivering software at all. You might be delivering a service that software just enhances.

- So, it's really important. Finally, DevOps is not just about automation. This is the one where the DevOps engineer jobs come in. People think that if they hire somebody who knows all the DevOps tools, then that person can automate everything and make them "DevOps."

- PIThis is not DevOps. The Devs are still doing the same thing. This is not just Ops automation. That’s all it is, automating Ops. DevOps is not just Dev and it is not just Ops. It is DevOps. One word, one team, one set of measurements.
- PIn this video, you learned that: DevOps is not just Dev and Ops working together while remaining in their separate silos. DevOps is a cultural change in which development and operations engineers work together during the entire development lifecycle.



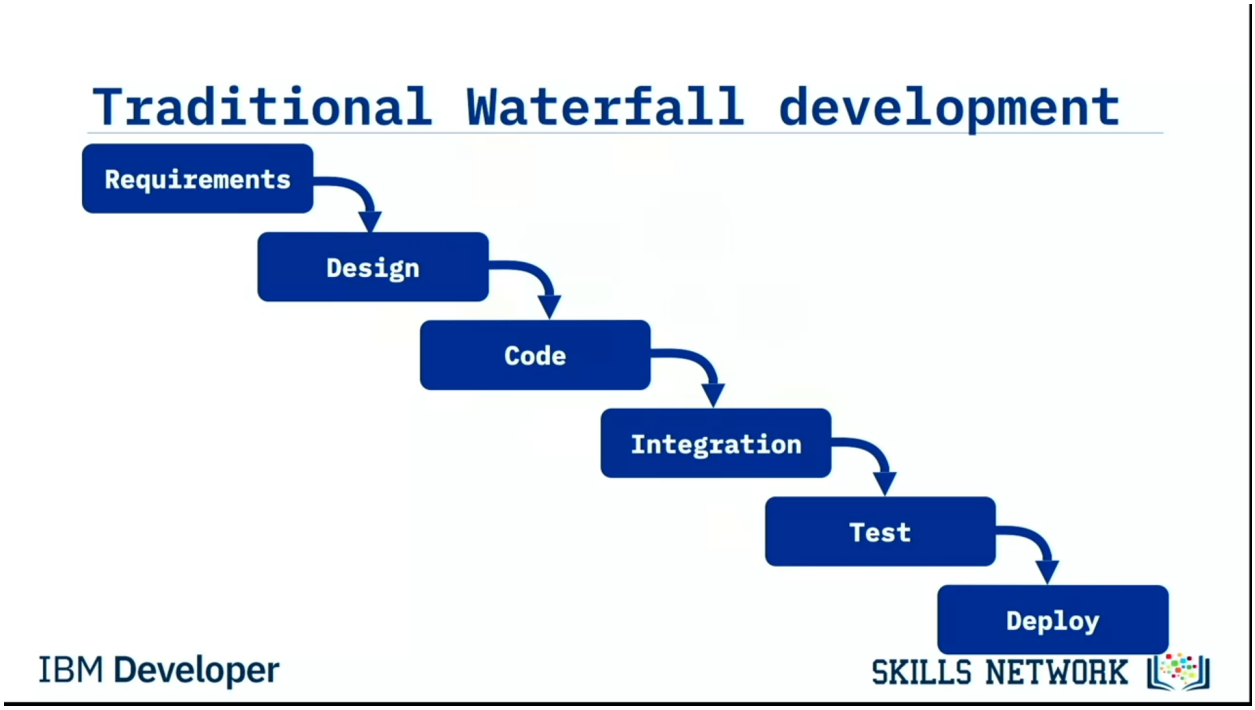
Leading Up to Dev ops

1. Architect desgins the structure using huge ample of time.
2. Then they do development for months .
3. Then it's send to the testing phase .After testing the code is send back for the development.

4.After’ testing phase, the code is released for the development.

Traditional Waterfall system Development.

Devops architecture design took worked for months



* Demerits of this system is we cant swim back and get back to the development phase which we have crossed.

* Not fix output in the coming phase. Till end it will work or not . No idea for it.

Problems with Waterfall system

very phase has entrance and exit criteria. When one phase ends, the next one begins. There's just no provision for going back and changing the design or changing the requirements or anything like that. Another problem is that you don't know if it works until the end. There is no intermediate delivery. Nothing is delivered, until that last step, when we give it to the operations team and say, go deliver this thing into production.

Play video starting at :3:48 and follow transcript

3:48

In addition, with each phase, there are issues that are just passed down. And, of course, every one of these is an opportunity to lose information, and to have a mishap. People get blocked, because they can't accept the work from the previous phase. Then the next phase is delayed.

Play video starting at :4:5 and follow transcript

4:05

Mistakes that are found later on are very, very costly, it is especially costly to find something that's designed wrong in the testing phase and go all the way back and redesign it.

Play video starting at :4:16 and follow transcript

4:16

Finally, these delays create long lead times between deliveries.

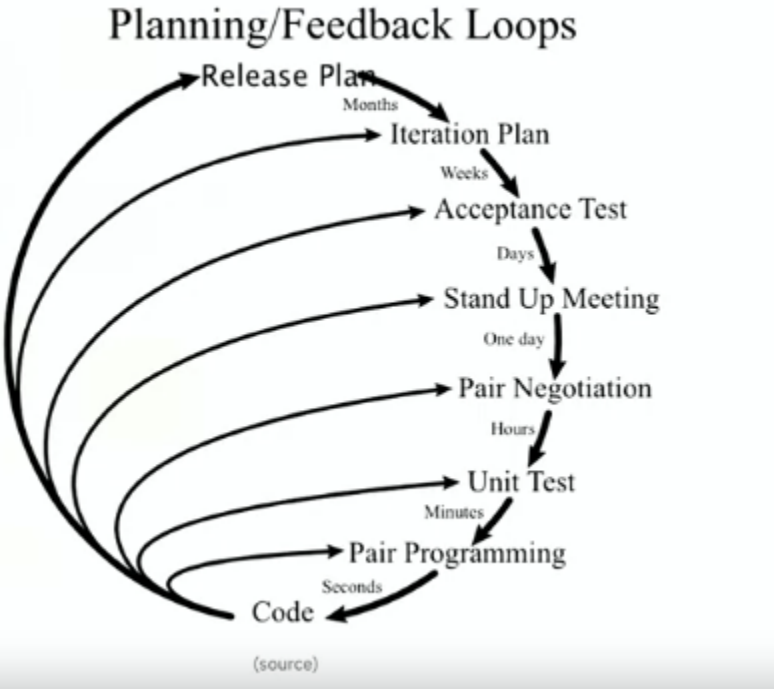
Play video starting at :4:21 and follow transcript

4:21

Also, the overall problem with Waterfall is that teams are working separately, and they're unaware of their impact on each other. Designers are unaware of the impact on the code, and the coders are unaware of their impact on testing or the impact on integration before we get all the code together. Everybody is working in their silos for their phase. And the scariest thought is that the people who are furthest away from the code, the poor operations team, have to deploy it, run it, and manage it in production. They know the least about the code, and they're the ones expected to run it.

