BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

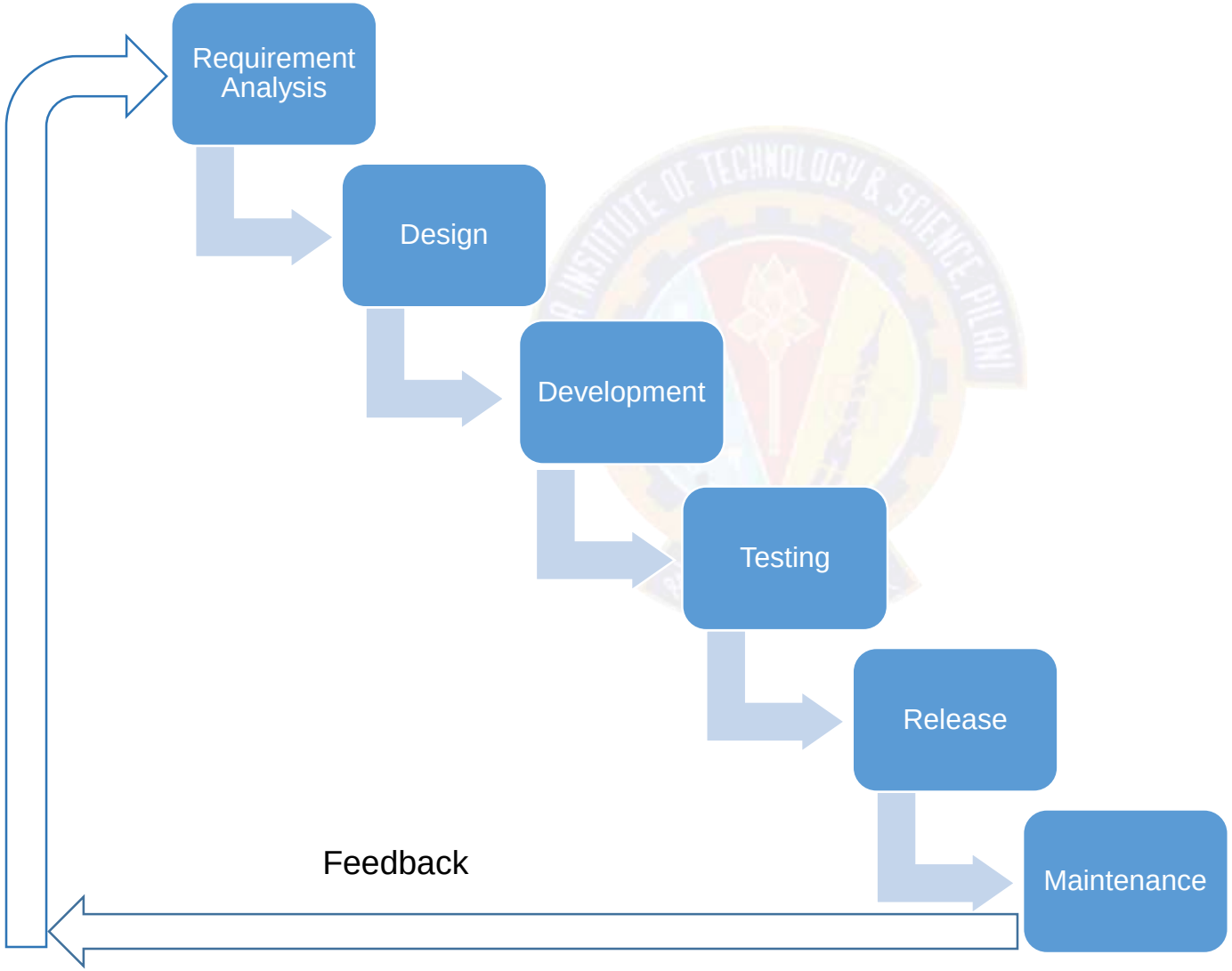
Agenda

Agile Methodology

- Software Development Life Cycle
- Waterfall Model & its Amendment
- Need of Agile
- Agile Methodology
- Principles of Agile Methodology
- Pillars of Agile Methodology
- Agile Methodologies
- Roles in Agile

Software Development Life Cycle

SDLC Phases



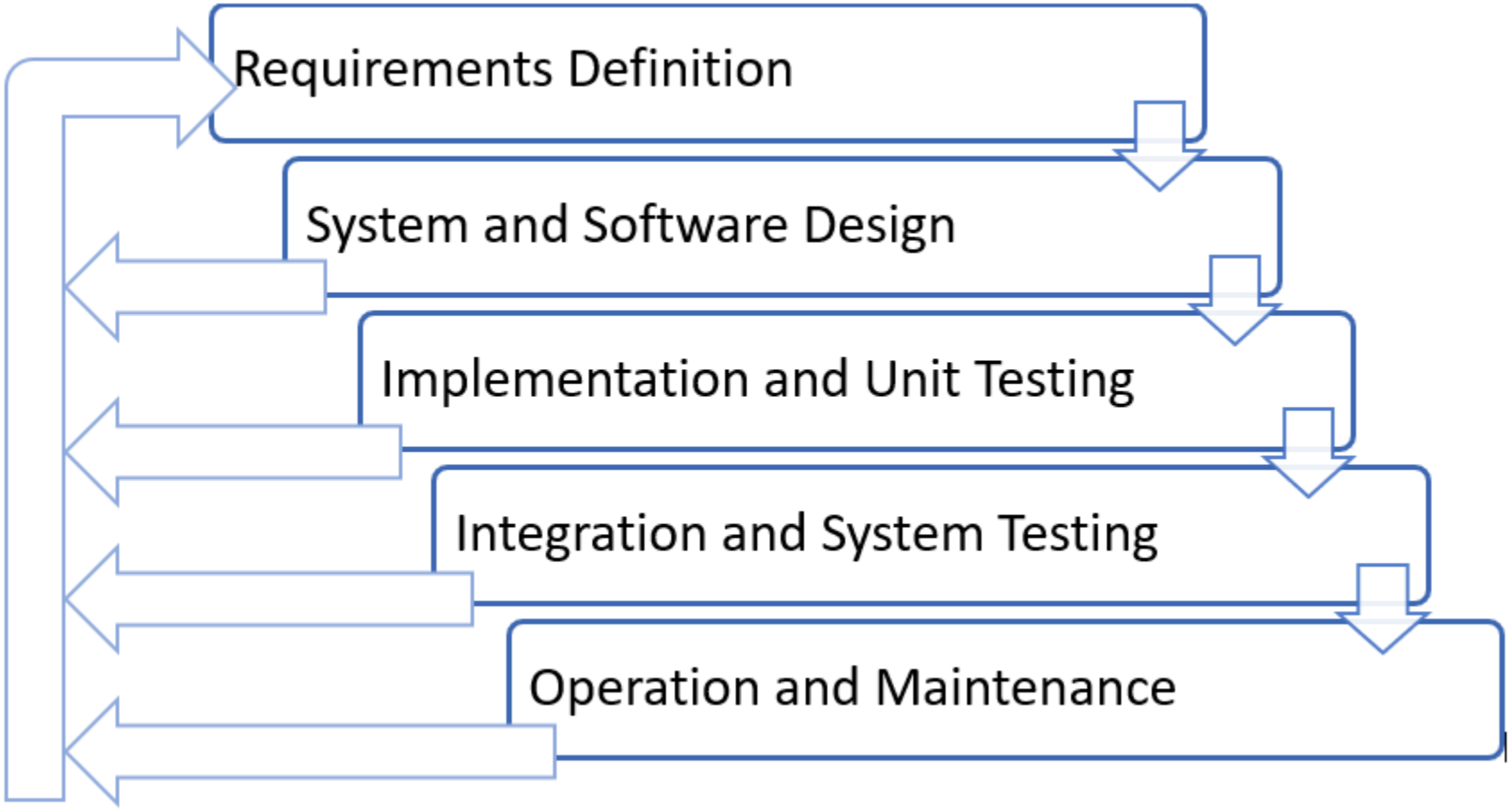
Waterfall Model

Intorduction

- It is also called as the classic life cycle
- Waterfall Model is considered as Black Box Model
- One can not go back to previous phase
- Sequential in Nature
- This suggests a systematic, sequential approach to software development that begins with customer specification of requirements
- And progresses through
  - Planning
  - Modeling
  - Construction
  - Deployment
  - and Culminating in ongoing support of the completed software

Waterfall Model Contd..

Feedback Amendment in Waterfall Model



Waterfall Model Contd..

Advantages

- Easy to use and follow
- Cost effective
- Each phase completely developed
- Development processed in sequential manner, so very less chance of rework
- Easy to manage the project
- Easy documentation

Waterfall Model Contd..

Waterfall Model Problems

- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway
- In principle, a phase has to be complete before moving onto the next phase
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process
  - Few business systems have stable requirements

Need of Agile

Why Agile?

- The project will produce the wrong product
- The project will produce a product of inferior quality
- The project will be late
- We'll have to work 80 hour weeks
- We'll have to break commitments
- We won't be having fun

- Storm called Agile
- According to VersionOne's State of Agile Report in 2017 says 94% of organizations practice Agile, and in 2018 it reported 97% organizations practice agile development methods

Agile Methodology Contd..

Agile – Background

- Agile was formally launched in 2001
- 17 technologists drafted the Agile Manifesto

Kent Beck	Mike Beedle	Arie van Bennekum	Alistair Cockburn	Ward Cunningham	Martin Fowler
James Grenning	Jim Highsmith	Andrew Hunt	Ron Jeffries	Jon Kern	Brian Marick
Robert C. Martin	Steve Mellor	Ken Schwaber	Jeff Sutherland	Dave Thomas	

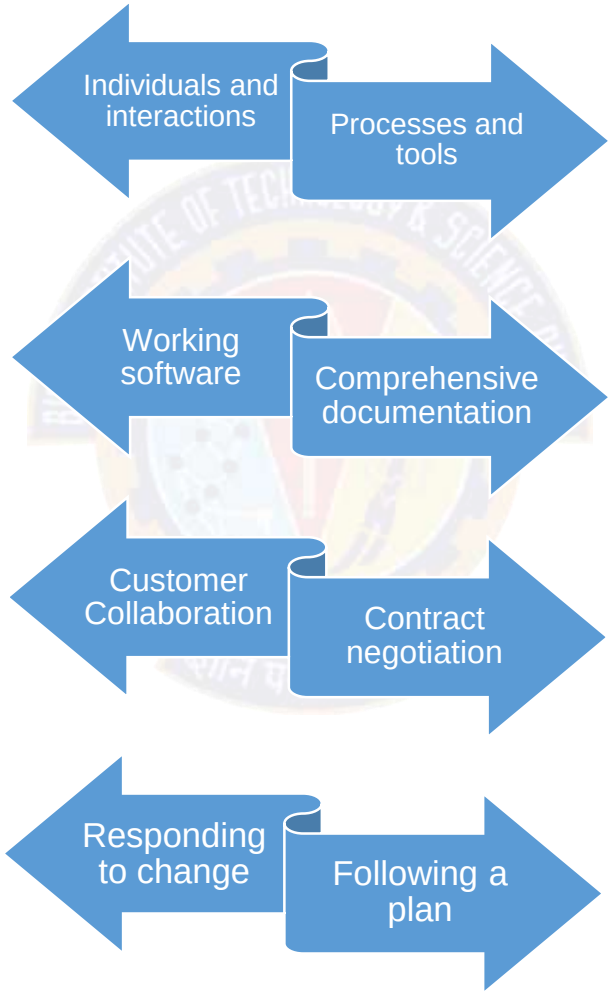
Principles of Agile Methodology

Twelve Principles

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
4. Business people and developers must work together daily throughout the project
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
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
10. Simplicity--the art of maximizing the amount of work not done--is essential
11. The best architectures, requirements, and designs emerge from self-organizing teams
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Pillars of Agile Methodology

Agile focuses on



Agile Methodologies

Few of Agile Methodologies

- Scrum
- Extreme Programming [XP]
- Test driven Development [TDD]
- Feature Driven Development [FDD]
- Behavior-driven development [BDD]

Roles in Agile

Basic roles involved

- User
- Product Owner
- Software Development Team

Reference

- https://explore.versionone.com/state-of-agile/

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Operational Methodologies: ITIL

- Introduction to ITIL
- IT Service Management (ITSM) Lifecycle
- ITIL Framework

Operational Methodology

ITIL

- ITIL is a framework of best practices for delivering IT services
- The ITIL processes within IT Service Management (ITSM) ensure that IT Services are provided in a focused, client-friendly and cost-optimized manner
- ITIL's systematic approach to IT service management can help businesses
- Manage risk
- Strengthen customer relations
- Establish cost-effective practices
- And build a stable IT environment
- that allows for
  - Growth
  - Scale and
  - Change

ITIL

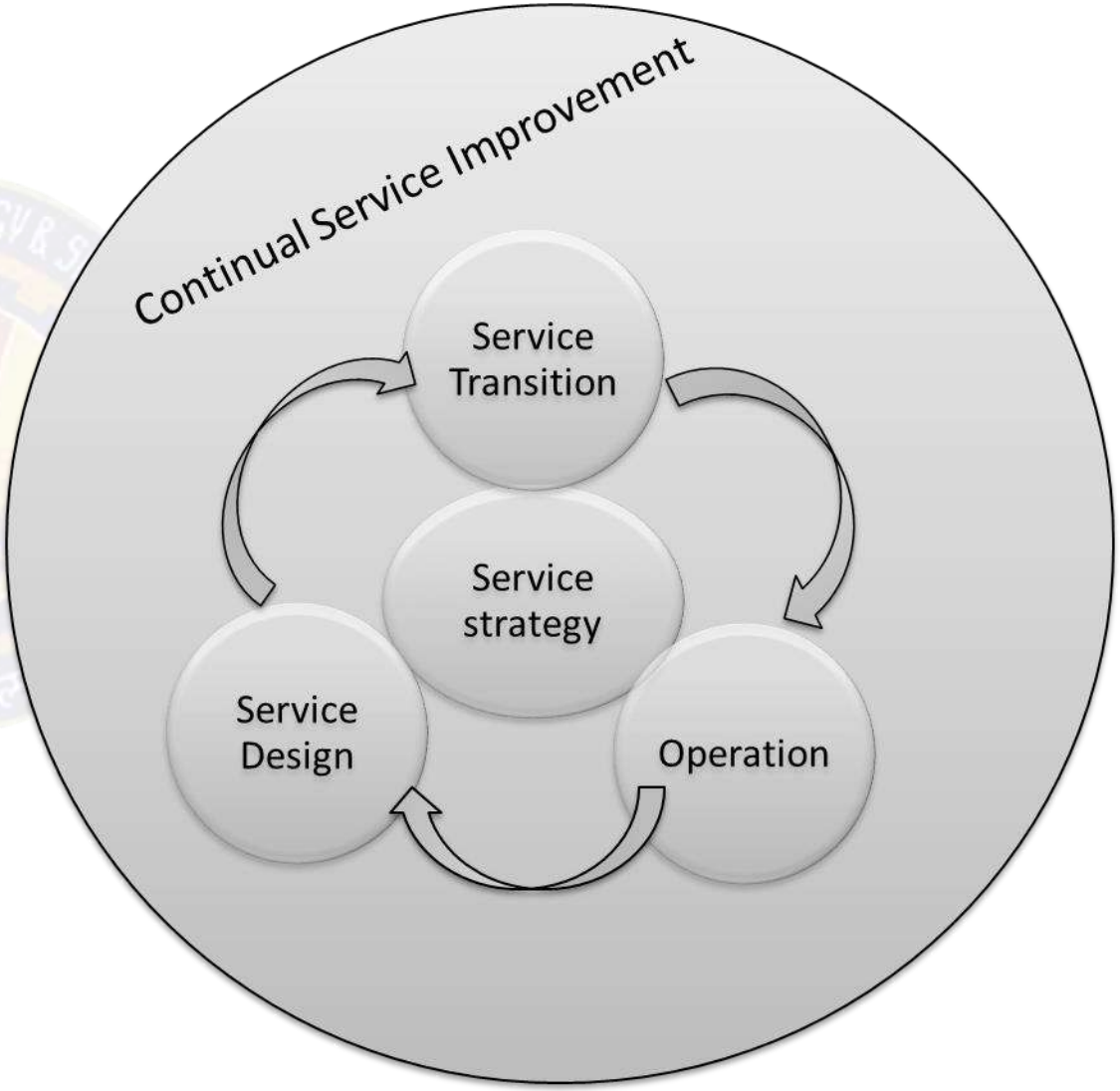
ITIL : Origin

- Developed by the British government's Central Computer and Telecommunications Agency (CCTA) during the 1980s
- The ITIL first consisted of more than 30 books, developed and released over time, that codified best practices in information technology accumulated from many sources (including vendors' best practices) around the world
- ITIL’s contribution aligned with the ISO/IEC 20000 Service Management standard in 2005
- Since 2013, ITIL is owned by Axelos — a joint venture between the Cabinet Office and Capita

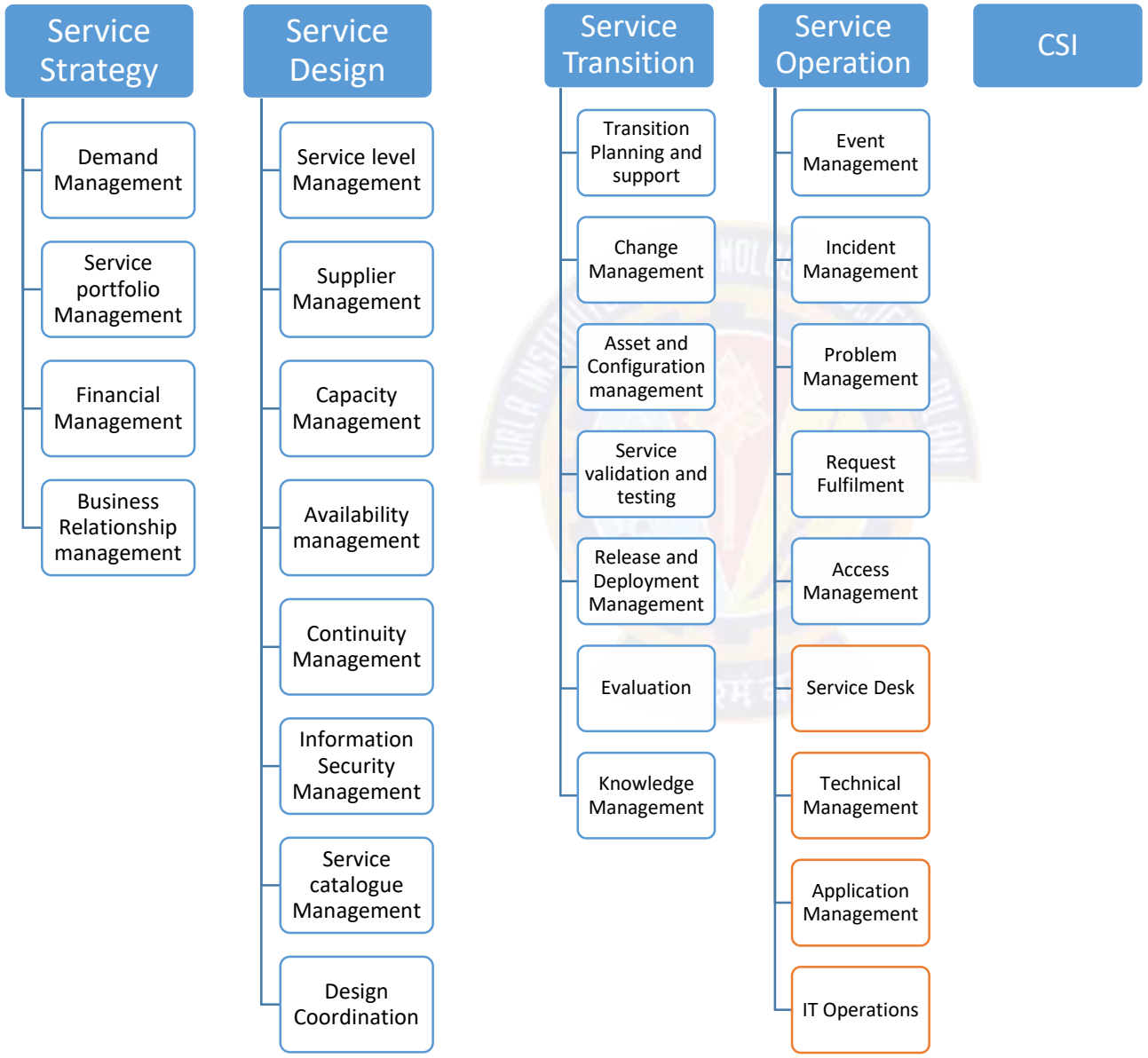
ITIL

IT Service Management (ITSM) Lifecycle

- ITIL views ITSM as a lifecycle
- Five Phases:
  - Service Strategy [design, development, and implementation]
  - Service Design [design and development]
  - Service Transition [development and design]
  - Service Operation [delivery and support]
  - Service Improvement [create and maintain value]



ITIL Framework



ITIL

ITIL: Is a Project?

- ITIL is not a “Project”
- ITIL is ongoing journey
- Benefits of ITIL:
  - Organizations to reduce the operations cost by optimizing the available resource utilizations
  - Reduce system outages and minimize the impacts of service unavailability
  - Even most organizations testified with reduction in business critical incidents by 80 to 90% in few years

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Problems of Delivering Software

- Problems of Delivering Software
- Release Antipatterns
- Principles of Software Delivery

Problems of Delivering Software

- Converting Idea to Product / Service?
- Reliable, rapid, low-risk software releases
- Ideal Environment
- Generic Methodologies for Software Methodologies
- More focus on Requirement Gathering

Problems of Delivering Software

Release Antipatterns

- Common Quote “The day of a software release tends to be a tense one”
- Antipattern: Deploying Software Manually
  - The production of extensive and detailed documentation for steps to be performed
  - Reliance on manual testing
  - Frequent calls to the development team to explain
  - Frequent corrections to the release process
  - Sitting bleary-eyed in front of a monitor at 2 A.M

Problems of Delivering Software

Release Antipatterns Cond..

