**Total 24 classes**

**Software**: takes the input from the user and act according to the instruction.

**Software development**: gathers the inputs and requirements from the user/ clients and develops the software.

**Software life cycle: 1. Requirement** (the need and purpose) 🡪2. **Design** (how it should be and which tool to be used to develop the software) 🡪3. **Implementation** (actual development of software by writing the code) 🡪 4. **Verification** (checks whether the developed application works as per the client’s requirement) 🡪 5. **Maintain** (Maintain and monitoring the software)

**Software development life cycle models:**

**Waterfall Model life cycle:** linear sequential model i.e until first phase completes we cannot start the next phase. There is no going back, once done is done.

**Steps: Feasibility check** (same as requirement analysis) 🡪 **Analysis** (Technical people/ architect will analyze the details which has been decided in the requirement phase whether it is feasible or not) 🡪 **Design** (how the software going to be, what tools can be used and etc.) **🡪 Testing** (unit testing, system testing) 🡪 Maintenance (maintain and Supporting the software in case of any issues)

**Advantages:** clear objectives, well understood milestone and the documentation will be very clear, each and everything will be captured in the document.

**Disadvantages**: software will not be available till the final stage is completed. The requirement keep changes in over time of period. High risk and uncertainty

**Agile**- To overcome the disadvantages of waterfall model agile have been introduced in 2000. Developing the software feature by feature. Creating the shorter development life cycle. Will be taking the customer feedback frequently in the process of developing the software.

**Steps**: Process request 🡪 design 🡪 coding 🡪 Test and debugging 🡪 Release and Deployment

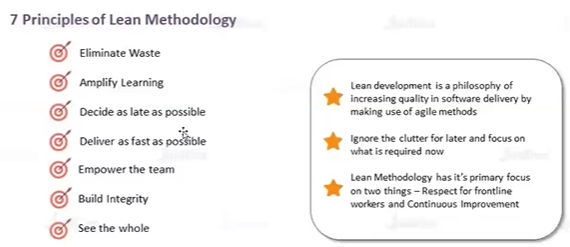
**Advantages:** customer satisfaction is high. Requirements can be dynamic in nature, less planning is required.

**Disadvantages**: KT can be difficult as there is no documentation, not suitable for complex dependencies project.

**Lean**: Same as Agile model. The only difference is eliminating waste. So the process will be faster than agile.

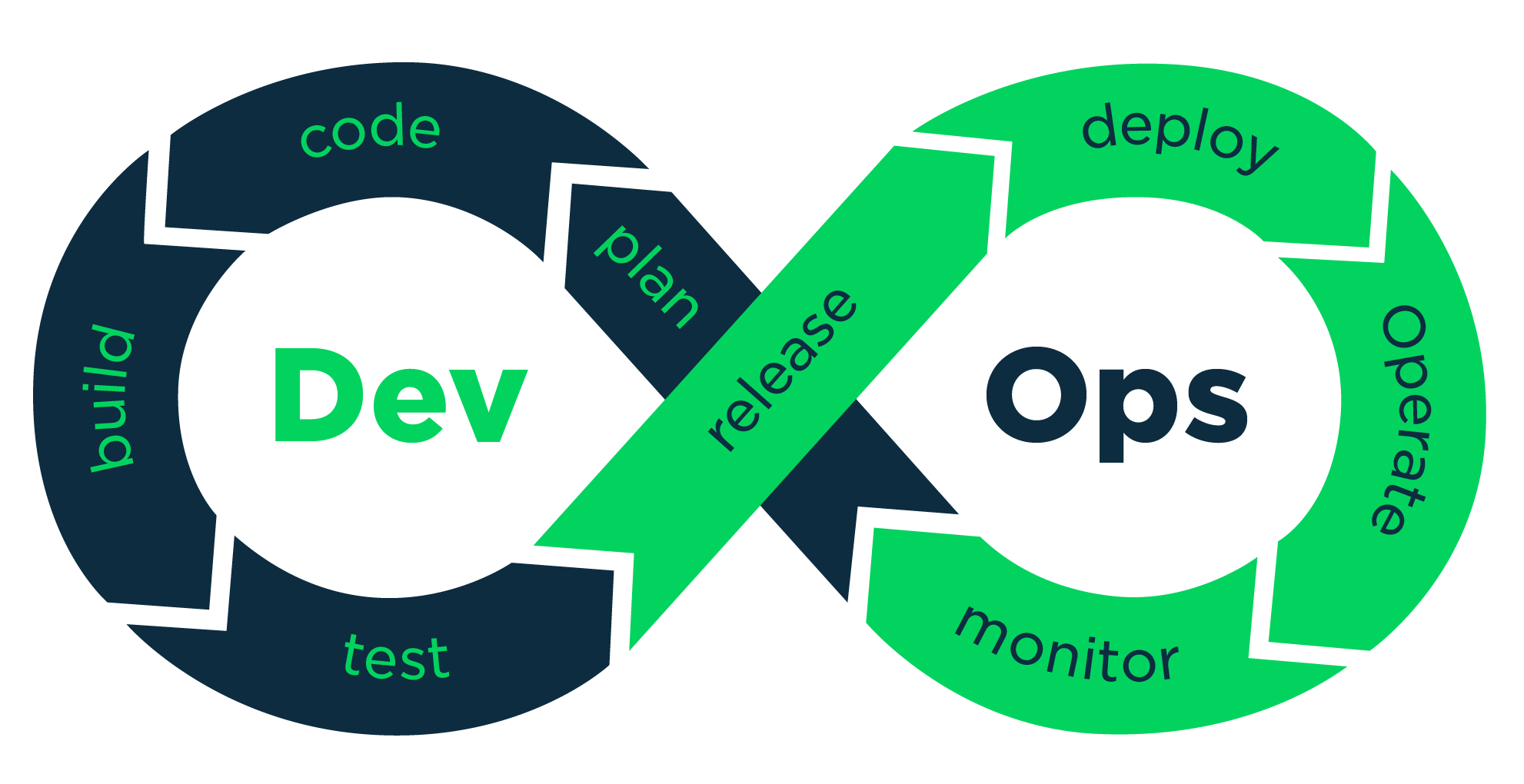
**Advantages**: same advantage as agile. Creates positive working environment, limiting wastage saves the time and money.