Extreme Programming

In extreme programmming 1996 Ken beck introduced it.
In this programming we can programm and change the software source code from any part of the software.
Based on interactive software development
Intend to improve software quality and responsiveness to changing coustomers requirenments.
One of the best agile method



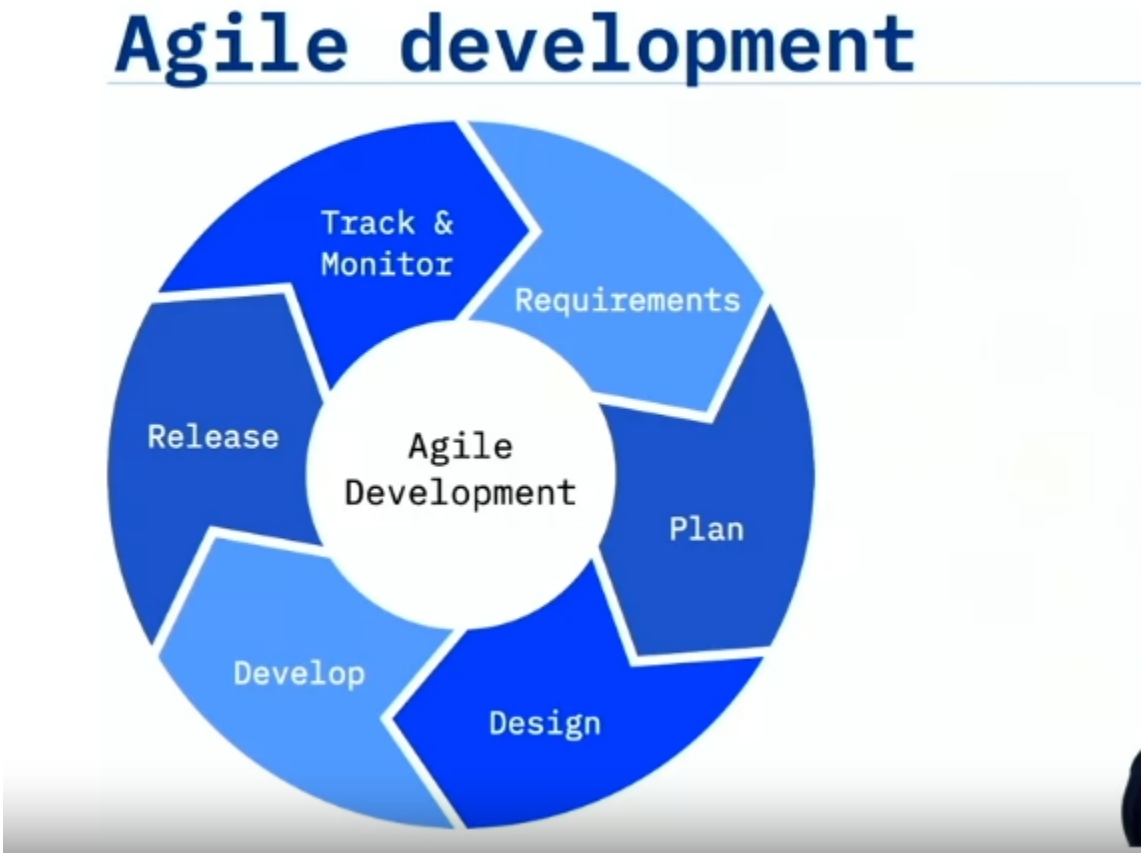
What is Agile manifesto ?

Its the manifesto where the top software developers came and gathered here for

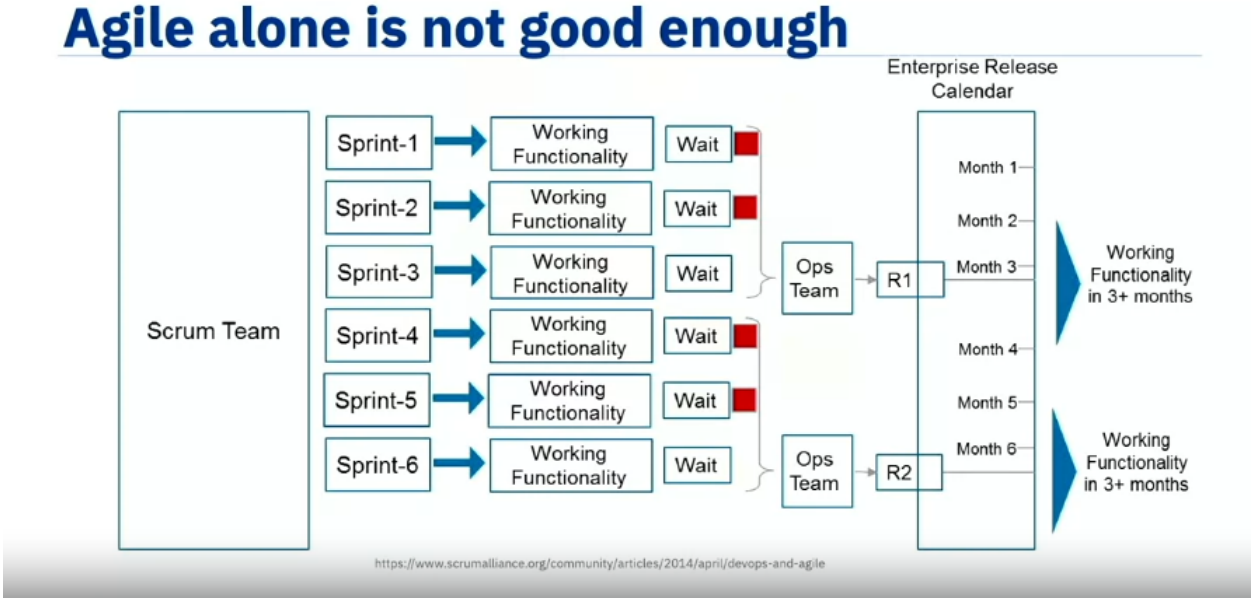
better software development and deployment.