- Antipattern: Deploying to a Production-like Environment Only after Development Is Complete
  - Tester tested the system on development machines
  - Releasing into staging is the first time that operations people interact with the new release
  - Who Assembles? The Development Team
  - Collaboration between development and Operations?
- Antipattern: Manual Configuration Management of Production Environments
  - Difference in Deployment to Stage and Production
  - Different host behave differently
  - Long time to prepare an environment
  - Cannot step back to an earlier configuration of a system
  - Modification to Configuration Directly

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Continuous Delivery

- Introduction to Continuous Delivery
- Principle of Continuous Delivery

Continuous Delivery

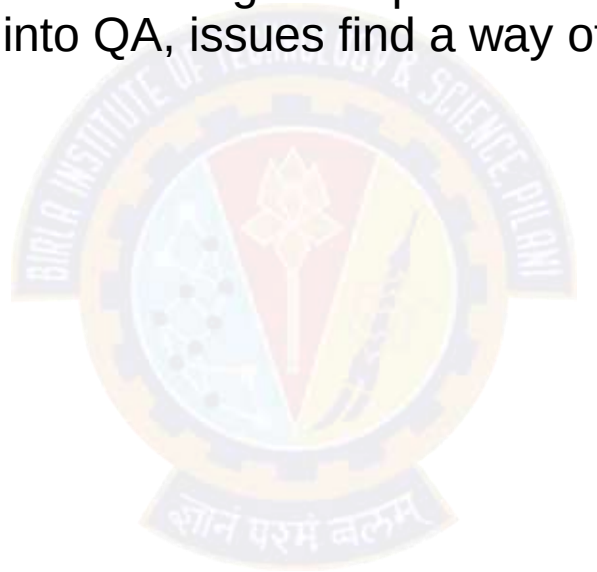
Introduction to Continuous Delivery

- Promotes the adoption of an automated deployment pipeline to release software into production reliably and quickly
- Its goal is to establish an optimized end-to-end process, enhance the development to production cycles, lower the risk of release problems and provide a quicker time to market
- To that point, Jez Humble and Dave Farley define the 8 principles of continuous delivery in their Book “Continuous Delivery” as:

Principle of Continuous Delivery

Build Repeatable and Reliable Process

- Use the same release process in all environments
- If a feature or enhancement has to work through one process on its way into the integration environment, and another process into QA, issues find a way of popping up



Principle of Continuous Delivery

Try to Automate Everything

- Automate builds, testing, releases, configuration changes and everything else
- Manual processes are inherently less repeatable, more prone to error and less efficient
- Once we automate a process, less effort is needed to run it and monitor its progress – and it will ensure to have consistent results

Principle of Continuous Delivery

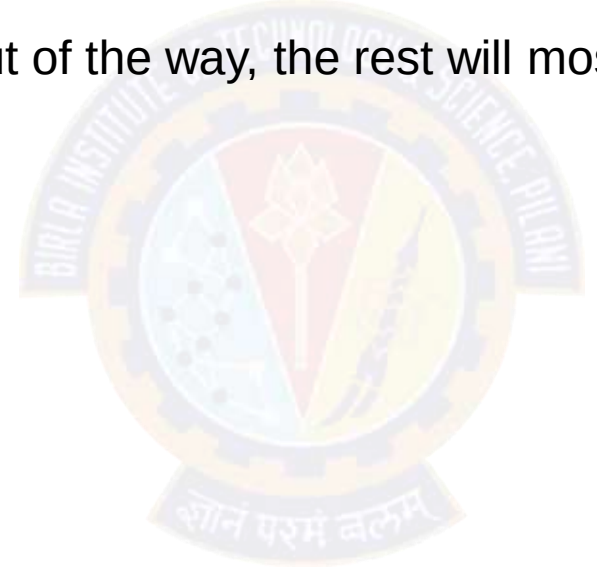
Keep everything in Version Control

- Have one source of truth
- Keep Code, configuration, scripts, databases, documentation and everything versioned

Principle of Continuous Delivery

Bring the Pain Forward

- Deal with the hard stuff first
- Time-consuming or error prone tasks should be dealt with as soon as you can
- Once you get the painful issues out of the way, the rest will most likely be easier to perfect



Principle of Continuous Delivery

Ensure Quality – in Build

- Create short feedback loops to deal with bugs as soon as they are created
- By having issues looped back to developers as soon as they fail post-build test, it will enable them to produce higher quality code quicker
- In addition, fewer issues will be found later on in the process, when it will be more expensive to fix

Principle of Continuous Delivery

Set expectation as "Done" means Released

- A feature is done only when it is in production
- Having a clear definition of “done” right from the start will help everyone communicate better, and realize the value in each feature

Principle of Continuous Delivery

Everyone is Responsible

- "It works on my workstation" is never a valid excuse
- Responsibility should extend all the way to production
- Cultural change can be the hardest to implement
- However, having management support and an enthusiastic champion will certainly help

Principle of Continuous Delivery

Continuous Improvement

- This principle is the most important for effective automation
- If "practice makes perfect" then automation represents the ultimate expression of a practice-making-perfect iterative process.

Principle of Continuous Delivery

Summary

- Create a Repeatable, Reliable Process for Releasing Software
- Automate Almost Everything
- Keep Everything in Version Control
- If It Hurts, Do It More Frequently, and Bring the Pain Forward
- Build Quality In
- “The Earlier you catch the defects, the cheaper they are to fix”
- Done, Means Released
- Everybody Is Responsible for the Delivery Process
- Continuous Improvement

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

DevOps Basics

- Introduction to DevOps
- Need for DevOps
- Why DevOps

DevOps Basics

Introduction

- A clipped compound of Development and Operations
- Buzzword
- Innovation for automation (pillar of DevOps)
- DevOps is an approach based on lean and agile principles
- There are more tangential DevOps Initiative
  - WinOps
  - DevSecOps
  - BizDevOps

DevOps Basics

Introduction Contd..

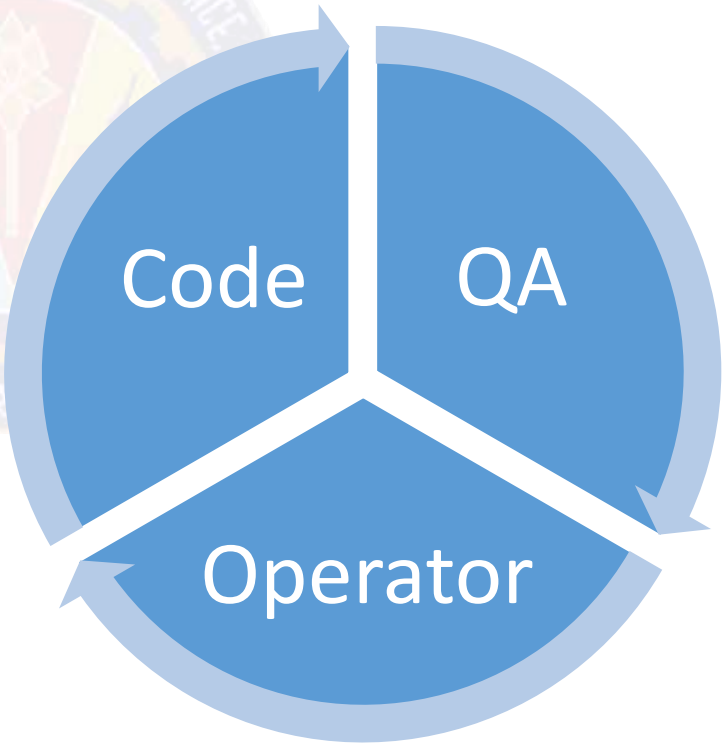
- Definition of DevOps:
  - “DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality”

- Implications of this definition
  - Practices and tools
  - Do not restricted scope of DevOps to testing and development

DevOps Basics

Need for DevOps

- Developer :- Writing Code
- Quality Assurance :- Testing
- Operations personnel :- Deploy



DevOps Basics

Need for DevOps Contd..

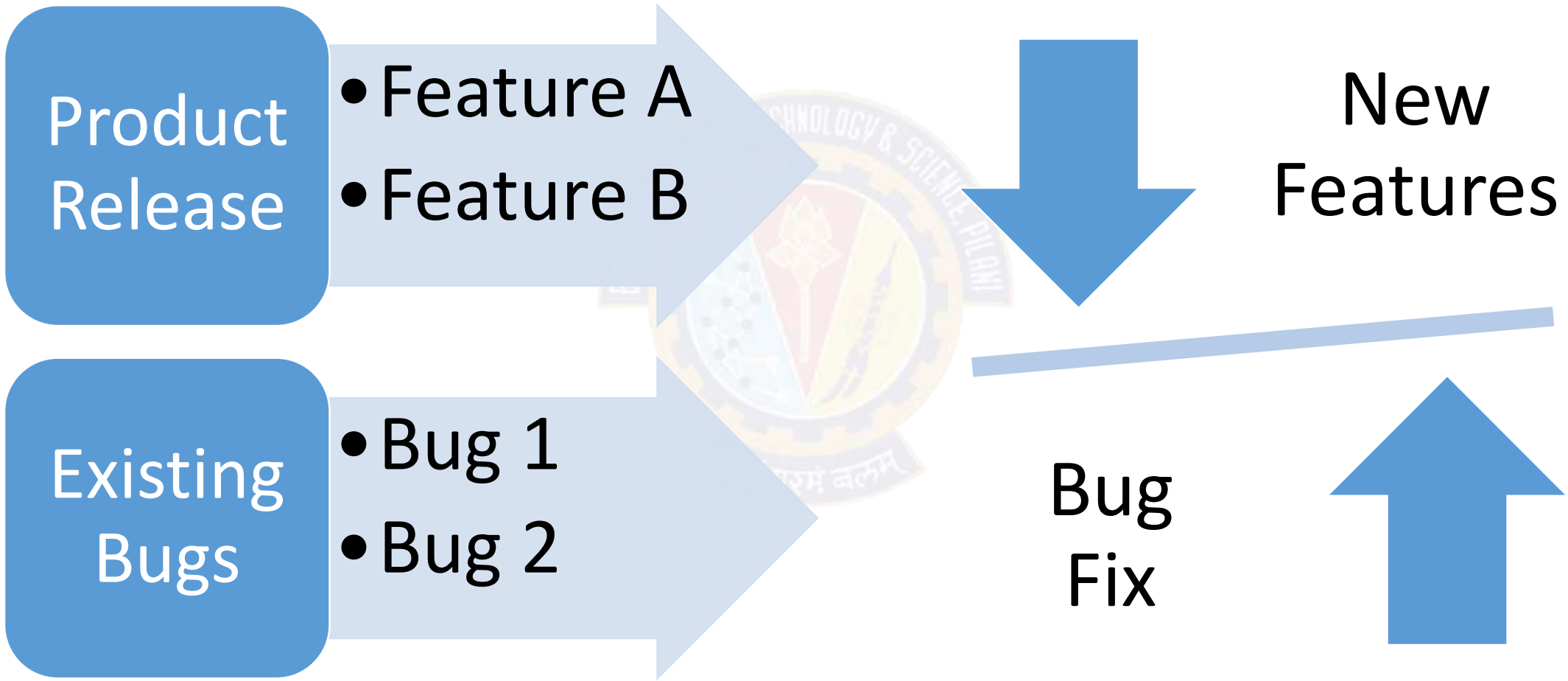
BUG

FIX

When

DevOps Basics

Need for DevOps Contd..



DevOps Basics

Need for DevOps Contd..

https://in.pinterest.com/

DevOps Basics

Need for DevOps Contd..

- Waterfall Model
- Dominated Project Management Until 2001

- Still in Use :
  - Disciplined Approach
  - Thorough Scoping
  - Clear Stages
  - Concrete Deadlines

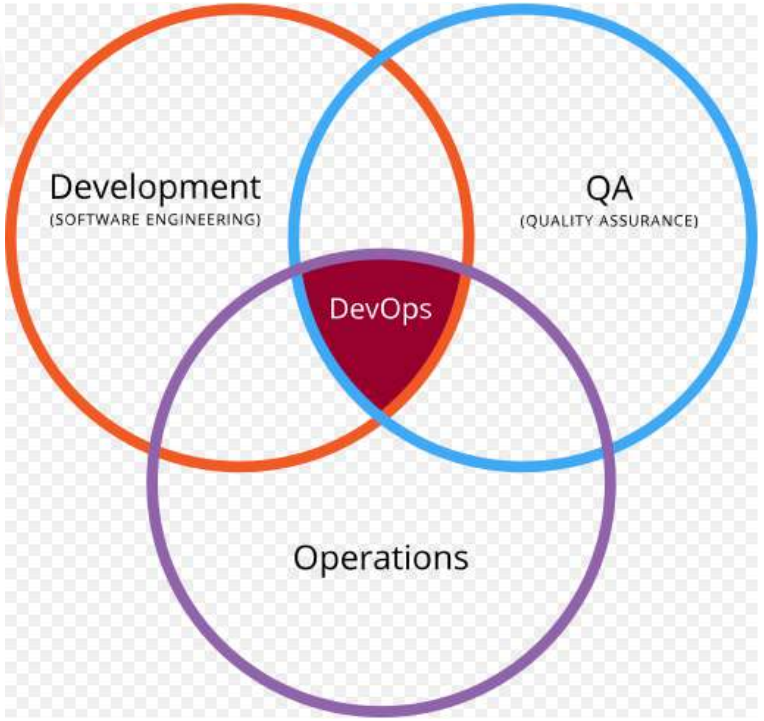
- Disadvantages:
  - Sequential
  - Inflexible
  - Assumptions

We are loosing Time to market

DevOps Basics

Where is Operations?

- Development is All Well
- Where is Operations?
- Where is System Administrators?



DevOps Basics

Why DevOps? :: Lets Summaries

- Release Process
  - Releasing a new system or version of an existing system to customers is one of the most sensitive steps in the software development cycle
  - The following release planning steps are adapted from Wikipedia
    - Define and agree on release and deployment plans with customers/stakeholders
    - Ensure that each release package consists of a set of related assets and service components that are compatible with each other
    - Ensure that the integrity of a release package and its constituent components is maintained throughout the transition activities and recorded accurately in the configuration management system
    - Ensure that all release and deployment packages can be tracked
- Poor Coordination
- Limited Capacity of Operations Staff

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

DevOps Practices

- Five categories of DevOps Practice
- Common terminology

DevOps Practices

Five different categories of DevOps practices

- Treat Ops as first-class citizens from the point of view of requirements
- Make Dev more responsible for relevant incident handling
- Enforce the deployment process used by all, including Dev and Ops personnel
- Use continuous deployment
- Develop infrastructure code, such as deployment scripts

DevOps Practices

DevOps life cycle processes by Porter's value chain

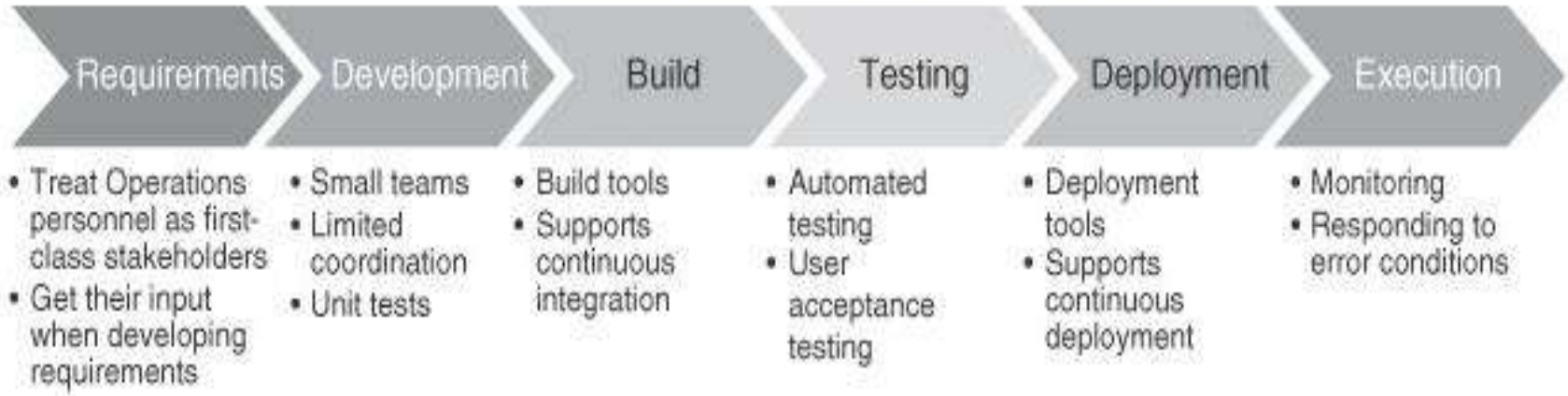


FIGURE 1.1 DevOps life cycle processes [Notation: Porter's Value Chain]

DevOps Practices

Common terminology used

- IT professional
  - help desk support
- operator, and operations personnel
  - hands-on access to the hardware—installing and configuring hardware, managing backups, and maintaining printers
- System administrator
  - responsible for uptime, performance, resources, and security of computer systems

- Note :We will use term operator though out the course refer to anyone who performs computer operator or system administration tasks (or both)

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

DevOps Misconceptions and Dimensions

- DevOps Misconceptions
- DevOps Antipatterns
- Three Dimensions of DevOps

DevOps Misconceptions

DevOps Only Involves Developers and System Administrators

- As the name indicates Development and Operations
- **However the Truth is:**
- DevOps tagline is bring development and operations together
- The concepts and ideas of DevOps include all roles within an organization
- There is no definitive list of which teams or individuals should be involved
- Any team within the organization should be considered as a candidate for DevOps, it may include security, QA, support, and legal

DevOps Misconceptions Contd..

DevOps is a Team

- Creating a team called DevOps, OR
- Renaming an existing team to DevOps is neither necessary nor sufficient for creating a DevOps culture
- **However truth is:**
- An additional team is likely to cause more communication issues, not fewer
- the development and operations teams must connect frequently or it should work together
- In a startup environment, having a single team that encompasses both functions can work

DevOps Misconceptions Contd..

DevOps is a Job Title

- The “DevOps engineer” job title has started a controversial debate
- The job title has been described in various ways
- A system administrator who also knows how to write code
- A developer who knows the basics of system administration
- **However truth is:**
- In DevOps it makes sense to have people become more specialized in their job role
- DevOps is at its core a cultural movement, and its ideas and principles need to be used throughout entire organizations in order to be effective

DevOps Misconceptions Contd..

You Need a DevOps Certification

- Certification exams are testing knowledge where there are clear right or wrong answers, which DevOps generally does not have
- **However truth is:**
- A significant part of DevOps is about culture: how do you certify culture?
- DevOps doesn’t have required technology or one-size-fits-all solutions

DevOps Misconceptions Contd..

DevOps Means Doing All the Work with Half the People

- Get both a software developer and a system administrator in one person and with one person's salary
- **However truth is:**
- Not only is this above perception incorrect, but it is often harmful
- At a time startups are offering perks such as three meals a day in the office and on-site laundry as a way of encouraging
- DevOps doesn't save money by halving the number of engineers your company needs
- Rather, it allows organizations to increase the quality and efficiency of their work

DevOps Misconceptions Contd..

DevOps Is About Automation

- Many innovations in DevOps-adjacent tools help to codify understanding, bridging the gaps between teams and increasing velocity through automation
- Practitioners have focused on tools that eliminate tasks that are boring and repetitive, such as with infrastructure automation and continuous integration
- **However truth is:**
- In both of these above cases, automation is a result of improved technology
- Automating repetitive tasks automation will help to human from having to do them, that automation helps that person work more efficiently
- Example: Automating server builds saves hours per server that a system administrator can then spend on more interesting or challenging work

DevOps Misconceptions Contd..

DevOps Is a FAD

- It’s a Buzzword in Market
- Just a new word in place
- **However truth is:**
- As DevOps is not specific to a technology, tool, or process however, It is movement about improving organizational effectiveness and individual employee happiness
- One of the primary differences between DevOps and methodologies like ITIL and Agile is:
- It does not have strict definition
- DevOps is movement defined by stories and ideas of individuals, teams, and organizations
- DevOps is the evolution of processes and ideas that lead growth and change
- Harnessing and leveraging effective DevOps will lead to growth and evolution in tools, technology, and processes in your organization

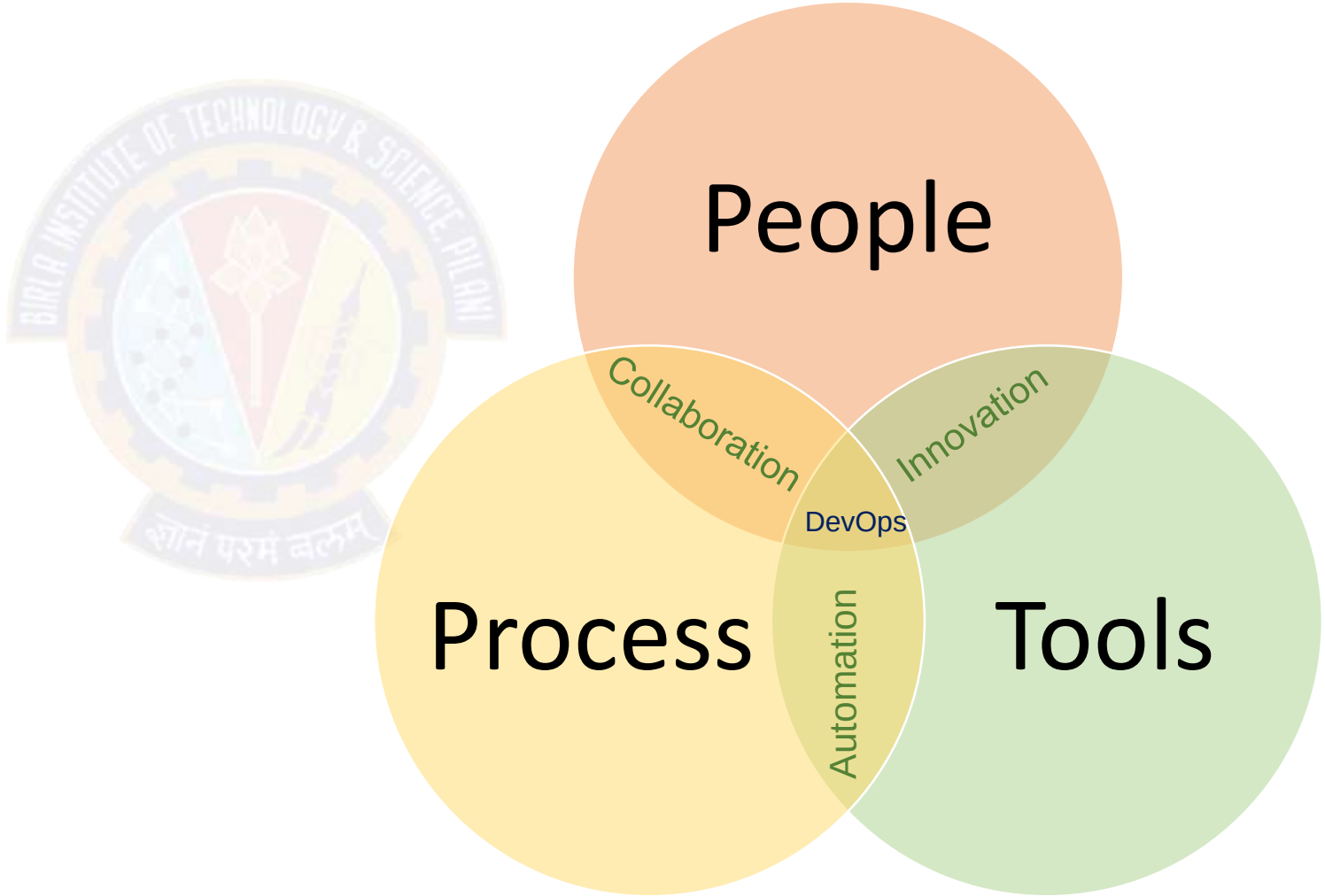
DevOps Anti-Patterns

- Blame Culture
  - A blame culture is one that tends toward blaming and punishing people when mistakes are made, either at an individual or an organizational level
- Silos
  - A departmental or organizational silo describes the mentality of teams that do not share their knowledge with other teams in the same company
- Root Cause Analysis
  - Root cause analysis (RCA) is a method to identify contributing and “root” causes of events or near-misses/close calls and the appropriate actions to prevent recurrence
- Human Errors
  - Human error, the idea that a human being made a mistake that directly caused a failure, is often cited as the root cause in a root cause analysis