**Disadvantages:** Strong skill set team is required. No room for proper meetings and etc.



**WHY DEVOPS:** Developers will write the code for the software and handover to the operation team to do the deployment on the servers for end users. But when Ops team tries to deploy on their machine the code may not work due to some reasons. To avoid such kind situations DevOps came into picture.

**What is DevOps: which improves the collaboration between developers and operation team using the various automation tools.**



**Plan:** getting inputs from the customer/ clients (i.e gathering requirements).

**Code:** writing the code for developing the software.

**Build**: convert the code into executable format.

**Test**: Testing the software whether its works fine or not

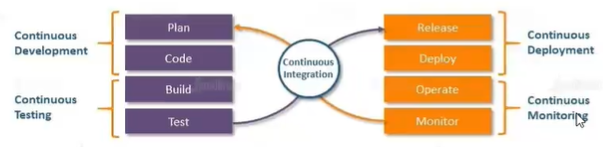
**Release and Deploy**: can be internal and external release. Internal relese will be within their own team to check if everything is ok. Once all fine, the software will be moved to external release (i.e releasing the software for end user)

**Deploy**: deploying the software in the server to make it available. Release and Deploy will happen together.

**Operate**: whether the applications works fine or not by checking the traffic and other things.

**Monitor**: Monitoring the application.

**How devops works?**



Devops is developed based on the agile and follows the same process except the containerization.

**Devops Tools:**

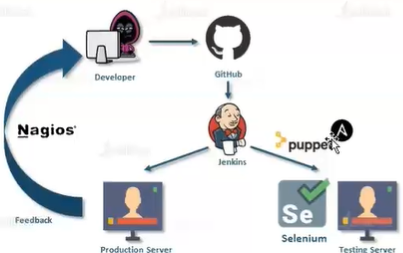
Continuous development: git hub, MAVEN (build tool)

Continuous Integration: jenkins

Continuous Deployment: ansible, terraform, kubernetes and Docker

Continuous Testing: self-paced classes

Continuous Monitoring: ELK



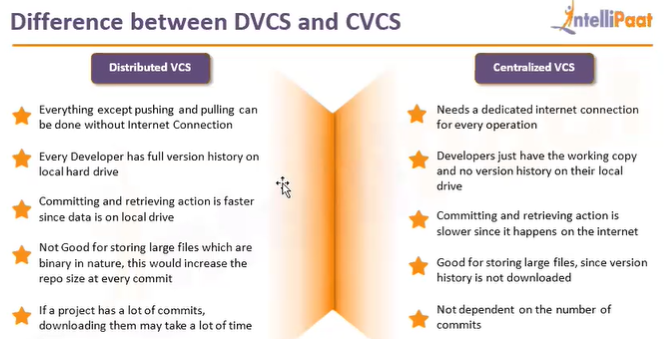
**Version Control system:**

Different developers will write the code and place the codes in one centralized location called GIT. So that other people in the team may aware of what other people are working and it is easy to track who has done what.

