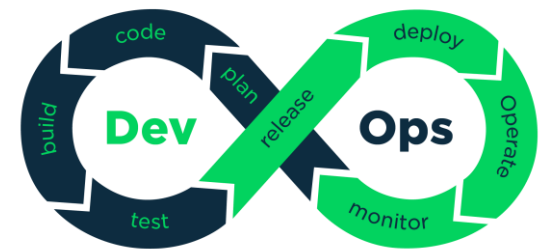


DevOps – Let the Journey Begin



Raman Khanna

Introduction

Name -

Total Experience -

Background –

Any Exposure of AWS/Git/Docker/Kubernetes/Jenkins/Terraform/Ansible

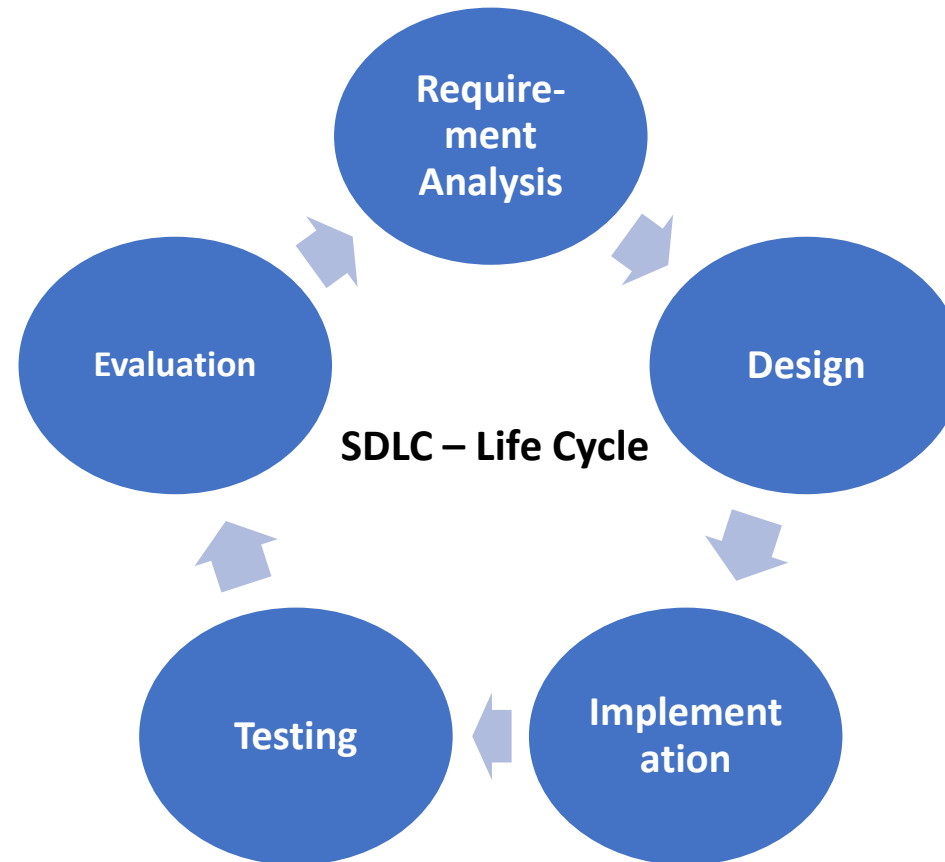
DevOps

What is DevOps?

SDLC Model

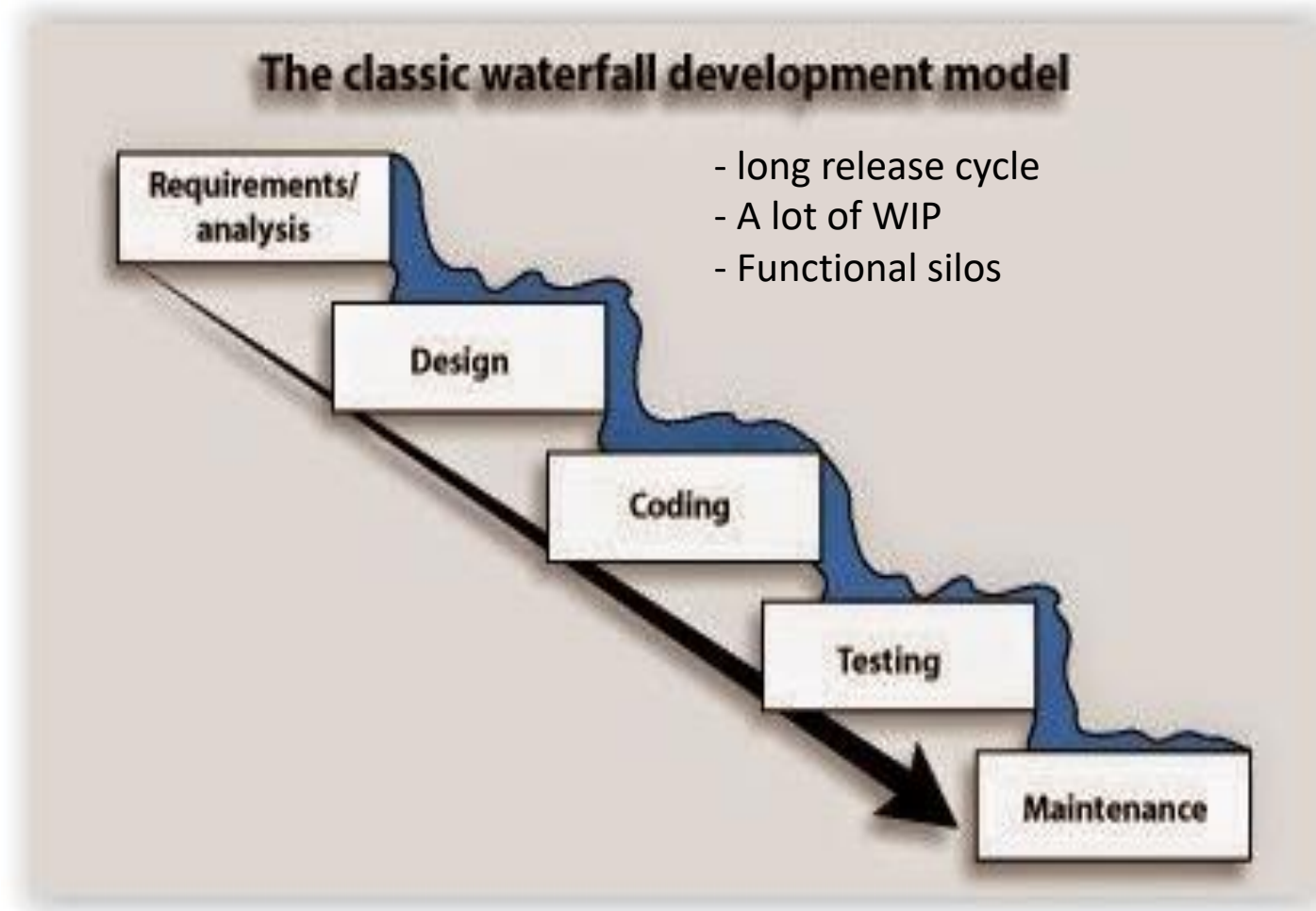
SDLC Model

- A systems development life cycle is composed of **several clearly defined and distinct work phases** which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems



Waterfall Development Model

1. Determine the Requirements
2. Complete the design
3. Do the coding and testing (unit tests)
4. Perform other tests (functional tests, non-functional tests, Performance testing, bug fixes etc.)
5. At last deploy and maintain



Agile Development Model

Agile

Agile Methodology



shorter release cycle

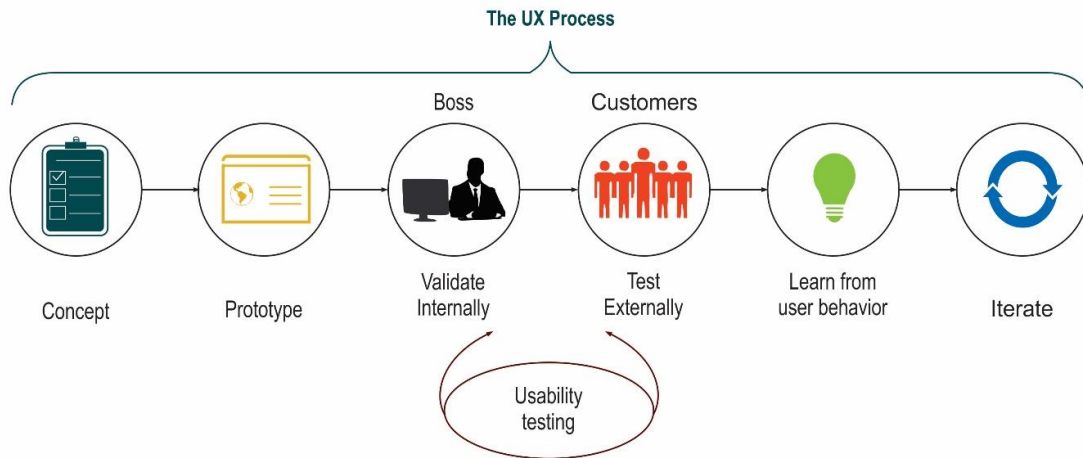
small batch sizes (MinimumViableProduct)

Cross-functional teams

incredibly agile

Lean Development Model

Lean Development (LD)



Not like this...