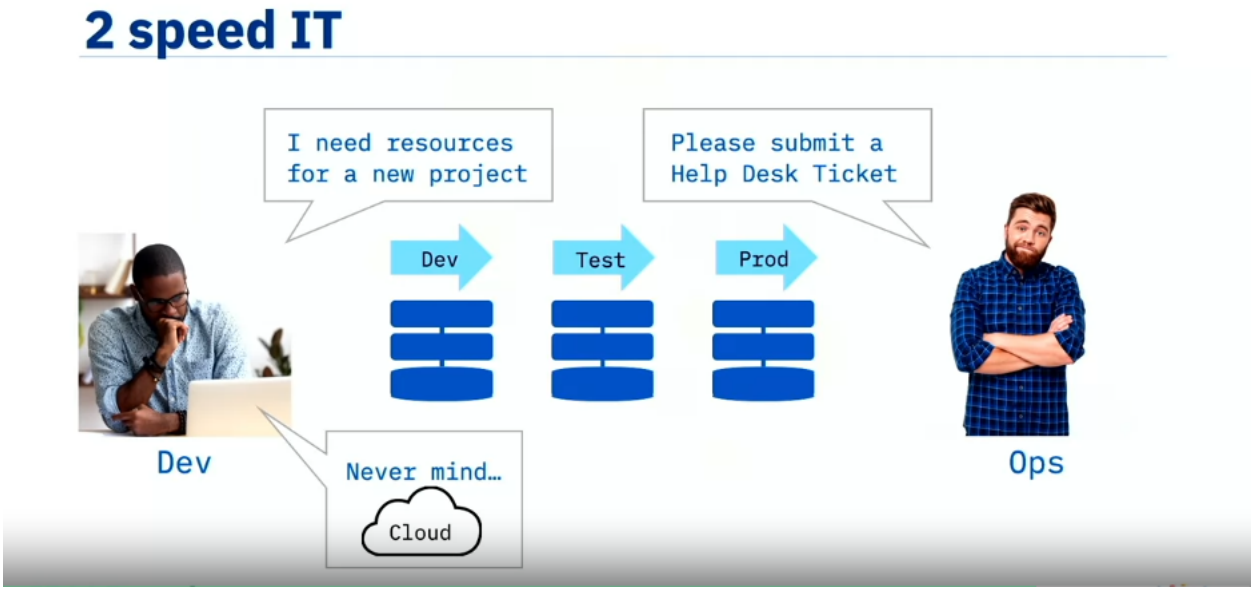
- Customers collaborations and work on negotiations .
- Responding to change the plan

Agile Development



* being agile doesnt solve the problem





Brief history of Devops.

The devops was started by Patric debois and many revoulutionary came

for the DEvops till 2019. For it. They said it can deploy 10 software through devops.Its a revolutionary .

Summary

Congratulations! You have completed this lesson. At this point in the course, you know:

Technology is the enabler of innovation, rather than the driver of innovation. You must have an innovative business idea to leverage technology.

-

In 2009, John Allspaw described an innovative approach to managing development and operations that enabled Flickr to complete over ten

deploys per day, when many companies were completing fewer than one deploy every six months. This was a key moment in the growth of DevOps.

-

DevOps is the practice of development and operation engineers working together during the entire development lifecycle, following Lean and Agile principles that allow them to deliver software in a rapid and continuous manner.

-

DevOps is not it is not just Dev and Ops working together. It is a cultural change and a different way to work. DevOps has three dimensions: culture,

methods, and tools. Of these, culture is the most important.

-

The essential characteristics of DevOps include cultural change, automated pipelines, infrastructure as code, immutable infrastructure, cloud native application design, the ecosystem of containers, and how to deploy with immutable infrastructure.

-

DevOps started in 2007 when Patrick Debois and Andrew Clay Shafer began to gather like-minded people together at conferences to talk about common experiences.

-

In 2009, Allspaw delivered his now famous “10+ Deploys Per Day – Dev and Ops Cooperation at Flickr” presentation and the idea gained ground. Also in 2009, Patrick Debois started a conference called DevOpsDays that helped spread the DevOps message.

-

Books such as *Continuous Delivery* in 2011, *The Phoenix Project* in 2015, and *The DevOps Handbook* in 2016, helped practitioners understand how DevOps worked.

-

The major influential people of the early DevOps movement: Patrick Debois,

Andrew Clay Shafer, John Allspaw, Jez Humble, Gene Kim, John Willis, Bridget Kromhout, and Nicole Forsgren, went out and made a difference, showing the results that could be achieved with DevOps.

-
- The message spread from practitioner to practitioner until they began to realize what was possible with DevOps and that it was a better way to work.

Week 2

Social Codings.

The Social Coding principles involves contributing in Open Source. Learn in Public Type culture and environment.