Three dimensions of DevOps

3-D of DevOps

- People
- Process
- Tools / Technology



Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Agile Methodology SCRUM

- Scrum

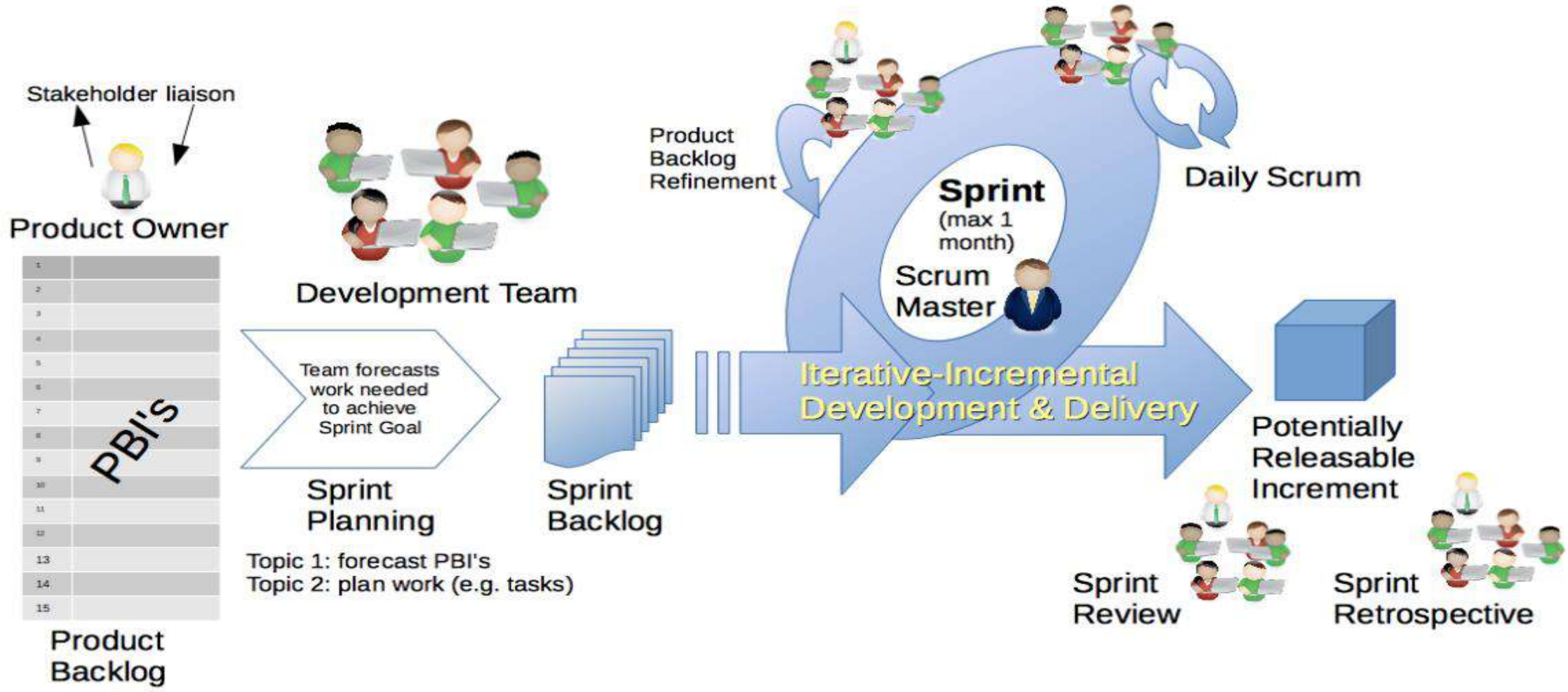
SCRUM Methodology

About Scrum

- Scrum is part of the Agile movement
- Scrum is an agile way to manage a project, usually software development
- Scrum's early advocates were inspired by empirical inspect and adapt feedback loops to cope with complexity and risk
- Scrum emphasizes decision making from real-world results rather than speculation
- By Mountain Goat Software – Pioneer of Agile
- "Agile software development with Scrum is often perceived as a methodology; but rather than viewing Scrum as methodology, think of it as a framework for managing a process"

SCRUM

Scrum Sprints



SCRUM

Benefits of Scrum

- Flexibility
- Quality
- Incremental Working Product
- Early to Market

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

DevOps and Agile

- Relationship of DevOps practices to agile practices
- DevOps Practices over Agile
- Traditional, Agile and DevOps – A comparison

DevOps and Agile

Relationship of DevOps practices to agile practices

- One of the characterizations of DevOps emphasizes the relationship of DevOps practices to agile practices
- We will focus on what is added by DevOps
- We interpret transition as deployment

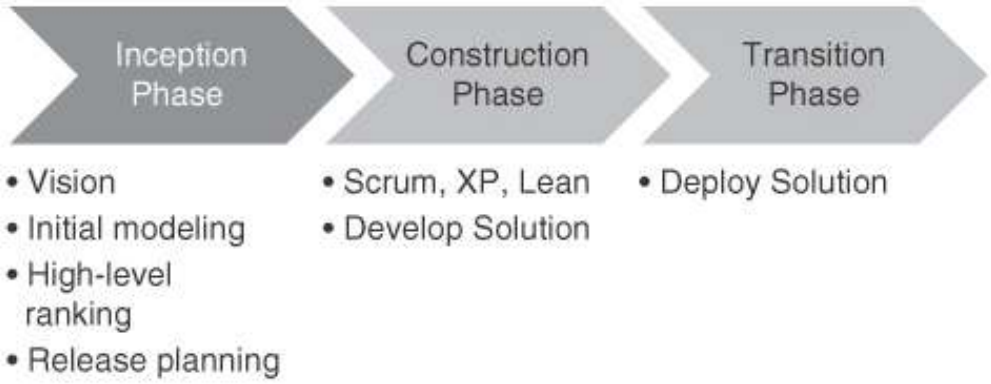


FIGURE 1.2 Disciplined Agile Delivery phases for each release. (Adapted from *Disciplined Agile Delivery: A Practitioner's Guide* by Ambler and Lines) [Notation: Porter's Value Chain]

DevOps and Agile Contd..

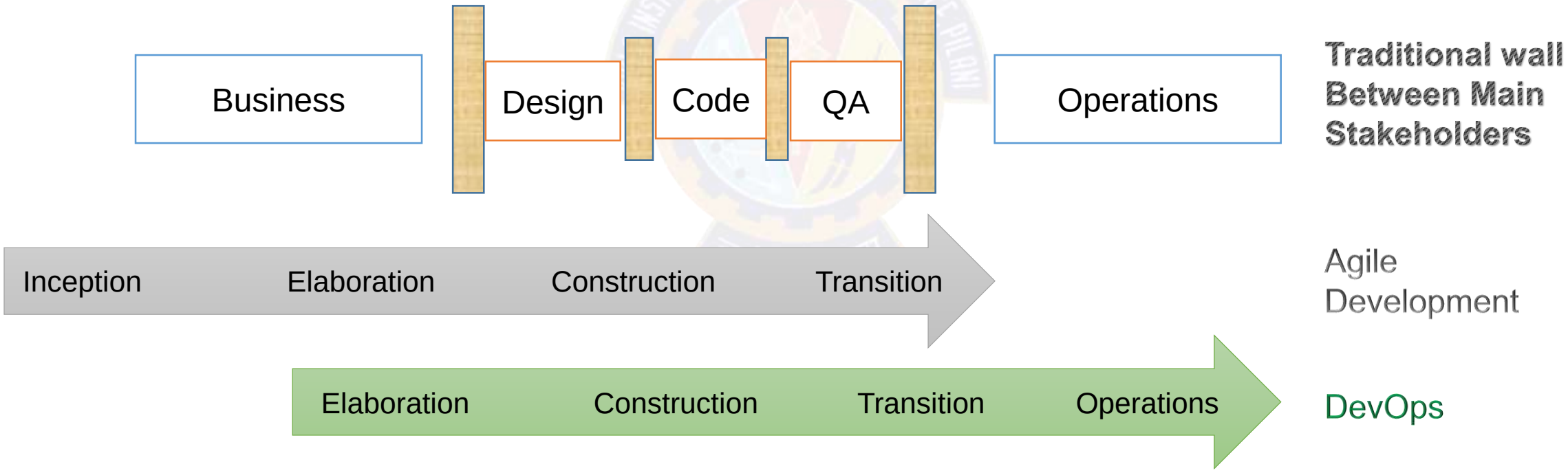
DevOps Practices over Agile

- DevOps practices impact all three phases
- Inception phase : During the inception phase, release planning and initial requirements specification are done
  - Considerations of Ops will add some requirements for the developers
  - Release planning includes feature prioritization but it also includes coordination with operations personnel
- Construction phase: During the construction phase, key elements of the DevOps practices are the management of the code branches,
  - the use of continuous integration
  - continuous deployment
  - incorporation of test cases for automated testing
- Transition phase: In the transition phase, the solution is deployed and the development team is responsible for the deployment, monitoring the process of the deployment, deciding whether to roll back and when, and monitoring the execution after deployment

DevOps and Agile Contd..

Traditional, Agile and DevOps – A comparison

- Agile breaks the wall between Business and Development team
- DevOps breaks the wall between Development and Operations team
- DevOps centers on the concept of sharing: sharing ideas, issues, processes, tools and goals



Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Agile methodology for DevOps Effectiveness

- Why DevOps Process?
- DevOps Process: Flow based vs Non flow based
- DevOps Process: Component team vs Feature Team

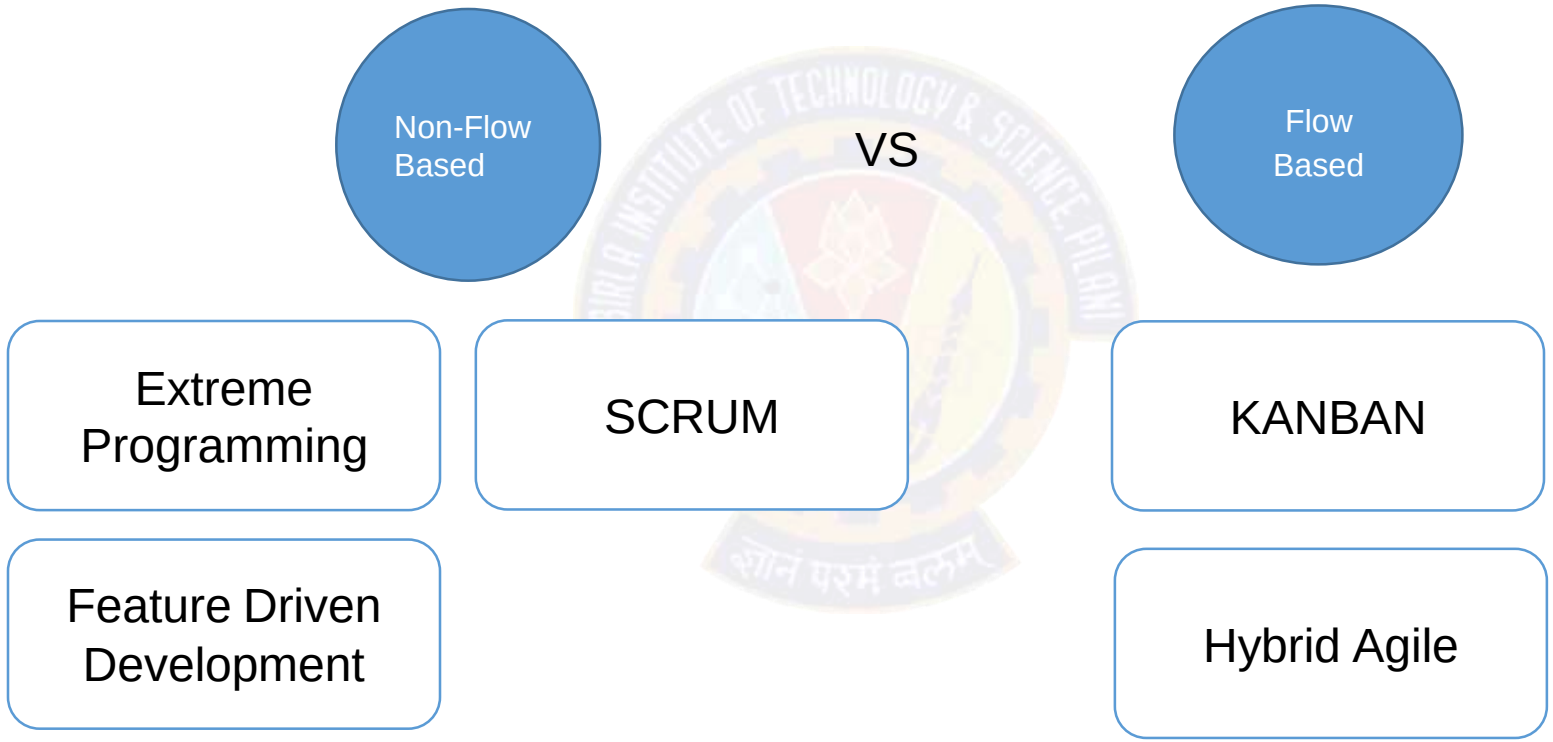
DevOps Process

Why DevOps Process?

- We need processes and policies for doing things in a proper way and standardized across the projects so the sequence of operations, constraints, rules and so on are well defined to measure success
- We need to set processes for the following things:
  - Agile planning
  - Resource planning and provisioning
  - Configuration management
  - Role-based access control to cloud resources and other tools used in automation
  - Static code analysis – rules for programming languages
  - Testing methodology and tools
  - Release management

Agile methodology for DevOps Effectiveness

Flow based and Non flow based



DevOps Process

Selecting a Teams Structure

- Choosing appropriate team structure (Resource planning)

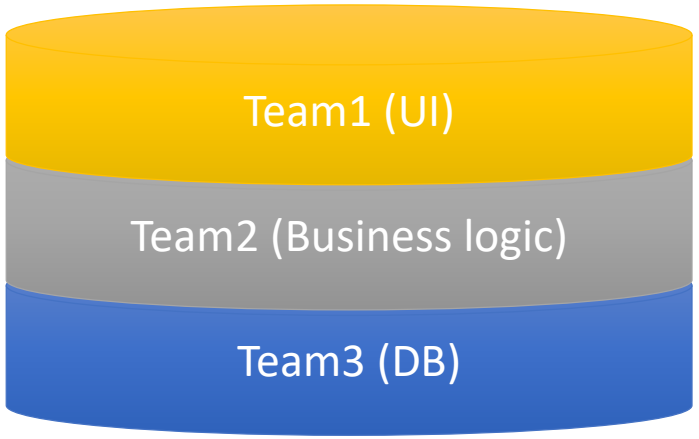
Component Team Structure

Featured Team Structure

DevOps Process

Component teams

- Made up of experts that specialize in a specific domain
- Architecture diagram can be represented as layer of the system
- Advantages:
  - work is always done by people specifically qualified to do the work
  - work is done efficiently with a low defect density

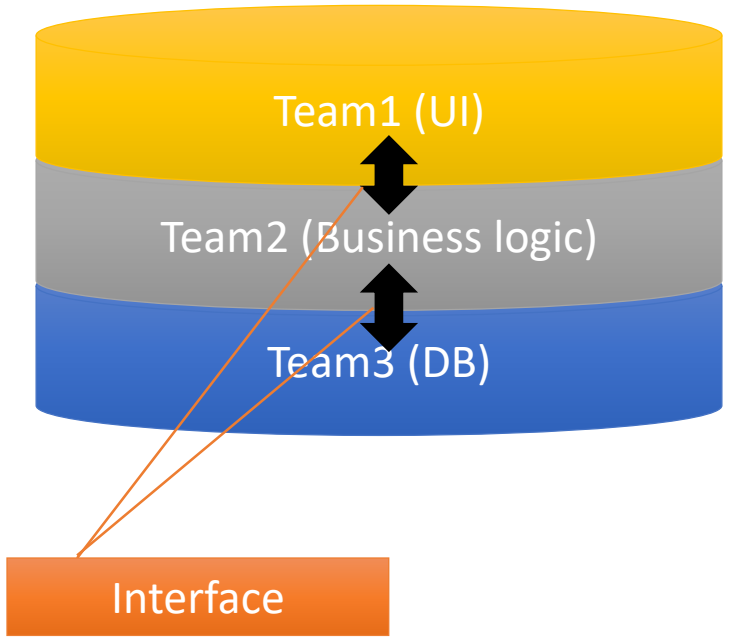


Architecture diagram

DevOps Process

Component teams Disadvantages

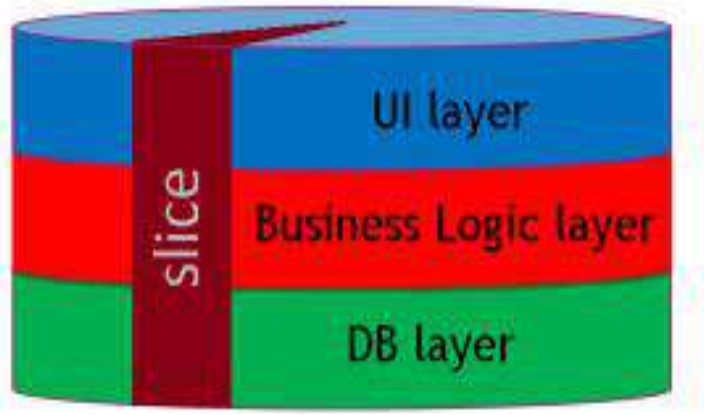
- Working as Component Teams increases the amount of time it takes to deliver
- working in Component Teams will have a negative effect on productivity that cannot be over emphasized
  - (Example: if business logic team starts on a piece of work, For them to complete the work, they need to define the interface between their layer and the other layers)
- Running Component Teams leaves us vulnerable to the risk that they will have to perform non-optimum work at the expense of items that are more valuable to the business



DevOps Process

Feature Teams

- Feature Teams contain multi-disciplined individuals that have the ability and freedom to work in any area of the system
- Feature Team usually has at least one member with specialist knowledge for each layer of the system
- This allows Feature Teams to work on what are known as ‘vertical slices’ of the architecture



DevOps Process

The characteristics of a feature team

- long-lived—the team stays together so that they can ‘jell’ for higher performance; they take on new features over time
- cross-functional and cross-component
- co-located
- work on a complete customer-centric feature, across all components and disciplines (analysis, programming, testing, …)
- composed of generalizing specialists
- in Scrum, typically 7 ± 2 people

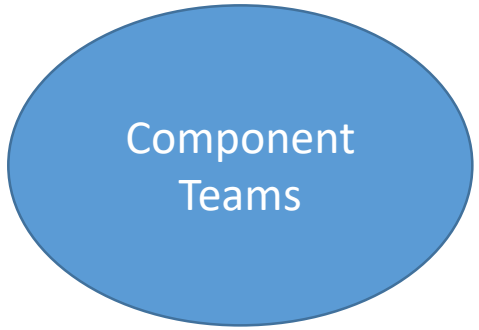
DevOps Process

Feature Team Disadvantages

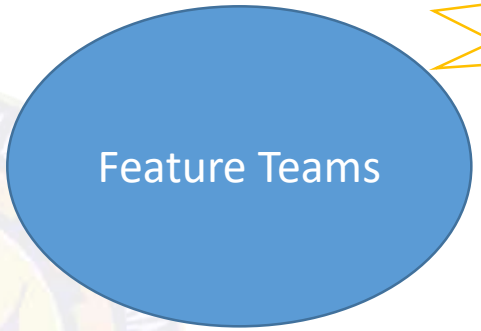
- What if untrained or inexperienced employee to deliver a poorly designed piece of work into a live environment
- Where do you get the personnel (‘generalizing specialists’)?

DevOps Process

DevOps – Process: Summary on How do you structure your teams



- Component Based structure
- Mostly aligned to technology cluster
- Faster component Development
- Slower feature realization
- Integration nightmare
- Suitable for platform teams



- Business Feature Based structure
- aligned to business domains
- Faster feature realization
- Scale-up problem due to need of cross-functional team
- Full-stack engineer
- Lower hand-off

References

- https://www.agil8.com/blog/component-teams-vs-feature-teams/

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Agile Methodologies

- Test Driven Development
- Feature Driven Development
- Behavior Driven Development

TDD

TDD Introduction

- Kent Beck said “Test-first code tends to be more cohesive and less coupled than code in which testing isn’t a part of the intimate coding cycle”

- “If you can’t write a test for what you are about to code, then you shouldn’t even be thinking about coding”

TDD

TDD Three steps

- RED
  - Write a new TEST which fails
- GREEN
  - Write simplest possible code to make it succeed
- REFACTOR
  - Refactor the code (including test code)

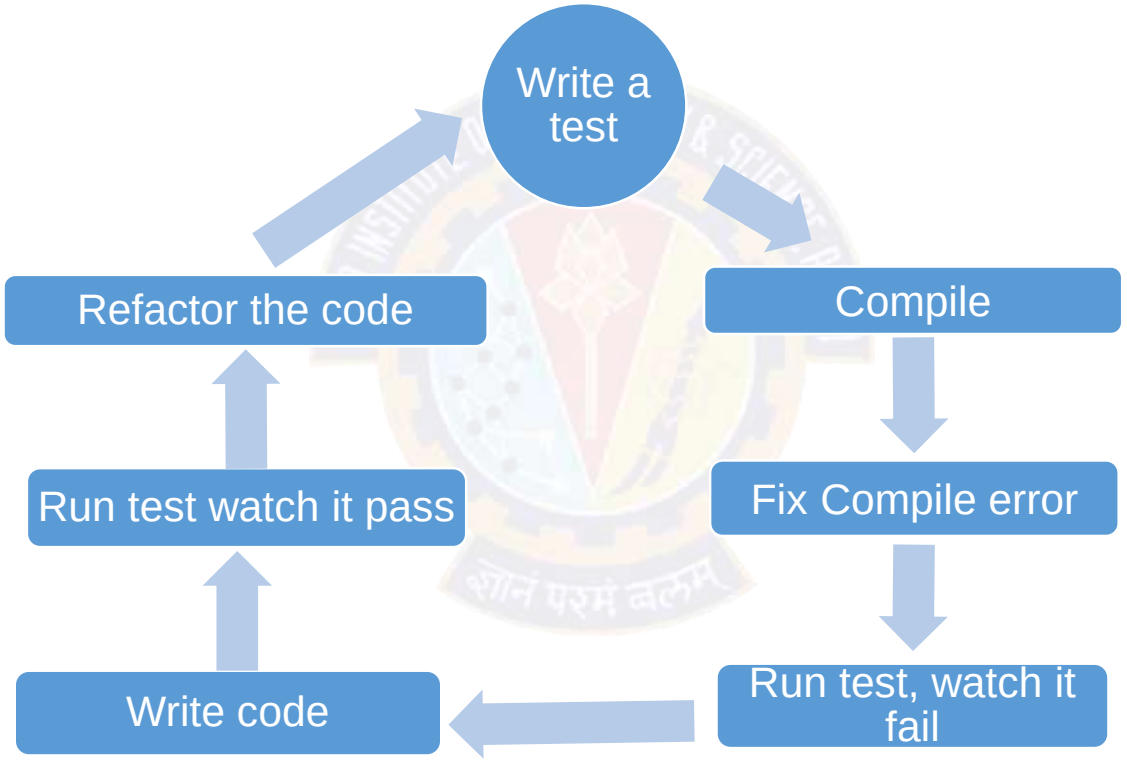
TDD

TDD Technique

- TDD is a technique whereby you write your test cases before you write any implementation code

TDD

TDD Overview



TDD

TDD Overview Contd..