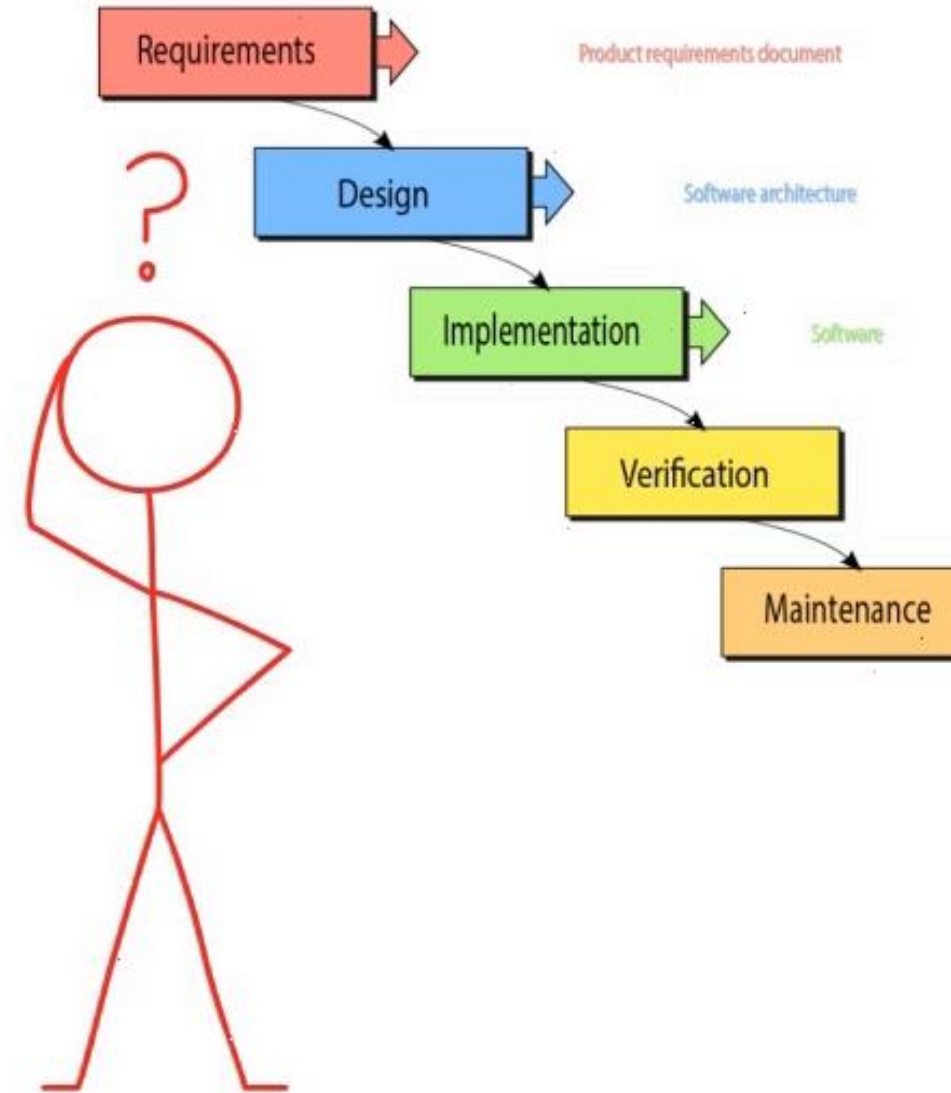


...instead like this!



Values of Agile Principle

Individuals and interactions over process and tools.



Working software over comprehensive documents.



Features



Specifications

Layouts

Test Cases

Requirements



Customer collaboration over contract negotiation



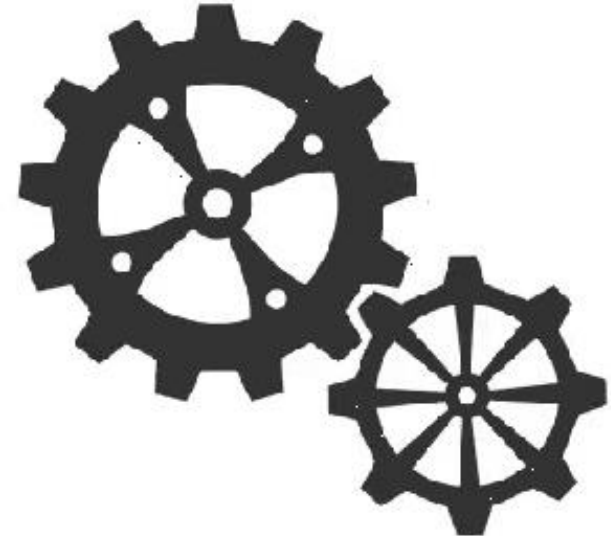
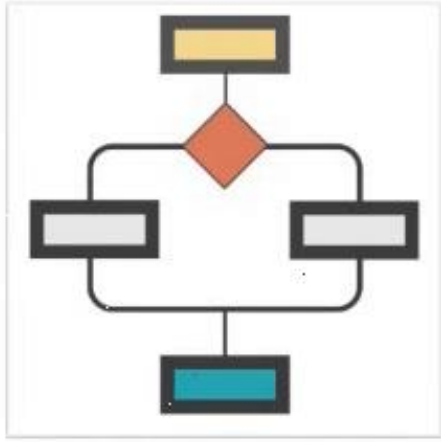
CONTRACT
NEGOTIATION

FURTHER
CHANGES

PROJECT
COMPLETION



Respond to change over following plan



Principles of Agile Project Management



1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.



2. Welcome changing requirements, even late in development.
Agile processes harness change for the customer's competitive advantage.



3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

JANUARY						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1	2	3	4	Delivery 5	6
7	8	9	10	11	12	13
14	15	Delivery 16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	Delivery 31	1	2	3





4. Business people and developers must work together daily throughout the project.



> Development team gets an end user view from the business side



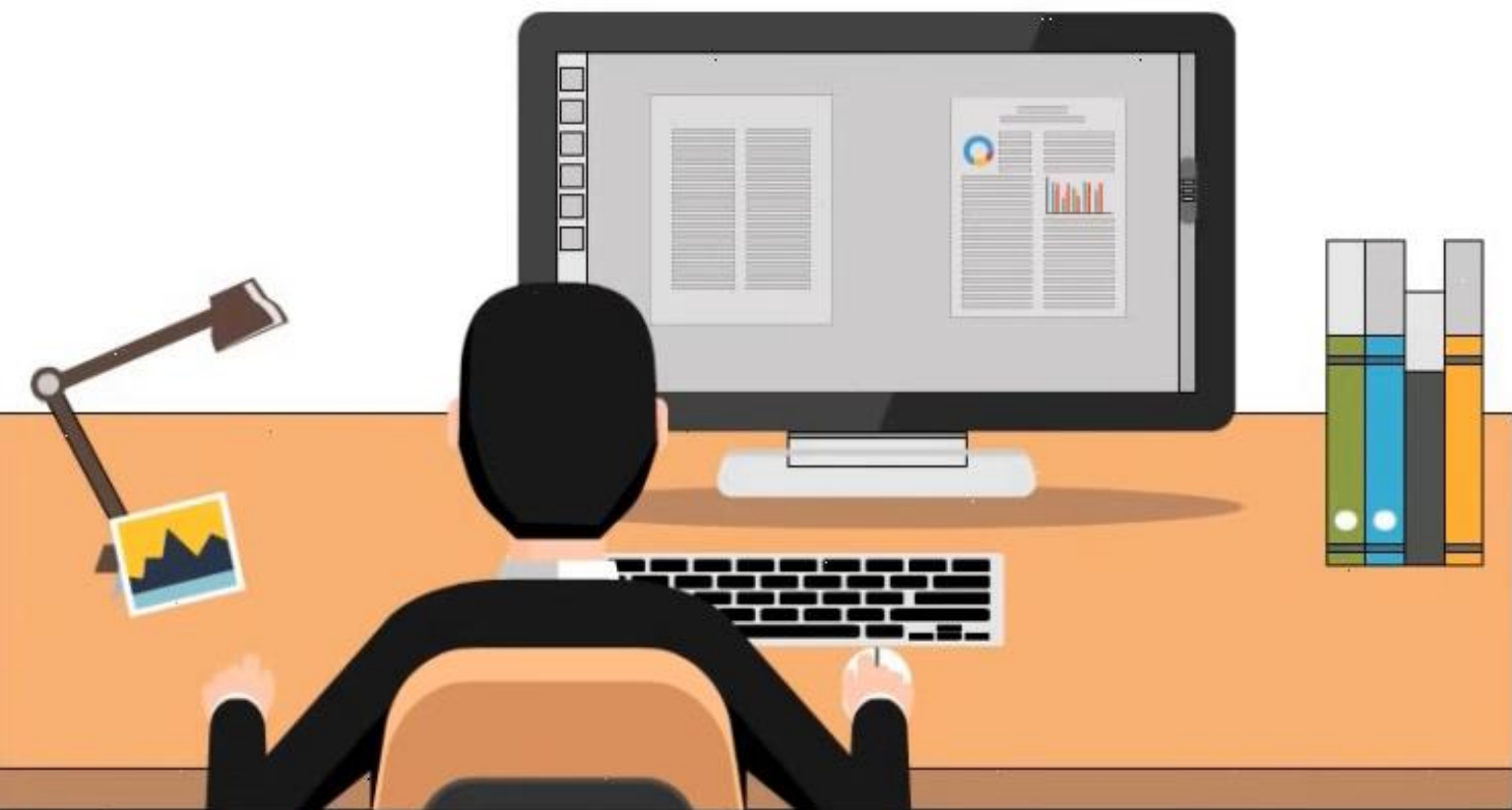
5. Build projects around motivated individuals.

Give them the environment and support they need, and trust them to get the job done.





