Welcome

DevelopIntelligence

A PLURALSIGHT COMPANY

Prerequisites

This course assumes you

- Know some basics of programming in any languages.
- Have never taken a formal course in Dev Fundamentals and know little or nothing about Software Development.

Please Note

If you have never programmed before that's also ok .In this course we will write some programs to demonstrate dev fundamentals.

Please send your query in chat so we can direct you towards the right resource.

Why study this subject?

- Why to take a course on Dev Fundamentals ?
- Services as an entry point in the world of Software Development.
- Understand how to work with Software Developers.

We teach over 400 technology topics

You experience our impact on a daily basis!

My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Create an inclusive learning environment
- Use an on-screen timer for breaks

…also, if you have an accessibility need, please let me know

Objectives

At the end of this course you will be able to:

- Have hands-on experience in build, integration & deployment of code.
- Clear your fundamentals on DevOps process flow.
- Understand Software and Its development lifecycle.
- Discuss software development process with various stakeholders and learn to document them as a Business Analyst.

Agenda

- Understand the influences on a project
- Understand what a software process is
- Understand two common models

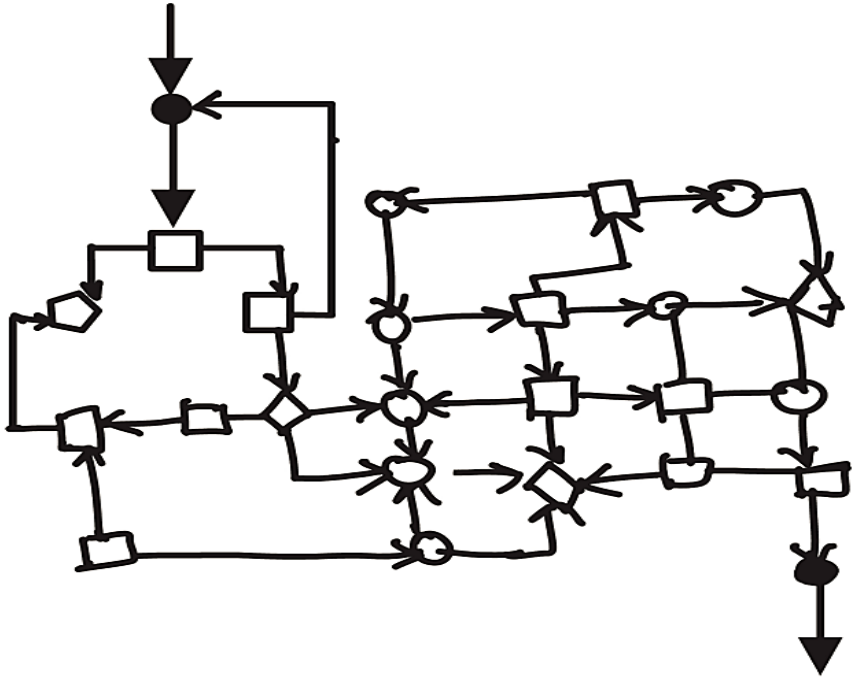
How we're going to work together

- Shortly you'll have a copy of all the course materials.
- we will be using Virtual lab environment for all our exercises which will be discussed shortly.
- You will be following along in the virtual lab environment doing coding exercises.

Timings

- You will get access to **individual Virtual Lab environments.**
- You will use this account to complete the lab exercises. You don't need to open or use your office computer browser for the labs.
- **Training time :** 08:30 AM to 12:00 PM
- **Breaks**
  - 1 break for 30 Minutes - **10 am to 10.30 am approx**
- If you have any questions please feel free to let me know.

Software Requirement



Great Software

We Trying to Accomplish ?

- Understand what is SDLC
- Describe the process of requirements gathering.
- Understand two common & most popularly used SDLC models

CIRCLE of Life

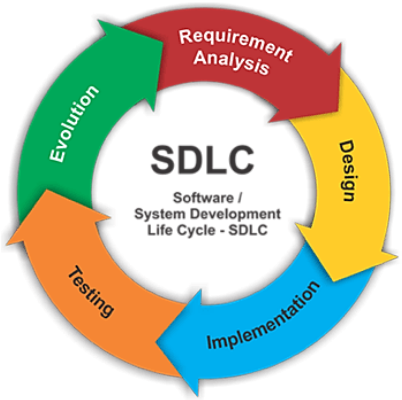
1. Teams come, operate, evolve and disband
2. People come, grow, and eventually move on
3. Projects come, grow, evolve and enter stasis

Your project has to accommodate these facts of project lifecycle.

Software Development Life Cycle (SDLC): Why?

Purpose

- Leads to good software
- Reduces risk
- Enables visibility and measurement
- Enables team teaming



Key attributes

- Outcomes of processes are key deliverables or products
- Roles are clear
- Pre- and post-conditions are understood and held true

SDLC Key Elements

1. Feasibility
2. Specification
3. Architecture and Design
4. Development
5. Validation
6. Maintenance
7. Evolution

The devil is in the details of how the steps are organized and executed

The Promise and the Reality

Nothing is ever as simple as it seems

SDLC: Process Models

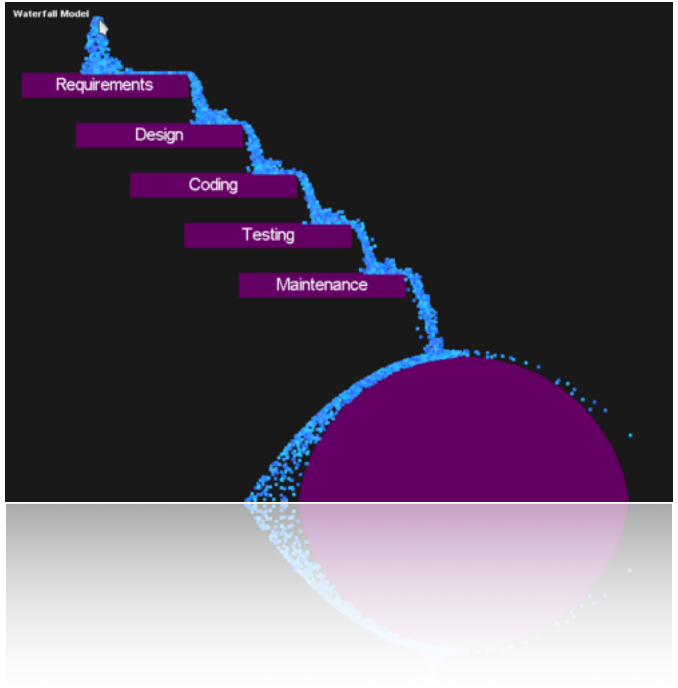
There are various software development life cycle models defined which are designed to organize software development process.

- Waterfall
- Iterative
- Agile
- Rapid Application Development (RAD)

SDLC: Waterfall Model

Waterfall

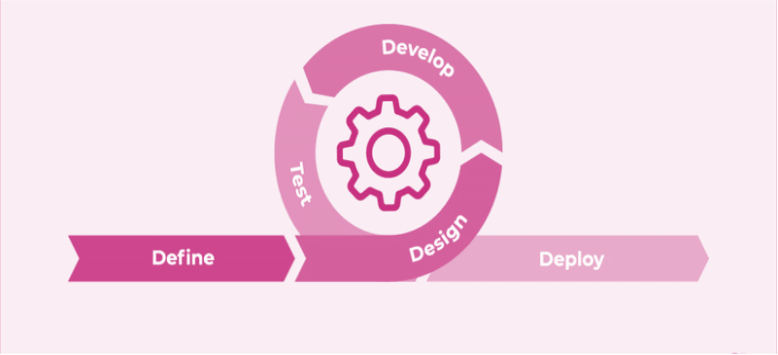
Follows a step-by-step sequential process where each step waits on the previous one
Design and develop all features of an application in one go.



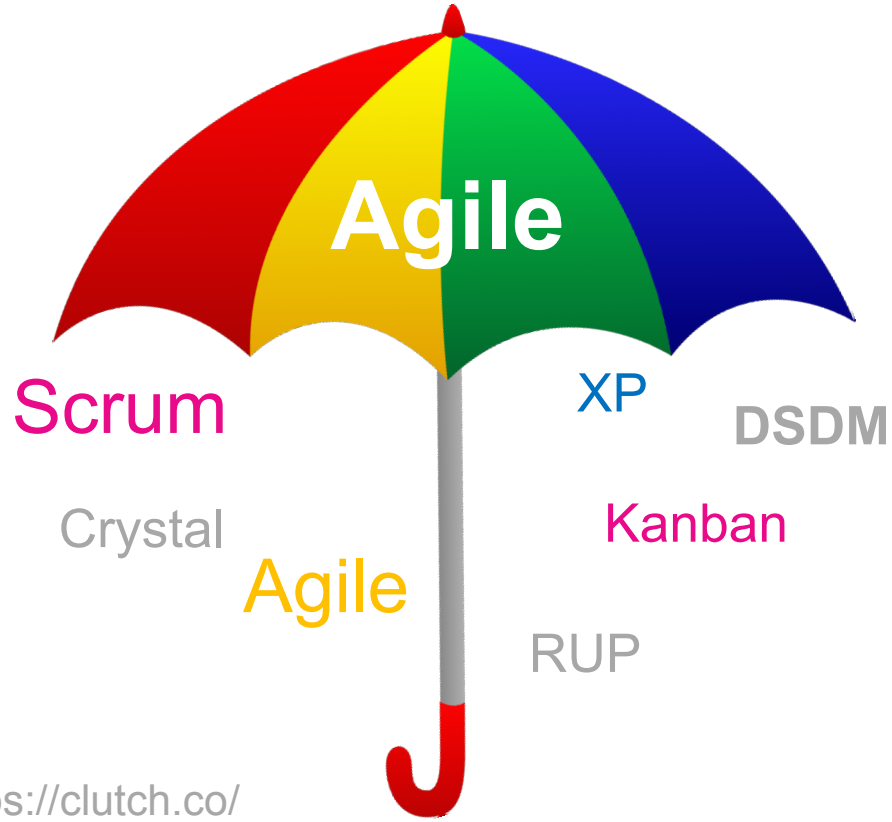
SDLC: Agile Model

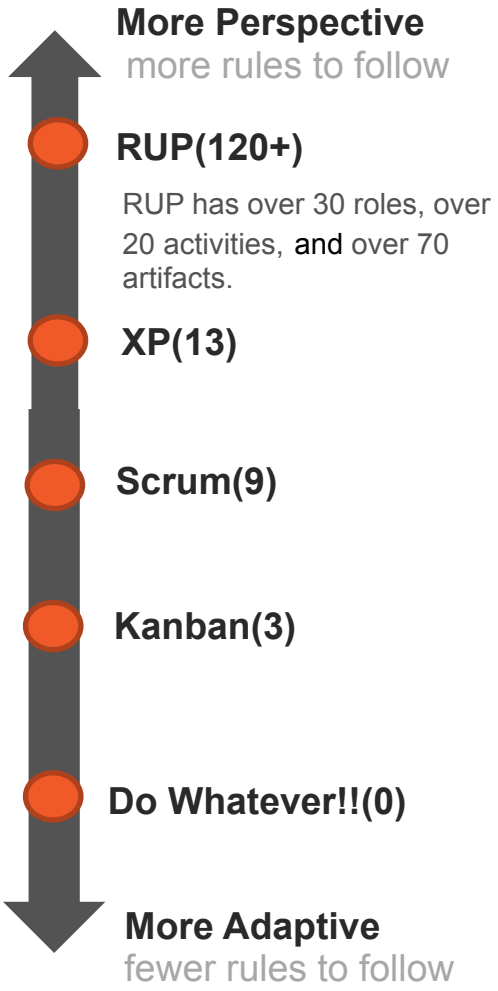
Agile

- Built to adapt to changes
- Getting to MVP is the first goal
- Relies on quick and tight feedback loops
- Experimentation is good, as long as we measure the results
- Highly analytical



SDLC: Agile Umbrella





source https://clutch.co/

SCRUM

In rugby, a **scrum** is how you restart the game after a minor infraction like a goal or timeout..