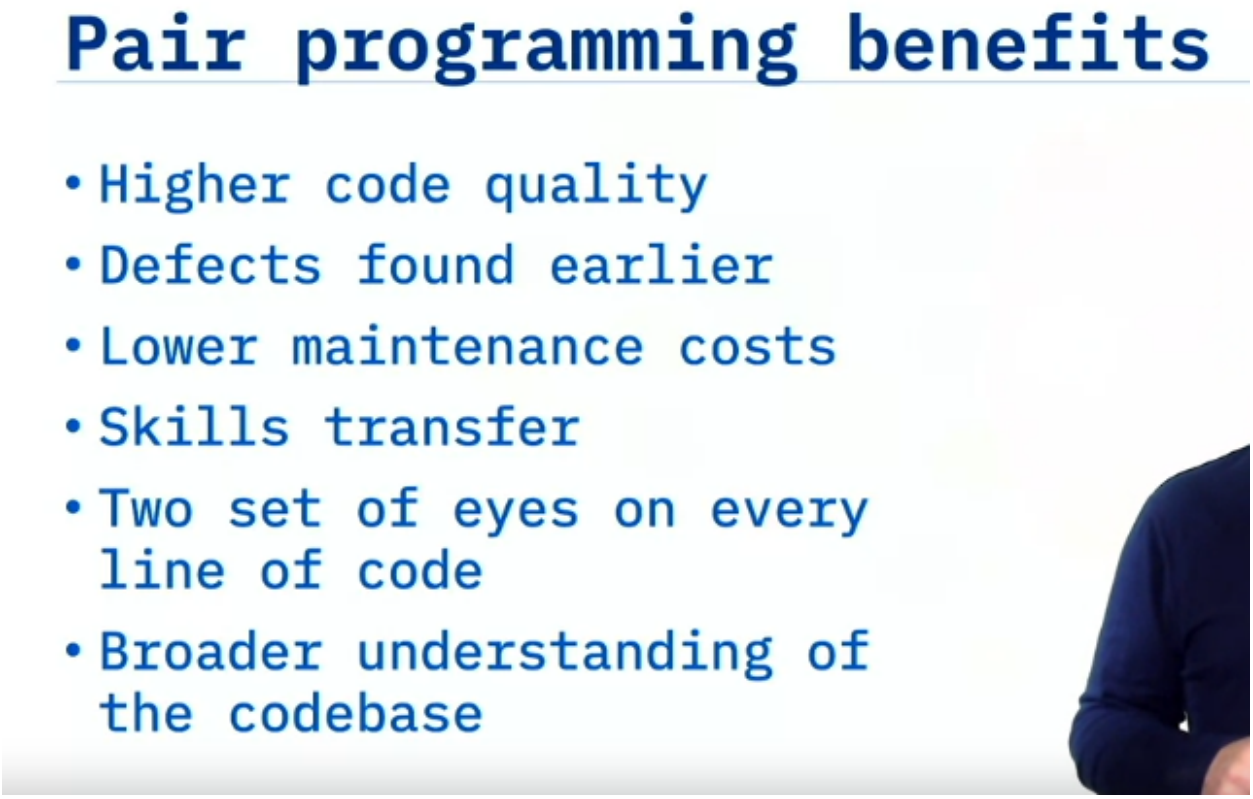
This is because of the Social coding nature and principals.

There is narrative and Driver in a Development community . This vary Like Driver and narrative work on same workstation. Driver writes the Code and narrative reads the code

How does adopting social coding principles solve this? You discuss the new feature with the repo owner, and you agree to develop it for them. This allows you to leverage everything that they've done and add the feature that you need. You open up a GitHub Issue and you assign it to yourself so that everyone knows you’re working on it. Then you fork the repo, create a branch, and make your changes. When you’re all done and have something to contribute back, then you issue a pull request signaling that you are ready for a review and the repo owner can decide whether to merge your code back into the main project. The repo owners are in full control. They can ask for changes because they do the merge. They can ask that you write more test cases if you don't have adequate test coverage. They treat you and your contribution like any other member of the team. This is a win-win situation. You get to leverage another team’s code and all the functionality that you need, and the other team gets a feature added for free. The company saves money

because code is being reused instead of rewritten and everyone is happy. This is how open-source works and this is how companies should treat their inner source. Pair programming is an aspect of social coding that is taken from Extreme Programming.This allows you to leverage everything that they've done and add the feature that you need. You open up a GitHub Issue and you assign it to yourself so that everyone knows you’re working on it. Then you fork the repo, create a branch, and make your changes. When you’re all done and have something to contribute back, then you issue a pull request signaling that you are ready for a review and the repo owner can decide whether to merge your code back into the main project. The repo owners are in full control. They can ask for changes because they do the merge. They can ask that you write more test cases if you don't have adequate test coverage. They treat you and your contribution like any other member of the team. This is a win-win situation. You get to leverage another team’s code and all the functionality that you need, and the other team gets a feature added for

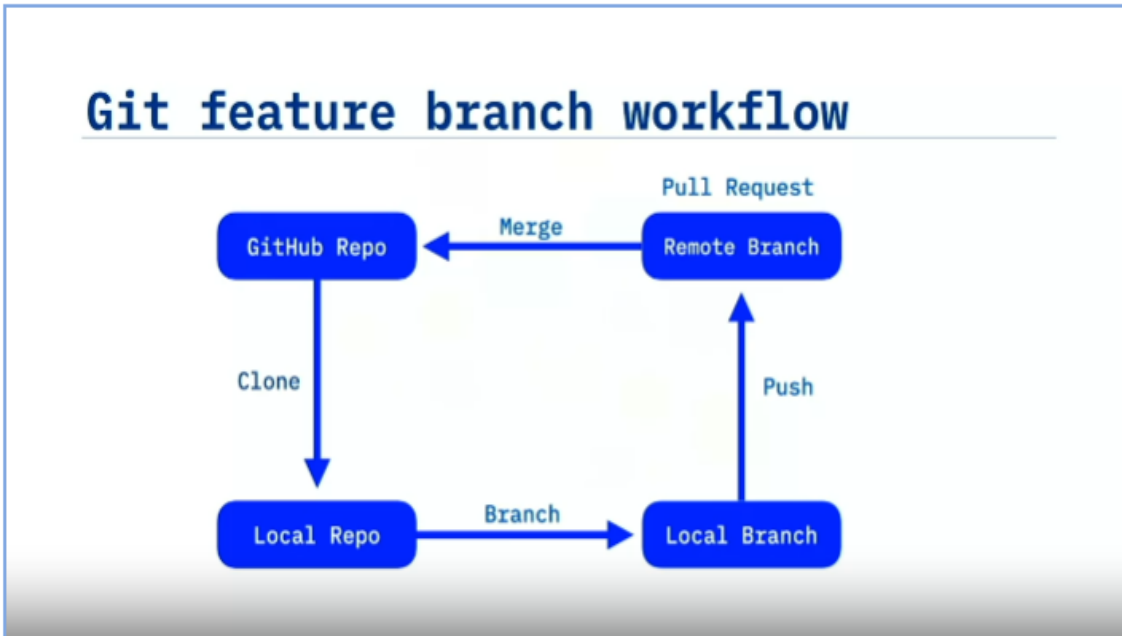
free. The company saves money because code is being reused instead of rewritten and everyone is happy. This is how open-source works and this is how companies should treat their inner source. Pair programming is an aspect of social coding that is taken from Extreme Programming. It consists of two programmers sharing a single workstation (one screen, one keyboard, and mouse between the pair). The programmer at the keyboard is usually called the “driver.” The other programmer, also actively involved in the programming task, but focused more on the overall direction, is called the “navigator.” While the driver is typing, the navigator is checking their work, or perhaps looking something up, or thinking about what's coming next. Then, after about 20 minutes they swap roles. This way they both get to play each role. I pair program whenever I can at work. I like this aspect of social coding.