6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

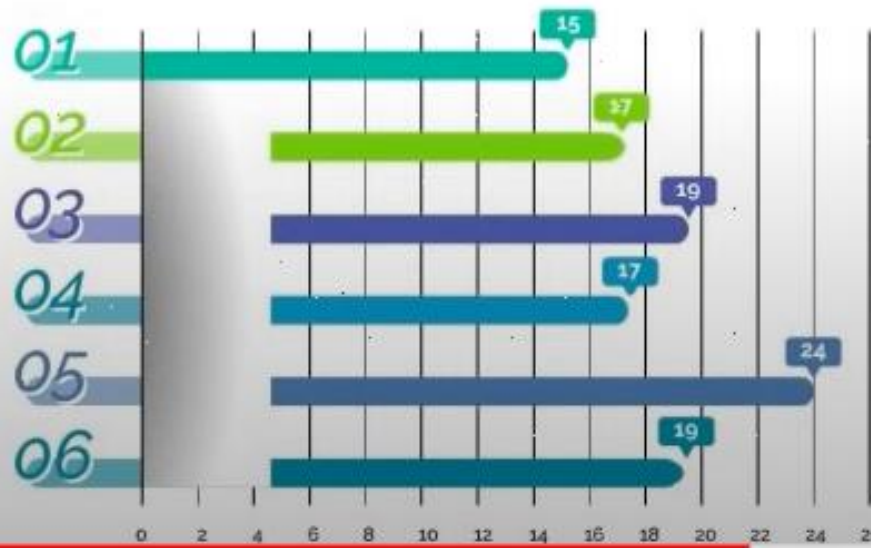


7. Working software is the primary measure of progress.



A software is not finished when it is successfully tested and delivered, it is finished when it is tested and accepted by the end user.

8. Agile processes promote sustainable development.
The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

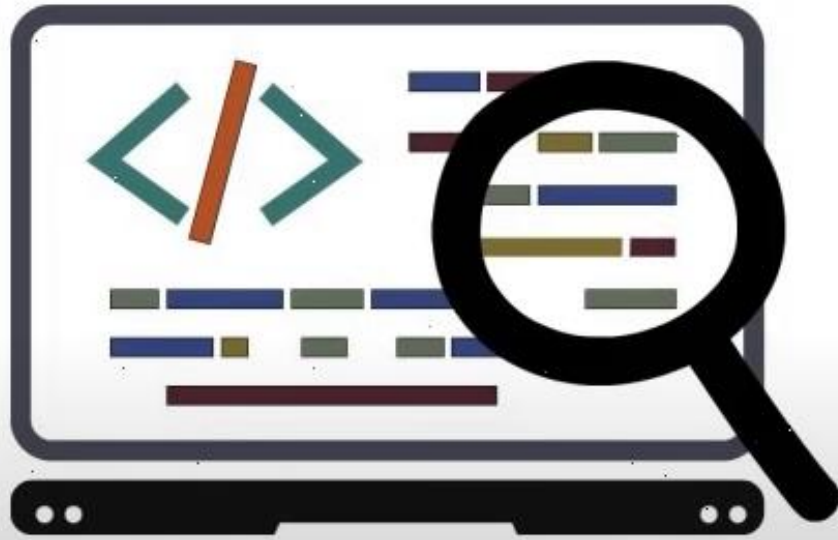


9. Continuous attention to technical excellence and good design enhances agility.



Continuous attention to technical excellence and good design

10. Simplicity--the art of maximizing the amount of work not done--is essential.



COMPLEX CODE



>DEVELOPMENT
>MAINTENANCE





11. The best architectures, requirements, and designs emerge from self-organizing teams



Teams find their own work and manage the associated responsibilities and timelines.



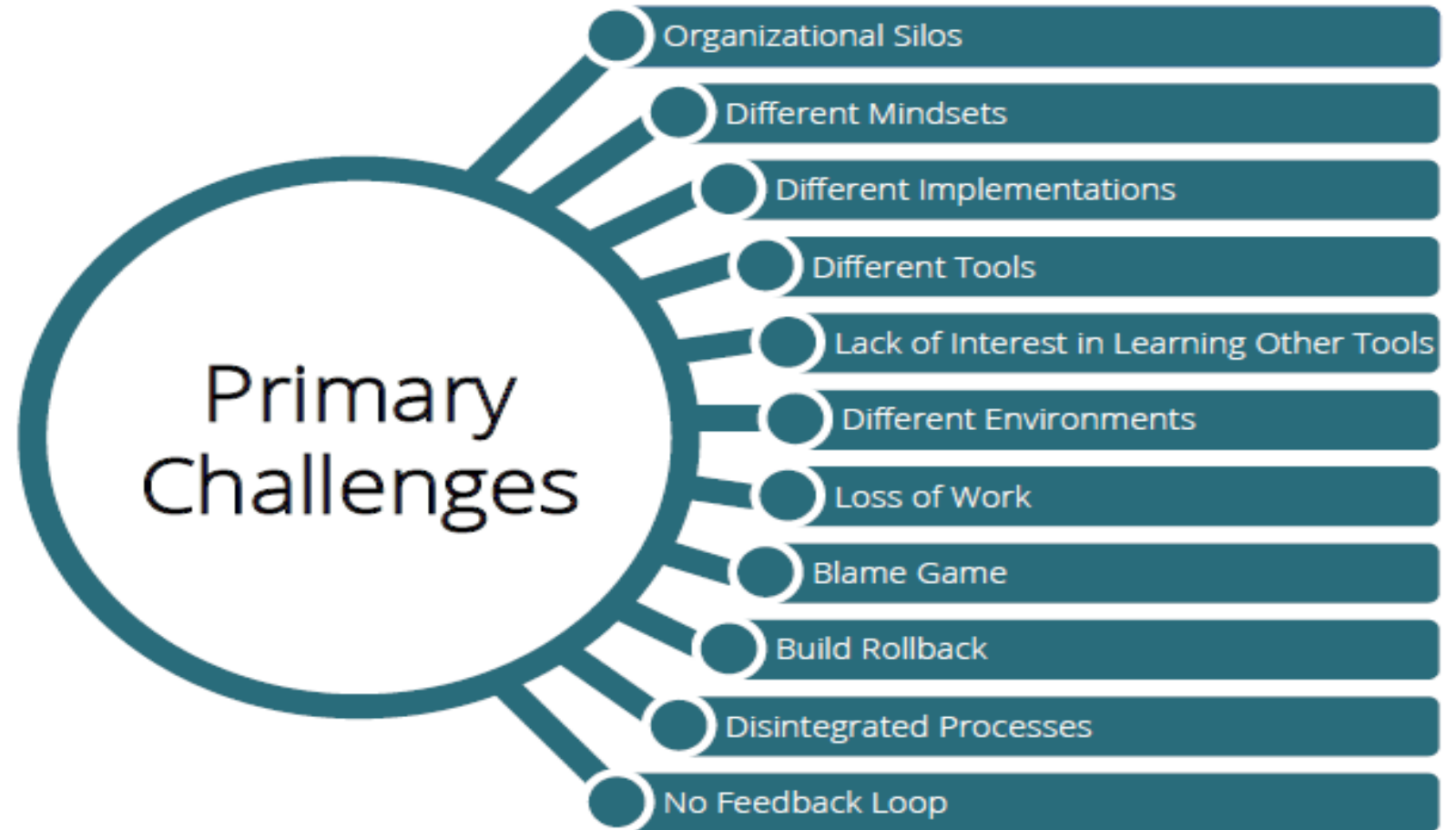
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.



Challenges

Challenges

Some of the challenges with the traditional teams of Development and Operations are:



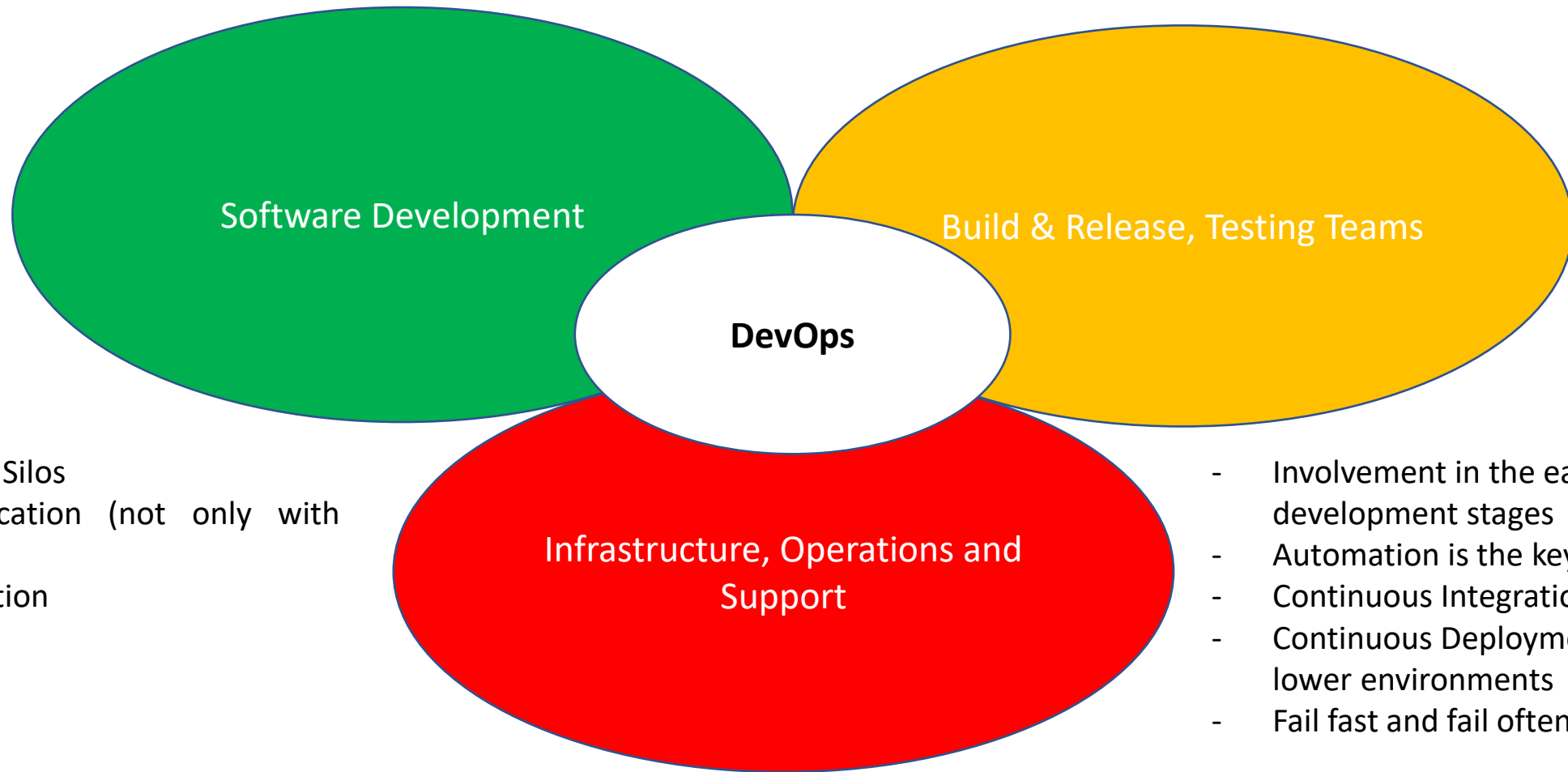
A Typical Case Study

- **Development Team:**
 - Monday Morning, the writing of code done, unit tests completed, code delivered to the Integration teams to get the code included in CI builds.
 - To get the services tested, a ticket is opened for QA teams
- **Build/Release/Testing/Integration Team:**
 - Tuesday Morning, ticket accepted, a tester put an email to the developer asking deployment instructions. There is not automated deployments, developer updated to the tester, lets come online and we will deploy the services to the QA environment together.
 - Call started, developer identified the “test environment” is not compatible.
 - Tuesday afternoon, a ticket raised in Ops Team with new specifications.
- **Ops Team:**
 - Wednesday morning, ticket accepted, specifications checked , a new port open request was identified.
 - Ticket raised for Security team, ticket accepted, change approved, port opened, email received by the Ops team the work is done.

A Typical Case Study

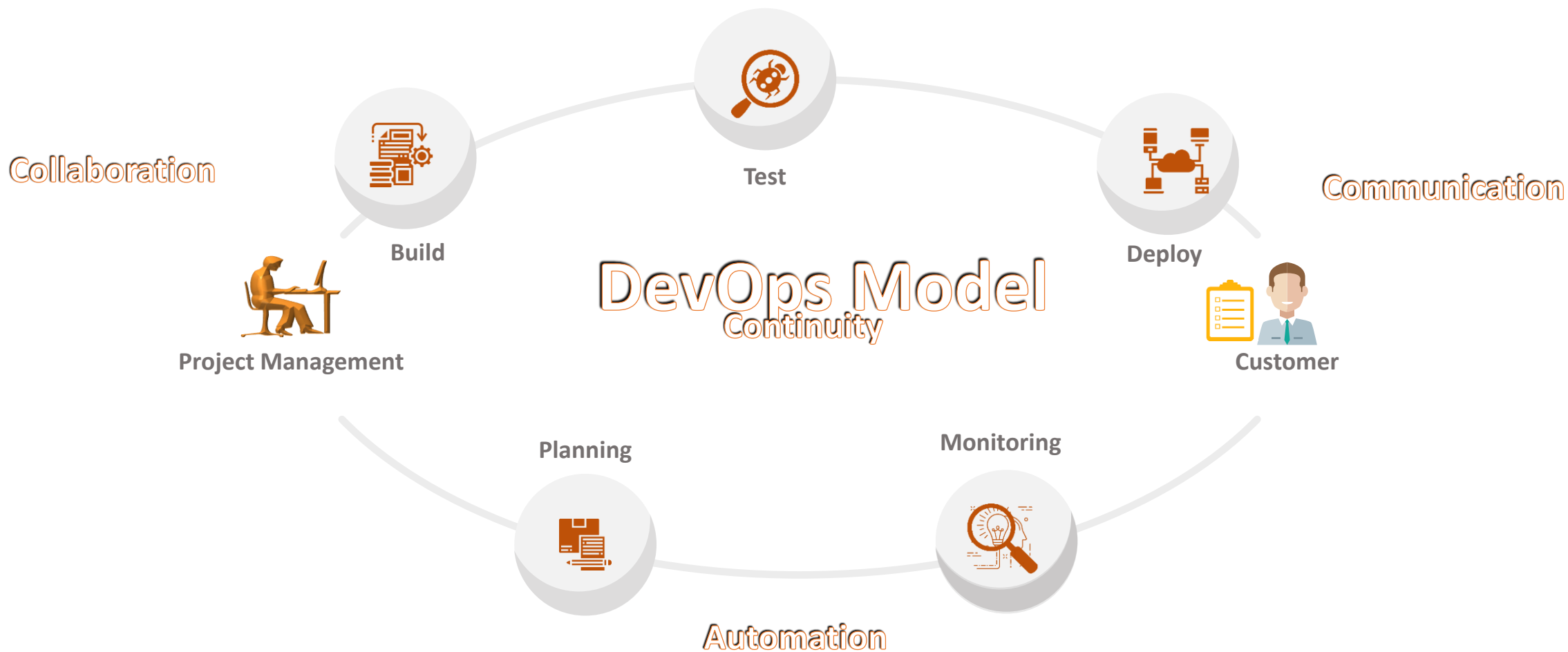
- **Ops Team:**
 - Identified the provisioning requirements again and started work on building the environment.
- **Build/Release/Testing/Integration Team:**
 - Thursday Morning, updates received - the environment is ready. Developer and Tester again on call to deploy new services. Services deployed; tester is running test scripts. Next phase is to run regression test cases. Again a new ticket is raised for new test data with production teams and day ends.
- **Ops Team:**
 - Its Friday and the work is not on full swing, ticket accepted but not worked as production team has to complete rest of the works. Somehow the test data is gathered by Friday Evening.
- **Build/Release/Testing/Integration Team:**
 - Monday morning, tester gets the data, regression tests run, a defect found, and ticket returned to the development team.

DevOps



- Break the Silos
- Communication (not only with emails)
- Collaboration
- Trust

- Involvement in the early development stages
- Automation is the key
- Continuous Integration
- Continuous Deployments in the lower environments
- Fail fast and fail often



Continuous Feedback

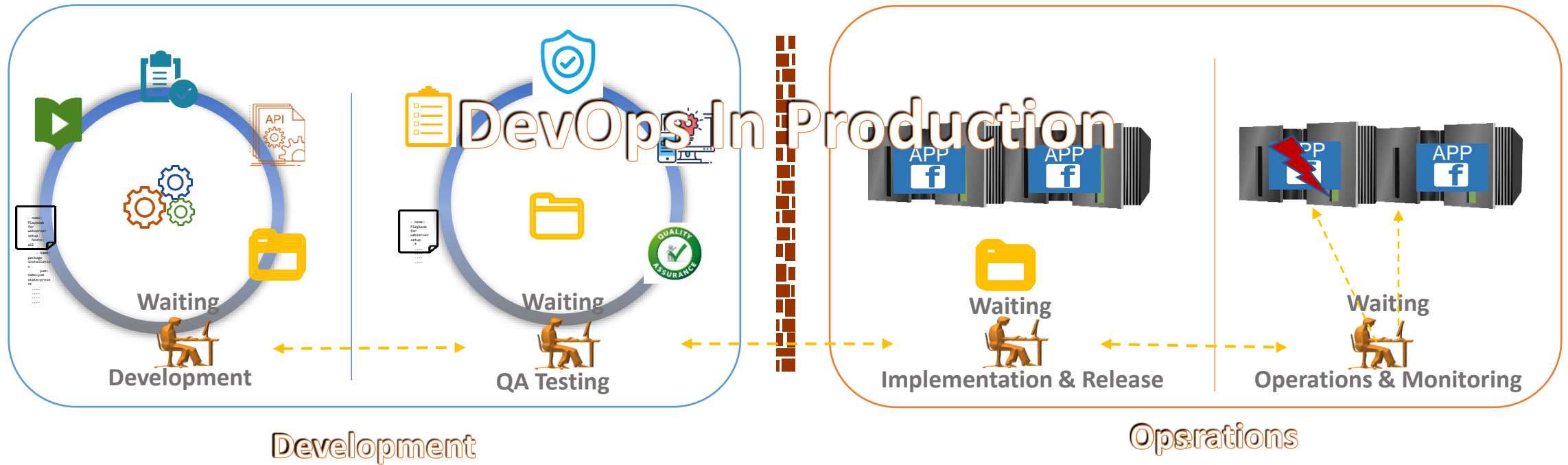
Continuous Improvement

Continuous Planning

Continuous Delivery

Continuous Deployment

Continuous Monitoring



DevOps Essence

Efficiency - Faster time to market

Predictability - Lower failure rate of new releases

Reproducibility – Version everything

Maintainability - Faster time to recovery in the event of a new release crashing or otherwise disabling the current system