- How It Works:
- Add a Test
  - Use Cases / User Stories are used to understand the requirement clearly
- Run all tests and see the new one fail
  - Ensures test harness is working correctly
  - Ensures that test does not mistakenly pass
- Write some code
  - Only code that is designed to pass the test
  - No additional functionality should be included because it will be untested
- Run the automated tests and see them succeed
  - If tests pass, programmer can be confident code meets all tested requirements
- Refactor code
  - Cleanup the code
  - Rerun tests to ensure cleanup did not break anything
- Repeat

TDD

TDD Overview Contd..

- In Extreme Programming Explored (The Green Book), Bill Wake describes the test / code cycle:
  1. Write a single test
  2. Compile it. It shouldn't compile because you've not written the implementation code
  3. Implement just enough code to get the test to compile
  4. Run the test and see it fail
  5. Implement just enough code to get the test to pass
  6. Run the test and see it pass
  7. Refactor for clarity and "once and only once"
  8. Repeat

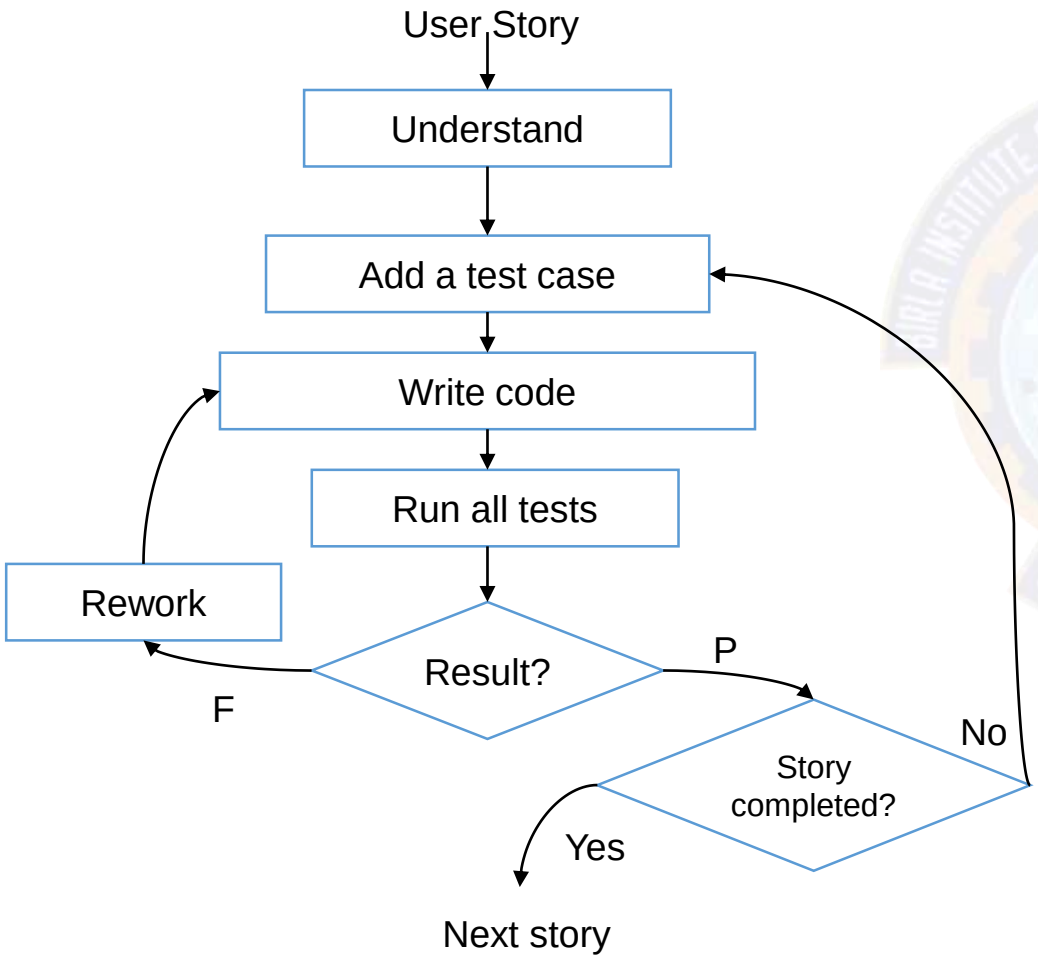
TDD

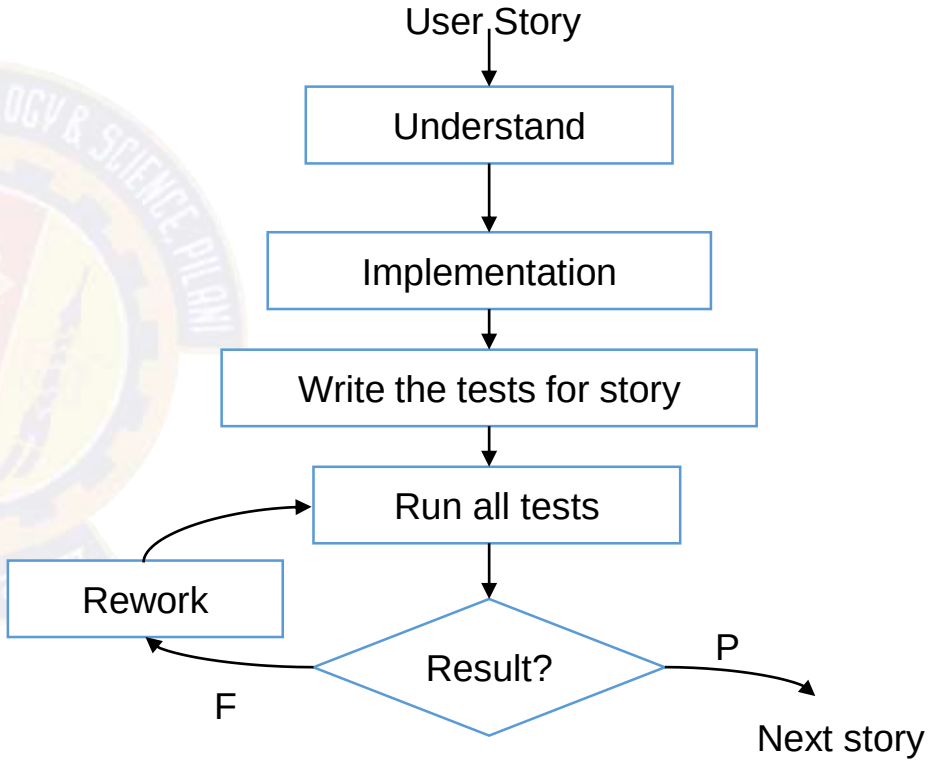
Why TDD

- TDD can lead to more modularized, flexible, and extensible code
- Clean code
- Leads to better design
- Better code documentation
- More productive

TDD

Test First vs. Test Last





Feature Driven Development

FDD History

- Original Creator: Jeff De Luca
  - Singapore in late 1997
- FDD evolved from an actual project
  - Bank Loan Automation
  - Luca was Project manager
  - 50 member developer team
  - Peter Coad : Chief Architect
    - 1990’s object-oriented analysis and design expert

FDD

What is a Feature?

- Definition: small function expressed in client-valued terms

- FDD’s form of a customer requirement

FDD

FDD Primary Roles

- Project Manager
- Chief Architect
- Development Manager
- Domain Experts
- Class Owners
  - This concept differs FDD over XP
  - Benefits
    - Someone responsible for integrity of each class
    - Each class will have an expert available
    - Class owners can make changes much quicker, if needed anytime
    - Easily lends to notion of code ownership
    - Assists in FDD scaling to larger teams, as we have one person available for complete ownership of feature.
- Chief Programmers

FDD

FDD Supporting Roles

- Release Manager
- Language Guru
- Build Engineer
- Toolsmith
- System Administrator
- Tester
- Deployers
- Technical Writer

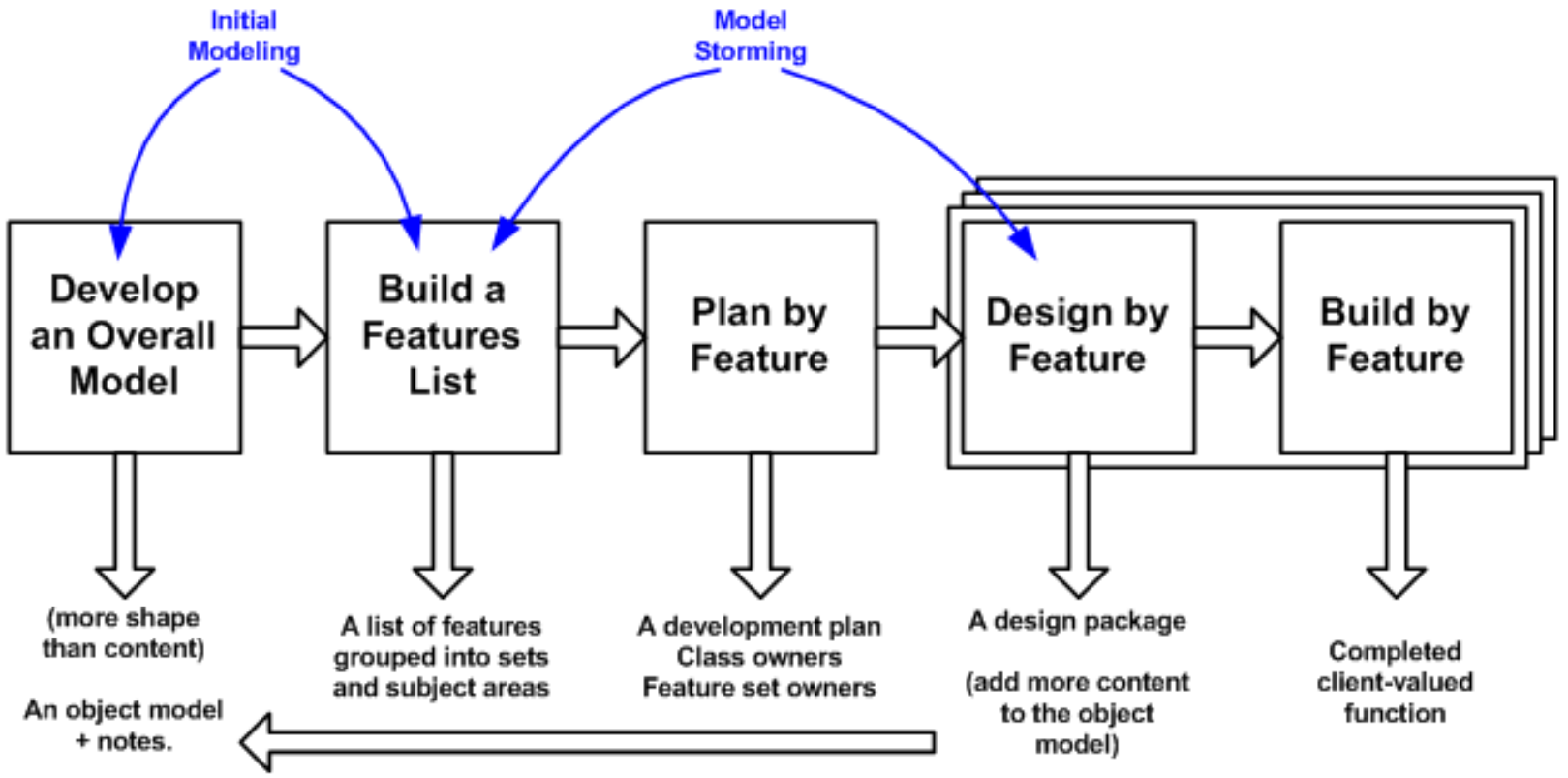
FDD

Feature Driven Development Process

- Process #1: Develop an Overall Model
- Process #2: Build a Features List
- Process #3: Plan By Feature
  - Constructing the initial schedule, Forming level of individual features, Prioritizing by business value , As we work on above factors we do consider dependencies, difficulty, and risks.
  - These factors will help us on Assigning responsibilities to team members, Determining Class Owners , Assigning feature sets to chief programmers
- Process #4: Design By Feature
  - Goal: not to design the system in its entirety but instead is to do just enough initial design that you are able to build on
  - This is more about Form Feature Teams: Where team members collaborate on the full low level analysis and design.
- Process #5: Build By Feature
  - Goal: Deliver real, completed, client-valued function as often as possible

FDD

Feature Driven Development Process Contd..



Behavior Driven Development

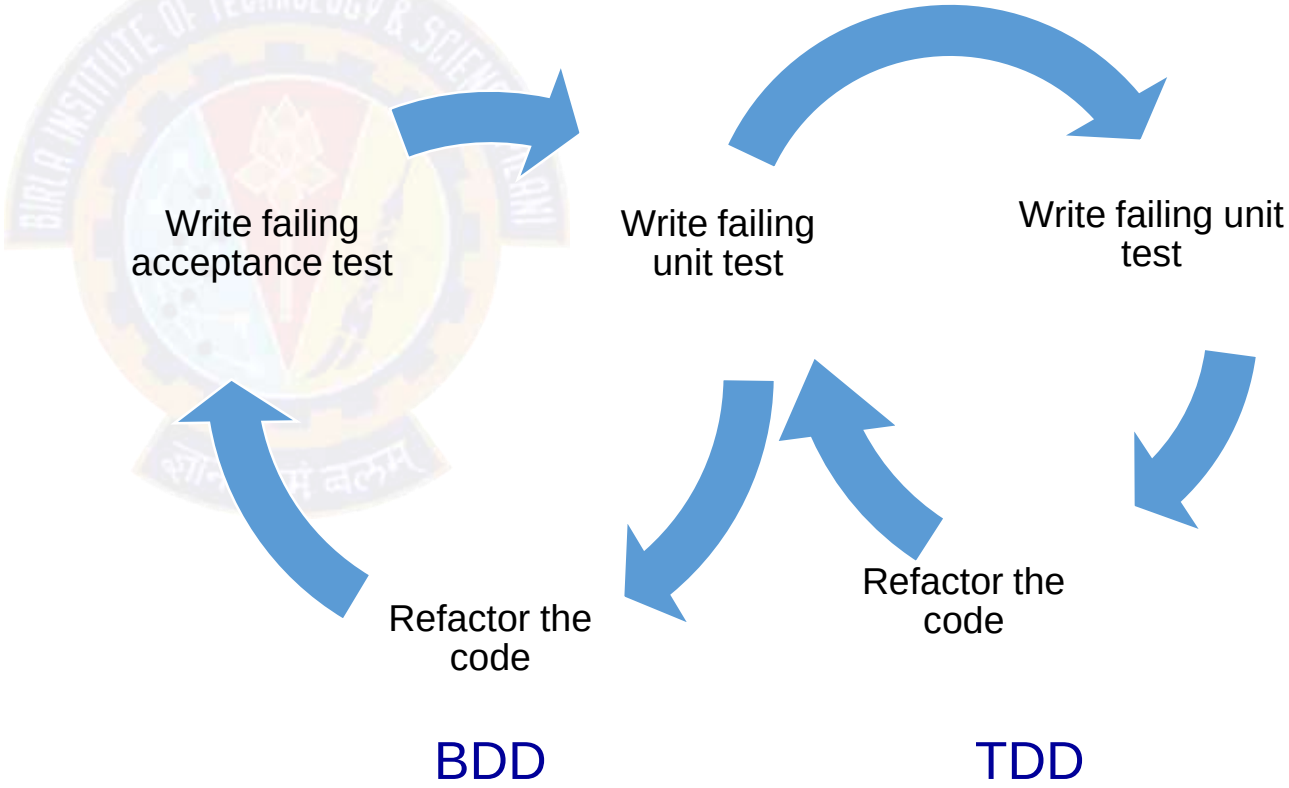
BDD Introduction

- Behavior-Driven Development (BDD) is a software development process
- BDD is considered an effective technical practice especially when the "problem space" for the business is complex
- What is BDD?:
  - General Technique of TDD
  - Follows same principle as TDD
  - Shared tools
  - Shared Process

BDD

BDD Focus

- Where to start in the process
- What to test and what not to test
- How much to test in one go
- What to call the tests
- How to understand why a test fails



BDD

BDD Basic structure

- BDD consists of cycles of a set of steps to follow.
  - Identify business feature
  - Identify scenarios under the selected feature
  - Define steps for each scenario
  - Run feature and fail
  - Write code to make steps pass
  - Refactor code, Create reusable automation library
  - Run feature and pass
  - Generate test reports

BDD

BDD Example

- User story:
- As someone interested in using the Mobile app, and want to sign up in app to enjoy app membership

- Mobile App Signup:
- Scenario 1:
  - Given that I am on the app's "Create new account" screen
  - Then I should see a "Sign up using Facebook" button
  - And I should see a "Sign up using Twitter" button
  - And I should see a "Sign up with email" form field
- Then I should see a new screen that asks permission to use my Facebook account data to create my new Mobile app account

BDD

BDD Sign Use

- Functional Documentation in terms of User Stories & Executable Scenarios
- Instead of referring to "tests", a BDD practitioner will prefer the terms "scenario" and "specification"
- Rather than refer to "functional tests", the preferred term will be "specifications of the product's behavior"
- Specification Names are more expressive

References

- http://agilemodeling.com/

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

DevOps – People

- DevOps Team Structure
- Team Coordination

Team Structure

Team Size

- The size of the team should be relatively small
- Amazon has a “two pizza rule”
- That is, no team should be larger than can be fed from two pizzas

- The advantages of small teams are
  - They can make decisions quickly
  - It is easier for individuals to express an opinion
  - It is easier for individuals to express an idea in front of a small group than in front of a large one

- The disadvantage
  - Some tasks are larger than can be accomplished by a small number of individuals
  - Task has to be broken up into smaller pieces
  - A small team, by necessity, works on a small amount of code

Team Structure

Team Roles

- We will study roles in the team from Scott Ambler’s description of roles in an agile team
- Team lead :
  - This role you can relate to “Scrum Master” of SCRUM TEAM.
  - Also you can relate to Project Lead
  - Is responsible:
    - For facilitating the team
    - Obtaining resources for it
    - And protecting it from problems
  - This role encompasses the soft skills of project management but not the technical ones such as planning and scheduling, activities which are better left to the team as a whole.

- Team member
  - This role, sometimes referred to as developer or programmer
  - Is responsible for the creation and delivery of a system
  - This includes modeling, programming, testing, and release activities, as well as others

Team Structure

Additional roles

- Service owner
- Reliability engineer
- Gatekeeper
- And DevOps engineer

Team Structure

Service Owner

- This role on the team responsible for outside coordination
- The service owner participates:
  - System-wide requirements activities
  - Prioritizes work items for the team
  - And provides information both from the clients of the team's service and about services provided to the team.
  - The requirements gathering and release planning activities for the next iteration can occur in parallel with the conception phase of the current iteration
  - The service owner maintains and communicates the vision for the service
  - That is, the vision involves the architecture of the overall system and the team's role in that architecture

Team Structure

Reliability Engineer

- Reliability engineer monitors the service in the time period immediately subsequent to the deployment
- Second, the reliability engineer is the point of contact for problems with the service during its execution
- Google calls this role “Site Reliability Engineer.”
- Once a problem occurs, the reliability engineer performs short-term analysis to diagnose, mitigate, and repair the problem
- In any case, the reliability engineer has to be excellent at troubleshooting and diagnosis
- The reliability engineer should discover or work with the team to discover the root cause of a problem
- Increasingly, reliability engineers need to be competent developers, as they need to write high-quality programs to automate the repetitive part of the diagnosis, mitigation, and repair

Team Structure

Gatekeeper

Netflix uses the steps given in Figure 1.3 from local development to deployment.