Agile Scrum

Emphasis on small, semi-independent teams ideally delivering discrete pieces of a system.
Team ideally has total responsibility for the components it produces.

Leads to DevOps model

1. Team
    - Small, cross-functional, self-organizing units
2. Scope
    - Small deliverable scope delivered in consensus priority order
    - Priorities can be adjusted (typically at sprint start)
3. Timeline
    - Small iterations (2-3 weeks is typical) emphasizing delivery at the end

SDLC: Agile vs Waterfall

- Agile is a continuous iteration of development and testing in the software development process.
- Agile methodology is known for its flexibility
- Agile follows an incremental approach
- Agile performs testing concurrently with software development
- Agile allows changes in project development requirement
- Test plan is reviewed after each sprint

- Waterfall is a Linear or Sequential Life Cycle Model
- Waterfall is a highly structured software development methodology
- In Waterfall methodology testing comes after the “Build” phase.
- Waterfall has no scope of changing the requirements once the project development starts.
- The test plan is rarely reviewed during the test phase.

What is agile ?

The simple definition of agile is **step-by-step.** Take something big, and break the most urgent thing into smaller pieces. By taking small steps you can see clearly if you're doing the right thing. If not, you can adjust course.

What is agile ?

A way of getting things done

An empirical method to project management

Continuously develop the plan , process and product

A mindset

Why agile ?

Effectiveness

Perform better than traditional projects

Empower the Team

Leverage team knowledge and increase job satisfaction

Manage Complexity

Simple project management approach to increasing complexity

Diving into Agile Methodologies

Agile as an approach for projects is all about ensuring what is being built is valuable.

“Who knows” ? about a **project** that never been able to add value to a customer ?

THAT’S WHERE BA’s COME IN

Agile Manifesto

https://github.com/dataarjun/Training_Contents/blob/main/Excercise/The_Agile_Manifesto.pdf

Agile Tools

Jira

monday.com

Airtable

Trello

asana

ClickUp

What is Jira ?

JIRA is

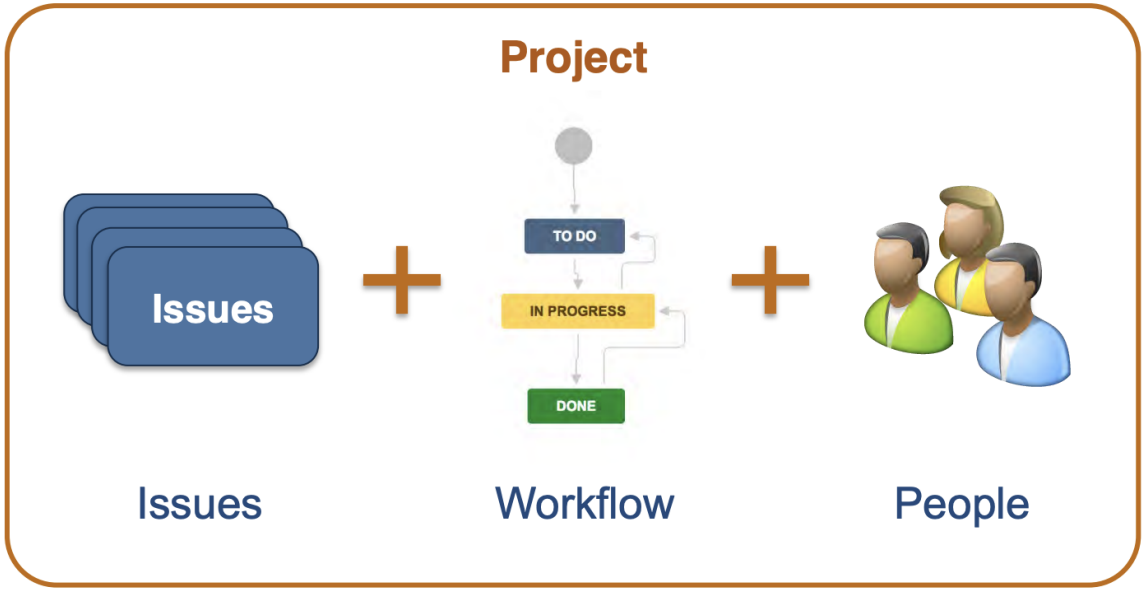
- A virtual work planning and delivery tool
- Customizable for your company and project needs
- Provides transparency and real-time reporting

What can JIRA do for you?

JIRA can help you manage your projects

- Coordinate efforts to deliver a product
- Manage and hold all your projects in one location
- Track and organize enterprise-wide initiatives
- Follow team activity with real-time reporting
- Monitor single or multiple projects
- Provide visibility and transparency

JIRA Projects



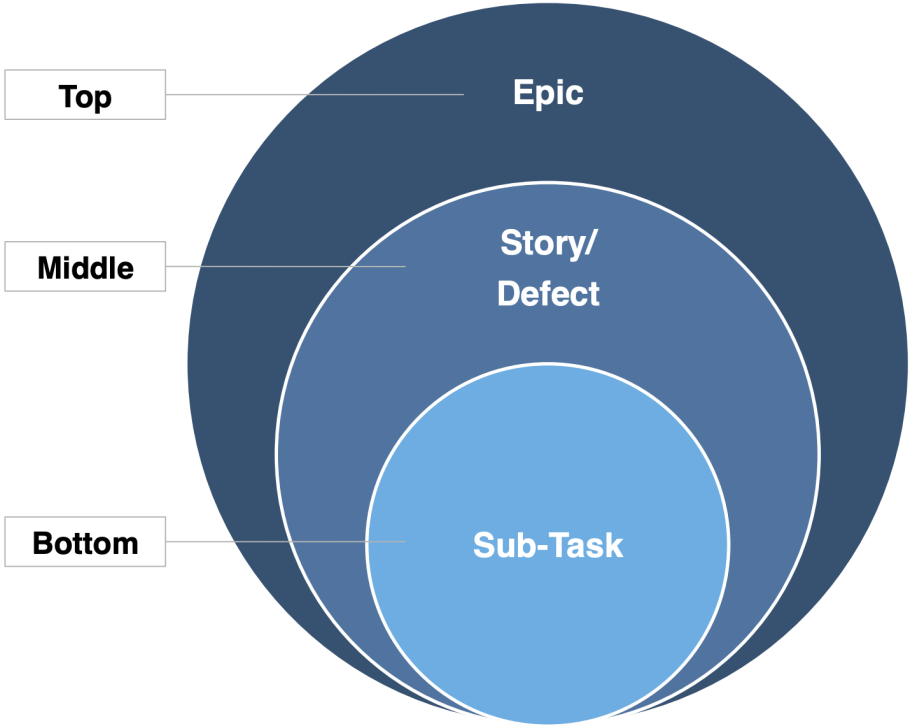
What is an Issue ?

Issues are the JIRA term for a requirements

- **Story**
- **Defect**
- **Improvement**
- **Request**
- **Epic**
- **Task**
- **Almost anything we want!**

- These are the basic elements of JIRA and they follow a predefined hierarchy
- Each Issue has specific fields that hold it’s details and attributes

JIRA Standard Issue Type Hierarchy



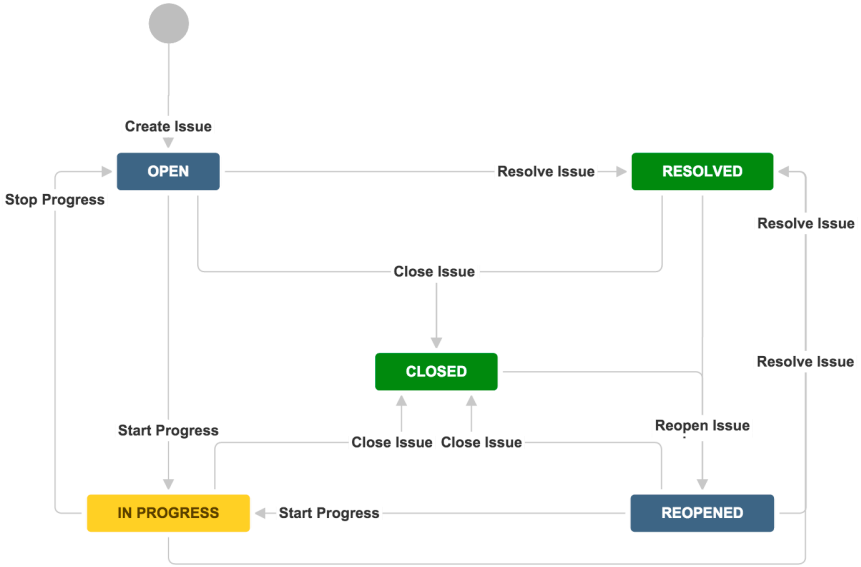
Common JIRA Agile Issue Types

Hierarchy	Issue Type	Description
Top	Epic	A large body of work that can be broken down into many Stories
Middle	Story	A requirement written in a few sentences, the most used Issue Type
Middle	Defect	An unplanned Issue related to improperly working software or systems written in a few sentences
Bottom	Sub-Task	A unit of work contained in a Story, called a Task in Scrum

JIRA Workflows

JIRA uses a workflow to manage Issue delivery

- Issues must follow the workflow states
- Customizable for each Issue Type
- Applied to projects and issue types by Project Admin