There are **2 types** of version control system.

1. Distributed VCS (ex: GIT)
2. Centralized VCS (ex: SVN-apache version)

In both methods developers will push and pull the codes from the centralized location/server. And the developers will always needs to be connected with the centralized location/server. The only difference is, in centralized vcs we need to be connected with centralized location/server even when we are working but in distributed vcs we need to be connected with centralized location/server only when we are pushing or pulling no need to be connected while we are working.



**Cluster**:

Collection/ group of machines/ nodes are called as **cluster**.

**Basic Linux commands:**

* To change the current user to root user- ***sudo su***
* To change the hostname ***– hostnamectl set-hostname name***
* To list the available files- ***-ls***
* To get detailed info about the available file ***– ls –ltr***
* To create a directory- **mkdir directoryname**
* To go inside the directory- **cd directoryname**
* To come outside of the directory- **cd ..**
* To see the hidden files/folders inside the directory- **ls –ltra**
* To create, edit, update file- **vi filename** (vi is an editor)
* To save and come out of that file- **press esc button then** **:wq**
* To see the content of the file- **cat filename**
* To come to previous directory- **cd -**

**GIT**: is called as SCM or VCS. Manage the versions of the code. Used to make collaboration with the team. In simple words GIT is a version control tool, to track who has done what.

Installation of GIT: **yum install git**

To know the version of installed GIT: **git –version**

To initialize the GIT- **git init**

To know the status**- git status (important)**

To move to staging area**- git add name of the file**

To commit a file**- git commit –m “message” name of the file**

**Error 1:** while doing the first commit we may face below error

Author identity unknown,\*\*\* Please tell me who you are. Run git config --global user.email "you@example.com" git config --global user.name "Your Name" to set your account's default identity. Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'root@agent.(none)')

**To resolve this just follow below steps:**

git config --global user.email [hellogirllara7@gmail.com](mailto:hellogirllara7@gmail.com)

git config --local user.name "hellogirllara"

git config –list (to check whether the user name and email go updated or not

To see who did what**- git log**

To see/ know about one commit- **git show “commit id”**

To move back from commit and staging are to working area- **git restore –staged filename**

**NOTE:** keep ruuning the git status command to know whats happening.

**Lifecycle of GIT:**

**Working directory** (is the place where we have initialized the git –untracked files) 🡪 **Staging area** (tracked files and ready to commit) 🡪 **commit (**we can’t make any changes to the file once the commit is done)

*In Git after moving to staging area if we are making any changes to the same file, but we are trying to commit means all the changes will get committed without doing staging also as git works on file level.*

To uncommit or restore the latest changes**- git reset commit id –hard** (we cannot use the latest commit id to remove the latest changed, we should use the previous commit id to delete the latest one.)

**Head** refers to the latest commit id.

**GITHUB:** is a centralized location where developers will place their code.

**Repository**: collection of files/ directories/ folders. There are 2 types of repo.

1. **Local** repo which is on our local machine
2. **Remote** repo which is available in the github

A repository can either public(people who has the url can access) or private(only who has access).

**Creation of repository:** just click on create repository in github, enter the name and click on create repo.

Pushing the committed file from our local to remote repo: **git push url of our repo master** this remote repo is called as GITHUB

After entering this command it will ask for the username and password. Username is our github username, and for password we need to generate the token.

Steps to generate the token:

Go to your profile right hand side- click on settings scroll down and select developer settings – select personal access tokens- under that select Token classic- then in right hand side click on generate new token (classic)- after that enter a name-choose the expire date- click all the check boxes- then click on generate token- save the token for future purpose.

**Master branch: (Important for interview) - why do we need to create a branch?**

The branch which is created first becomes the master branch. In the master branch, live code will be available which is running successfully and used by the end users. When a developer or a product owner wants to do some enhancement, we can’t go ahead and edit the master branch as the live code is running without any issues. So in such case we create a new branch from the master branch. As we creating new branch from the master all files will be inherited to the newly created branch.

**How do we name a branch?**

When we create a repo, the master/ main branch will get created automatically. Naming of the branch is based on the environments we works like (dev, Test, staging, preprod, PROD-Master)

Prod- where the live (bug free) code wil be available and used by the users.

Dev- where the developers write the code, fixing the bug.