DevOps Core Principals

1. Customer-Centric Action



4. Cross-Functional Autonomous Teams



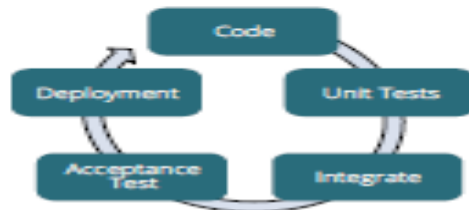
2. Create with the End in Mind



5. Continuous Improvement



3. End-to-End Responsibility



6. Automate Everything You can



How to Build DevOps Organization Culture

Retention is as important as recruitment

Establish Cross-functional team structure

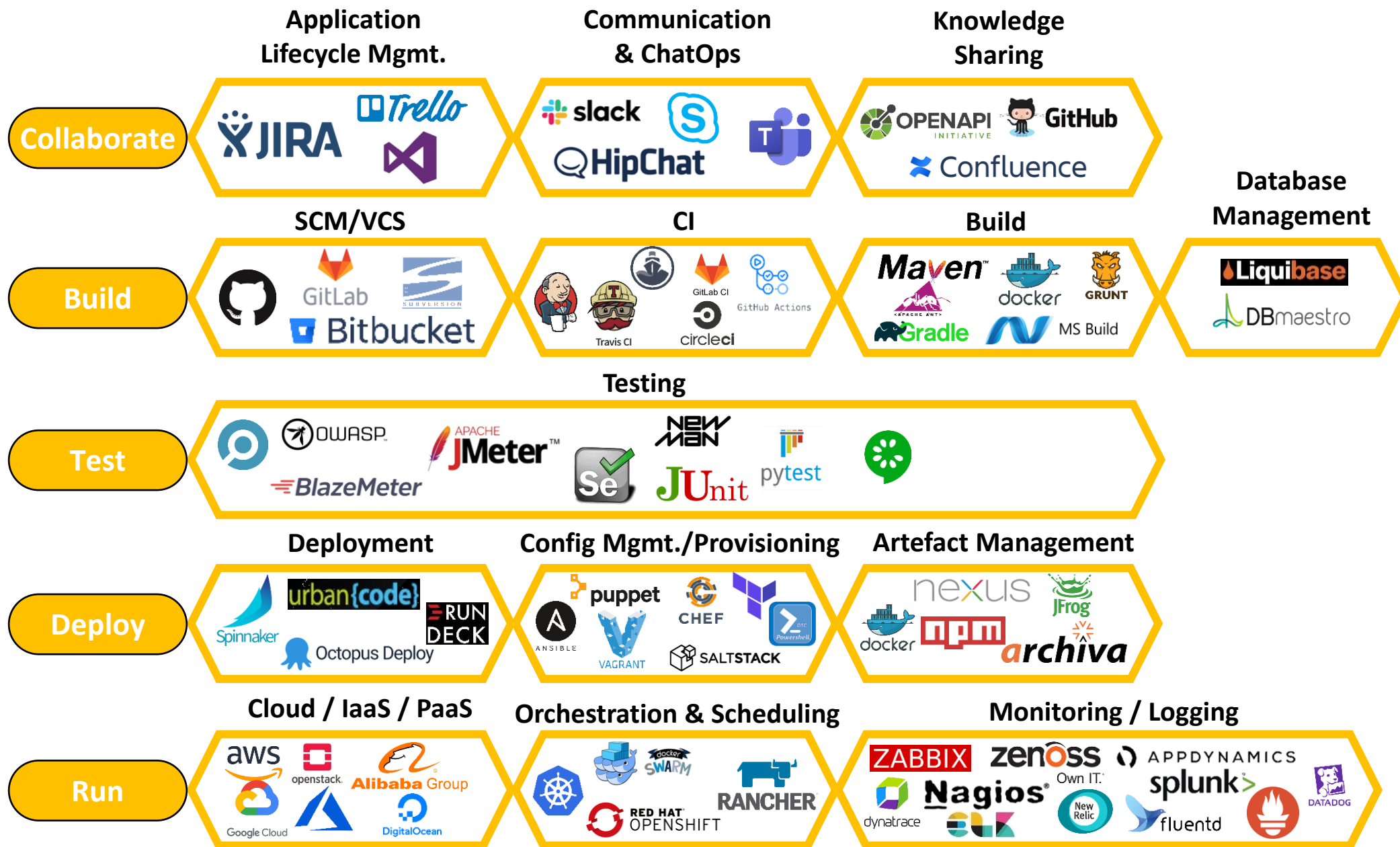
Small teams are better

Give autonomy

Automate with existing staffs and give them a chance to learn

Take out few resources from each team, build a new virtual team for automation.

DevOps Toolsets







JIRA – A Project Management tool



JIRA

A Product from Atlassian family




PLAN, TRACK, & SUPPORT

-  **Jira Software**
Project and issue tracking
-  **Jira Align**
Enterprise agile planning
-  **Jira Core**
Essential business management
-  **Jira Service Desk**
Collaborative IT service management
-  **Opsgenie**
Modern incident response
-  **Statuspage**
Incident communication



COLLABORATE

-  **Confluence**
Document collaboration
-  **Trello**
Collaborate visually on any project

CODE, BUILD, & SHIP

-  **Bitbucket**
Git code management
-  **Sourcetree**
Git and Mercurial desktop client
-  **Bamboo**
Integration and release management

SECURITY & IDENTITY

-  **Atlassian Access**
Security and control for cloud
-  **Crowd**
User management for self-managed environments

JIRA

A Project Management Tool made by Atlassian (Australian company).

You can manage Project, Tests cases, defects, Scrum Meeting etc.

A #1 tool for Project management due to unmatched benefits of the tool.

Everything which you need to manage projects and team is available.

Fully customizable

Hundreds of Apps to enhance the functionality.

Agile Principles with JIRA

Iterative Development

Adaptive to changing Requirements

Frequent Delivery

Cross-functional team structure

Agile methodologies

	Scrum	Kanban
Cadence	Regular fixed length sprints (ie, 2 weeks)	Continuous flow
Release methodology	At the end of each sprint	Continuous delivery
Roles	Product owner, scrum master, development team	No required roles
Key metrics	Velocity	Lead time, cycle time, WIP
Change philosophy	Teams should not make changes during the sprint.	Change can happen at any time

Scrum

Sprint
1

Sprint
2

Plan

Plan

Build

Build

Test

Test

Review

Review

Several
incremental
releases
called
Sprints

Potentially Shippable Product

Scrum

Breaks projects into Epics and Stories

Epics – Large Stories or Work Items, usually broken into small storeies

User Stories: Smallest Unit of work

User stories can be prioritized.

Development is performed in short cycles.

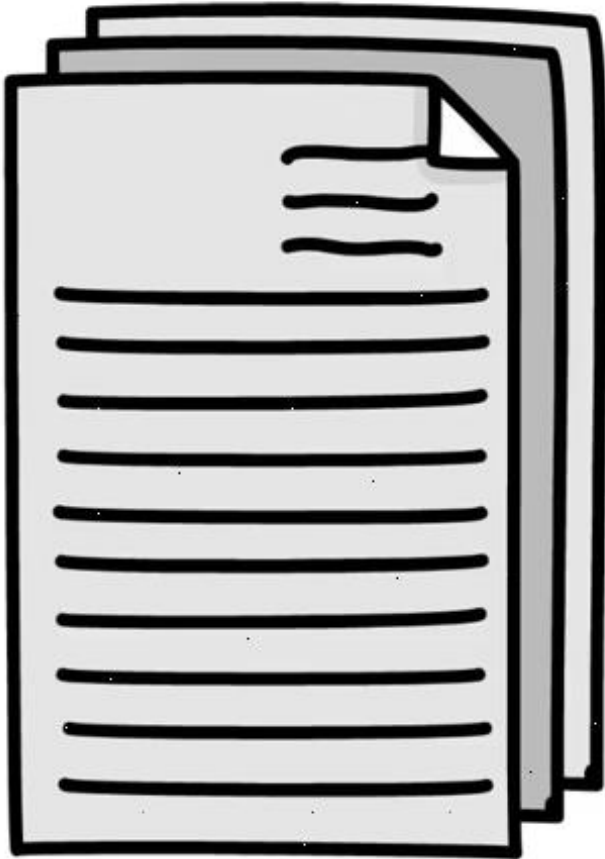
Terms like Product Owner, Development Team, Scrum Master