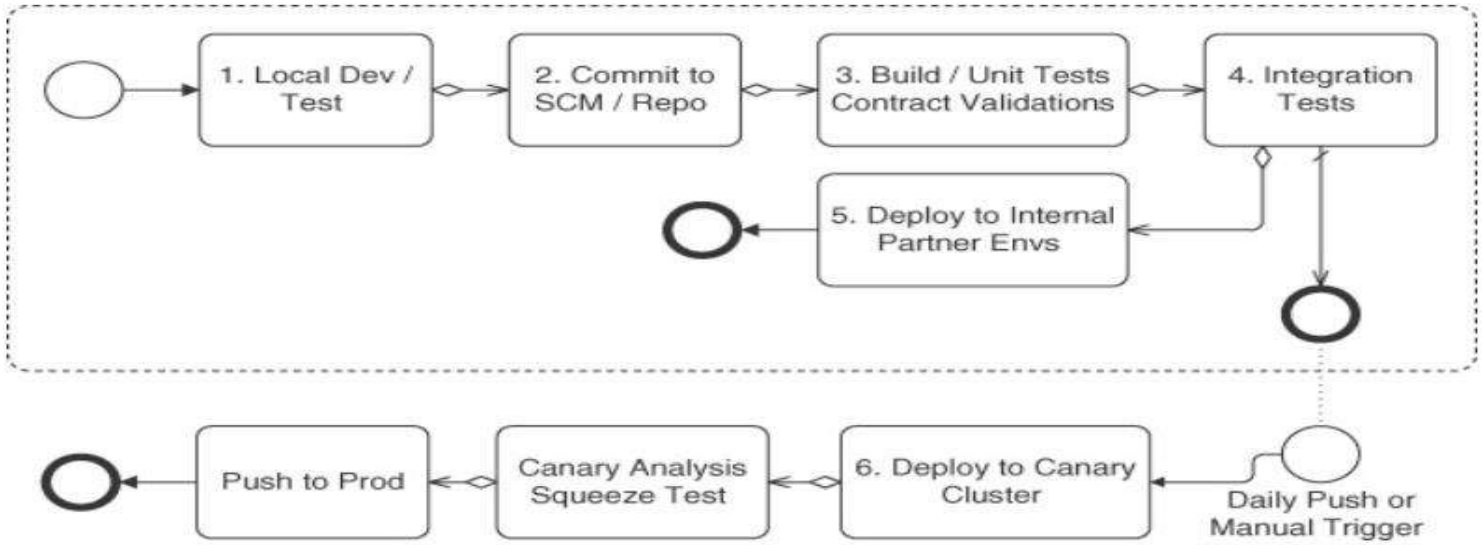


FIGURE 1.3 Netflix path to production. (Adapted from http://techblog.netflix.com/2013/11/preparing-netflix-api-for-deployment.html) [Notation: BPMN]

Team Structure

DevOps Engineer

- The DevOps engineer role is responsible for the care and feeding of the various tools used in the DevOps tool chain
- The DevOps engineer’s role, as are the tailoring of the tool for the development team and monitoring
- The DevOps engineering role is inherent in automating the development and deployment pipeline
- DevOps engineer also deals with configuration managements tools which are like Puppet, Ansible and Chef etc.
- These configuration management solutions are applied to the source code for the service

Team Coordination

Forms of Coordination

- One goal of DevOps is to minimize coordination in order to reduce the time to market
- Two of the reasons to coordinate are, first, so that the pieces developed by the various teams will work together and, second, to avoid duplication of effort
- Forms of Coordination
  - Direct—the individuals coordinating know each other (e.g., team members)
  - Indirect—the coordination mechanism is aimed at an audience known only by its characterization (e.g. Operators)
  - Persistent—the coordination artifacts are available after the moment of the coordination (e.g., documents, e-mail, bulletin boards)
  - Ephemeral—the coordination, per se, produces no artifacts (e.g., face to face meetings, conversations, telephone/video conferencing)
  - Synchronous—individuals are coordinating in real time, (e.g., face to face)
  - Asynchronous—individuals are not coordinating in real time (e.g., documents, e-mail)

Team Coordination

Team coordination mechanisms

- Team coordination mechanisms are of two types
  - Human Processes and
  - Automated Processes
- The DevOps human processes are adopted from agile processes and are designed for high-bandwidth coordination with limited persistence
  - Examples: Stand-up meetings
- Automated team coordination mechanisms are designed to protect team members from interference of their and others' activities:
  - To automate repetitive tasks (continuous integration and deployment),
  - And to speed up error detection and reporting (automated unit, integration, acceptance, and live production tests)

Team Coordination

Cross-team Coordination

- Coordination must occur with customers, stakeholders, other development teams, and operations. Therefore, DevOps processes attempt to minimize this coordination as much as possible
- There are three types of cross-team coordination:
  - Upstream Coordination [with stakeholders and customers]
  - Downstream Coordination [with operations]
  - And Cross-Stream Coordination [with other development teams]
- There are two reasons for a development team to coordinate with other development teams
  - To ensure that the code developed by one team works well with the code developed by another
  - And to avoid duplication of effort

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

DevOps People

- Transformation to Enterprise DevOps culture

Transformation to Enterprise DevOps culture

Constraints of Large Enterprise :: DevOps Transformation

- Organization Structure
- Legacy Technology Stack
- Organization Culture

Transformation to Enterprise DevOps culture

Concepts

- Executives do not like terms such as “Transformation”
- Somehow it is often interpreted as:
  - Cost Cutting or
  - Resource Realignment or
  - Outsourcing
- Why organizations are pushing to be DevOps
- The intent is to transform the way solutions are being delivered today and to do it faster, cheaper and better
- The techniques of Agile and DevOps have been successfully implemented in many large and small organizations to enable those outcomes
- Look in to the approach that worked quite well for most enterprises

Transformation to Enterprise DevOps culture

Initial Planning for Enterprise Readiness

- Participants should absolutely include all the towers that make up the solution delivery
- Design Thinking : It is a great method; it leverages the expertise of all stakeholders, enables them to come to a common understanding
- High Level Output

Transformation to Enterprise DevOps culture

Establish a DevOps Center of Excellence

- At the right organizational level and enterprise authority
- Create a Face: It has to be led by an enterprise leader who has the support and buy-in from all the towers
- Active Participant from All Towers of Delivery & Participants must be chosen wisely
- These phases help concrete the vision and strategies of organization and help in setup the best practices to support cultural movement

Transformation to Enterprise DevOps culture

Establish Program Governance

- Agile and DevOps, practitioners' roles and responsibilities will change
- Need awareness, enablement and empowerment to succeed
- KPIs must shift from individual metrics to holistic customer business outcomes

Transformation to Enterprise DevOps culture

Establish Project In-take Process

- DevOps SME’s conduct in-take workshops for Scrum Teams
- Automation scripts, Infrastructure assets
- Test Automation, Branching and Merging & Lessons Learned
- Fit for purpose tool selection and Application Criticality

Transformation to Enterprise DevOps culture

Identify and Initiate Pilots

- Value stream-mapping exercise
- Level of detail necessary:
  - To identify end to end as-is process, tooling, manual and automated processes
  - And skills and people

Transformation to Enterprise DevOps culture

Scale Out DevOps Program

- Onboard Parallel Release Trains
- Apply Intake Process
- Support, Monitor and Manage

References

- I would like to thank you Mr. Sunil Joshi
- https://devops.com/six-step-approach-enterprise-devops-transformation/

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

DevOps Culture

- DevOps People Dimension

DevOps People Dimension

DevOps is a culture

- People
  - DevOps is a cultural movement; it's all about people
  - Building a DevOps culture, therefore, is at the core of DevOps adoption

- An organization may adopt the most efficient processes or automated tools possible, but they're useless without the people who eventually must execute those processes and use those tools

- DevOps culture is characterized by a high degree of collaboration across roles, focus on business instead of departmental objectives, trust, and high value placed on learning through experimentation
- Building a culture isn't like adopting a process or a tool
- It requires social engineering of teams of people, each with unique predispositions, experiences, and biases
- This diversity can make culture-building challenging and difficult
- Lean and agile transformation practices are at the core of DevOps

DevOps People Dimension

Managing People, Not Resources

- “Managing teams would be easy if it wasn’t for the people you have to deal with”
- Some people react allergically when you call people resources and talk about managing resources
- A resource is something you can manage without much tailoring
- For an Example: you can treat one bag of sand like the next bag of sand when it comes to making dikes
- With people, that is never true
- For an Example: We have probably all been planning for projects and creating a plan that includes a certain amount of “anonymous full-time equivalents (FTEs).” Then a bit later, we start putting names to it and realize that we need more or less effort based on the actual people we have available for the project
- One Java developer is just not the same as another; and honestly, we would never consider ourselves to be just a resource whose job someone else could do with the same efficiency and effectiveness

DevOps People Dimension

Don’t change Organization; Change yourself

- You will not be able to get the organization to change
- But what you can do is change your part of the organization
- Manage your teams the way that you would like to be treated
- As you climb higher in the hierarchy, your area of influence will increase and, with that, a larger and larger part of the organization will work the way you would like it to

DevOps People Dimension

Identifying Business Objective

- Getting everyone headed in the same direction
- And working toward the same goal
- “Identify Common Business Objective for the Team and Organization”
- Incent the entire team based on business outcomes versus conflicting team incentives
- Easy to measure progress of goal for team and organization
- DevOps isn’t the goal. It helps you reach your goals

DevOps People Dimension

Effective Management of People

- One-On-Ones
  - Regular one-on-one meetings with the people who report directly to you
  - Don’t be Anonymous
  - Having an open-door policy is NOT the same as setting up one-on-ones
  - Let feel people are important to you
  - You are making time for them
  - Best Practices : have weekly or bi weekly 30 minutes one-on –one, learn more about person, let them raise first and provide your updates
  - Benefit to Manager?
  - Benefit to Employee?
- Feedback
  - Everyone would like to get more feedback from his or her boss
  - Dan Pink’s mastery, autonomy, and purpose : “When you do x, y happens, which is not optimal. Can you find a way to do it differently next time to lead to a better outcome? “
  - Focus on feedback?

DevOps People Dimension

Effective Management of People Contd..

- Delegation
  - Feel on delegation of JOB to others?
  - Managerial Economics 101
- Creating a Blameless Culture
  - Who is being blamed?
  - You will have to cover it without delegating the blame to the person in your team who is responsible
  - The team will appreciate this
  - The more you share with the rest of the organization the positive impact a member of your team has made, the better you will look too
  - These two practices empower the people on your team to do the best job they can for you
  - Whenever you have failures or problems, your root-cause analysis should not focus on who did what but on how we need to change the system so that the next person is able to avoid making the mistake again

DevOps People Dimension

Effective Management of People Contd..

- Measuring Your Organizational Culture
  - There are a few different ways to measure culture
  - The Westrum survey measures
    - On my team, information is actively sought.
    - On my team, failures are learning opportunities, and messengers of them are not punished.
    - On my team, responsibilities are shared.
    - On my team, cross-functional collaboration is encouraged and rewarded.
    - On my team, failure causes inquiry.
    - On my team, new ideas are welcomed
  - There are few more suggestions:
    - I would recommend the team to my friends as a good place to work.
    - I have the tools and resources to do my role well.
    - I rarely think about leaving this team or my company.
    - My role makes good use of my skills and abilities.
- Remember that advice stated before: you can only control your part of the organization

DevOps People Dimension

Effective Management of People : Lets Summarize

- Measuring Your Organizational Culture
  - Provide your employees:
    - An organizational structure
    - The business context and
    - The necessary feedback
- People are factor that makes transformation succeed or fail
- Spend enough time and effort on People during the transformation

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Tools and Technology in DevOps

- What is DevOps Tools
- Local Development Environment
- Version Control
- Artifact Management
- Automation
- Monitoring
- Evolution of the Ecosystem

DevOps Tools

What is DevOps Tools:

- What is DevOps Tools:
- Tools can be used to improve and maintain various aspects of culture
- Software development tools help with the process of programming, documenting, testing, and fixing bugs in applications and services
- Not restricted to specific roles, these tools are important to anyone who works on software in some capacity
  - Local development environment,
  - Version control,
  - Artifact Management,
  - Automation and Monitoring

DevOps Tools

Local Development Environment

- A consistent local development environment is critical to quickly get employees started contributing to product
- Don’t Consider this as limitation:
- This is not to say that individuals should be locked into a single standard editor with no flexibility or customization, but rather it means ensuring that they have the tools needed to get their jobs done effectively
- Minimal requirements may vary in your environment depending on individual preferences:
- Multiple Displays
- High-Resolution Displays
- Specific Keyboards, Mice, and other Input Devices
- Identify a shared area for documenting the local development environment
- Documentation can be stored within a version control repository, an internal wiki, or even a Google Doc

DevOps Tools

Version Control

- Having the ability to commit, compare, merge, and restore past revisions of objects to the repository allows for richer cooperation and collaboration within and between teams
- Version control enables teams to deal with conflicts that result from having multiple people working on the same file or project at the same time, and provides a safe way to make changes and roll them back if necessary
- When choosing the appropriate version control system (VCS) for your environment, look for one that encourages the sort of collaboration in your organization that you want to see
  - Opening and forking repositories;
  - Contributing back to repositories;
  - Curating contributions to your own repositories;
  - Defining processes for contributing; and
  - Sharing commit rights
- Your Wrong Ways may be:
  - Lines of code is not an accurate measure of value

DevOps Tools

Artifact Management

- An artifact is the output of any step in the software development process
- When choosing between a simple repository and more complex feature-full repository, understand the cost of supporting additional services as well as inherent security concerns
- An artifact repository should be:
  - Secure;
  - Trusted;
  - Stable;
  - Accessible; and
  - Versioned
- You can store a versioned common library as an artifact separate from your software version control, allowing all teams to use the exact same shared library

DevOps Tools

Automation

- Automation tools reduce labor, energy, and/or materials used with a goal of improving quality, precision, and accuracy of outcomes
- Server Installation
  - Server installation is the automation of configuring and setting up individual servers
- Infrastructure Automation
  - Configuration Management
  - Capacity Management
- System Provisioning
- Test and Build Automation
  - On-demand automation
  - Scheduled automation
  - Triggered automation
  - Smoke testing
  - Regression testing
  - Usability testing

DevOps Tools

Monitoring

- Metrics
  - Metrics are the collection of qualitative and quantitative measurements
  - Generally they are compared against some benchmark or established standard, tracked for analytics, or tracked for historical purposes
- Logging
  - Logging is the generation, filtering, recording, and analysis of events that occur on a system due to operating system or software messages
  - When tracking down the source of a software issue, one of the first things that engineers often do is to check the logs for any relevant error messages
  - A single system can generate hundreds or even thousands of lines of logs a day
- Alerting
  - Monitoring and alerting are important not only from a performance perspective, but also from a preventative one, in that they help you find out about potential issues before they become actual issues for your customers

DevOps Tools

Monitoring Contd..

- Events
  - Event management is the element of monitoring that acts on existing knowledge around impacts to systems and services
  - This generally reflects the need for real-time information about the status of all of the different components of infrastructure
  - A system is configured to monitor a specific metric or log based on a defined event and to signal or alert if a threshold is crossed or an alert condition has been met
  - Many alerting and monitoring systems have built-in ways to automatically respond to a given event
  - Nagios, Zabbix, etc.

DevOps Tools

Evolution of the Ecosystem

- There is a trend to simplify and remove the repetitive tasks that can be subject to human error
- As automation is added to the different parts of the environment, new patterns are discovered
- Continuous delivery and continuous deployment have freed humans up to focus on what matters
- Automated shortened feedback cycles through build automation with tests give us additional confidence and insight into our systems
- The ecosystem will continue to evolve as application development adopts increased operability
- These trends bring into focus the tools that stress the “we” over “me,” building understanding across teams and encouraging time spent on valuable outcomes

DevOps Tools

Summarize

- We have been through an overview of the current ecosystem of tools
- How tools are used, and the ease with which they can be used, impacts the acceptance and proliferation of specific aspects of culture
- The DevOps culture is one of collaboration across teams, organizations, and industries
- When developing solutions, it is important to think about their impact on teams and organizations, not just individuals
- DevOps tools stress “we” over “me”; they allow teams and organizations to build mutual understanding to get work done

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Cloud as a catalyst for DevOps

- Introduction
- Characterization of the cloud
- Cloud services

Cloud as a Catalyst for DevOps

Introduction

- Unfortunately for many organizations, the cloud is still somewhat mysterious
- Not the concept of the cloud but why they are not seeing the benefits that were promised to them
- The focus is on adopting cloud-based infrastructure and managing it
- While we will discuss cost benefits primarily, one should not forget the other benefits and risks of the cloud
- Benefits:
  - Redundancy for resilience
  - The scalability of resources
  - The strong ecosystem of related
- Risks:
  - The dependency on a third-party provider
  - The challenges with data sovereignty
  - Risk of being attacked because you are on a popular platform

Cloud as a Catalyst for DevOps

Characterization of the cloud

- Characterization of the cloud by National Institute of Standards and Technology (NIST)
  - On-demand self-service
  - Broad network access
  - Resource pooling
  - Rapid elasticity
    - It is the Capabilities can be elastically provisioned and released
- Measured service
  - Software as a Service (SaaS)
  - Platform as a Service (PaaS)
  - Infrastructure as a Service (IaaS)

Cloud as a Catalyst for DevOps

Software as a Service (SaaS)

- In this the consumer is provided the capability to use the provider's applications running on a cloud infrastructure
- The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based e-mail) or an application interface.
- The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, storage.
- For an example, you can relate google apps, Cisco WebEx, as a Service, Office 365 service, where Provider deals with the licensing of software's

Cloud as a Catalyst for DevOps

Platform as a Service (PaaS)

- The consumer is provided the capability to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider
- The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, or storage, but consumer has control over the deployed applications and possibly configuration settings for the application-hosting environment
- For an Example: .NET Development platform is considered as a platform

Cloud as a Catalyst for DevOps

Infrastructure as a Service (IaaS),

- The consumer is provided the capability to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications
- The consumer does not manage or control the underlying cloud infrastructure but consumer has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls). For this you can consider any Server Provisioning is IaaS,

Cloud as a Catalyst for DevOps

Summary

- The cloud has emerged as a major trend in IT during recent years
- Its characteristics include metered usage (pay-per-use) and rapid elasticity, allowing the scaling out of an application to virtually infinite numbers of VMs
- The cloud rests on a platform that is inherently distributed and exploits virtualization to allow rapid expansion and contraction of the resources available to a given user
- From an operational perspective, controlling the VMs, managing different database management systems, and ensuring the environments meet the needs of the development and operations tasks are new considerations associated with the cloud

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Evolution of Version Control

- What is Version Control
- Version Control System
- Benefits of Version Control System

Evolution of Version Control

What is Version Control

- What is “version control”, and why should you care?
- Defination: Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later
- The history of version control tools can be divided into three generations

Generation	Networking	Operations	Concurrency	Example Tool
First Generation	None	One file at a time	Locks	RCS, SCCS
Second Generation	Centralized	Multi-file	Merge before commit	CVS, Subversion
Third Generation	Distributed	Changesets	Commit before merge	Bazaar, Git

Evolution of Version Control

Version Control System

- A category of software tools that help a software team manage changes to source code over time
- Version control software keeps track of every modification to the code
- If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake
- For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected
- Great version control systems facilitate a smooth and continuous flow of changes to the code

Evolution of Version Control

Benefits of Version Control System (VCS)

- A complete long-term change history of every file
- Restoring previous versions
- Branching and merging; having team members work concurrently
- Traceability; being able to trace each change made to the software
- Backup

Evolution of Version Control

Summary

- The vast majority of professional programmers are using second generation tools
- However the third generation is growing very quickly in popularity
- The most popular Version Control System is Apache Subversion
- Commonly known as SVN:
  - Subversion is an open source version control system
  - Founded in 2000 by CollabNet, Inc.
  - Subversion is developed as a project of the Apache Software Foundation
  - It’s a centralized version control system
  - Current Stable release is “Apache Subversion 1.11.1” as on date 11-Jan-2019

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Version control system and its types

- Centralized Version Control System
- Distributed Version Control System
- Version Control System Basics
- Version Control System Common Terminology
- Version Control System Basic Operations

Version Control System

Version Control System Types

- Centralized Version Control System:[CVCS]
  - With centralized version control systems, you have a single “central” copy of your project on a server and commit your changes to this central copy
  - You pull the files that you need, but you never have a full copy of your project locally
  - Some of the most common version control systems are:
    - Subversion (SVN) by Apache
    - Perforce by Perforce Software
- Distributed Version Control System: [DVCS]
  - With distributed version control systems (DVCS), you don't rely on a central server to store all the versions of a project’s files
  - Instead, you clone a copy of a repository locally so that you have the full history of the project
  - Some of most common distributed version control systems are:
    - Git
    - and Mercurial

Version Control System

Centralized Version Control System:[CVCS]

- Centralized Version Control Systems were developed to record changes in a central system and enable developers to collaborate on other systems
- Advantages of Centralized Version Control Systems:
  - Relatively easy to set up
  - Provides transparency
  - Enable admins control the workflow
- Disadvantages of Centralized Version Control Systems:
  - If the main server goes down, developers can’t save versioned changes
  - Remote commits are slow
  - If the central database is corrupted, the entire history could be lost (security issues)

Version Control System

Distributed Version Control System:[DVCS]

- They allow developers to clone the repository and work on that version. Develops will have the entire history of the project on their own hard drives
- Advantages of Distributed Version Control Systems:
  - Performing actions other than pushing and pulling changesets is extremely fast because the tool only needs to access the hard drive, not a remote server
  - Everything but pushing and pulling can be done offline
  - Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback
- Disadvantages of Centralized Version Control Systems:
  - If your project contains many large files then the space will be more on local drives
  - If your project has a very long history then downloading the entire history can take an impractical amount of time

Version Control System

Version Control System Available:[Centralized]

- Open source:
  - Subversion (SVN) – versioning control system inspired by CVS
  - Concurrent Versions System (CVS) – originally built on RCS, licensed under the GPL
  - Vesta – build system with a versioning file system and support for distributed repositories
  - OpenCVS – CVS clone under the BSD license, with emphasis put on security and source code correctness
- Commercial:
  - AccuRev – source configuration management tool with integrated issue tracking based on "Streams" that efficiently manages parallel and global development. Now owned by Micro Focus
  - Helix Core, formerly Perforce Helix - for large scale development environments
  - IBM Rational ClearCase – SCC compliant configuration management system by IBM Rational Software
  - Team Foundation Server (TFS) - Development software by Microsoft which includes revision control

Version Control System

Version Control System Available:[Distributed]

- Open source:
  - Git – written in a collection of Perl, C, and various shell scripts, designed by Linus Torvalds based on the needs of the Linux kernel project; decentralized, and aims to be fast, flexible, and robust
  - Bazaar – written in Python, originally by Martin Pool and sponsored by Canonical; decentralized, and aims to be fast and easy to use; can losslessly import Arch archives
  - Mercurial – written in Python as an Open Source replacement to BitKeeper; decentralized and aims to be fast, lightweight, portable, and easy to use
- Commercial:
  - Visual Studio Team Services - Services for teams to share code, track work, and ship software for any language by Microsoft
  - Sun WorkShop TeamWare – designed by Larry McVoy, creator of BitKeeper
  - Plastic SCM – by Codice Software, Inc
  - Code Co-op – peer-to-peer version control system (can use e-mail for synchronization)

Version Control System Basics

What do you want from your version control system?

