SDLC:

Software Development Lifecycle Cycle: model for project management

**Waterfall Model: Linear**

Requirement Analysis

Design

Implementation (40)

Testing (43rd day)

Deployed

Maintenance

1. Rigid
2. Not flexible

**AGILE**

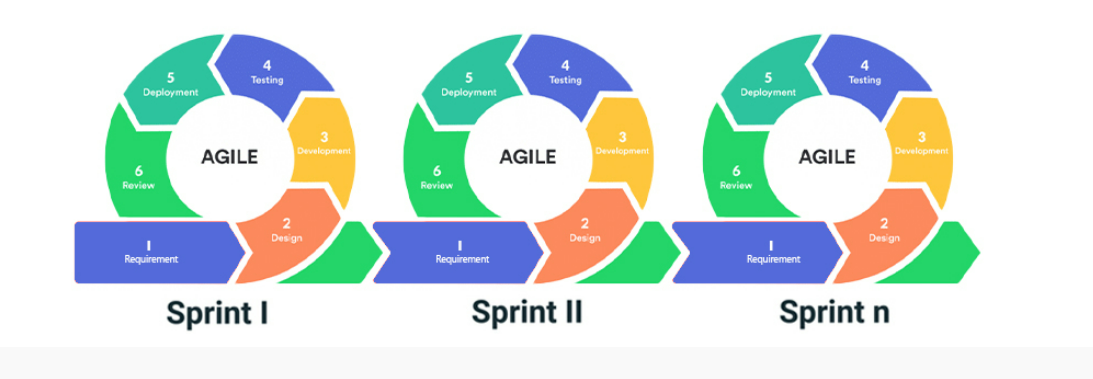
Ecommerce website: divide the entire project into incremental builds

1. Signup
2. Login page
3. Home page
4. Recommendation engine
5. Billing
6. Cart

1 Iterations: Signup

Req gathering -> design -> development(2 days) -> testing -> deploys

feedback



**DEVOPS**

**DEVELOPMENT + OPERATIONS**

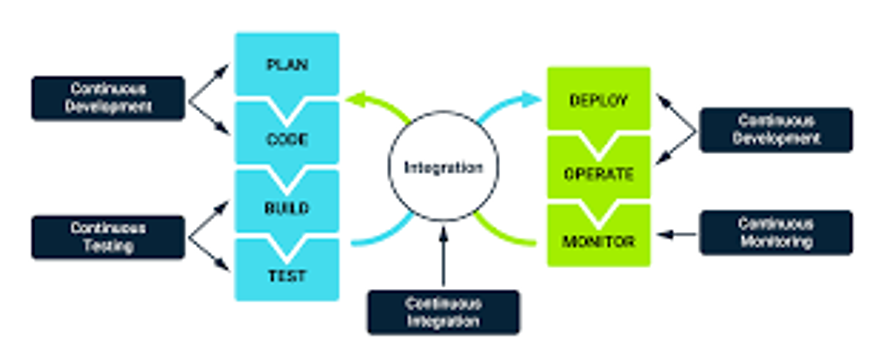
**Continuous development + continuous testing + continuous deployment**