User Stories

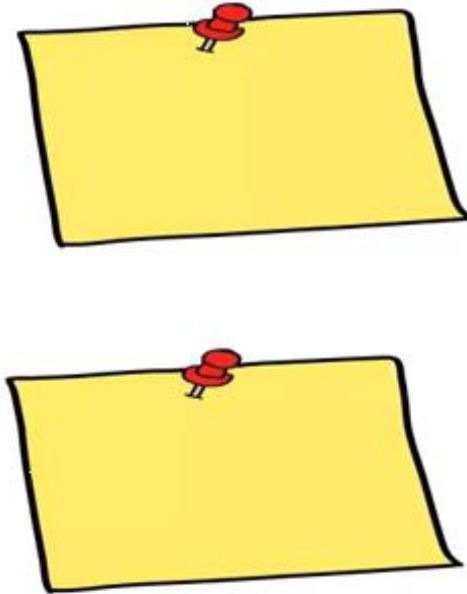


3 Artifacts

Product Backlog



Sprint Backlog



Burndown Chart



3 Roles



Product Owner



Scrum Master



Team

3 Ceremonies

Sprint Planning



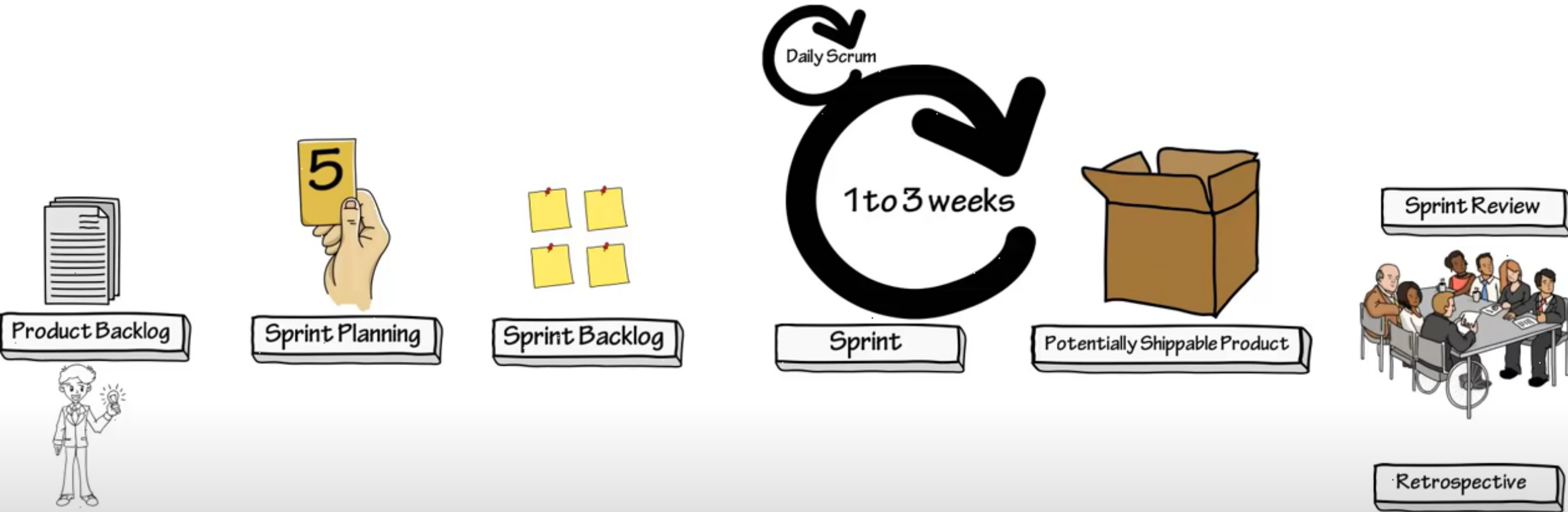
Daily Scrum



Sprint Review

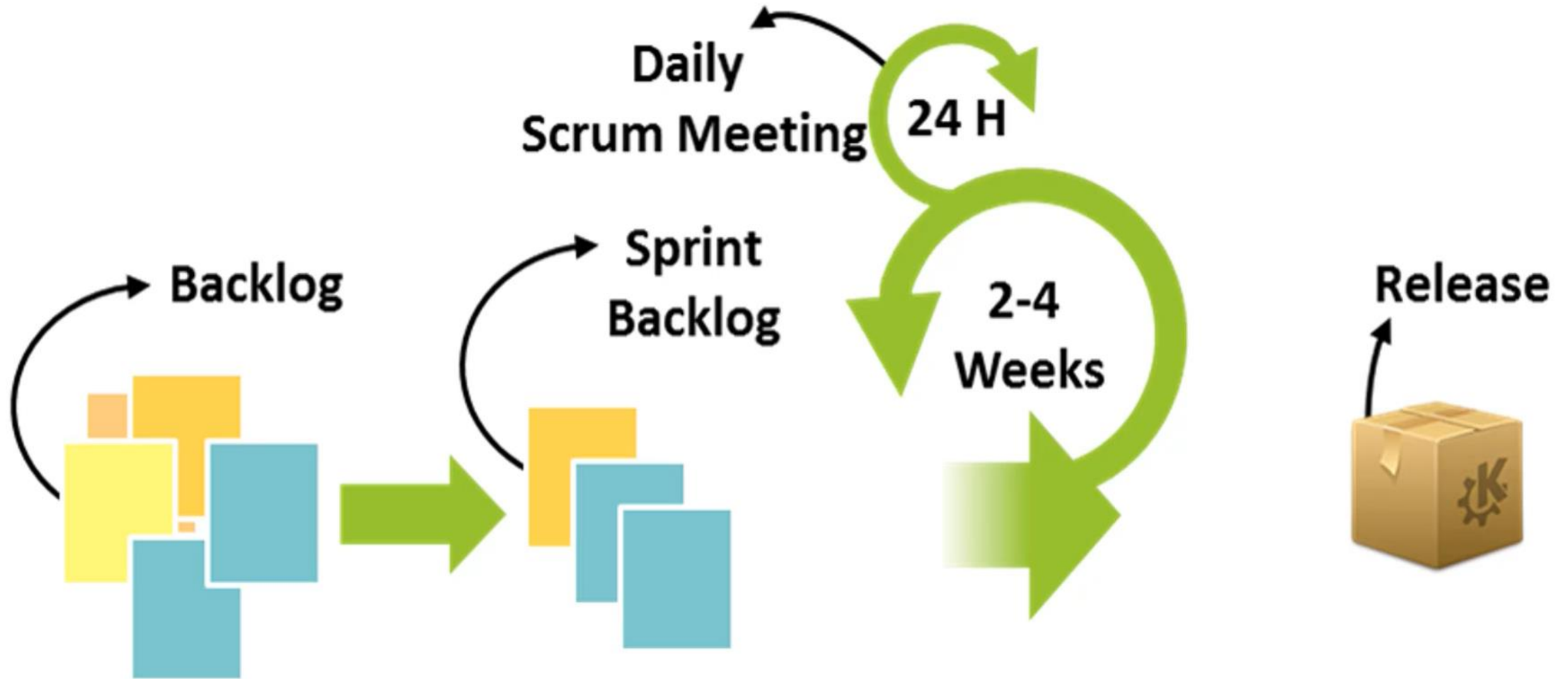


Scrum Workflow



Repeat this workflow for each sprint

Scrum flow



Demo : Scrum on Jira

Version Control Systems with GIT

What is Version Control System

As name states Version Control System is the “**Management of changes to anything**”.

Version Control is way of storing files in central location accessible to all team members and enabling them to keep track of changes being done in the source code by whom, when & why. It also help teams to recover from some inevitable circumstances.

Think about traditional versioning of file with names – Login001.java, Login002.java, Login_final.java.

Its not just for code, it also helps in

- Backups & Restoration

- Synchronization

- Reverts

- Track Changes

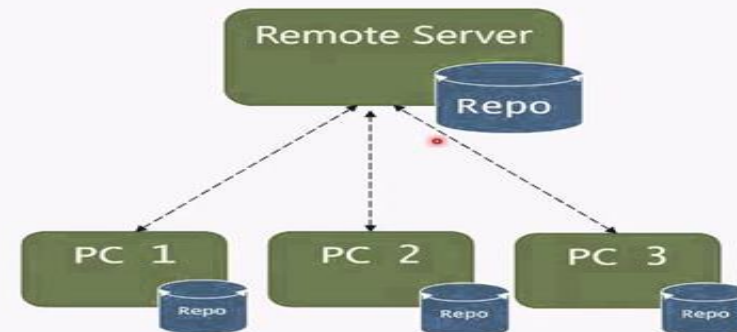
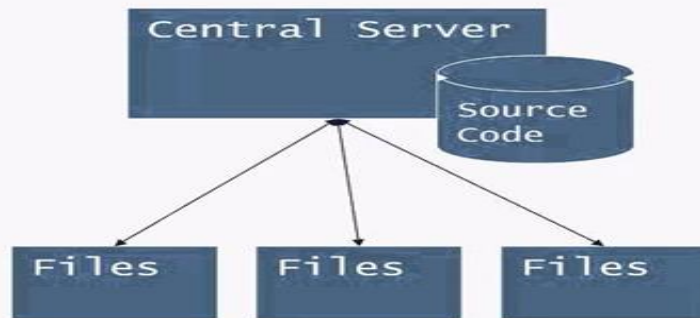
- Most importantly in Parallel Development

Types of VCS

Majorly VCS is divided into two parts:

- Centralized Version Control System – CVS, Subversion, Visual Source Safe
- Distributed Version Control System – Mercurial, Bitkeeper, Git

Centralized vs. Distributed



Git History

Linus uses BitKeeper to manage Linux code

Ran into BitKeeper licensing issue

Liked functionality

Looked at CVS as how not to do things

April 5, 2005 - Linus sends out email showing first version

June 15, 2005 - Git used for Linux version control

Why Git?

- **Branching:** gives developers a great flexibility to work on a replica of master branch.
- **Distributed Architecture:** The main advantage of DVCS is “no requirement of network connections to central repository” while development of a product.
- **Open-Source:** Free to use.
- **Integration with CI:** Gives faster product life cycle with even faster minor changes.

Git Installation

- `yum install autoconf libcurl-devel expat-devel gcc gettext-devel kernel-headers openssl-devel perl-devel zlib-devel -y`
- Visit git release page - <https://github.com/git/git/releases> and pick desired version.
- `curl -O -L https://github.com/git/git/archive/v2.14.0.tar.gz`
- `tar -zxvf v2.14.2.tar.gz`
- `cd git-v2.14.0`
- `make clean`
- `make configure`
- `./configure --prefix=/usr/local`
- `make`
- `make install`
- `ln -s /usr/local/bin/git /usr/bin/git`

Version Control Systems with GIT

Raman Khanna

What is Version Control System

- As name states Version Control System is the **“Management of changes to anything”**.
- Version Control is way of storing files in central location accessible to all team members and enabling them to keep track of changes being done in the source code by whom, when & why. It also help teams to recover from some inevitable circumstances.