Test- where the testers will test whether all the functionalities are working as expected or not.

Branching names can be based on the features also. Like feature/status, feature/message, etc.

**Branching Strategies:**

1. **Multi branch Strategies like** prod, dev, test
2. **Single branch Strategies** will have only one master/main branch where the live will be available (not recommended) - It is easy to manage in the high level. Different stages will be created in the available branch. If the condition passes in one stage then it will be moved to next stage.

To see on which branch we are currently working on- **git branch**

To create a new branch**- git branch branch name**

To switch to another branch**- git checkout branch name**

To create new branch and checkout to newly created branch- **git checkout –b branch name**

To delete any branch- **git branch –D branch name** (we cant delete a branch where we are currently working, if we want delete we need to switch to another branch then delete that one)

Instead of copying and using the url every time we push the files to remote repo, we can create and use the alias name for the url.

To create an alias name for the url- **git remote add [aliasname] [repo url]**

**UnCooked file:**

In dev there is one file created in dev but it is not yet committed, meanwhile if we switch to any other branch we can see the uncommitted file which available in dev. That particular file is called as uncooked file. In this scenario we cant commit any new changes until that file gets committed.

To avoid this issue, first we need to switch to the branch where we have uncooked or uncommitted file. Then we can use stash command- **git stash –u** (u- untracked)

If we want to work on the uncommitted file, to bring back that file- **git stash pop**

To get a remote repo from github to our local- **git pull url of repo**

To delete any file – **rm –rf file name**

To get any other repo apart from master branch we need to use clone command- **git clone –b branch name url repo** – when we use this clone command it will automatically create new directory then pulls the files from the particular branch

To get any particular file from particular branch- **git pull url of repo branch name** (from which branch we want to pull any file)

To push any branch from our local to remote repo github: **git push url of remote repo branch name**

To delete any branch in our remote repo from our local cli: **git push –d url of our remote repo branch name (which branch needs to be deleted)**

If we want to have the committed files from one branch to another branch we cann use the merge command. Before running the merge command we need to checkout to the branch where we wanted to have the committed files- **git merge branchname(from where we wanted to merge)**

**PR (Pull Request): (important)-** it is used to merge the file from one branch to another in the Github directly in the GUI without using the cli.

Click on the PR- select from where we want to merge the file- enter the message – add the reviewers and assignee – then create a pull request- once done the PR will be in open state- if no issues it will show merge pull request- then confirm merge.

For security, we can add the branch protection rules as per the requirement.

**Conflict:**

We will face the conflicts, when 2 developers are working on same file in the 2 different machines, and tried to merge their files back to the main branch. To resolve the conflicts git provides **mergetool**. To avoid this situation everytime before we start working on any branch or life will make sure it is updated.

**Git mergetool-** once we enter it will show us all the 3 files, the original file, changes done by developer1 and developer2. As a devops person or 3rd person we cannot decide which line can be removed and which line need to be kept. So here just remove the extra spaces in the final file then save it **:wq**. Remaining files just do **:q**.

Once done with above step, we will be able to see one extra file with the extension of .orig. we can delete this file as it is no longer needed.

Then we need to commit the final file. Here if we are giving name of the file or . at the end we will get error like “fatal: cannot do a partial commit during a merge.” So no need to mention the file name.

**Rebase: Git rebase branchname**(from where we are merging the file)

Rebase same like merge only. When we use merge we can see the detailed explanation about the file, from which branch merging has happened but if we are using Rebase there will not be much info about the merge or file.

**Fetch Command**: is used to get file from github gui to our local cli

We need to 2 commands to get the file to local.- **git fetch url of repo branch name + git merge url of repo branchname(from where we are pulling file)**

**Forking:** cloning the repo from one github account to another github account.

**Revert**: **git revert commitid.** (Recovering the old change)

To save the username and password: **git config --credential.helper store+ git cofig –credential.helper ‘cache --timeout=3600’** (using the above commands we no need to use the user name and passwords everytime)

To set the email ID or username: **git config --global user.email “**[**lathajasesh@gmail.com**](mailto:lathajasesh@gmail.com)**” instead of global we can use --system for all users and all repositories in one machine. --Global is for one user all repositories.**

To unset the email ID or username: **git config --global --unset user.email “**[**lathajasesh@gmail.com**](mailto:lathajasesh@gmail.com)**”**

Tags: when we do a production release, in case of any issue to fix the bug we will create a tag using **git tag v1.0 (name of the tag**) then we need to push the tag **git push urlrepo tagname.**