Development Team || Operations team

Application -> tested deploy the application on client servers

Working fine! Dependency issue -> dependency mismatch

1. Cultural philosophy
2. Practices
3. tools
4. efficiency
5. faster





**GIT: Version control system**

**Subversion, cvs, perforce and clearcase**

**Manage the history of your project**

**Revert back to prev changes**

**Mumbai**

**Rahul -> Ajay**

**Code\_v1**

**Code\_v2**

**Code\_v3**

**Code\_v3\_1**

**GIT ->**

**WD(Working Directory) (add) -> SA(Staging Area) -> Local Repository**

**imdb**

**Directory / Folder (git) -> movies.txt**

**Godfather**

**Matrix**

**LOTR**

**Adding to staging area -> tracking the changes in the file ->commit to the LR (local repository)**

**Second version -> one new line with LOTR**

**Git init: initialise a git folder/ create a local repository**

**Git status**

**Git add <filename>**

**Git config –global user.email** [**xyz@abc.com**](mailto:xyz@abc.com)

**Git config –global user.name “xyz@abc.com”**

**Git commit -m “long message” : whatever is in staging area it saves a version in LR**

**#in commit you don’t get to pickup what to commit from staging area**

**Git log**

**Git log –oneline**

**Git show <commit\_id> : details of commit**

**Git diff: MULTIUSE**

**Git diff <filename> (diff btw staging area and WD)**

**Git diff --staged (diff btw the last commit and what is there in staging area)**

**Git diff <commit\_id\_1> <commit\_id\_2>**

**Git diff <branch1> <branch2>**

**RESTORE: Discard the changes in the WD permanently (restore back to what we have in staging area)**

1. **create a new file, add some content**
2. **add it to staging area**
3. **change the file**
4. **git status**
5. **got restore <filename>**

**Git restore <filename>**

**REVERT COMMAND**

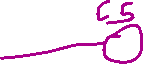
**Git revert <commit\_id>**

* **If a change is committed on a file and you want to go back to previous version of the file then use revert**
* **This cmd only works at LR, it doesn’t work at staging area**
* **New commit id is generated logging the revert that u have done**
* **Targets individual commits at an arbitrary point in the history**

**RESET COMMAND**

**Git reset --hard <commit\_id>**

* **This is command is applied to your commit history**
* **Destructive cmd: never generates**
* **Soft reset: deletes permanently from LR. Files would be available in the Wd and staging area**
* **--hard reset: deletes permanently from LR, WD and staging area**
* **--mixed: deletes permanently from LR and staging area but not from WD**



**branching**



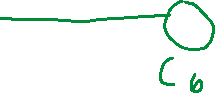
**isolated from each**



1. **Bug Fixing**



**Master Branch**



1. **Experiment**
2. **Features**

**Git branch: list down all the branches**

**Git branch <branch\_name>: create a new branch**

**Git checkout <branch\_name>: switch to a branch**

**Merging: merge the changes from the feature branch to master branch**

**Git merge <sourcebranch> <destination\_branch>**

**Change name of a branch (Modified)**

Git branch -M <old\_branch\_name> <new\_branch\_name>

**Delete a branch**

**Git branch -D <branch\_name>**

**List down all the branches that have been merged to master**

**Git branch –merged**

**Git branch –no-merged**

**Master -> 0 commits**

**Create a Branch -> error**

**Working -> 20 days ->**



**branch**

**Cheery Picking**

**Git cherry-pick <commit\_id>**

**New commit (in master branch) is generated for the cherry-picked commit\_id: changes are reapplied to the master branch**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Merge conflict:**

* 1. **Same part of the code**
  2. **Creating/ modifying a file in the featuire branch deleting the file**

1. **Master -> home.txt -> add 1 to first line, 2 to second line -> commit**
2. **Ajay - > branch**
   1. **Home.txt 1,2,3 -> commit ->**
3. **Master -> home.txt -> add 3 to third line -> commit**
4. **Ajay - > branch**

**changes 1 to 4 🡪 4,2,3 -> commit**

1. **Master merges ajay’s changes in his branch**

**RESOLVE a merge conflict**

**Discussion with team members - > Incorporate the changes in the file in which conflict is occurring -> stage that file -> commit the change -> merge operation will be completed**

**TOOLS:**

**Easy to visualise where the conflict is**

**Mkdir <folder\_name> : create a new folder**

**Pwd: current path**

**Touch <file\_name>**



**Vi <file\_name>**

**Edit: press on “i” button on your keyword**

**Close and save- :wq**

**Cat <file\_name>**

**echo "salad" >> food.txt**

**ls: list file in current directory**

**#PS: client “Zomato”**

**Version control ur project using git**

1. **food.txt -> 5 food items -> save a version of that file**
2. **change the first food item -> save this version of the file as well**
3. **check the diff btw the 2 versions (LR)**
4. **add two more items to the food.txt**
5. **add it to staging area**
6. **check the diff btw the staging area and last commit**

**-1 +1: line one has changed**

**@@-24, 6+24,8 @@**

**6 lines are extracted starting from line number 24. Additionally 8 lines have been addes stating ar line no. 24**

**-3,4: version A, extracting four lines starting from line3**

**+3,2 means version B extracting 2 lines sratiung frome line2**