A History Lesson

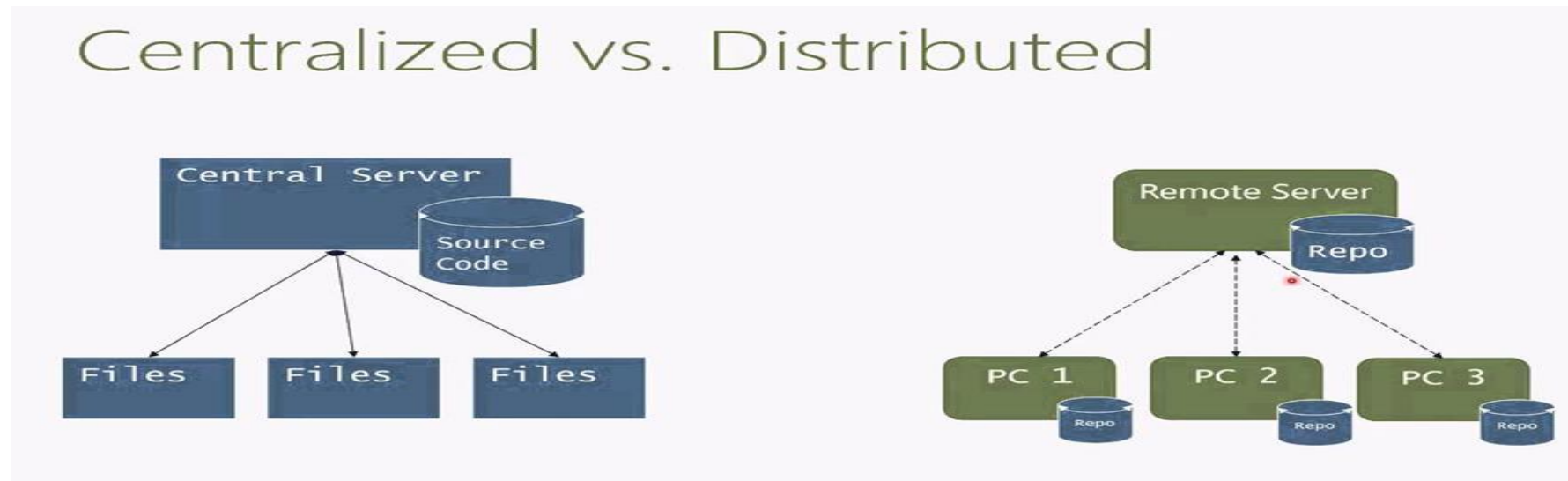
- Before Version Control System
 - File renaming
 - Login001.java
 - Login002.java
 - LoginFinal.java
 - Directories
 - June_Release_Code
 - August_Release_Code
 - Zip Files
 - Package_May.zip
 - Package_June.zip
 - Nothing at all

Version Control helps in

- Its not just for code, it also helps in
 - Backups & Restoration
 - Synchronization
 - Reverts
 - Track Changes
 - Most importantly in Parallel Development

Types of Version Control Systems

- Majorly VCS is divided into two parts:
 - Centralized Version Control System
 - Distributed Version Control System



Centralized Version Control System

- Traditional version control system
 - Server with database
 - Clients have a working version
- Examples
 - CVS
 - Subversion
 - Visual Source Safe
- Challenges
 - Multi-developer conflicts
 - Client/server communication

Distributed Version Control Syetem

- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial
- Other distributed systems include
 - Mercurial
 - BitKeeper
 - Darcs
 - Bazaar

GIT History

- Linus uses BitKeeper to manage Linux code
- Ran into BitKeeper licensing issue
 - Liked functionality
 - Looked at CVS as how not to do things
- April 5, 2005 - Linus sends out email showing first version
- June 15, 2005 - Git used for Linux version control

GIT is not a SCM

Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable file system, used to track directory trees.

Linus Torvalds, 7 Apr 2005



Why GIT

- **Why Git:**
 - **Branching:** gives developers a great flexibility to work on a replica of master branch.
 - **Distributed Architecture:** The main advantage of DVCS is “**no requirement of network connections to central repository**” while development of a product.
 - **Open-Source:** Free to use.
 - **Integration with CI:** Gives faster product life cycle with even more faster minor changes.

GIT Installation

- GIT comes as default offering for all major Linux flavors
- Though you can download the installers from below link for Windows, Mac OS X, Linux, Solaris
 - <https://git-scm.com/downloads>
 - you can install same with yum too:
 - yum install git
- [Installing GIT Bash on Windows](#)

GIT Installation on Linux

- `yum install autoconf libcurl-devel expat-devel gcc gettext-devel kernel-headers openssl-devel perl-devel zlib-devel -y`
- Visit git release page - <https://github.com/git/git/releases> and pick desired version.
- `curl -O -L https://github.com/git/git/archive/v2.14.0.tar.gz`
- `tar -zxvf v2.14.2.tar.gz`
- `cd git-v2.14.0`
- `make clean`
- `make configure`
- `./configure --prefix=/usr/local`
- `make`
- `make install`
- `rm -rf /usr/bin/git`
- `ln -s /usr/local/bin/git /usr/bin/git`

GIT VERSION

```
[root@user20-master plays]# git --version  
git version 1.8.3.1  
[root@user20-master plays]#
```

GIT HELP

```
k Mayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git --help
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
    clone      Clone a repository into a new directory
    init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
    add        Add file contents to the index
    mv         Move or rename a file, a directory, or a symlink
    reset      Reset current HEAD to the specified state
    rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
    bisect     Use binary search to find the commit that introduced a bug
    grep       Print lines matching a pattern
    log        Show commit logs
    show       Show various types of objects
    status     Show the working tree status

grow, mark and tweak your common history
    branch     List, create, or delete branches
    checkout   Switch branches or restore working tree files
    commit     Record changes to the repository
    diff       Show changes between commits, commit and working tree, etc
    merge      Join two or more development histories together
    rebase     Reapply commits on top of another base tip
    tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
    fetch      Download objects and refs from another repository
    pull       Fetch from and integrate with another repository or a local branch
    push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```


Setting Identity in GIT

```
[root@Techlanders ~]# git config --global user.email "raman.khanna@Techlanders.com"  
[root@Techlanders ~]# git config --global user.name "raman.khanna"  
[root@Techlanders ~]# git config --global -l  
user.name=Gagandeep Singh  
user.email=Gagandeep.singh@Techlanders.com  
[root@Techlanders ~]#
```

GIT Repository

- A **repository** is usually used to organize a single project.
- Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs.
- Repository is like a unique shared file system for a project.

Initializing a Repository

```
[root@master git]# mkdir /Repo1
[root@master git]# cd /Repo1/
[root@master Repo1]# git init
Initialized empty Git repository in /Repo1/.git/
[root@master Repo1]# ls -lrt /Repo1/.git/
total 12
drwxr-xr-x. 4 root root 31 Dec 19 19:32 refs
drwxr-xr-x. 2 root root 21 Dec 19 19:32 info
drwxr-xr-x. 2 root root 242 Dec 19 19:32 hooks
-rw-r--r--. 1 root root 73 Dec 19 19:32 description
drwxr-xr-x. 2 root root 6 Dec 19 19:32 branches
drwxr-xr-x. 4 root root 30 Dec 19 19:32 objects
-rw-r--r--. 1 root root 23 Dec 19 19:32 HEAD
-rw-r--r--. 1 root root 92 Dec 19 19:32 config
[root@master Repo1]#
```

Adding file to a Repository

```
[root@master Repo1]# git status
# On branch master
# Initial commit - nothing to commit (create/copy files and use "git add" to track)
[root@master Repo1]# echo "File1 content" >> file1
[root@master Repo1]# ll
-rw-r--r--. 1 root root 14 Dec 19 19:35 file1
[root@master Repo1]# git add file1
[root@master Repo1]# git status
# On branch master
# Initial commit
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#       new file:   file1
#
[root@master Repo1]#
```

Checking Repository Status

```
[root@user20-master Repo1]# touch file2
[root@user20-master Repo1]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       file2
[root@user20-master Repo1]#
```

Committing changes to a Repository

```
[root@user20-master Repo1]# git add --all
[root@user20-master Repo1]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1
#       new file:   file2
#
[root@user20-master Repo1]# git commit -m "Commit one - Added File1 & File2"
[master (root-commit) 9c301cb] Commit one - Added File1 & File2
2 files changed, 1 insertion(+)
create mode 100644 file1
create mode 100644 file2
[root@user20-master Repo1]#
```

Committing changes to a Repository

```
[root@master Repo1]#echo "File2 content added" > file2
[root@master Repo1]#git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   file2
#
no changes added to commit (use "git add" and/or "git commit -a")
[root@master Repo1]#git commit -am "second commit - Changes done in File2"
[master 77de849] second commit - Changes done in File2
1 file changed, 1 insertion(+)
[root@master Repo1]#
```

Git Log

```
[root@master Repo1]#git log
commit 77de8496c39c8d442d8e1212f9f3879a33253a1c
Author: admin.gagan@gmail.com <admin.gagan@gmail.com>
Date: Wed Dec 19 19:48:56 2018 +0000
```

second commit - Changes done in File2