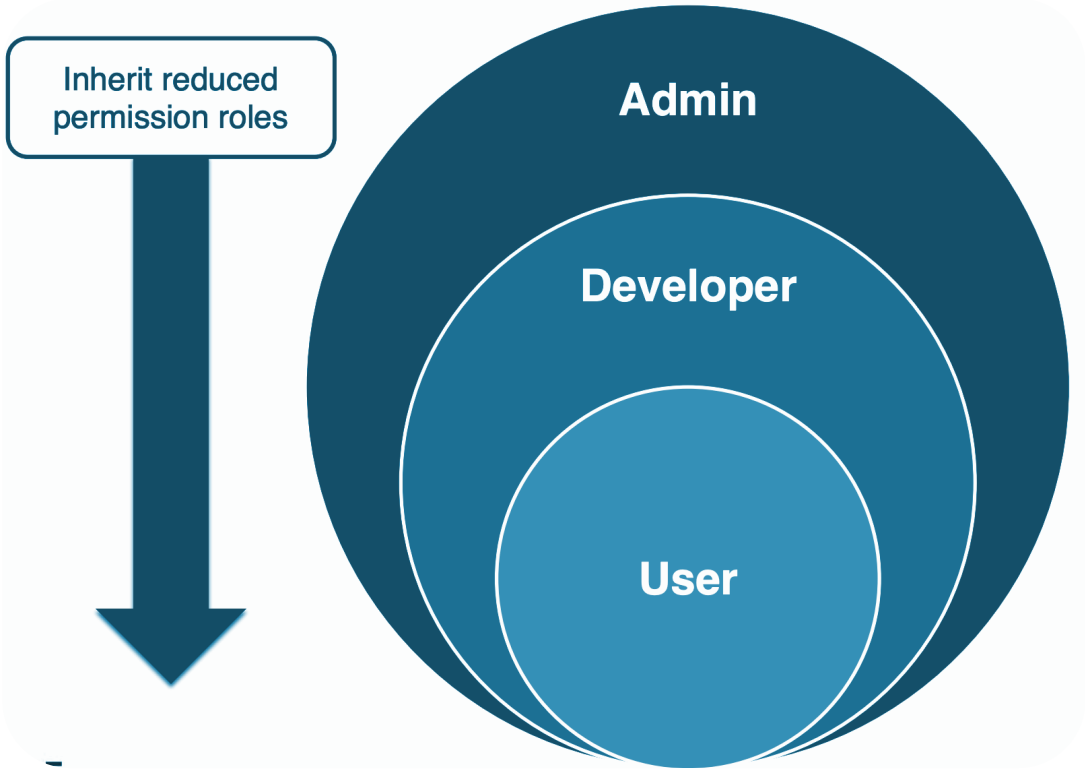


Project Roles

- Project roles are a flexible way to associate users and groups with a particular project
- Used in conjunction with Permission Schemes to define access and activity permissions to your project

Default Project Role	Definition
Administrator	Usually associated with resources who administer the Project
Developer	Usually associated with resources who work on and manage Issues
User	Usually associated with resources who will view work

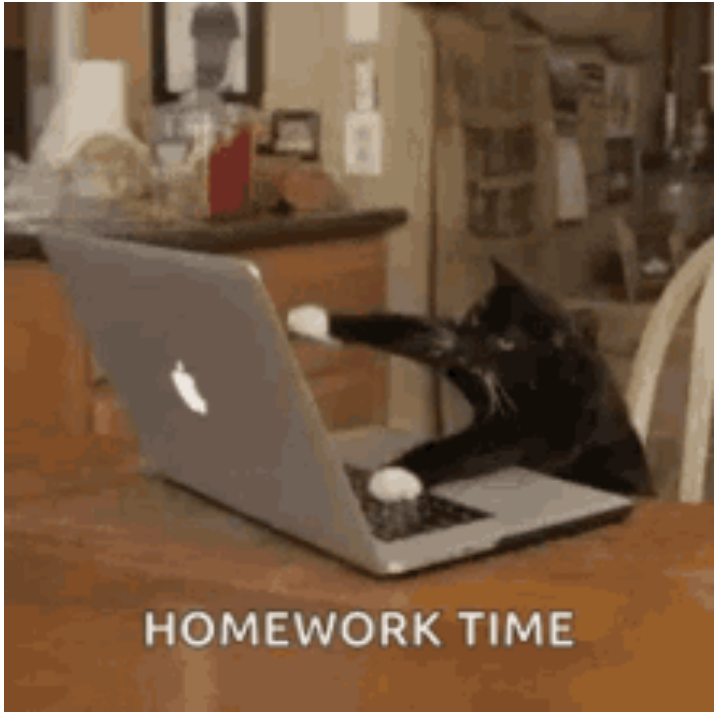
Project Roles



Demo

Exercise

Using JIRA Atlassian https://www.atlassian.com, add the user stories as a storyboard based on the case study given in the earlier exercise.



What are we going to learn?

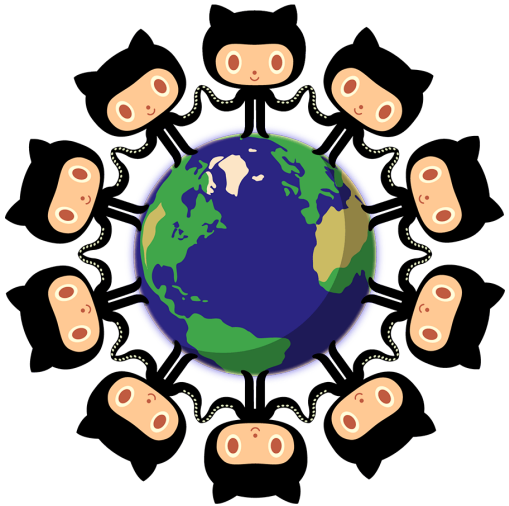
- What makes Git special ?
- Git and the basics of Git
- What is version control ?
- Popular Git-based services

What is Git?

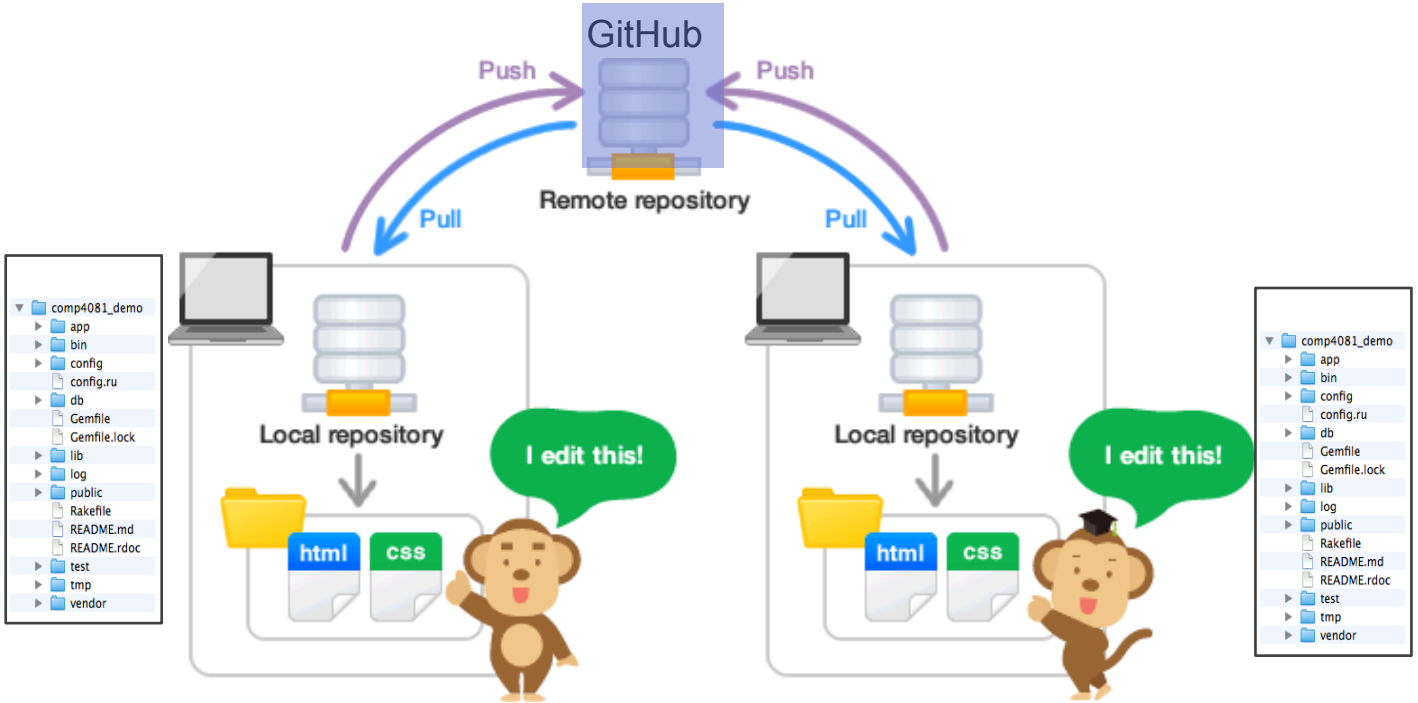
Git is a Version Control System (VCS) designed to make it easier to have multiple versions of a code base, sometimes across multiple developers or teams. (**It is not same as GitHub**)

Then what is Github?

Github.com is a website that hosts git repositories on a remote server



How GitHub works?



By pushing your repositories to Github, you will pretty much automatically create your own developer portfolio as well!

Some popular Git Hosting Services

Here the three most popular Git hosting services:

GitHub: Owned recently by Microsoft — Launched in 2008 (31 million users in October 2018).

GitLab: Owned by GitLab Inc. — Launched in 2011.

BitBucket: Owned by Atlassian — Launched in June 2008.

Confirm that you have git

VS Code

1. Open your vs code terminal and run ‘git’
2. If you see a ‘command not recognized’ error, you probably haven’t installed git yet.
3. We can solve your issue once we start the lab session – sit tight for now!

Basic Git Commands

command description	command description
`git clone url [dir]`	copy a Git repository so you can add to it
`git add file`	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help [command]`	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository

others:init,reset,branch,checkout,merge,log,tag

Git Workflows

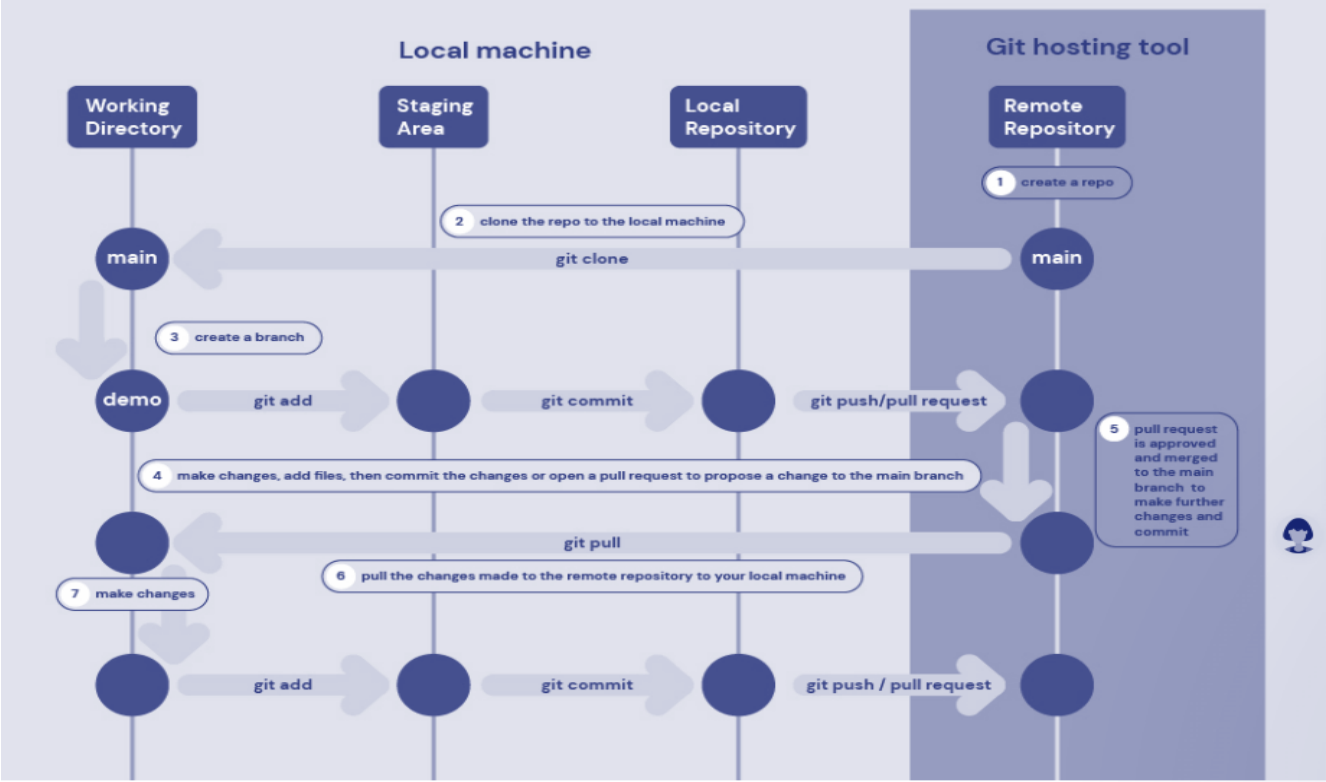


Image source: https://www.hostinger.com/

Initialize Git

Initializing an empty git repository.