- Store all versions of your files ("version control")
- Associate versions of each file with appropriate versions of all other files ("configuration management")
- Allow many people to work on the same files, toward a common goal or release ("concurrency")
- Allow groups of people to work on substantially the same files, but each group towards its own goal or release ("branching")
- Recover, at any time, a coherent configuration of file versions that correspond to some goal or release, either for investigation or extension like bug fixing ("release management")

Version Control System Basics

Version Control System Terminology

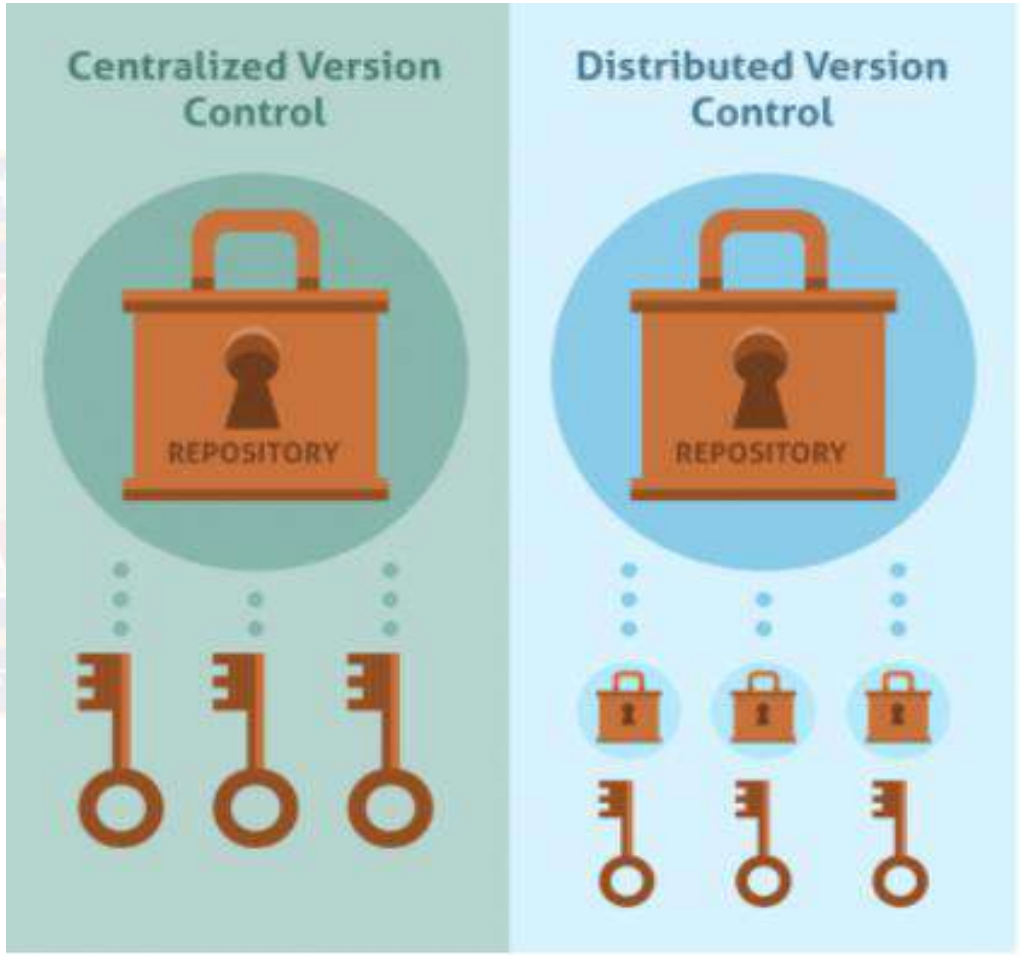
- Baseline : An approved revision of a document or source file from which subsequent changes can be made
- Branch : A set of files under version control may be branched or forked at a point in time so that, from that time forward, two copies of those files may develop at different speeds or in different ways independently of each other
- Change : A change represents a specific modification to a document under version control
- Checkout : To check out is to create a local working copy from the repository. A user may specify a specific revision or obtain the latest
- Clone : Cloning means creating a repository containing the revisions from another repository
- Commit : To commit is to write or merge the changes made in the working copy back to the repository
- Conflict : A conflict occurs when different parties make changes to the same document, and the system is unable to reconcile the changes
- Fetch : Fetch is initiated by the receiving repository
- Initialize : To create a new, empty repository
- Merge : A merge or integration is an operation in which two sets of changes are applied to a file or set of files
- Repository : The repository is where files' current and historical data are stored, often on a server
- Tag : A tag or label refers to an important snapshot in time, consistent across many files. These files at that point may all be tagged with a user-friendly, meaningful name or revision number

Version Control System Basics

Summary to Understand:

- Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy
- “Committing” a change simply means recording the change in the central system

- Distributed Version control systems do not necessarily rely on a central server to store all the versions of a project’s files. Instead, every developer “clones” a copy of a repository and has the full history of the project on their own hard drive
- Then you collect all copies and prepare a final copy



Version Control System Basics

Centralized VCS over Distributed VCS

- Advantages of Centralized VCS over Distributed VCS :
  - If your project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly
  - If your project has a very long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space

- Advantages of Distributed VCS over Centralized VCS :
  - Performing actions other than pushing and pulling changesets is extremely fast because the tool only needs to access the hard drive, not a remote server
  - Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once
  - Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone

- Misconceptions: Distributed systems do not prevent you from having a single “central” repository, they just provide more options on top of that

Version Control System Basic Operations

Create

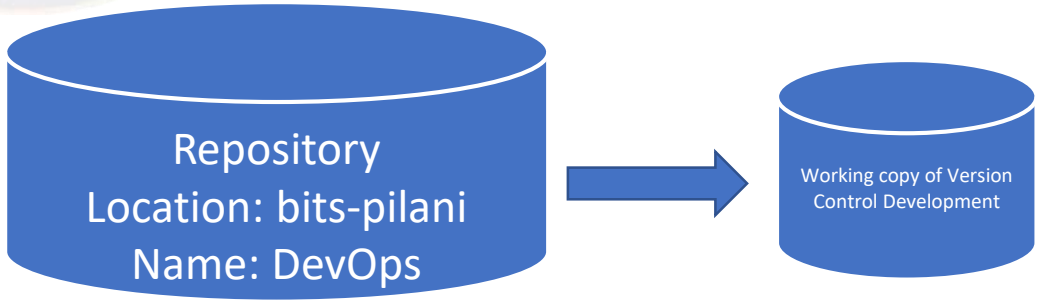
- Create a new, empty repository
- A repository is the official place where you store all your work.
- It keeps track of your tree i.e. all your files, as well as the layout of the directories in which they are stored
- A version control repository would be a network filesystem ✗

- A repository is much more than that. A repository contains history
  - Repository = Filesystem * Time
- A filesystem is two-dimensional: Its space is defined by directories and files
- Where as a repository is three-dimensional: It exists in a continuum defined by directories, files, and time
- What you need to create a Repository?
  - Location [Where you want to create it]
  - Name

Version Control System Basic Operations

Checkout

- Create a working copy
- This operation is used when you need to make a new working copy for a repository that already exists
- A working copy is a snapshot of the repository used by a developer as a place to make changes
- The working copy provides developer with a private workspace where developer can do his / her work isolated from the rest of the team
- The working copy is actually more than just a snapshot of the contents of the repository
- It also contains some metadata so that it can keep careful track of the state of things



Version Control System Basic Operations

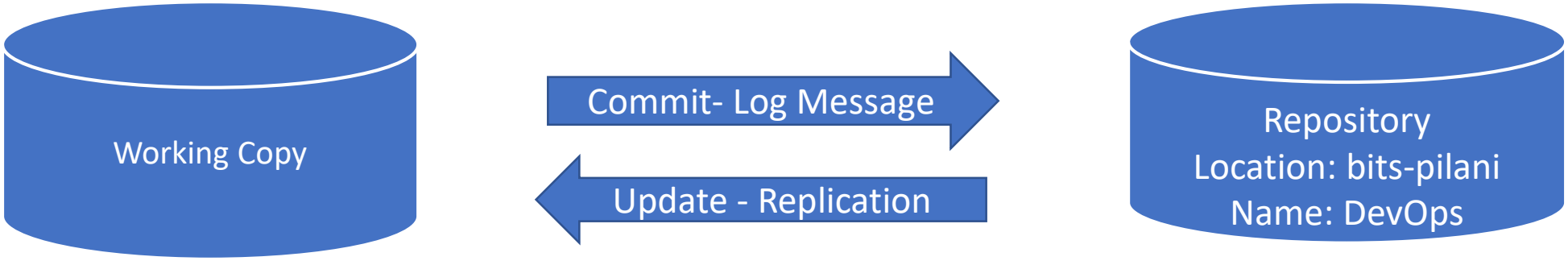
Commit

- Apply the modifications in the working copy to the repository as a new changeset
- This is the operation that actually modifies the repository
- Several others modify the working copy and add an operation to a list we call the pending changeset, a place where changes wait to be committed
- The commit operation takes the pending changeset and uses it to create a new version of the tree in the repository
- It is typical to provide a log message (or comment) when you commit, explaining the changes you have made. This log message becomes part of the history of the repository

Version Control System Basic Operations

Update

- Update the working copy with respect to the repository
- Update brings your working copy up-to-date by applying changes from the repository, merging them with any changes you have made to your working copy if necessary
- When the working copy was first created, its contents exactly reflected a specific revision of the repository
- Update is sort of like the mirror image of commit
- Both operations move changes between the working copy and the repository
- Commit goes from the working copy to the repository; Update goes in the other direction



Version Control System Basic Operations

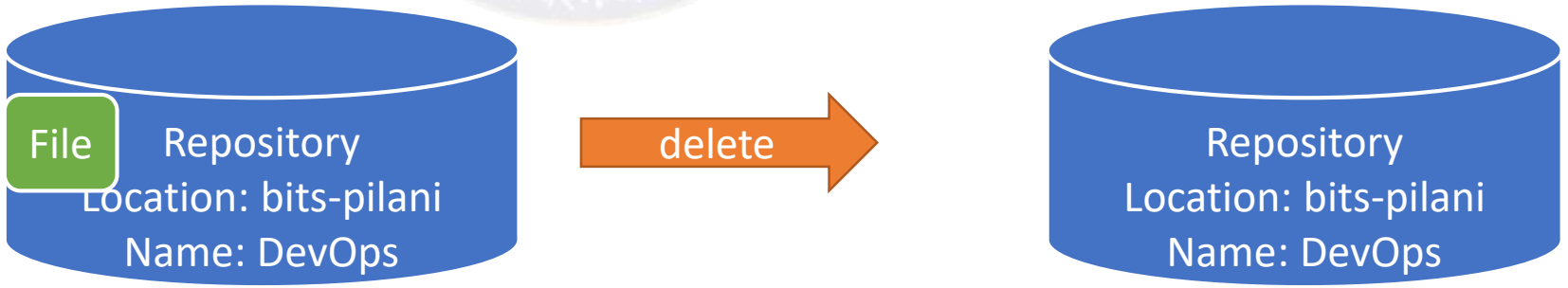
Add

- Add a file or directory
- Use the add operation when you have a file or directory in your working copy that is not yet under version control and you want to add it to the repository

Version Control System Basic Operations

Delete

- Delete a file or directory
- Use the delete operation when you want to remove a file or directory from the repository
- Typically, the delete operation will immediately delete the working copy of the file, but the actual deletion of the file in the repository is simply added to the pending changeset



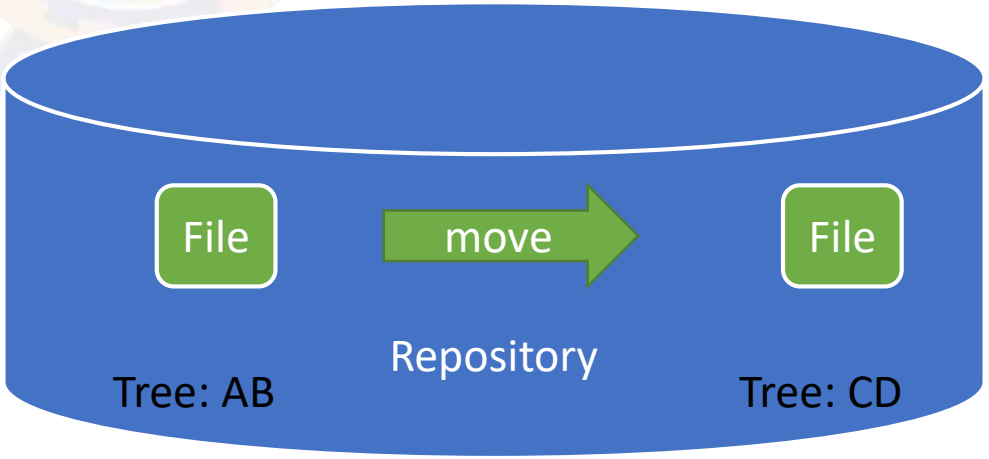
Repo 1.7

Repo 1.8

Version Control System Basic Operations

Move

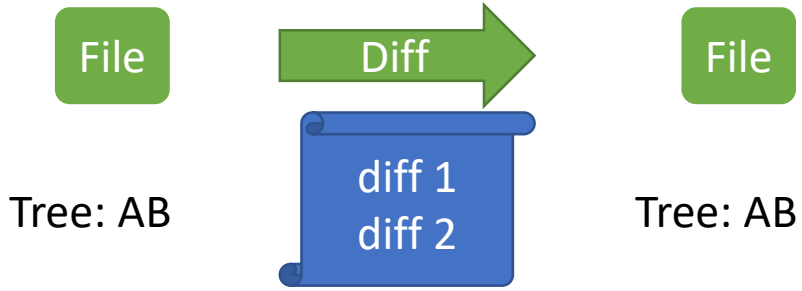
- Move a file or directory
- Use the move operation when you want to move a file or directory from one place in the tree to another
- Some tools treat rename and move as the same operation like Unix



Version Control System Basic Operations

Diff

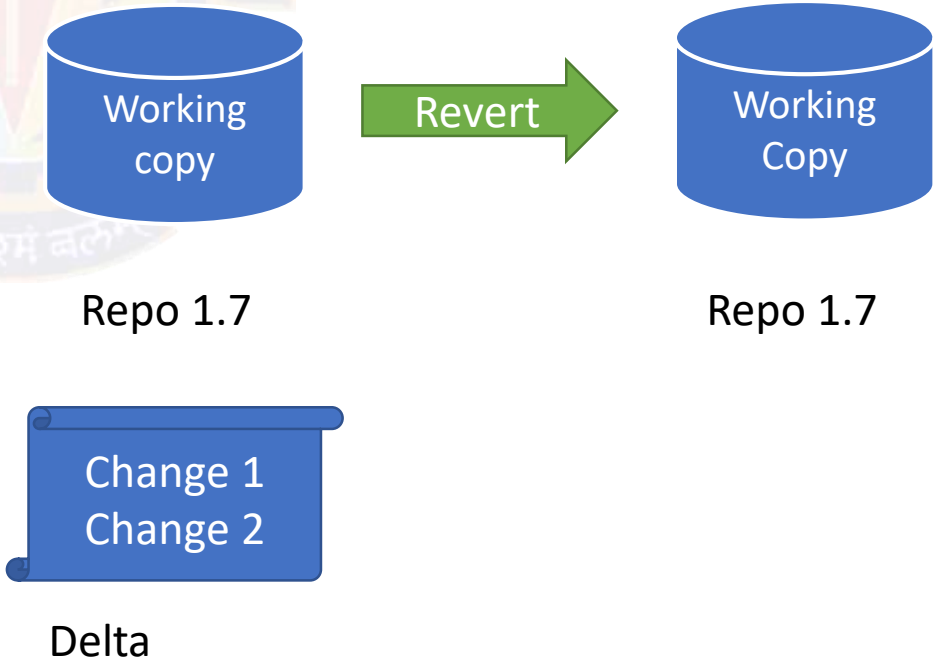
- Show the details of the modifications that have been made to the working copy
- In other hand Status provides a list of changes but no details about them
- To see exactly what changes have been made to the files, you need to use the diff operation



Version Control System Basic Operations

Revert

- Undo modifications that have been made to the working copy
- A complete revert of the working copy will throw away all your pending changes and return the working copy to the way it was just after you did the checkout



Version Control System Basic Operations

Log

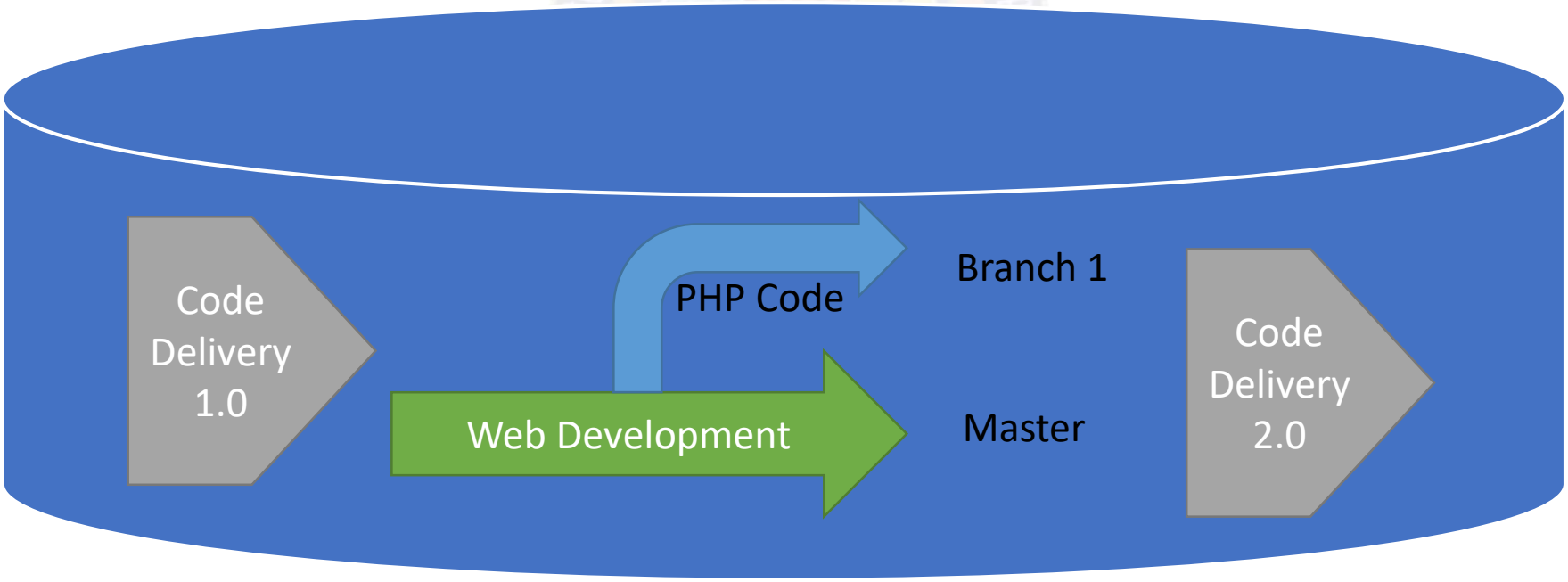
- Show the history of changes to the repository
- Your repository keeps track of every version that has ever existed
- The log operation is the way to see those records
- It displays each changeset along with additional data:
  - Who made the change?
  - When was the change made?
  - What was the log message?

- Repo Created – 1 Jan 2019 – user Rathi – For Devops @ bits
- Add File 1 – 10 Jan 2019 – user 1 – For Devops Source Code
- Add File 2 – 12 Jan 2019 – user 1 – For Devops Source Code
- Rename File 1 – 12 Jan 2019 – user A – For Java Ammend Code
- Created Working Copy – 15 Jan 2019 – User B
- Deleted File 2 – 5 Feb 2019 – User 1 – not need
- Moved File 1 – 5 Feb 2019 – User Rathi - rename

Version Control System Basic Operations

Branch

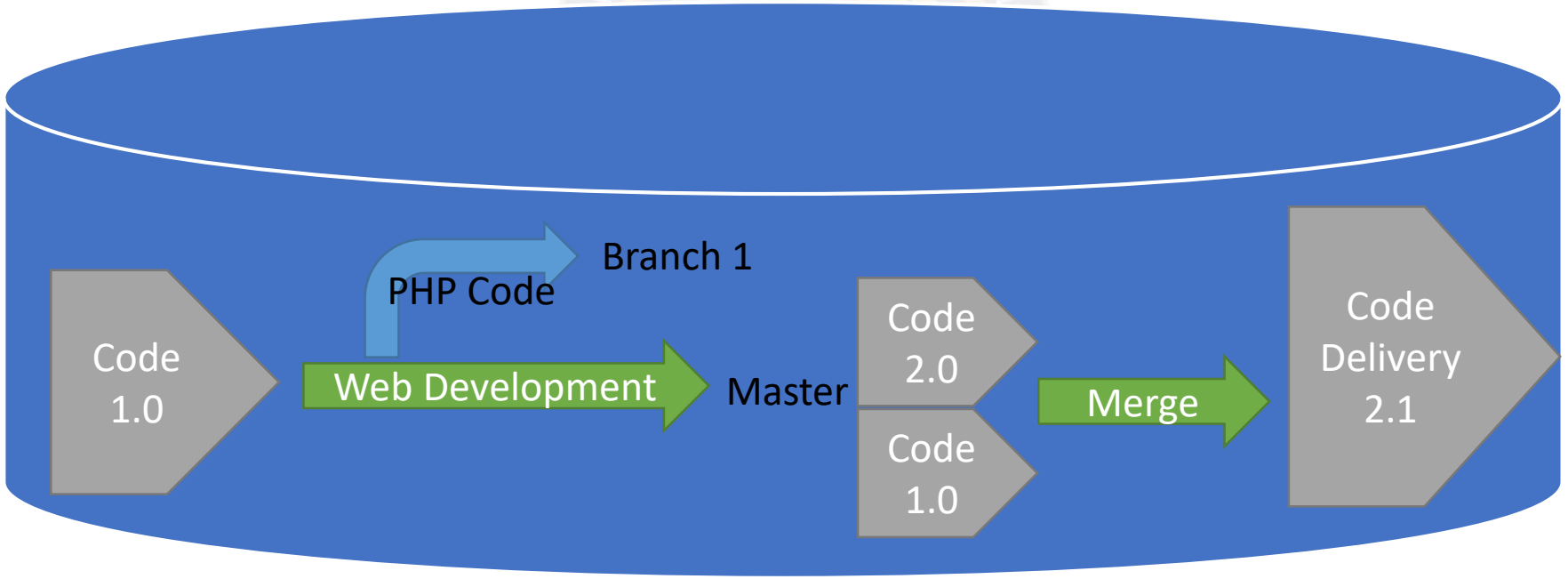
- Create another line of development
- The branch operation is what you use when you want your development process to fork off into two different directions



Version Control System Basic Operations

Merge

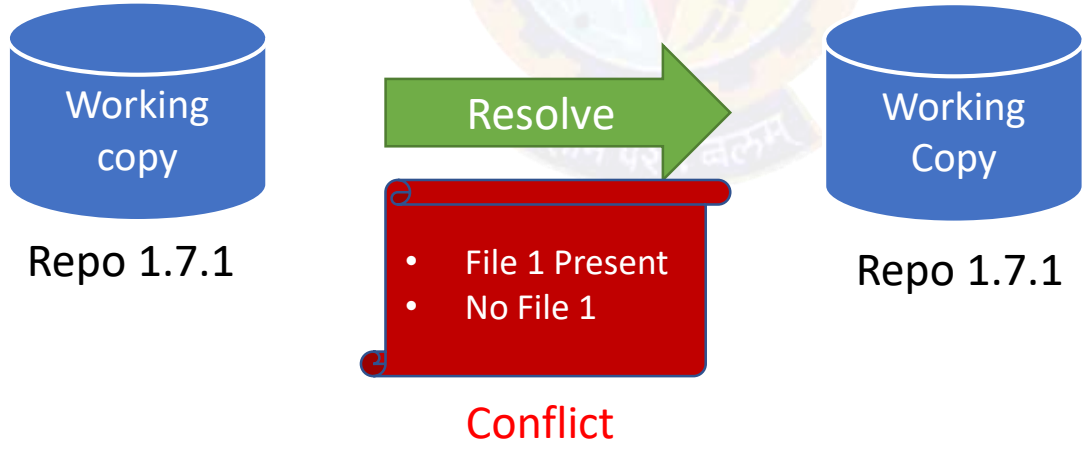
- Apply changes from one branch to another
- Typically when you have used branch to enable your development to diverge, you later want it to converge again, at least partially



Version Control System Basic Operations

Resolve

- Handle conflicts resulting from a merge
- Merge automatically deals with everything that can be done safely
- Everything else is considered a conflict
- The resolve operation is used to help the user figure things out and to inform the VCS how the conflict should be handled