Git Repo guidelines

- In this we learn about Git And adding branch in Github. So how we can contribute to open source. So how Git work branch flow help social coding.
- Create a repository in github for every component. Dont make in one create separate.
- Dont put multiple microservices in single repo. This is called mono repo
- Use Pull request to merge to master.Every pull request has an opportunity for a pull review.

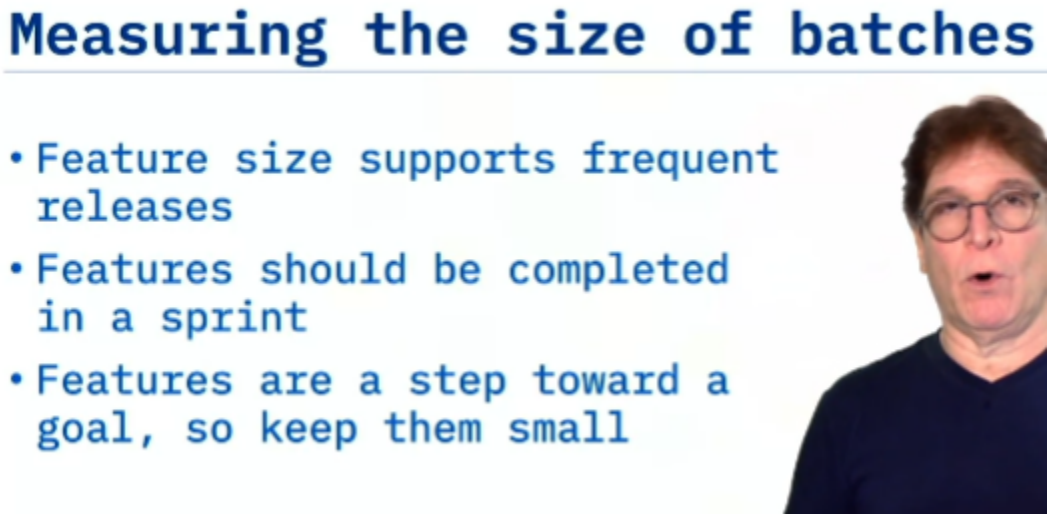
- If we are using some one else Git repo Fork it and this is how it begans.



Working in Small batches.

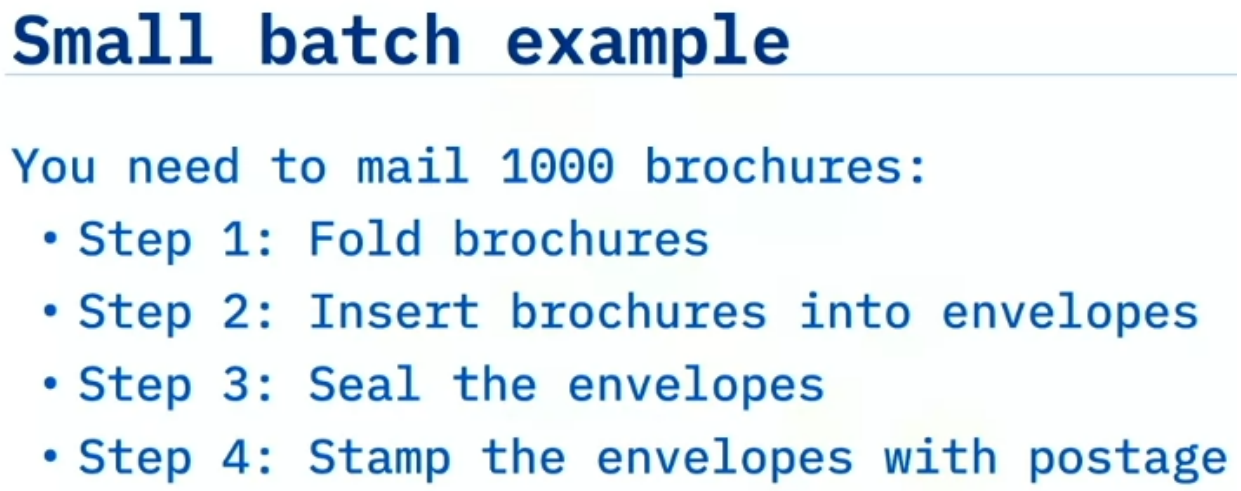
In small Batches we get feedback quickly. The work is done faster and We can deploy the code and software more faster.

How to know our batch is small or not/?