`git init` # Initialize an empty git repository under current directory.

Cloning a repository

`git clone https://github.com/UTXXXYZ/git-test.git` # Clone this repository to your current working directory

Commit Files

Make a commit

`git commit` # Make a new commit with the changes in your staging area. This will open an editor for a commit message.
`git commit -m` "I love documentation" # Make a new commit with a commit message from the command line
`git commit -a` # Make a new commit and automatically "add" changes from all known files git commit -am "I still do!"
A combination of the above
`git commit --amend` # Re-do the commit message of the previous commit (don't do this after pushing!)

We never change "public history“

`git reset <file>` # Unstage a staged file leaving in working directory without losing any changes.
`git reset --soft` [commit_hash] # resets the current branch to <commit>. Does not touch the staging area or the working tree at all.

Remote Git

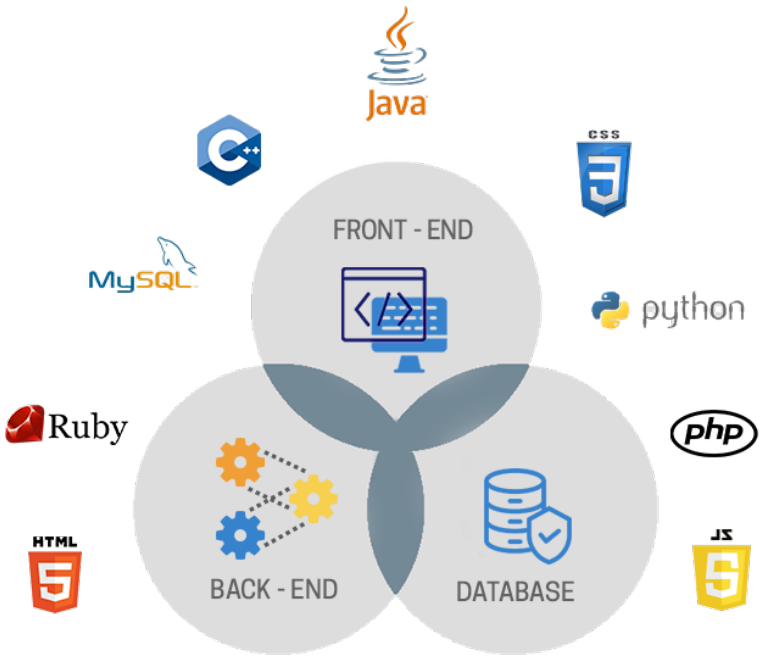
Working with Remotes

`git remote` # Show your current remotes
`git remote -v` # Show your current remotes and their URLs
`git push` # Publish your commits to the upstream master of your currently checked out branch
`git push -u origin my-branch` # Push newly created branch to remote repo setting up to track remote branch from origin.
No need to specify remote branch name, for e.g., when doing a 'git pull' on that branch.
`git pull` # Pull changes from the remote to your currently checked out branch

What are we going to learn?

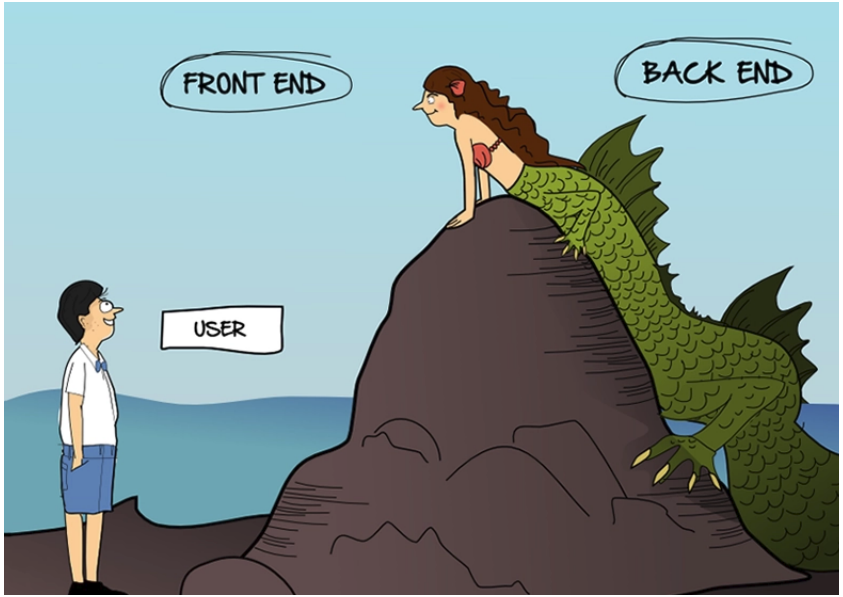
- What’s Full Stack Development?
- Full Stack technologies & Frameworks
- Hosting and Infrastructure

What is Full Stack Development



Who is a Full Stack Developer?

A full stack web developer is a person who can develop both **frontend** and **backend** software.