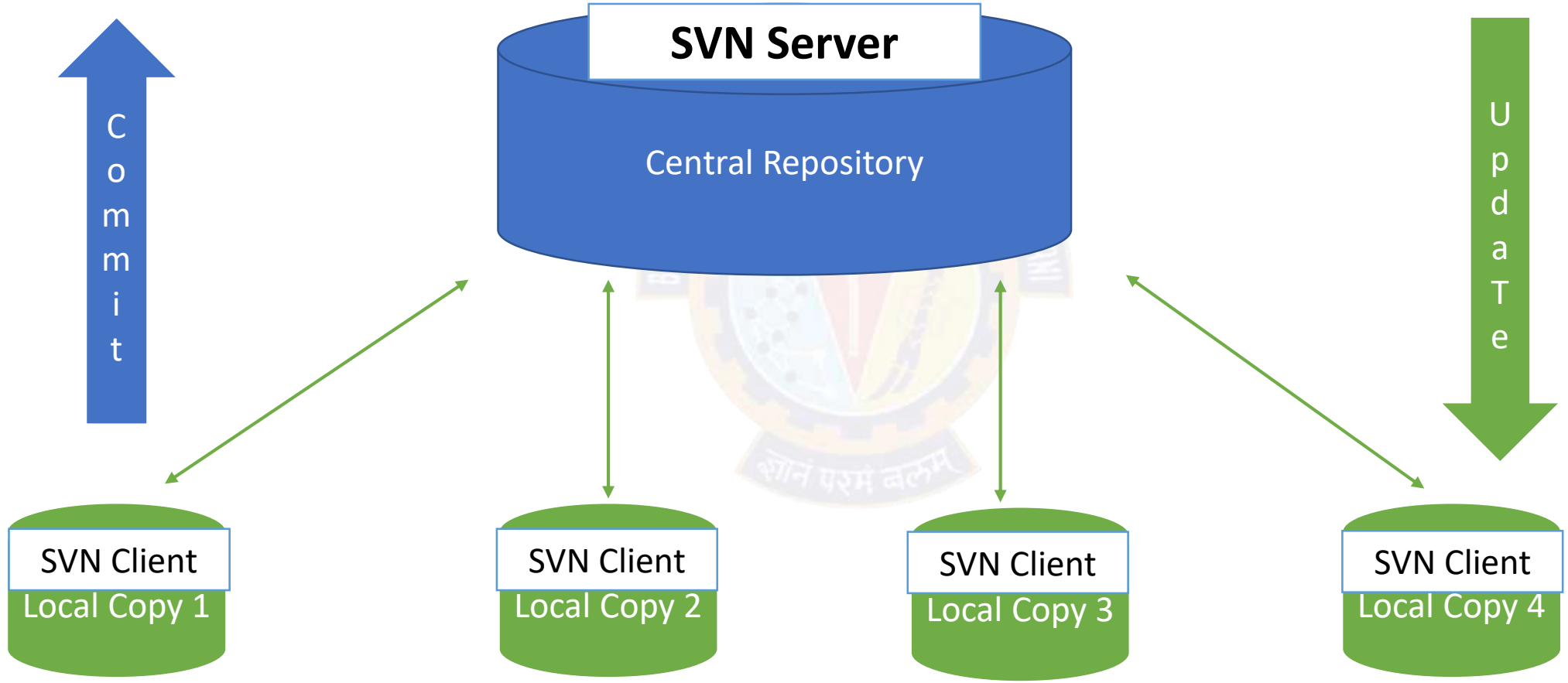
Examples of Centralized Version Control Systems

Apache Subversion (SVN)

- Subversion is a free/open source version control system
- Subversion manages files and directories, and the changes made to them, over time
- This allows you to recover older versions of your data, or examine the history of how your data changed
- Subversion is the most popular and its usage is growing larger
- Its centralized architecture make it easy to maintain a security hierarchy, access control, and backups, which is why it's preferred in many enterprise organizations over Perforce, VSS, and ClearCase
- Subversion is still widely popular amongst the open source community

Examples of Centralized Version Control Systems

Apache Subversion (SVN) Architecture



Examples of Centralized Version Control Systems

Features of Subversion

- CVS is a relatively basic version control system. For the most part, Subversion has matched or exceeded CVS's feature set where those features continue to apply in Subversion's particular design
- Subversion versions directories as first-class objects, just like files
- Copying and deleting are versioned operations. Renaming is also a versioned operation
- Subversion allows arbitrary metadata ("properties") to be attached to any file or directory
  - These properties are key/value pairs, and are versioned just like the objects they are attached to
  - These properties are not versioned, since they attach metadata to the version-space itself, but they can be changed at any time
- No part of a commit takes effect until the entire commit has succeeded
- Revision numbers are per-commit, not per-file, and commit's log message is attached to its revision
- Branches and tags are both implemented in terms of an underlying "copy" operation
- Unix users can place symbolic links under version control
- Subversion supports (but does not require) locking files so that users can be warned when multiple people try to edit the same file
- All output of the Subversion command-line client is carefully designed to be both human readable
- Subversion uses gettext() to display translated error, informational, and help messages, based on current locale settings
- The Subversion command-line client (svn) offers various ways to resolve conflicting changes, include interactive resolution prompting

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

About GIT

- Examples of Distributed Version Control Systems
- Introduction to GIT
- States of GIT
- A GIT Project
- GIT workflow

About GIT

Examples of Distributed Version Control Systems

- Distributed Version Control Systems (DVCSs) don’t rely on a central server
- They allow developers to clone the repository and work on that version
- Developers will have the entire history of the project on their own hard drives
- Revise:
  - Because of local commits, the full history is always available
  - No need to access a remote server (faster access)
  - Ability to push your changes continuously
  - Saves time, especially with SSH keys
  - Good for projects with off-shore developers
- Lets Discuss our Distributed Version Control System GIT

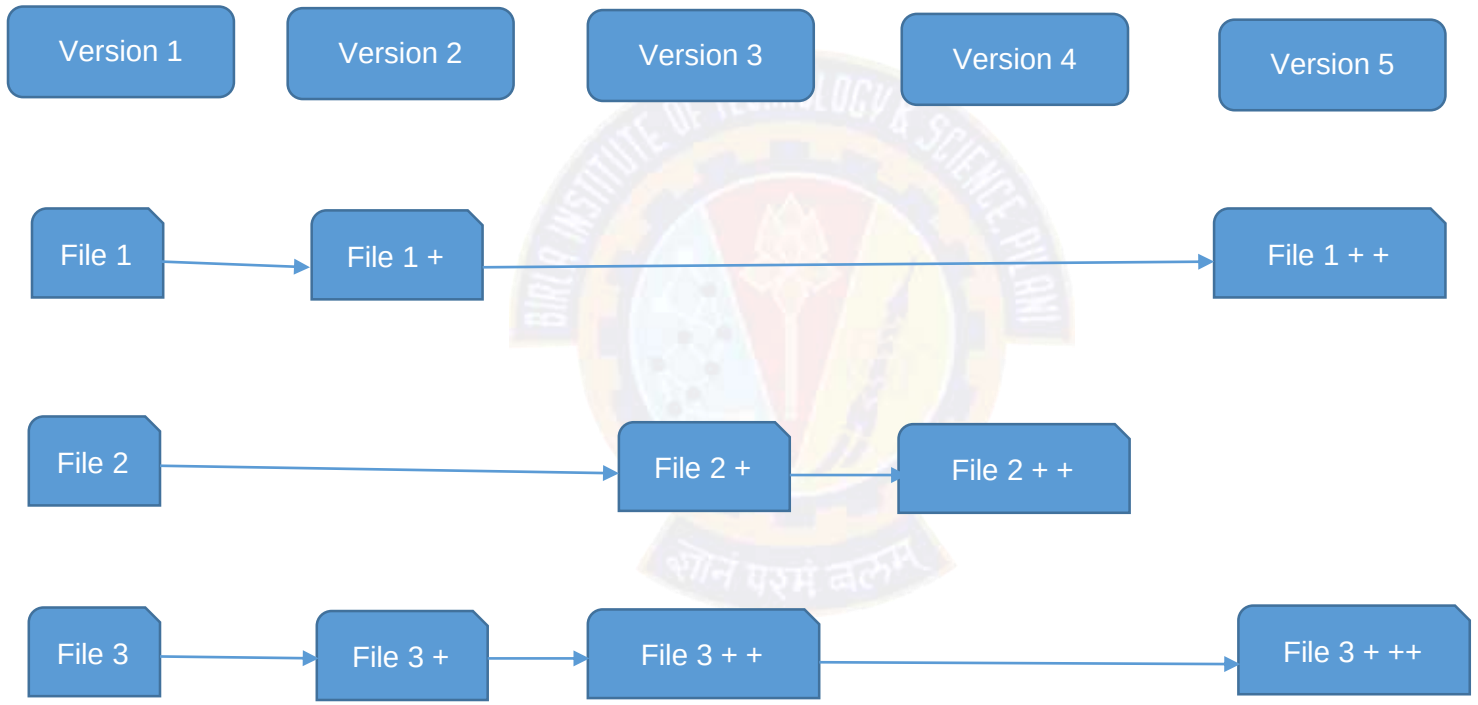
Introduction to Git

A Short History of Git

- The Linux kernel is an open source software project of fairly large scope
- For most of the lifetime of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files
- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper
- In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down
- This prompted the Linus Torvalds the creator of Linux to develop their own tool based on some of the lessons they learned while using BitKeeper
- Since its birth in 2005, Git has evolved and matured to be easy to use
- The qualities considered for the development of Git was:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

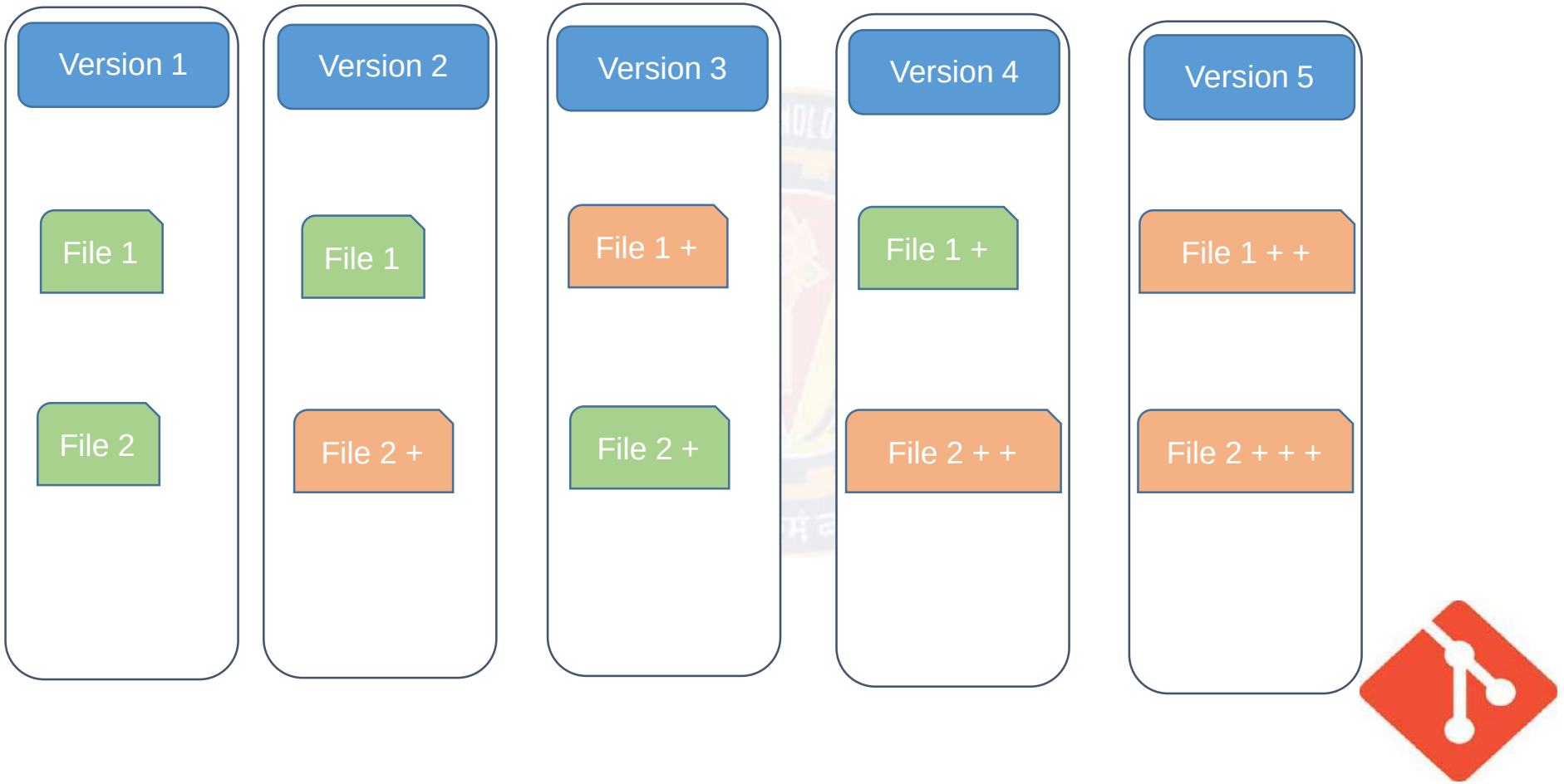
Introduction to Git

What is Different in Git?



Introduction to Git

This is How Git treats:



Introduction to Git

How it works?

- Nearly Every Operation Is Local:
  - Most operations in Git need only local files and resources to operate
  - you have the entire history of the project right there on your local disk, most operations seem almost instantaneous
- Git Has Integrity
  - Everything in Git is check-summed before it is stored and is then referred to by that checksum
  - You can’t lose information in transit or get file corruption without Git being able to detect it.
- Git Generally Only Adds Data
  - When you do actions in Git, nearly all of them only add data to the Git database
  - After you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository

States of Git

The three states of Git

- Git has three main states that your files can reside in:
- Committed : Committed means that the data is safely stored in your local database
- Modified : Modified means that you have changed the file but have not committed it to your database yet
- Staged : Staged means that you have marked a modified file in its current version to go into your next commit snapshot

A Git Project

A Git Project will have

- The Git directory:
  - It is where Git stores the metadata and object database for your project
  - It is copied when you clone a repository from another computer
- The working tree:
  - It is a single checkout of one version of the project
  - These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify
- The staging area:
  - It is a file, generally contained in your Git directory
  - It stores information about what will go into your next commit
  - Its technical name in Git parlance

Git workflow

The basic Git workflow

- You modify files in your working tree
- You selectively stage just those changes you want to be part of your next commit; which adds only those changes to the staging area
- You do a commit; which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Git

- Git basic commands

Git basic commands

Getting a Git Repository or Creating a Git Repository

- You typically obtain a Git repository in one of two ways
  - You can take a local directory that is currently not under version control, and turn it into a Git repository
  - OR
  - You can clone an existing Git repository from elsewhere
- Initializing a Repository in an Existing Directory
  - Go to the Project Directory which is not under version control
  - And type

#git init

Git basic commands

Cloning an Existing Repository

- If you want to get a copy of an existing Git repository
- You should notice that the command is "clone" and not "checkout“
- Instead of getting just a working copy, Git receives a full copy of nearly all data that the server has
- Every version of every file for the history of the project is pulled down by default when you run git clone
- Benefit: if your server disk gets corrupted, you can often use nearly any of the clones on any client to set the server back to the state it was in when it was cloned

#git clone <url>

Git basic commands

Create a working copy

- Create a working copy of a local repository

```
#git clone /path_to_repository
```

- When using a remote server:

```
#git clone username@host:/path_to_repository
```

Git basic commands

Getting a Git Repository Status:

- The Git command to know exactly what you changed, not just which files were changed
- git diff shows you the exact lines added and removed — the patch, as it were
- That command compares what is in your working directory with what is in your staging area

#git diff

Git basic commands

Add a file to Git Repository

- The Git command to add a file to the git repository is git add
- This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit
- Before using commit command you must use the add any new or modified files to the index

#git add <file name>

Git basic commands

Committing your changes

- Once the staging area is set up, you can commit your changes
- Remember that anything that is still unstaged i.e. any files you have created or modified that you haven't run git add on since you edited them, those won't go into this commit
- The default commit message contains the latest output of the git status

#git commit

#git commit –m "Commit Message"

Git basic commands

Removing Files from a Git Repository

- To remove a file from Git, you have to remove it from your staging area and then commit
- The git rm command does that

#git rm <filename>

Git basic commands

Moving Files or Renaming Files in a Git Repository

- Unlike many other VCS systems, Git doesn't explicitly track file movement
- If you rename a file in Git, no metadata is stored in Git that tells it you renamed the file
- To rename a file in Git, you can run

#git mv <file_from> <file_to>

Git basic commands

Viewing the Commit History of Git Repository

- After you have created several commits, or if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened
- The most basic and powerful tool to do this is the
- By default, with no arguments, git log lists the commits made in that repository in reverse chronological order i.e. the most recent commits show up first
- To see only the commits of a certain author

#git log

#git log --author=john

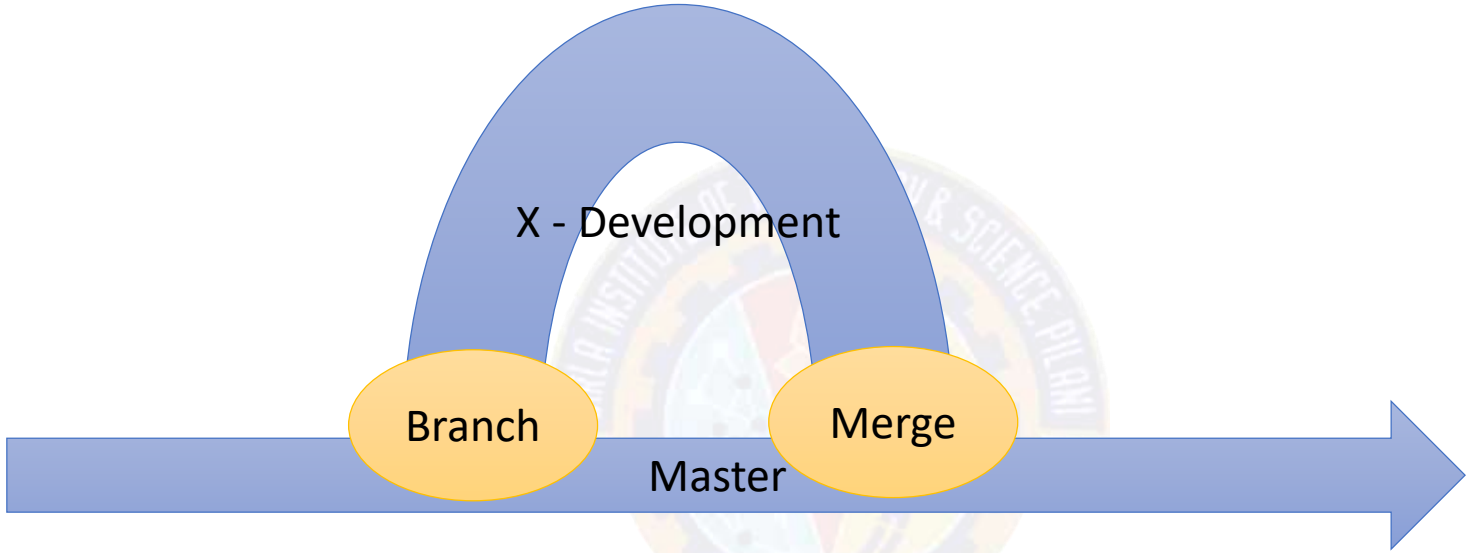
Git basic commands

Git Branching

- As we saw nearly every Version Control System has Branching Support
- Branching means you diverge from the main line of development and continue to do work without messing with that main line
- Branches are used to develop features isolated from each other
- The way Git branches is incredibly lightweight, making branching operations nearly instantaneous
- Git doesn’t store data as a series of changesets or differences
- However Git instead stores data as a series of snapshots
- Which means when you make a commit, Git stores a commit object that contains a pointer to the snapshot of the content you staged

Git basic commands

Git Branching Contd..



#git branch <branch name>

Git basic commands

Git Branching Contd..

- What happens if you create a new branch?
- Creation of Branch creates a new pointer for you to move around
- How does Git know what branch you're currently on?
- Git keeps a special pointer called HEAD; HEAD acts as a pointer to the local branch you're currently on
- The git branch command only created a new branch; it didn't switch to that branch, you will be still on master branch
- To switch to an existing branch, you run

#git checkout <branch name>

Git basic commands

Git Merging

- When to Merge?
- If you have completed the hotfix development or feature development or the Individual task branched, and up on completion of that now you need to add that component to the Master
- How does it work?
- First you need to checkout the branch you wish to merge in to, for an example here we will be merging in Master Branch
- Then Run

#git merge <name of branch>

Git basic commands

Getting a Git Repository Status & check conflicts

- The main tool you use to determine which files are in which state is the git status command
- The below output means you have a clean working directory — in other words, none of your tracked files are modified

#git status

Example:

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
```

Git basic commands

Git check conflicts or status

- git status output is pretty comprehensive, it's quite self explanatory
- Lets see how the conflicts output looks like