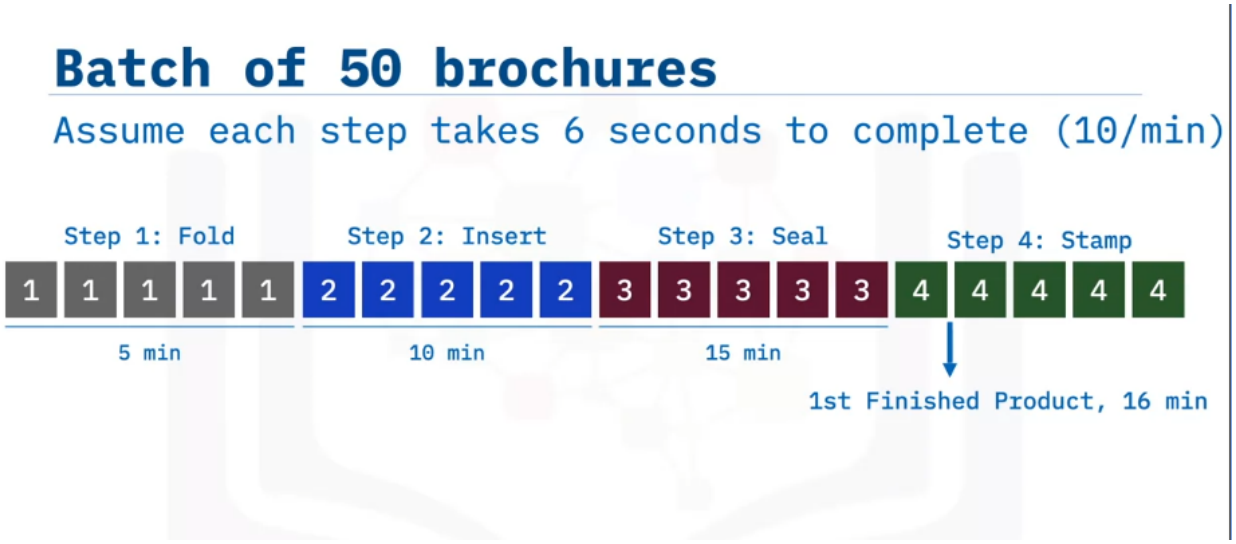
Benefits of working in small Batches?

The benefits of working in small batch is like , in pictures lets take an example , here we can see that packing a Stamp card is how much difficult.



Now see:

Now in this process we can see how the process is done
A envelope is made in 4 step process. Like Unlike we can see that each step for one Brochures take 6 sec.
So 5 min for 50 brochures in 1 step.
In Setp 2 5 more min overall it took 16min.

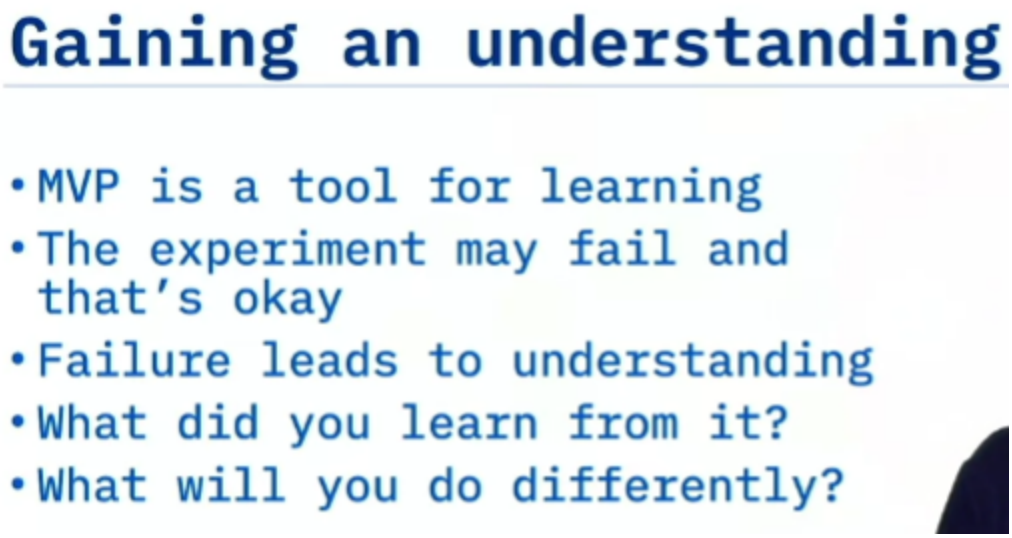


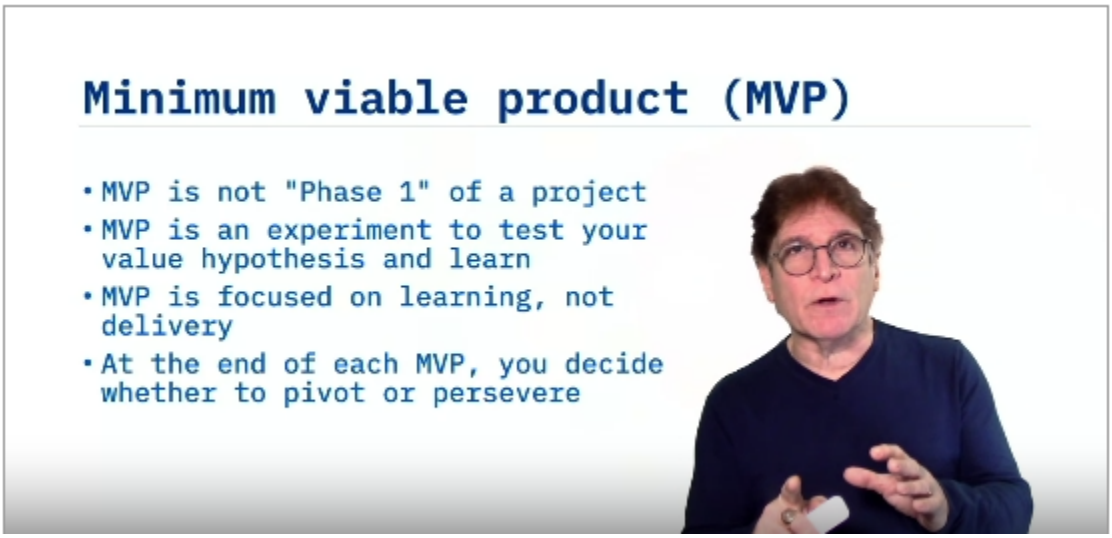
So here we can see that how Single step process takes more time than earlier. Where In single step all type of folding and other activities are done.

So how do you know if your batches are small enough? I would start with the size of your stories in your backlog. Are your application features decomposed in such a way as to support frequent releases? You have to ask yourself, how often are releases possible? Are there delays in the release because the features are too large? Can features be completed in a sprint? If you have features that take several sprints, then your batches are too large. Optimally a feature should be small enough to be completed in a week or less. The features you build are one step toward a goal. Many people feel that only a completed goal is worth shipping. Instead of thinking of useful subsets that can be delivered in increments to gain feedback toward the ultimate goal.