Front End Development Languages

Front End Development Frameworks

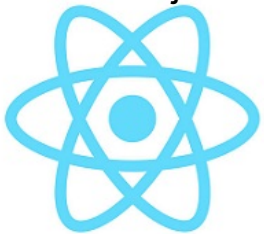
Bootstrap

Angular

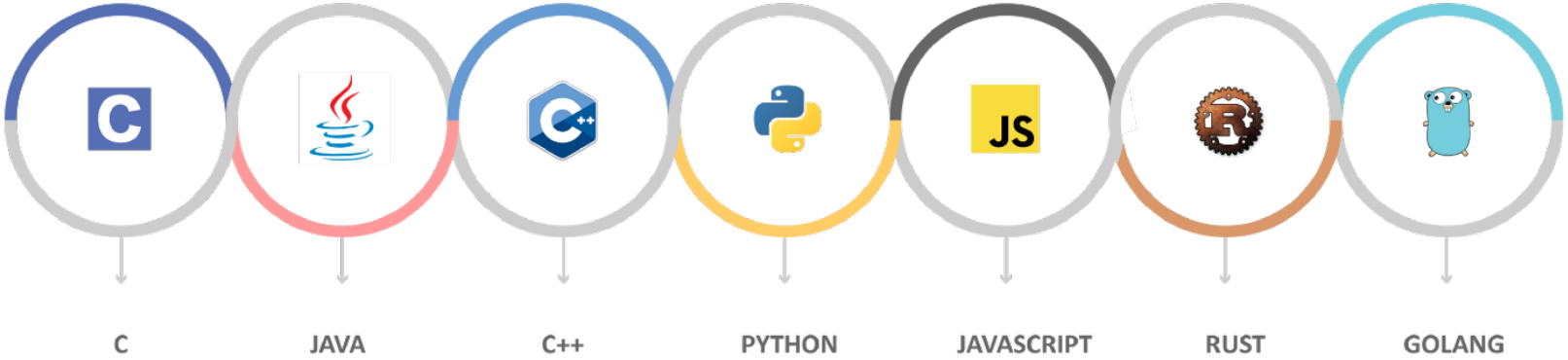
Vue.js

Materialize

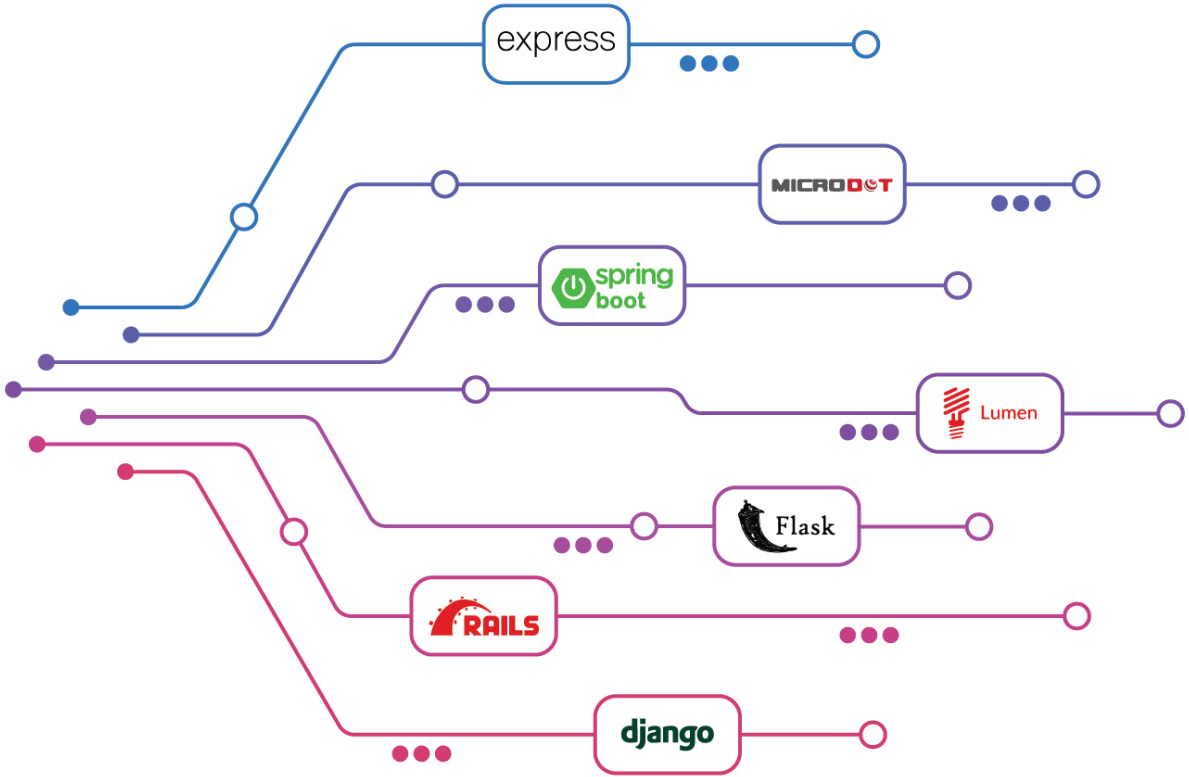
React.js



Back End Development Languages

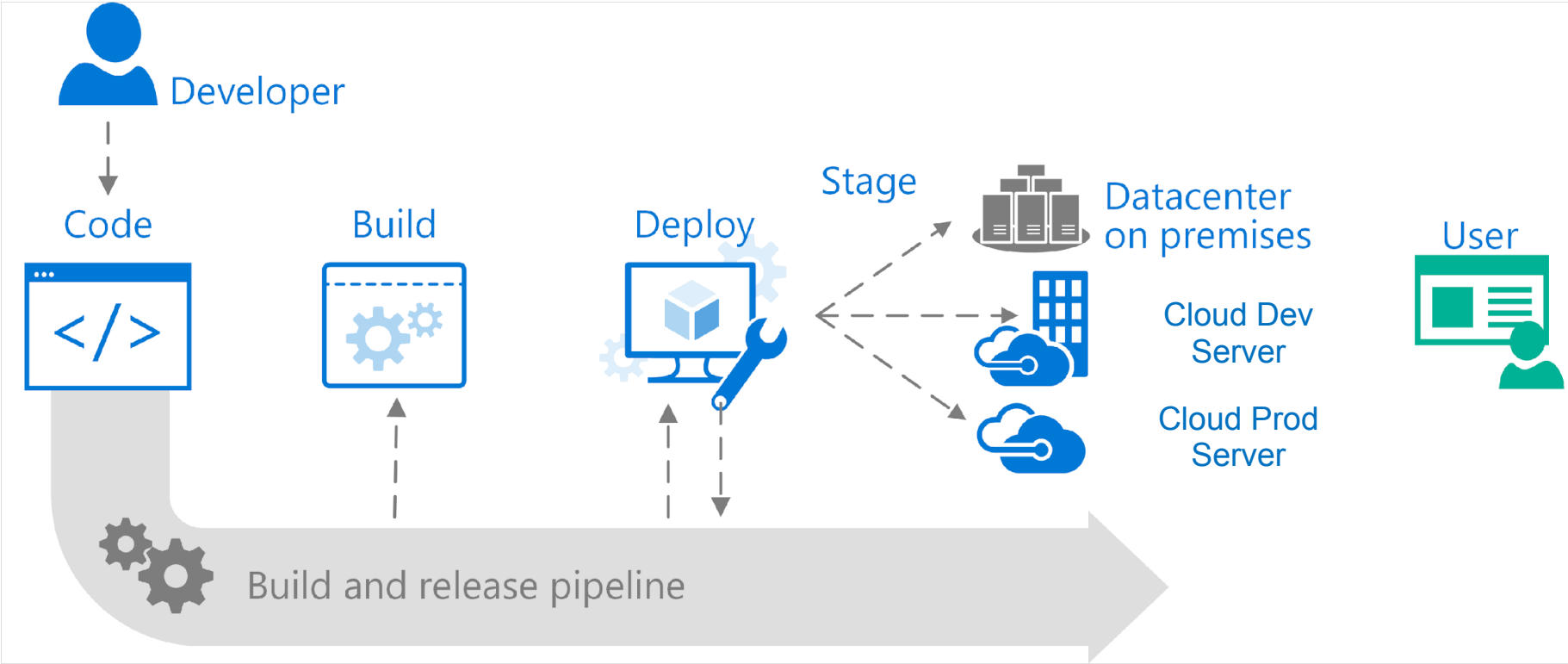


Back End Development Frameworks



Software Deployment

Software Hosting & Deployment

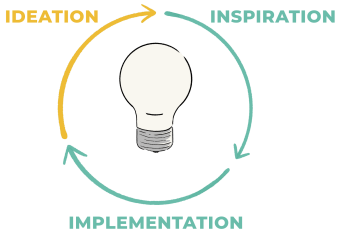


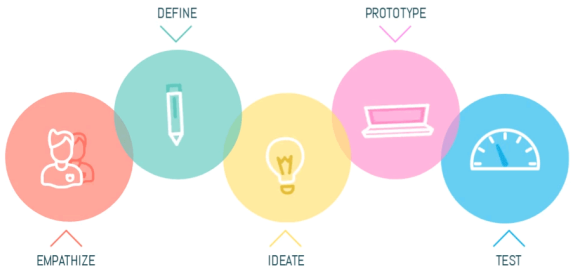
Full-stack Business Analyst

- Instead of a technology stack, the Full-Stack Business Analyst focuses on the Product Stack, which is comprised of:
  - **Ideation:** Discovering possible opportunities
  - **Validation**: Ensuring that this is a good/the right opportunity
  - **Visioning**: Creating a shared understanding and leading through development of a compelling vision
  - **Decomposition**: Breaking the vision into smaller achievable chunks and identifying experiments to test and learn
  - **Delivery**: Building the solution and getting those solutions into the hands of our customers
  - **Monitoring**: Analyzing metrics to validate success and identify new opportunities

Ideation: Discovering Opportunities

- Ideation is where you use data, your knowledge of the business environment, knowledge of your organization's capabilities, and your understanding of your customers to discover ideas for potential solutions.

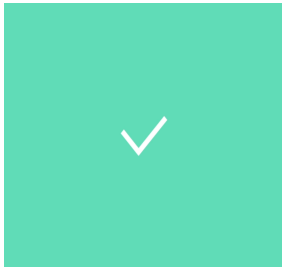


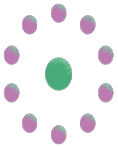


- Skills and knowledge needed for ideation include problem analysis and identification of opportunities through **Design Thinking** or **Lean Startup** and tools like personas, empathy maps, customer journeys, and experiments.

Validation: Confirming it’s the Right Opportunity

- Validation occurs when you analyze your idea to make sure there’s good problem/solution fit as well as good product/market fit.





- You’ll likely develop a business case as part of the validation to ensure that the product idea is worth the investment and to help with prioritization.

Visioning: Gaining Buy-In & Alignment

- Once you have validated the product idea, you'll need buy-in from your organization to get the time, money, and other resources you need to make the idea come to life.

- This involves **influencing, using data from your earlier validation, creating a compelling vision, and creating alignment** by developing a shared understanding of the **problem, proposed solution, metrics, and expected outcomes.**

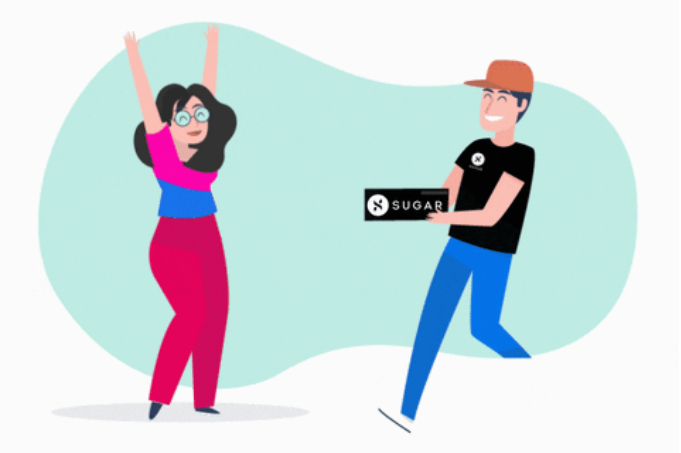
Decomposition: Making work Manageable

- Even if you have a great product idea and alignment in your organization, it's difficult to understand and deliver the solution in one big piece. We need to break the work down into smaller, more manageable pieces.

When you decompose work, you'll likely identify the **Minimum Viable Product (MVP) and Minimum Marketable Features (MMF).** You can also use techniques such as Story Mapping to create a holistic view of the initiative as well as thin slices to deliver.

Delivery: Getting the right solution to Customer

- Once you have alignment and an understanding of how the product can be delivered, it's time to build the solution. Whether you're developing in iterations or a big bang implementation, you'll want to create fast feedback loops and adjust your approach along the way.



The other component beyond building the solution and testing it is getting the product into the hands of customers. Knowledge about how your organization delivers products will help you provide more value and reduce delays. This may mean learning about **DevOps practices.**

Monitoring: Metrics for Success

- Just because you successfully deliver the product, your job as a **Full-Stack Business Analyst** isn't done. How do you know if you achieved the intended outcome?

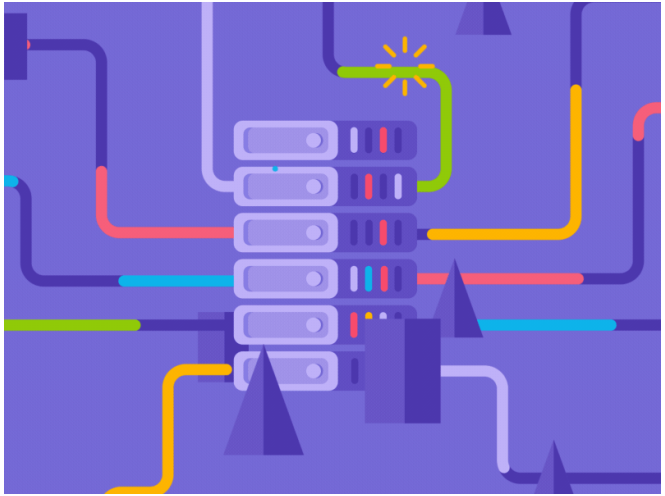
- You need to be able to **monitor the product, its usage, and metrics.** Doing so allows you to validate your original hypothesis and helps you to discover new opportunities.

DFD: Data Flow Diagrams

- A **Data Flow Diagram** (DFD) is a graphical representation of the flow of data through a computer system.

OR

- A Data Flow Diagram looks at how data flows through a system.
- It concerns things like where the data will come from and go to as well as where it will be stored.



DFD is not a Flow Chart

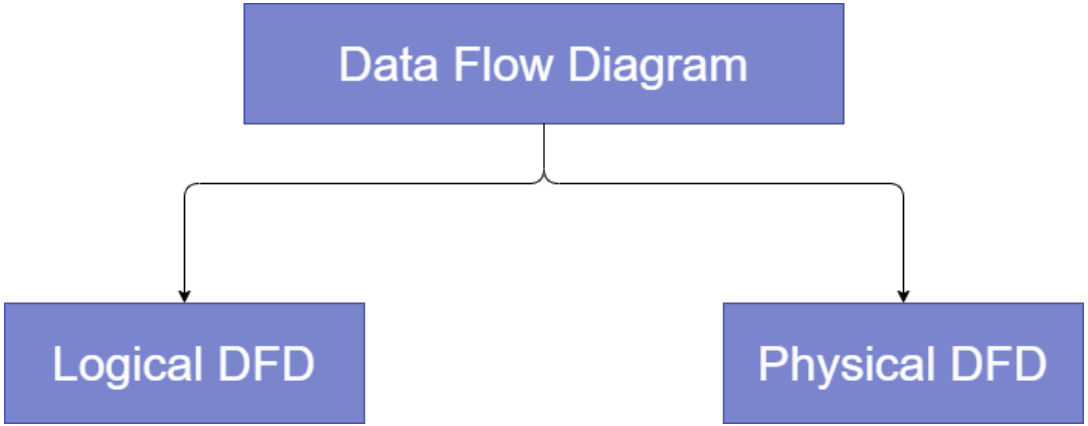
- Sometimes people get confused between data flow diagram and flowchart.

- The flowchart illustrates **flow of control** in program modules.
- Data flow diagrams illustrate **flow of data** in the system at various levels.

Data flow diagram does not have any **control or branch** elements.