```
commit 9c301cb93733f666e959a87c7a3f61142d1d9f48
Author: admin.gagan@gmail.com <admin.gagan@gmail.com>
Date: Wed Dec 19 19:43:25 2018 +0000
```

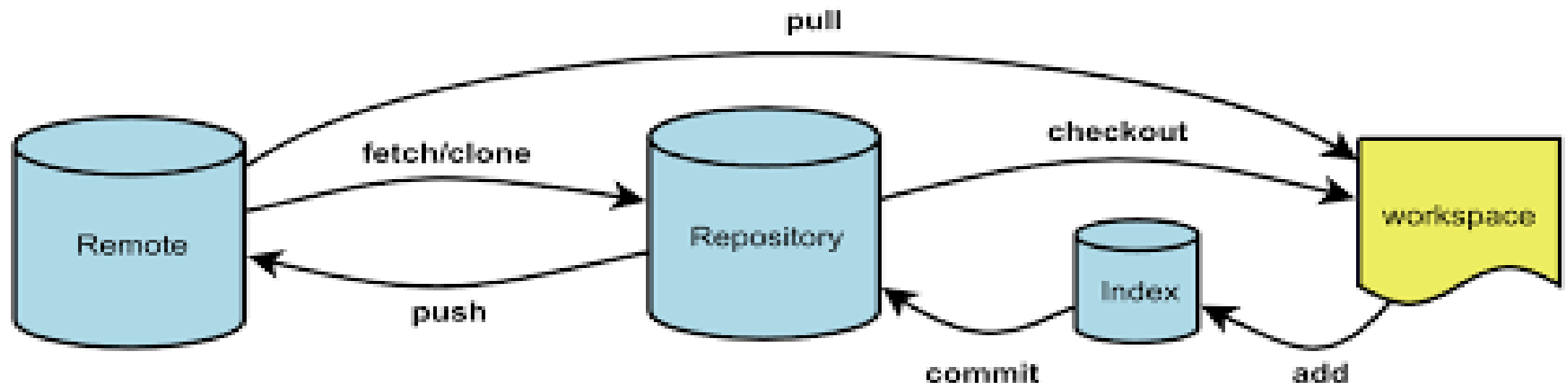
Commit one - Added File1 & File2

```
[root@master Repo1]#
```


Git Diff- Comparing two commits

```
[root@master Repo1]#git diff 9c301cb93733f666e959a87c7a3f61142d1d9f48  
77de8496c39c8d442d8e1212f9f3879a33253a1c  
diff --git a/file2 b/file2  
index e69de29..de51b99 100644  
--- a/file2  
+++ b/file2  
@@ -0,0 +1 @@  
+File2 content added  
[root@master Repo1]#
```

Git Flow



Git Push

```
[root@master Repo1]#git push origin master
Username for 'https://github.com': admin.gagan@gmail.com
Password for 'https://admin.gagan@gmail.com@github.com':
Counting objects: 14, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.22 KiB | 0 bytes/s, done.
Total 12 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/admingagan/ansibleplaybooks.git
   d9b5ae7..64bad4d  master -> master
[root@master Repo1]#
```

Initializing a Remote Repository

```
[root@master Repo1]#git remote add origin https://github.com/admingagan/repo1.git
[root@master Repo1]#
[root@master Repo1]#git pull origin master
From https://github.com/admingagan/repo1
 * branch      master    -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 README.md | 1 +
 abc      | 0
 ntp.yaml | 13 ++++++++
3 files changed, 14 insertions(+)
create mode 100644 README.md
create mode 100644 abc
create mode 100644 ntp.yaml
[root@master Repo1]#
```

Git Pull

```
[root@master Repo1]#ls
abc file1 file2 ntp.yaml readme5 README.md

[root@master Repo1]#git pull origin dev
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/admingagan/ansibleplaybooks
* branch      dev      -> FETCH_HEAD
Merge made by the 'recursive' strategy.
readme6 | 1 +
1 file changed, 1 insertion(+)
create mode 100644 readme6
[root@master Repo1]#ls
abc file1 file2 ntp.yaml readme5 readme6 README.md
[root@master Repo1]#
```

Git Clone

```
[root@user20-master git]# git clone https://github.com/admingagan/test.git
Cloning into 'test'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 23 (delta 5), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (23/23), done.
[root@user20-master git]# cd test
[root@user20-master test]# ll
total 16
-rw-r--r--. 1 root root 10 Dec 20 07:45 Readme
-rw-r--r--. 1 root root 24 Dec 20 07:45 Readme2
-rw-r--r--. 1 root root 57 Dec 20 07:45 readme3
-rw-r--r--. 1 root root  9 Dec 20 07:45 README4
[root@user20-master test]#
```

GIT BRANCH

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch training

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
  training
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch -d training
Deleted branch training (was 74e76df).

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
```

GIT CHECKOUT

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git checkout training
Switched to branch 'training'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training)
$ git checkout -b training_checkout
Switched to a new branch 'training_checkout'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git branch
  master
  training
* training_checkout
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git log
commit 76e137f900eb33b7eb4a4ac7c81f160af8124697 (HEAD -> training_checkout)
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Fri Jun 9 20:38:59 2017 +0530

    commit in branch

commit 74e76dfcaf297ae9b63f950988907cdce8d275de (training, master)
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Thu Jun 8 23:16:16 2017 +0530

    second commit

commit 3eaadebb9972b29b7463822e99b485b9d9b490eb
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Thu Jun 8 23:14:06 2017 +0530

    my initial commit
```


GIT MERGE

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git checkout master
Switched to branch 'master'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git merge training_checkout
Updating 74e76df..76e137f
Fast-forward
 file.txt | 2 ++
 1 file changed, 2 insertions(+)
```

- Run git checkout master
- Run git merge training

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git merge training
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master|MERGING)
$ ls -ltr
total 2
-rw-r--r-- 1 kmayer 197121 30 Jun  9 20:51 file-in-training-branch.txt
-rw-r--r-- 1 kmayer 197121 238 Jun 12 20:47 file.txt
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   file.txt
```

GIT MERGE – Resolving Conflicts

```
1 Adding dummy data - Kulbhushan Mayer
2 Second Update
3
4 <<<<<<< HEAD
5 Making changes in branch for demo
6
7 Making one more change - 09/06/2017 08:44
8
9 Making change at 08:46 PM
10 =====
11 Making change to check merging - 12/06/2017
12 >>>>>>> training
13
```

- <<<<<<< depicts changes from the HEAD or BASE branch
- ===== divides your changes from the other branch
- >>>>>>> depicts the end of changes
- Remove <<<<<<<, =====, >>>>>>> from the file and make then necessary changes
- Now you have to commit the changes explicitly

```
1 Adding dummy data - Kulbhushan Mayer
2 Second Update
3
4
5 Making changes in branch for demo
6
7 Making one more change - 09/06/2017 08:44
8
9 Making change at 08:46 PM
10
11 Making change to check merging - 12/06/2017
12
```

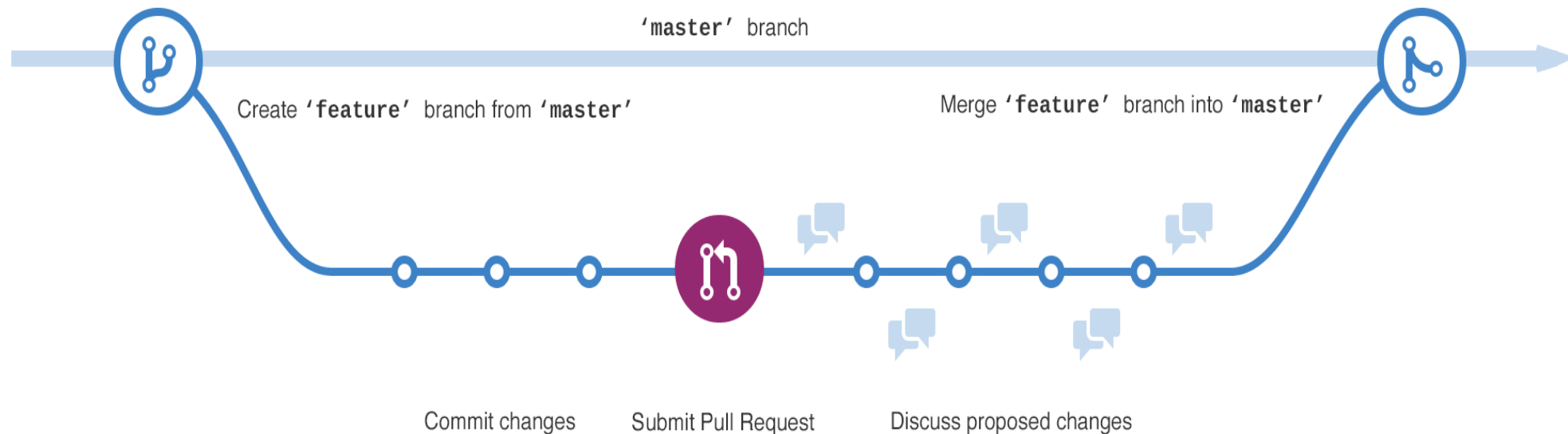
GIT Branch

- Branching is the way to work on different versions of a repository at one time.
- By default your repository has one branch named master which is considered to be the definitive branch.
- We use branches to experiment and make edits before committing them to master.
- When you create a branch off the master branch, you're making a copy, or snapshot, of master as it was at that point in time.
- If someone else made changes to the master branch while you were working on your branch, you could pull in those updates.

GIT Branch

Here you have:

- The master branch
- A new branch called feature (because we'll be doing 'feature work' on this branch)
- The journey that feature takes before it's merged into master



BitBucket/GitHub/GitLab

- **What is Bitbucket /GitHub/Gitlab?.**
- Bitbucket is a Git solution for professional teams. In simple layman language its a UI for Git, offered by Atlassian, similarly we have different available UI solutions from Github (most famous) and Gitlab.
- GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.
- **Host in the cloud:** free for small teams (till 5 users) and paid for larger teams.
- **Host on Your server:** One-Time pay for most solutions.
- Visit "<https://bitbucket.org/>" and click "Get Started" to sign up for free account.
- Visit "<https://github.com/>" for Github details