MVP

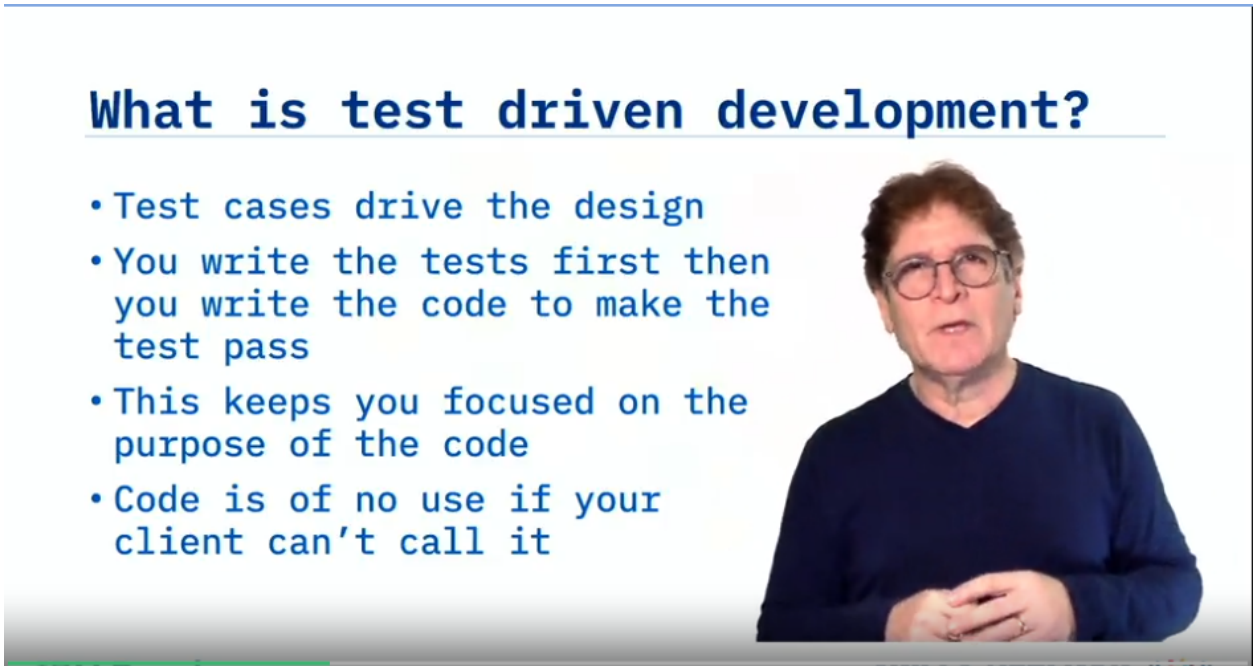
- Its not phase 1 of the project.
- Its minimum value hypothesis of understanding and learning





Test Driven Development.
What is TDD?
How TDD Produces higher quality code?
Describe red,green, refractor workflow?
What is the importance of TDD in Devops?

Tdd is test driven development is the testcase already defined for the coding means test case drive and ddesgin the code.



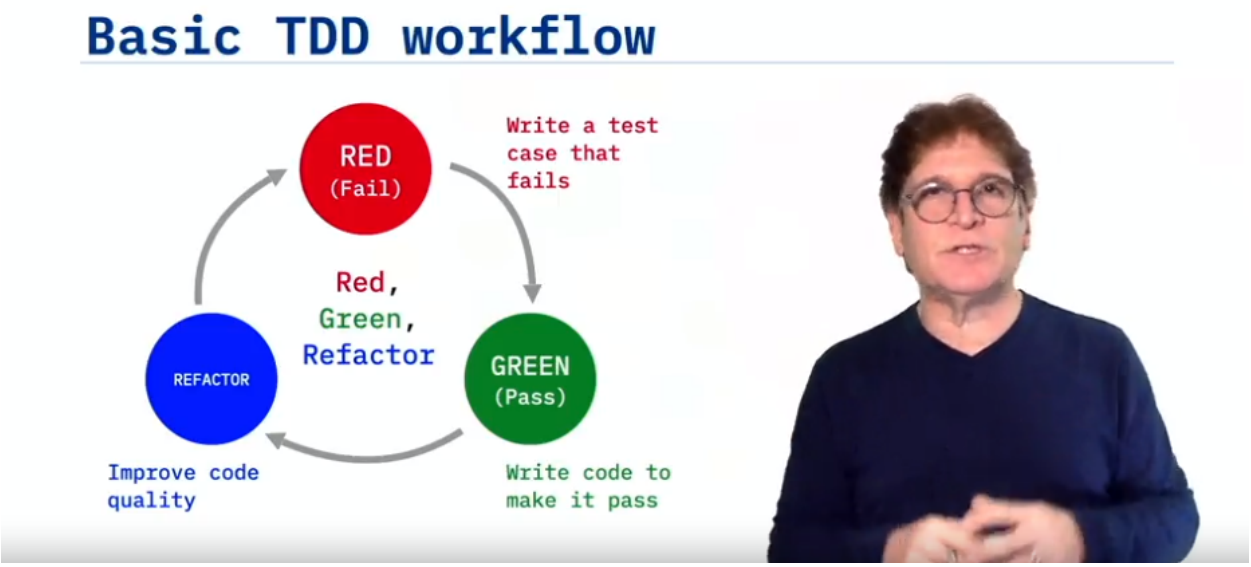
Why developers dont test:]