Types of DFD

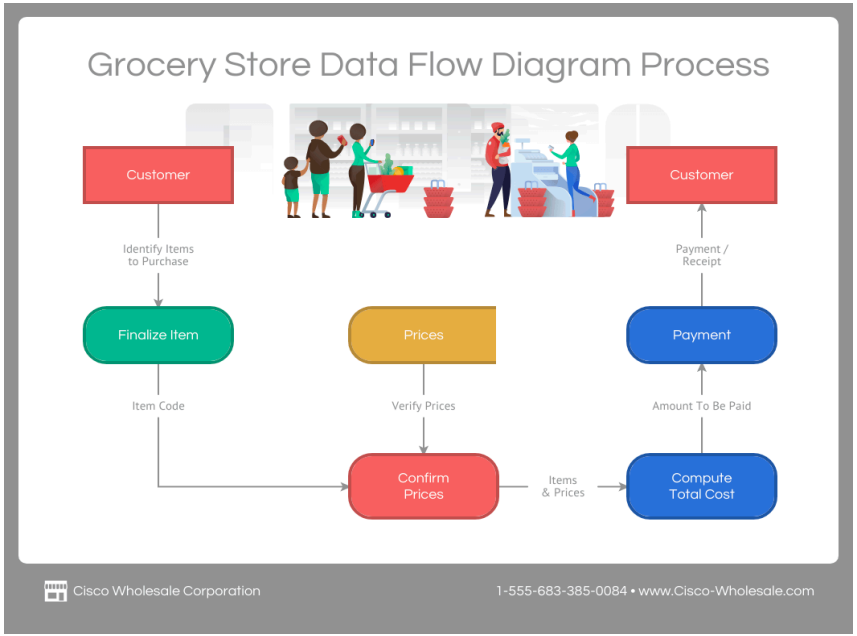
DFD if of two types



Logical DFD

1. **Logical DFD:**

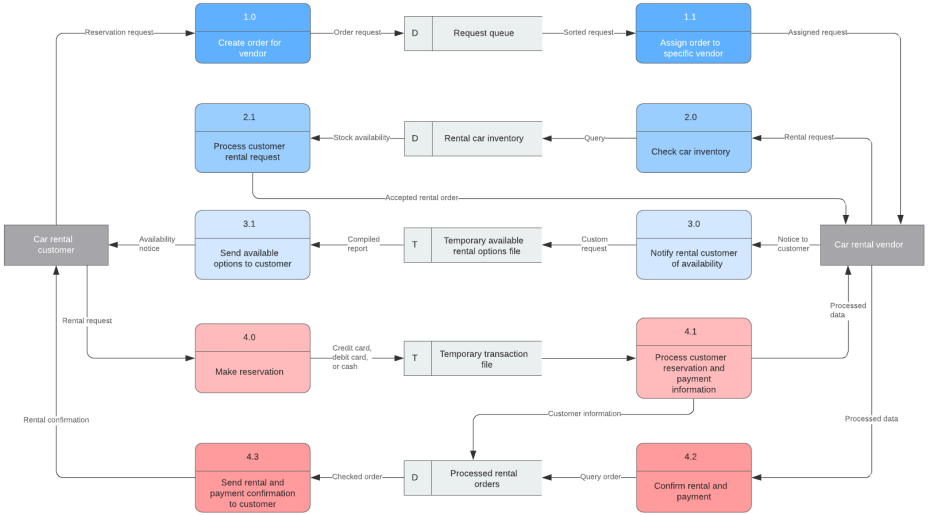
   A logical DFD shows the conceptual flow of information through a process, focusing on the kind of data needed, its source, destination, and the transformation desired from the process. Here's an example.



Physical DFD

1. **Physical DFD:**

   On the other hand, a physical DFD visualizes how the system plan will be executed, including the specific tools, hardware, or players needed to make it work. Just like this example.



DFD Symbols

Notation	Yourdon and Coad	Gane and Sarson
External Entity		
Process		
Data Store		
Data Flow		

Exercise

Create a simple high level data flow diagram that depicts the frontend and backend processes for the Online Pizza Delivery shop.

Example DFD in Lucid Chart

Why DevOps ?

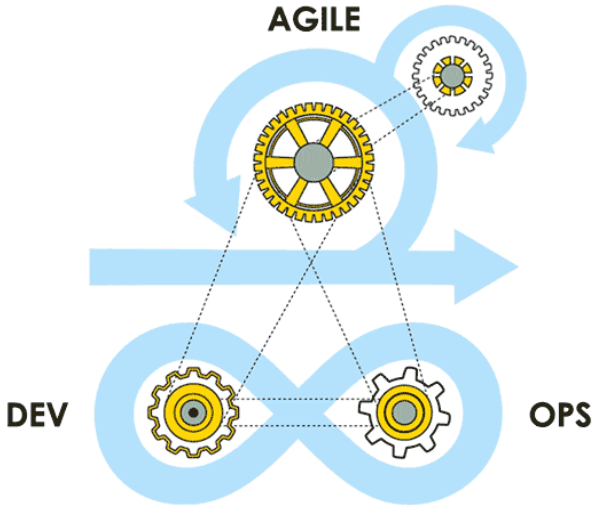
- Are you troubled by the strange ways your organization delivers software ?
- Do you experience feelings of dread when preparing long painful application release?
- Have you or your co-worker experienced buggy , unstable services after deployment?
- Does it take day, weeks or months to build and deploy a application ?

If your answer is **YES**

You need **DevOps**

What are we going to learn?

- Devops Fundamentals
- CI/CD
- Automated builds, testing and deploying code

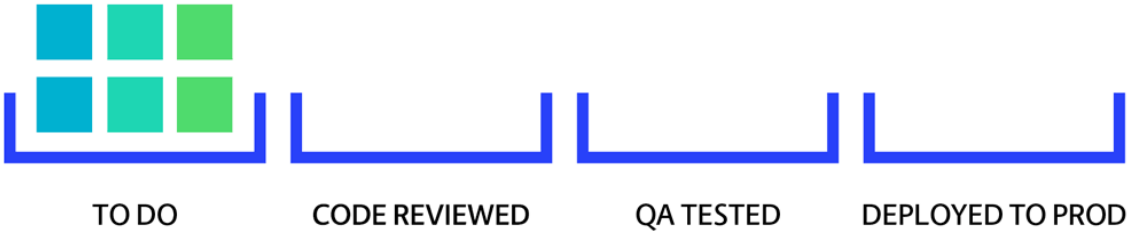


Continuous Integration

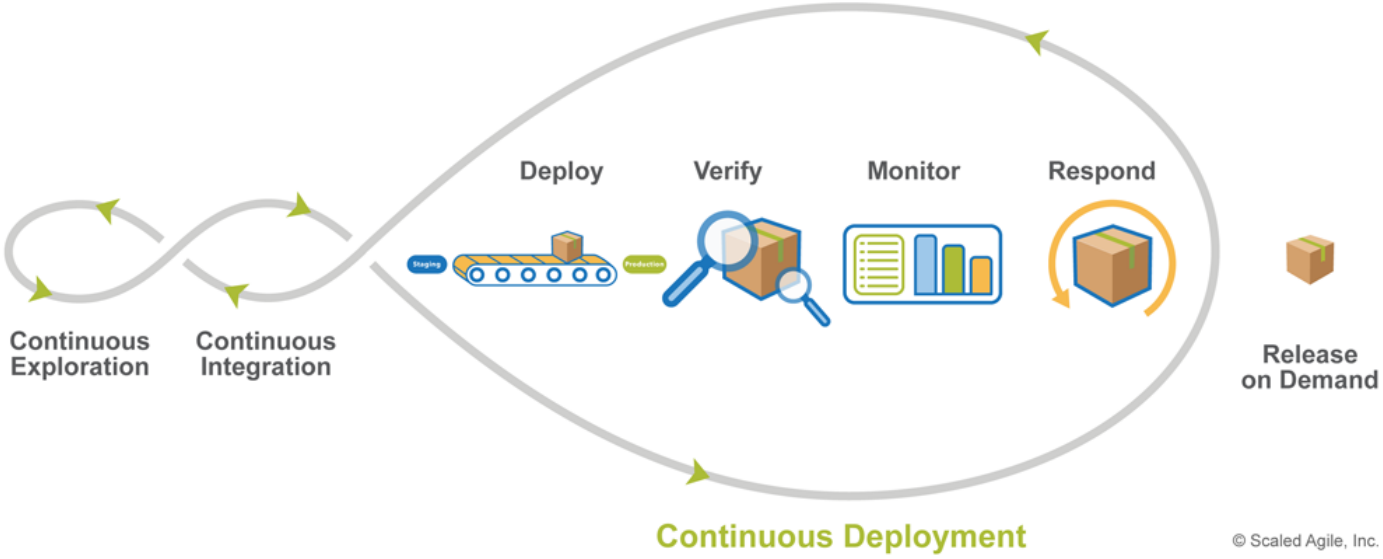
Continuous Integration (CI) is a DevOps software development practice that enables the developers to merge their code changes in the central repository. That way, automated builds and tests can be run.

Continuous Deployment

Continuous Delivery (CD) is a DevOps practice that refers to the building, testing, and delivering improvements to the software code. The phase is referred to as the extension of the Continuous Integration phase to make sure that new changes can be released to the customers quickly in a substantial manner.



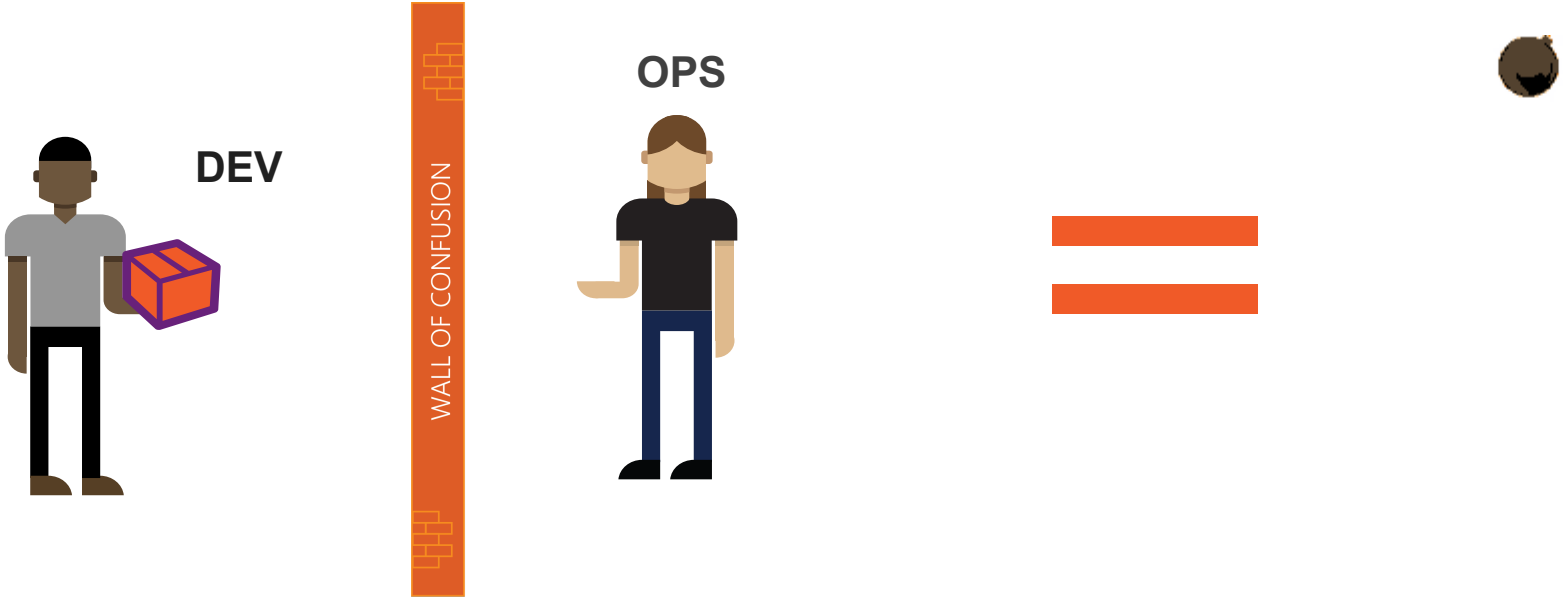
CI/CD



Source: https://puppetlabs.com/

The Devops Story

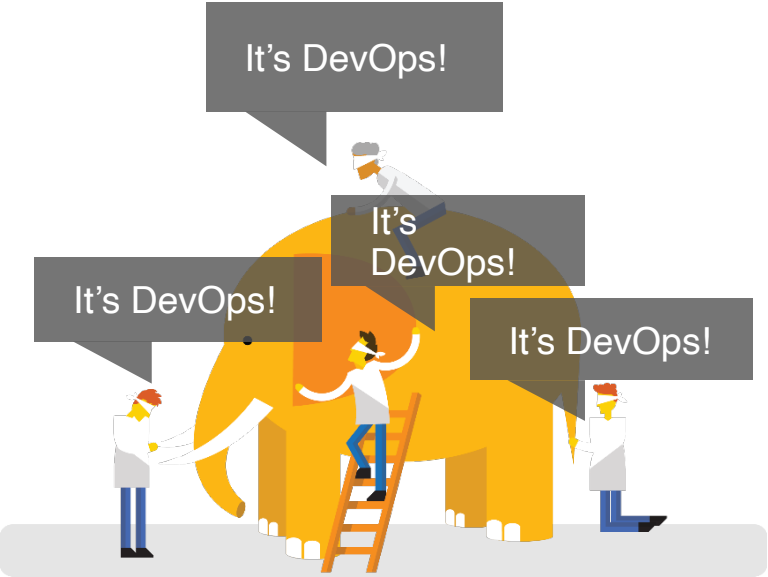
Traditional Development and Operations



“**DevOps** is development and operations collaboration”

“**DevOps** is using automation”

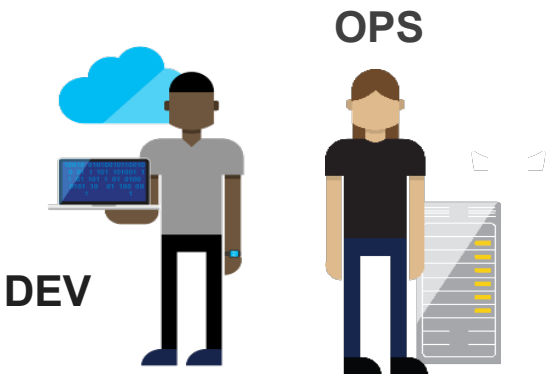
“**DevOps** is small deployments”

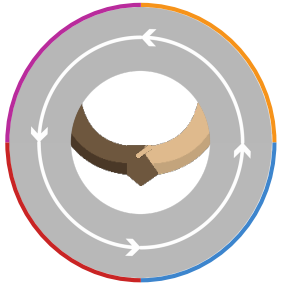


“**DevOps** is treating your infrastructure as code”

“**DevOps** is feature switches”

DevOps: the three stage conversation





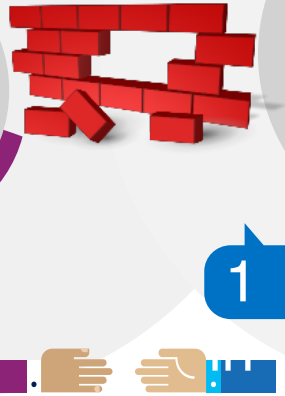
1 People

2 Process

3 Products

The consequences of inefficiency

4 % … of development budgets for software, IT staff and external professional services will be consumed by **poor requirements**



3/4 of team members **have** **Agile methodologie**

1 6 **IT decision makers** is still unfamiliar with the term DevOps

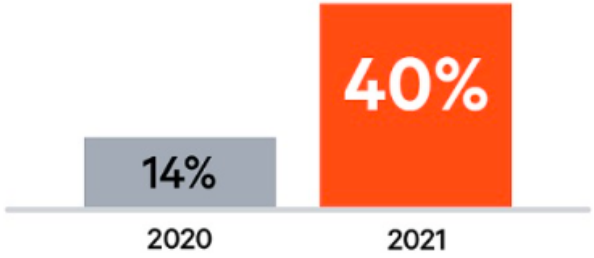
DevOps was being initiated by more development teams than IT Ops teams by about a **40%** to **33%** margin

Devops Benefits

- Strong IT Performance is a competitive advantage

Firms with high-performing IT teams were 2x as likely to exceed their profitability, market share, and productivity goals

- DevOps practices improve IT performance



- Deploy code 30x faster

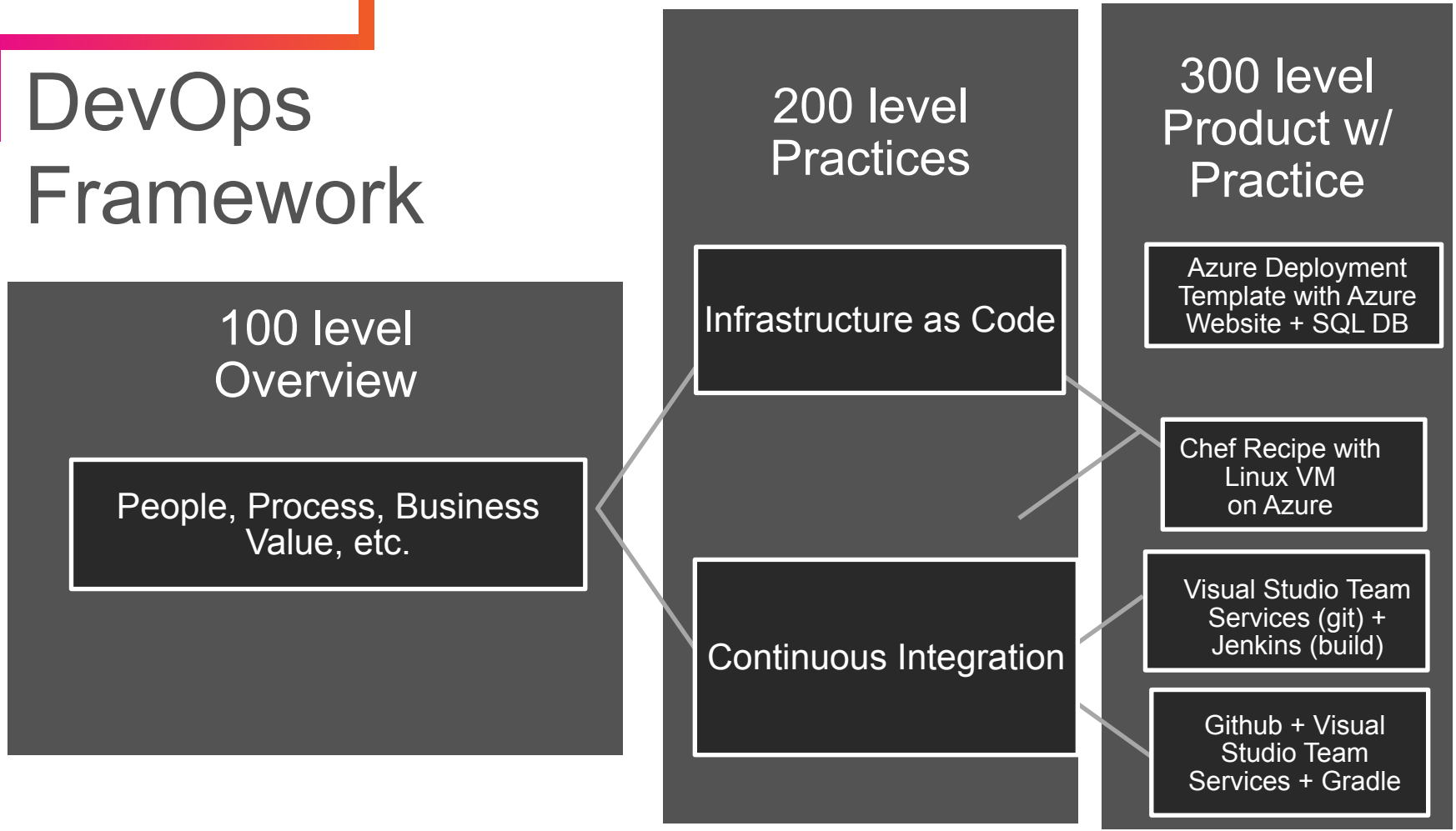
and with 200x shorter lead time as compared to your lower-performing peers

- Have 60x fewer failures

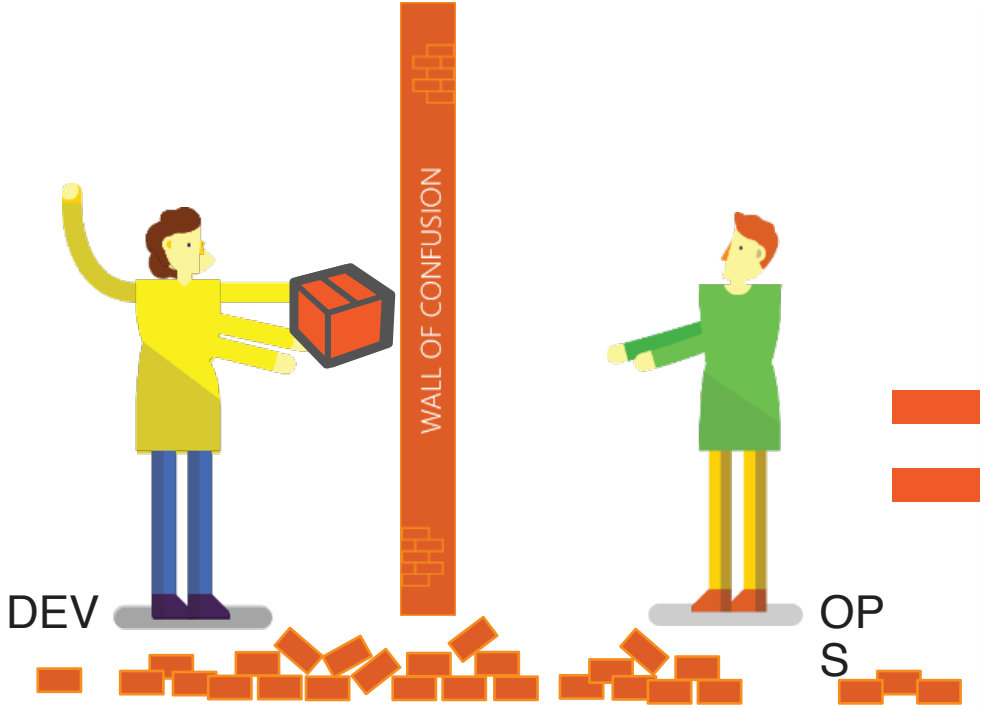
and recover from failure 168x faster as compared to lower-performing peers