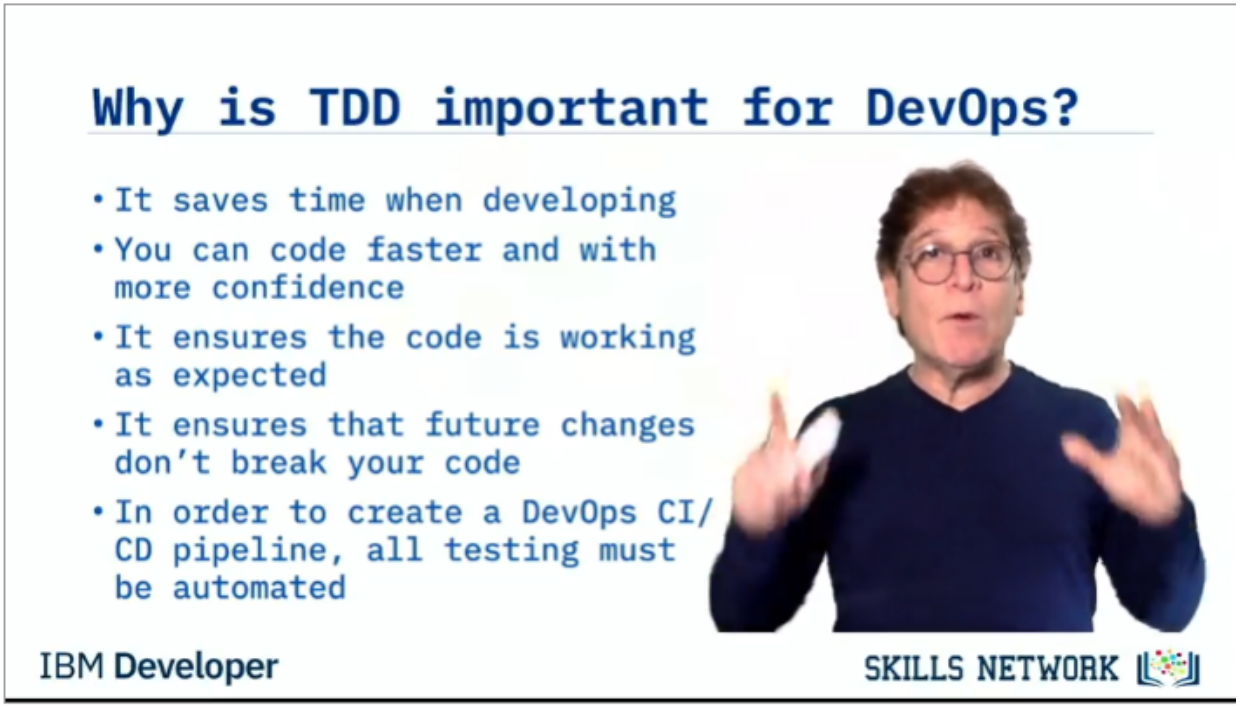
*Always test the test case while developing

*there is nothing like no time learn to deploy test case first as per user requirement .

Time spent of these test cases saves hours and hours debugging later.

TDD workflow





Finally, and most importantly, in order to create a DevOps pipeline (a CI/CD pipeline), all tests must be automated unless you want to push bugs to production faster. A lot of companies don't understand this. They want to automate using Continuous Integration (CI) and Continuous Delivery (CD), without automating their test. Unfortunately, you can't have a CI/CD pipeline without automating your tests

Behaviour Driven Development

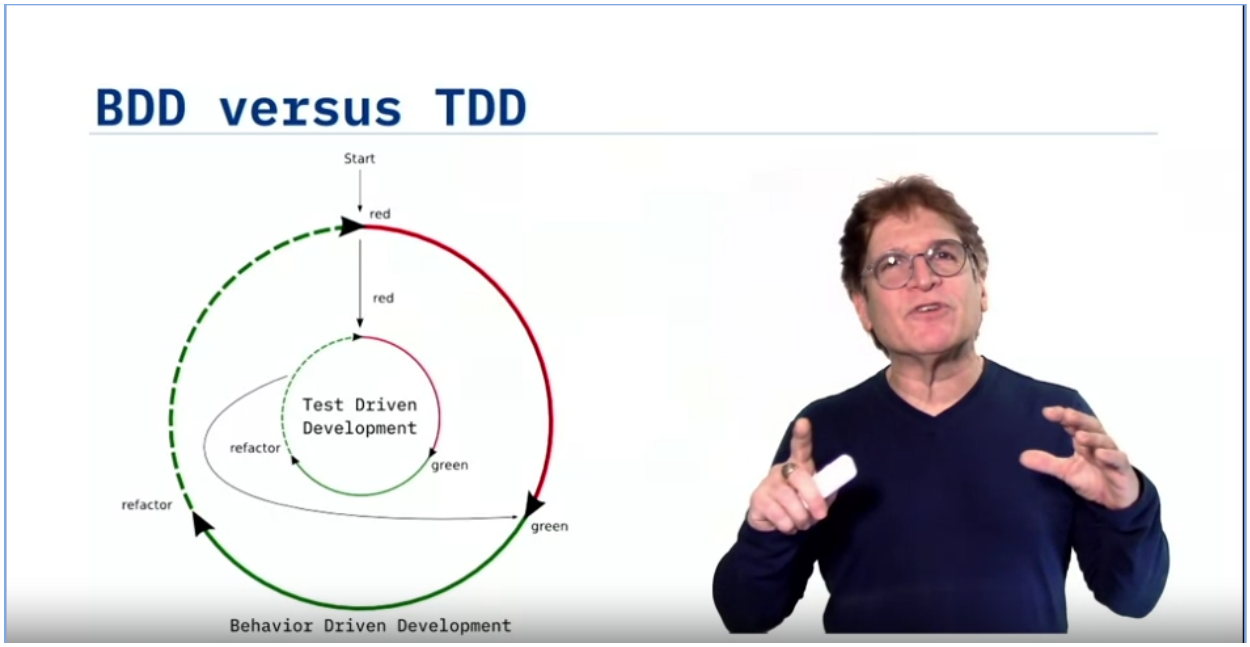
Describes behaviour from outside of the system to inside

Great for integration testing.

Uses syntax that both developer and stake holder can understand

TDD is vice-versa of BDD

And remember BDD is completely different.



Tdd tells all working correctly and BDD tells that all work areon higher level

BDD versus TDD

- BDD ensures that you are building the "right thing"
- TDD ensures that you are building the "thing right"

Gherkin syntax

- Given (some context)
- When (some event happens)
- Then (some testable outcome)
- And (more context, events, or outcomes)

Gherkin for acceptance criteria

- Add acceptance criteria to every user story
- Use Gherkin to do that
- Indisputable definition of "done"

IBM Developer

SKILLS NETWORK