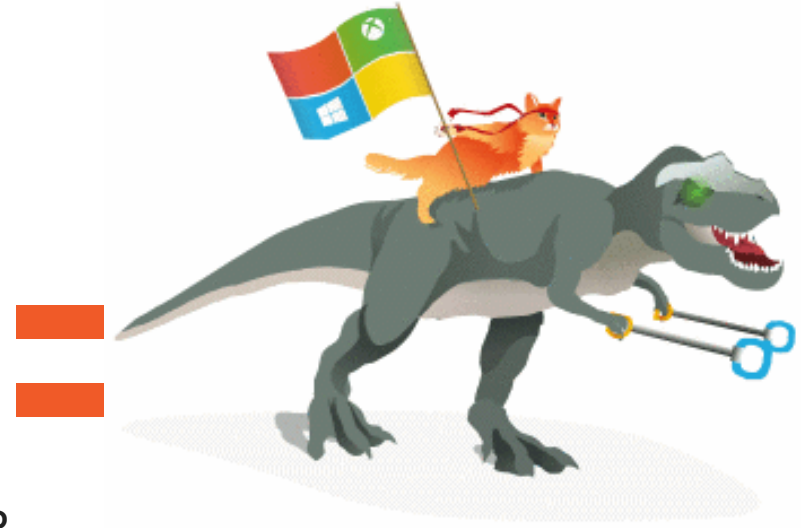
Source: www.devopsdigest.com

DevOps Framework



Happy DevOps





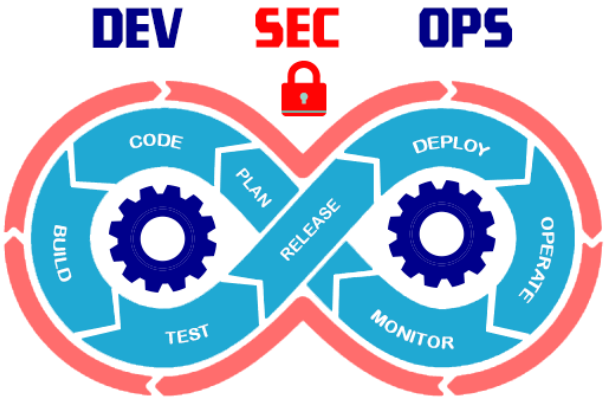
Inspired by Microsoft

Benefits of DevOps ?

- Empowering teams
- Lowered cycle time (shortens lead times for changes, time to market goes down, lower MTTR)
- Better security (and quality increases, not decreases)
- Rhythm of practice (limits your work in progress)
- More time to be productive

What is DevSecOps

- **DevSecOps** is a set of disciplines combining **development**, **security**, and **operations**.
- It is a philosophy that helps **software development businesses** deliver innovative products quickly without sacrificing security. T
- This allows **potential security issues** to be identified during the development process – and not after the product has been released in line with the emergence of continuous software development practices.



Exercise

1. Note down the steps in DevOps

1. Create an application diagram that depicts the end to end flow of the application OnlinePizza store application.

Types of Testing

- Unit testing
- Integration testing
- Code hygiene
- TDD, BDD, ATDD
- Infrastructure testing
- Performance testing
- Security testing

Unit Testing

- **Unit tests** are very low level and close to the source of an application.
- **Unit testing** is performed at build time on a single unit of code and/or artifact without use of external dependencies or deployment

Integration Testing

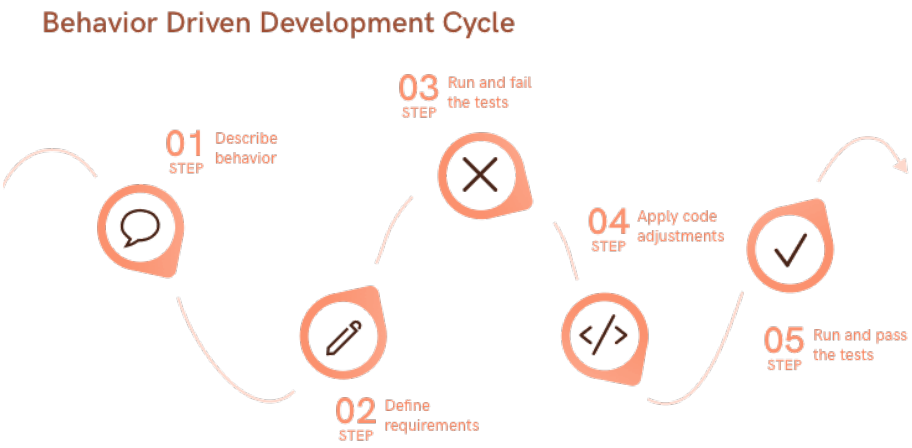
- **Integration testing** is performed as you bring together pieces of your application and as it needs to use external dependencies—databases—to actually do its thing

TDD Testing

- TDD, or **test-driven development**, is the practice of writing a failing test first, and then writing the code that causes the test to pass, and then refactoring it to make it cleaner

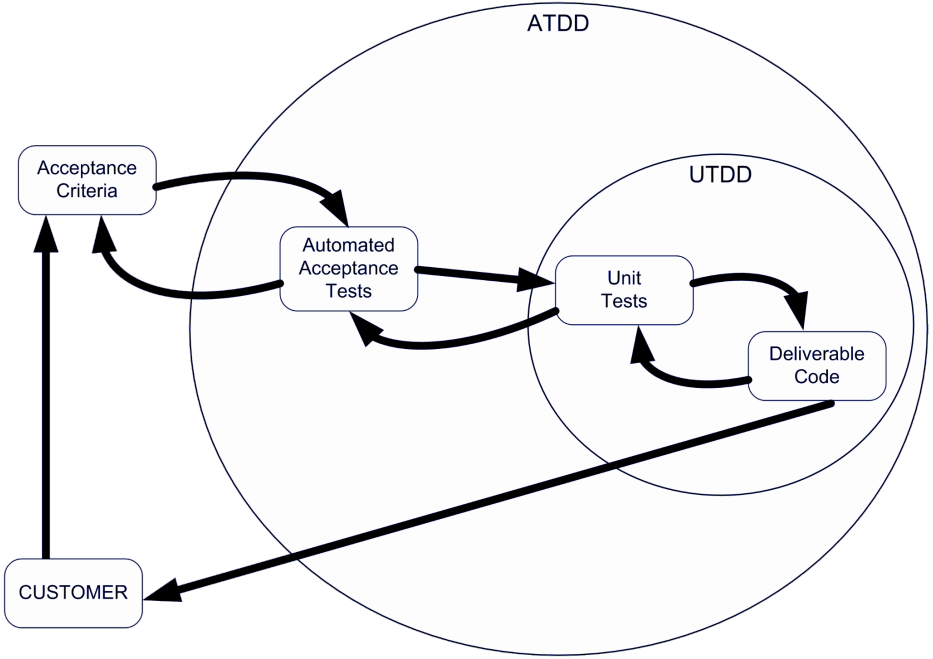
BDD Testing

- BDD, or behavior-driven development, is a refinement of TDD that focuses on simple sentence-driven testing
- The scenarios in BDD have several purposes:
  - Illustration of the requirements
  - Documentation of the behavior
  - Automated tests



ATDD Testing

- ATDD, or **acceptance test-driven development**, extends this to where the project team decides on a set of BDD acceptance tests before development begins

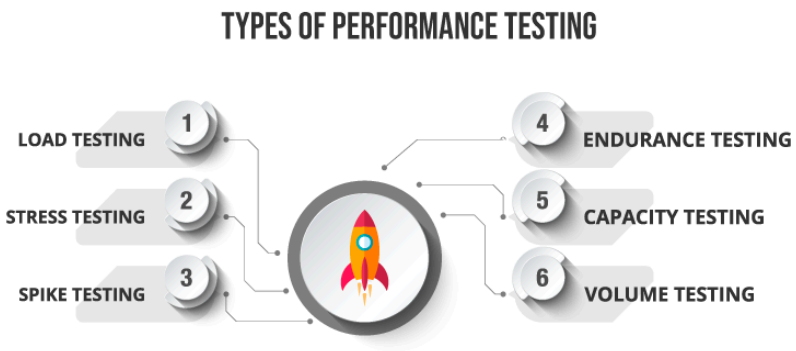


Infrastructure Testing

- Infrastructure testing is **the testing process that covers hardware, software, and networks, security etc**. It involves testing of any code that reads configuration values from different things in the IT framework and compares them to intended results. It reduces the risks of failure.

Performance Testing

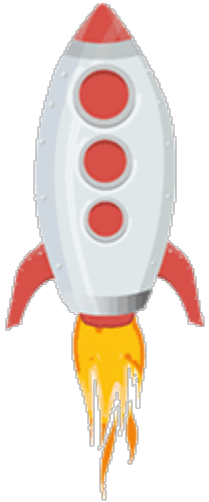
- **Performance testing** is a non-functional software testing technique that determines how the **stability, speed, scalability**, and **responsiveness** of an application holds up under a given workload.



Security Testing

- **Security Testing** is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders.
- Below are the six **basic principles of security testing**:
  - Confidentiality
  - Integrity
  - Authentication
  - Authorization
  - Availability
  - Non-repudiation

Product Demo



Key Takeaways

- Fundamentals of SDLC process and Documentation
- Version control systems
- Devops Fundamentals
- CI/CD pipelines
- Testing

Quiz

1. In waterfall model output of one phase is input to next phase.

- ❑ True
- ❑ False

Answer: It is True

Quiz

2. Selection of particular life-cycle model is based on ?

A. Requirements
B. Technical Knowledge of Development team
C. Users
D. Project Types & Associated Risks
E. All of the above

Answer: E. All of the above . Because it depends on all of this aspects .

Quiz

3. Which of the following is a characteristic of Agile Development?

A. Repeated Customer Collaboration
B. Focus on Business Goals
C. Rapid Application Development
D. Ans 1 & 2
E. Ans 1 & 3
F. Ans 2 & 3
G. All of the above

Answer: D

Quiz

4. Your new code ends with a broken build. How should you handle this?

A. Revert to a previous version of the code and fix it in your own time.

B. Commit the code anyway because the automatic testing will find the problem.

C. Try to recommit the code to determine whether the issue is in the version control.

D. None

Answer: **A**

It is a best practice that developers take responsibility for their own build and do not leave a build broken.

Quiz

5. Why does continuous delivery offer a team more time to be productive?

A. Teams spend less time doing rework or fixing problems.
B. It allows teams to implement one feature at a time to ensure each works.
C. Much of the process is automated, so teams have more time for personal projects.
D. None

Answer: **A**
Determining bugs and fixing code is taken care at the DevOps end.

Quiz

6. For the purposes of CI/CD, what are the primary benefits of Unit Tests?

A. speed of test execution and confidence to ship faster
B. business stakeholders can write them and run the tests themselves
C. testing all the edge cases and ability to do build matrixes
D. testing all the backend services needed and using real traffic

Answer: **A**
Yep, unit tests are fast to run and increase confidence of your team

Quiz

7. Why is integration testing needed even if we have good unit testing in place?

A. to see how our application is built
B. to test external dependencies to our application
C. to add chaos into our system
D. understanding how cosmic rays impact our system

Answer: **B**
Yep, we use integration testing to exercise our dependencies.

Thank you!

DevelopIntelligence
A PLURALSIGHT COMPANY