```
$ git status
On branch master

You have unmerged path
  (Fix conflict and run "git-commit")
```

Git basic commands

Git Revert

- The git revert command can be considered an 'undo' type command
- however, it is not a traditional undo operation
- Instead of removing the commit from the project history, it figures out how to invert the changes introduced by the commit and appends a new commit with the resulting inverse content
- This prevents Git from losing history, which is important for the integrity of your revision history and for reliable collaboration
- Reverting should be used when you want to apply the inverse of a commit from your project history
- Git revert expects a commit ref is passed in and will not execute without one

Git basic commands

Git Revert:

- Similar to a merge, a revert will create a new commit
- It's important to understand that git revert undoes a single commit
- It does not "revert" back to the previous state of a project by removing all subsequent commits
- The git revert command is a forward-moving undo operation that offers a safe method of undoing changes
- Instead of deleting or orphaning commits in the commit history, a revert will create a new commit that inverses the changes specified
- Git revert is a safer alternative to git reset in regards to losing work

#git revert HEAD

Git basic commands

Git Reset

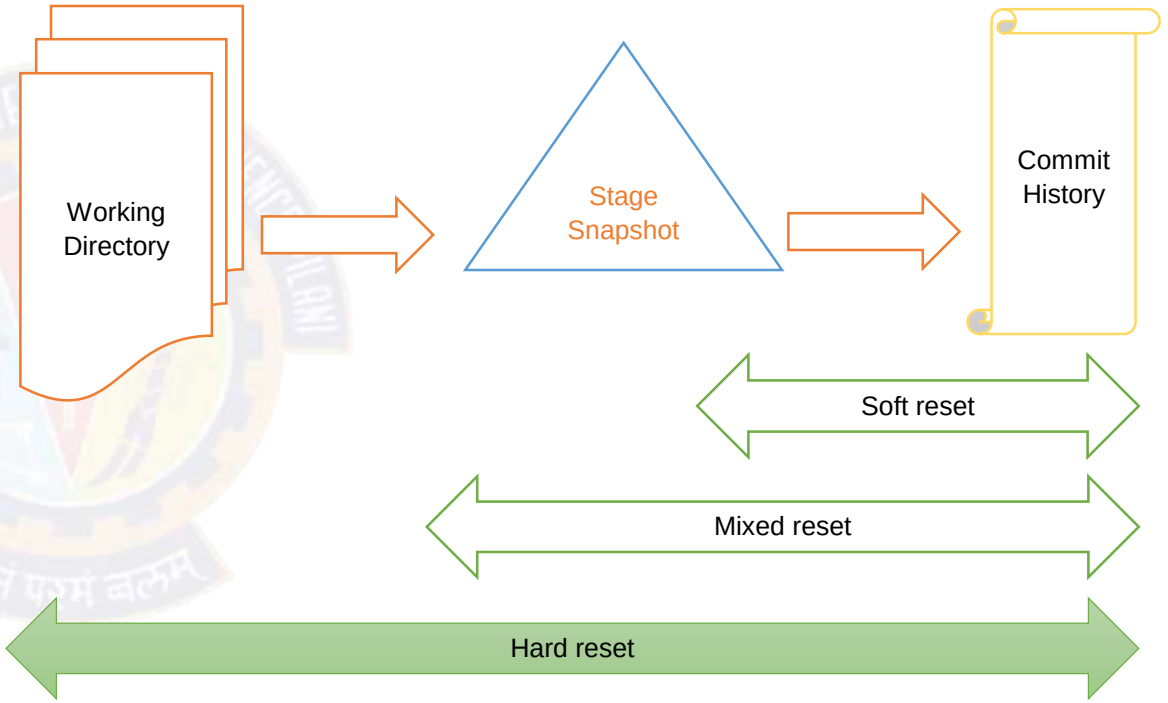
- The git reset command is a complex and versatile tool for undoing changes
- It has three primary forms of invocation
- These forms correspond to command line arguments
  - soft
  - mixed
  - hard
- The default invocation of git reset has implicit arguments of --mixed and HEAD
- This means executing git reset is equivalent to executing git reset --mixed HEAD

#git reset

Git basic commands

Git Reset: --hard

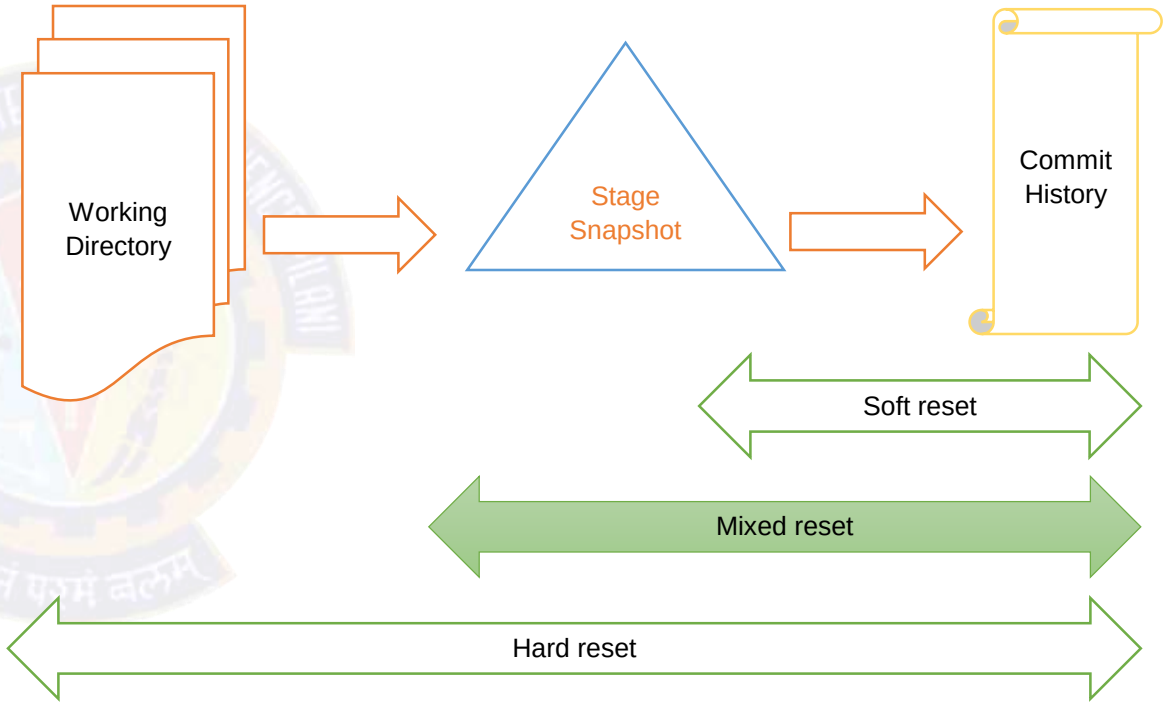
- This is the most direct, DANGEROUS, and frequently used option
- When passed --hard The Commit History ref pointers are updated to the specified commit
- Then, the Staging Index and Working Directory are reset to match that of the specified commit
- Any previously pending changes to the Staging Index and the Working Directory gets reset to match the state of the Commit Tree
- This means any pending work that was hanging out in the Staging Index and Working Directory will be lost



Git basic commands

Git Reset: --mixed

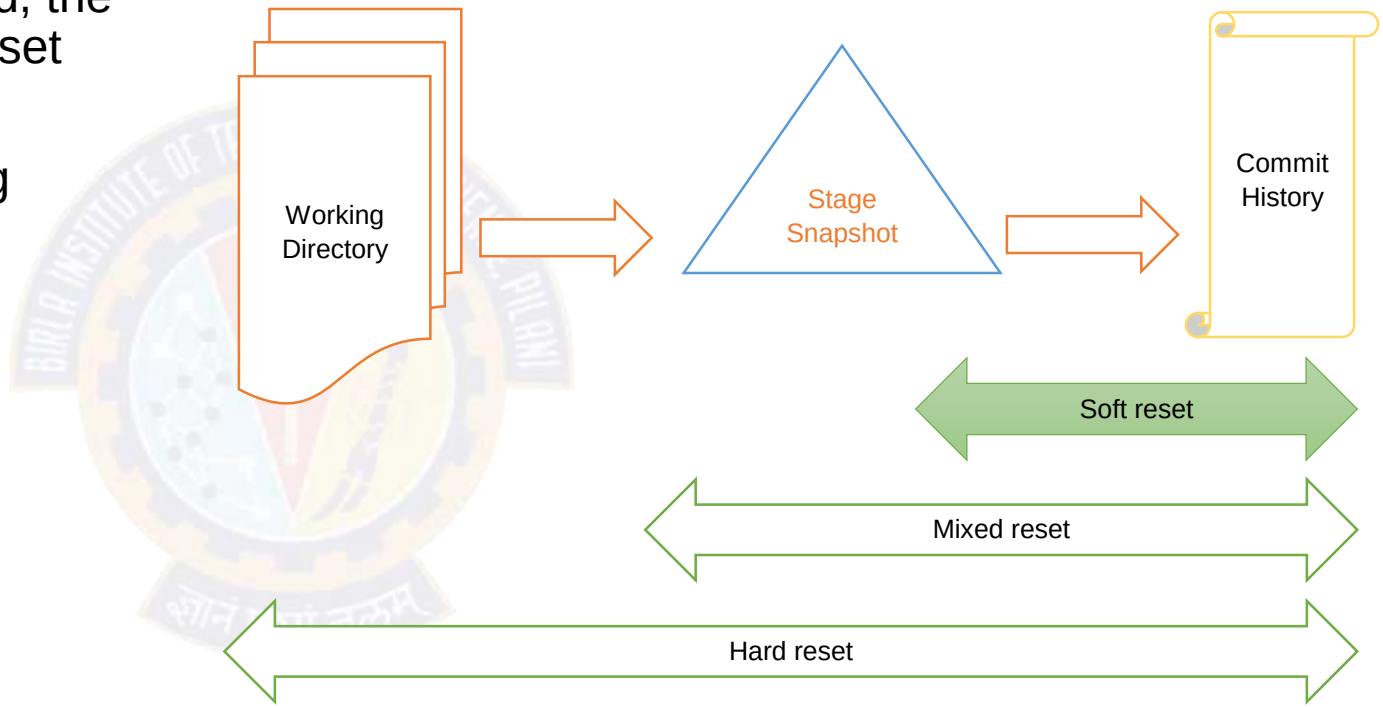
- This is the default operating mode
- The ref pointers are updated
- The Staging Index is reset to the state of the specified commit
- Any changes that have been undone from the Staging Index are moved to the Working Directory
- In other way we can say here Staging Index gets reset and the pending changes will be moved into the Working Directory



Git basic commands

Git Reset: --soft

- When the --soft argument is passed, the ref pointers are updated and the reset stops there
- The Staging Index and the Working Directory are left untouched



Git basic commands

Git Revert vs Git Reset

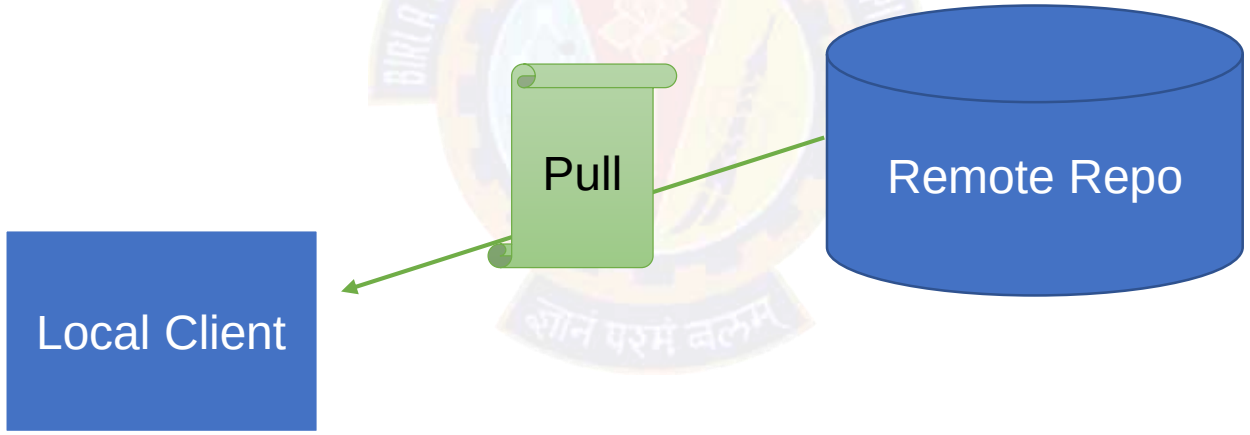
- Git Revert:
- It is a “safe” way to undo changes
- it doesn’t change the project history, which makes it a “safe” operation for commits that have already been published to a shared repository
- git revert is able to target an individual commit at an arbitrary point in the history
- git revert is a safer alternative to git reset in regards to losing work

- Git Reset:
- It is the dangerous method
- There is a real risk of losing work with git reset
- git reset can only work backward from the current commit
- git reset will never delete a commit, however, commits can become 'orphaned' which means there is no direct path from a ref to access them
- For example, if you wanted to undo an old commit with git reset, you would have to remove all of the commits that occurred after the target commit, remove it, then re-commit all of the subsequent commits

Git basic commands

Git Pull

- The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content
- The git pull command is actually a combination of two other commands, git fetch followed by git merge

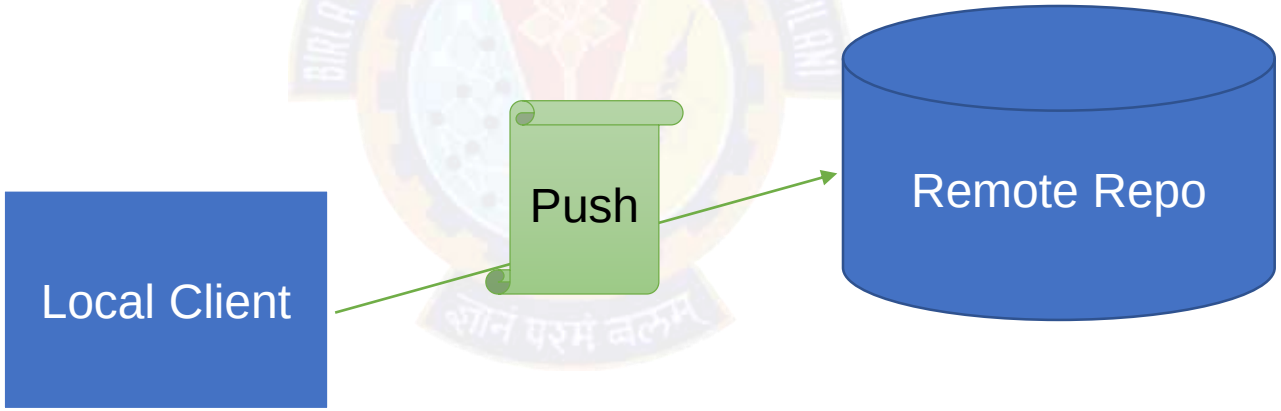


#git pull

Git basic commands

Git Push

- The git push command is used to upload local repository content to a remote repository
- Pushing is how you transfer commits from your local repository to a remote repo
- git push is one component of many used in the overall Git "syncing" process



#git push

Git basic commands

Git tagging:

- Tags are ref's that point to specific points in Git history
- Tagging is generally used to capture a point in history that is used for a marked version release (i.e. v1.1.1)
- A tag is like a branch that doesn’t change
- Unlike branches, tags, after being created, have no further history of commits
- We will have look in to tag operations:
  - Create tag
  - List tags
  - Delete tag
  - Sharing tag

Git Tagging

Git create tag

- To create a new tag execute the following command

#git tag <tag name>

- A common pattern is to use version numbers like git tag v1.4
- Git supports two different types of tags, annotated and lightweight tags
- A best practice is to consider Annotated tags as public, and Lightweight tags as private
- Annotated tags store extra meta data such as: the tagger name, email, and date so it helps for public release
- Lightweight tags are essentially 'bookmarks' to a commit, they are just a name and a pointer to a commit, so useful for creating quick links to relevant commits

Git Tagging

Git list tag and Git share tag

- Git list tag:
- To list stored tags in a repo execute the following command

```
#git tag
```

- Git share tag:
- Sharing tags is similar to pushing branches
- By default, git push will not push tags
- Tags have to be explicitly passed to git push

```
#git push origin <tag num>
```

Git Tagging

Git delete tag

- Deleting tags is a straightforward operation
- Passing the -d option and a tag identifier to git tag will delete the identified tag

```
#git tag -d v1
```

- Tagging is an additional mechanism used to create a snap shot of a Git repo
- Tagging is traditionally used to create semantic version number identifier tags that correspond to software release cycles

Git Tagging

Git gui

- Built in Git gui

```
#gitk
```

- Use colorful git output

```
#git config color.ui true
```

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

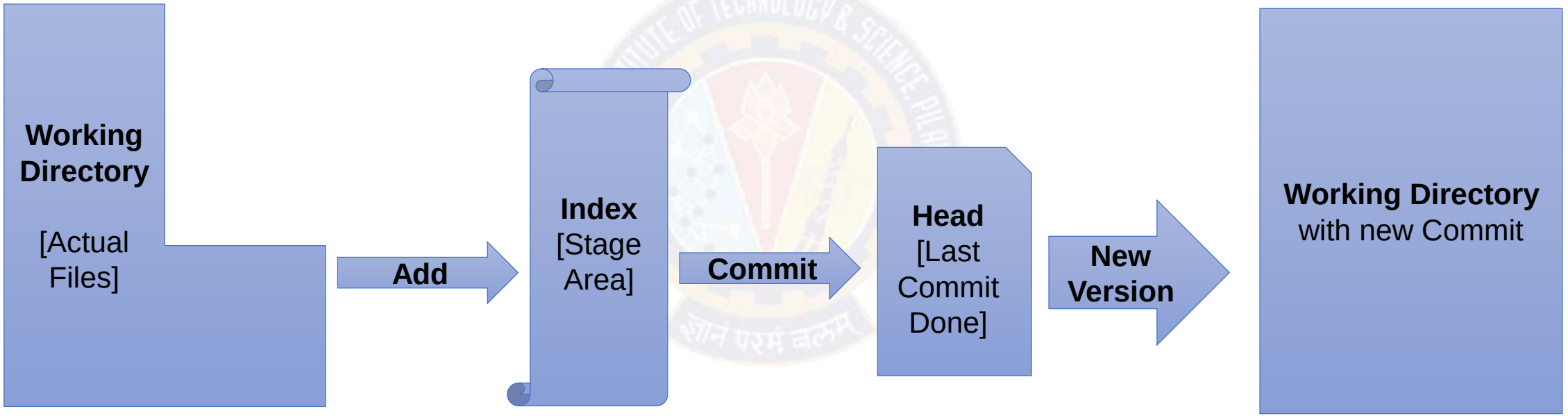
Git workflow

- Git tree Structure
- What is Git workflow?
- Git Centralized workflow

Git workflow

Git tree Structure

- Git local repository consists of three “trees”



Git workflow

Considerations

- To evaluate a Workflow of Git for a team you must consider:
  - Teams culture
  - Let team know the workflow is to enhance effectiveness of team and not to burden
  - Let team know the workflow is not to limit the productivity

- Selecting a Workflow? Basic conditions for decision
  - Will selected workflow scale with team size?
  - Is it easy to handle mistakes and errors with this workflow?
  - Does this workflow impose any new unnecessary cognitive overhead to the team?

Git workflow

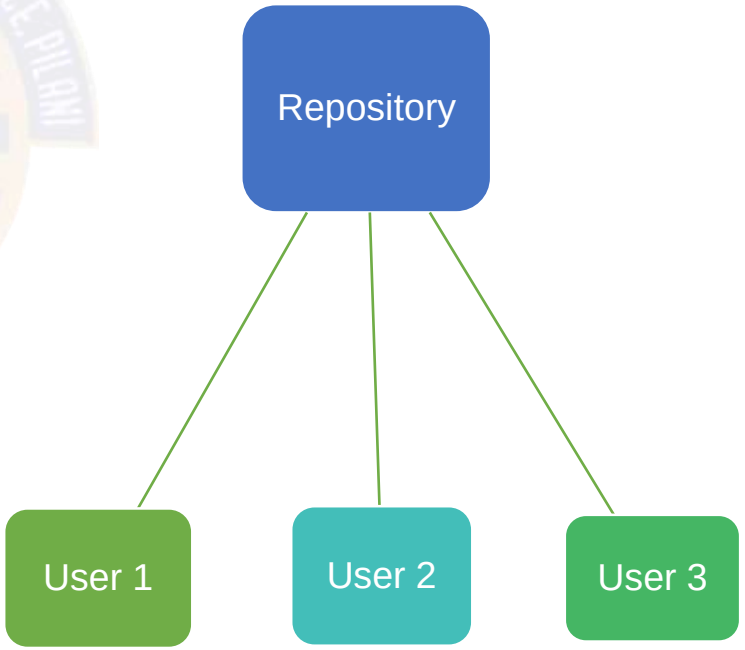
What is Git workflow?

- A Git Workflow is a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner
- Git workflows encourage users to leverage Git effectively and consistently
- Git offers a lot of flexibility in how users manage changes
- There are several publicized Git workflows that may be a good fit for your team
- We will discuss:
  - Centralized Workflow
  - Feature Branching Workflow

Git Centralized workflow

Introduction

- It is a great Git workflow for teams transitioning from SVN (Subversion)
- It uses a central repository to serve as the single point-of-entry for all changes to the project
- Here there will be only one branch i.e. Master; there will not be any other branch



Git Centralized workflow

Benefits

- Centralized workflow gives every developer their own local copy of the entire project
- This isolated environment lets each developer work independently of all other changes to a project
- Centralized workflow gives you access to Git's robust branching and merging model
- Git branches are designed to be a fail-safe mechanism for integrating code and sharing changes between repositories
- The Centralized Workflow is similar to other workflows in its utilization of a remote server-side hosted repository that developers push and pull form
- A Centralized Workflow is generally better suited for teams migrating from SVN to Git and smaller size teams

Git Centralized workflow

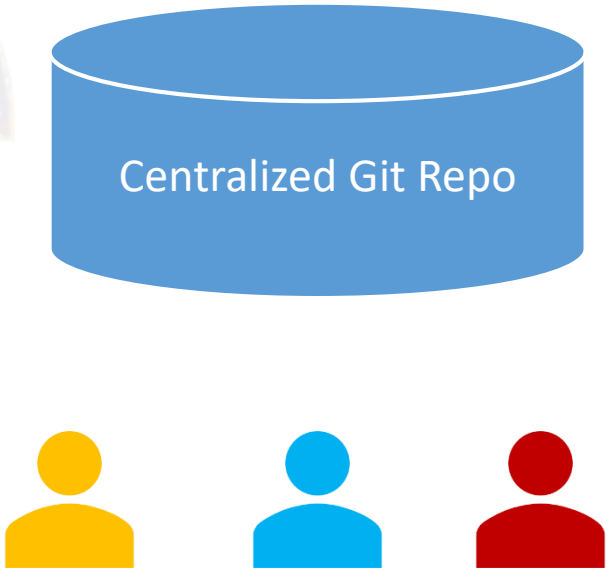
Git Centralized workflow: How it works?

- Developer Starts Cloning the central repository
- In local copy they edit files and commit changes as they would with SVN
- However, these new commits are stored locally - they’re completely isolated from the central repository
- This lets developers defer synchronizing upstream until they’re at a state where they would like to commit the code to the Master
- To publish changes to the official project, developers "push" their local master branch to the central repository

Git Centralized workflow

Git Centralized workflow: Example

- Lets say the team of three people named Orange, Blue and Red are working on a Centralized Git Repository
- They will be collaborating with Centralized Workflow



Git Centralized workflow

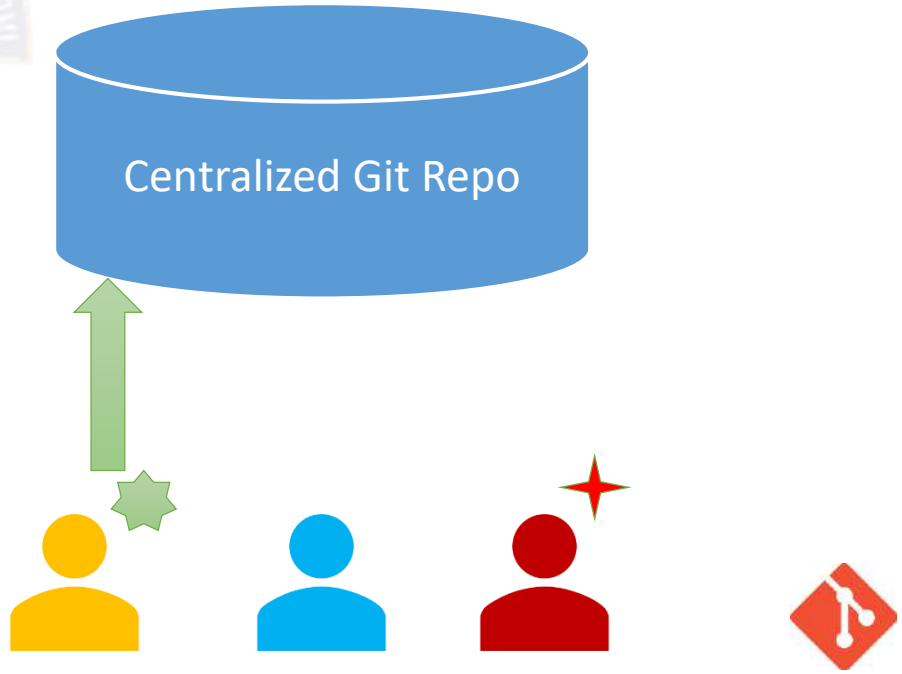
Git Centralized workflow: Example Contd..

- User Orange is working on a feature development in his local repository
- At the Same time User Red is also working on a separate feature in his local repository

Git Centralized workflow

Git Centralized workflow: Example Contd..

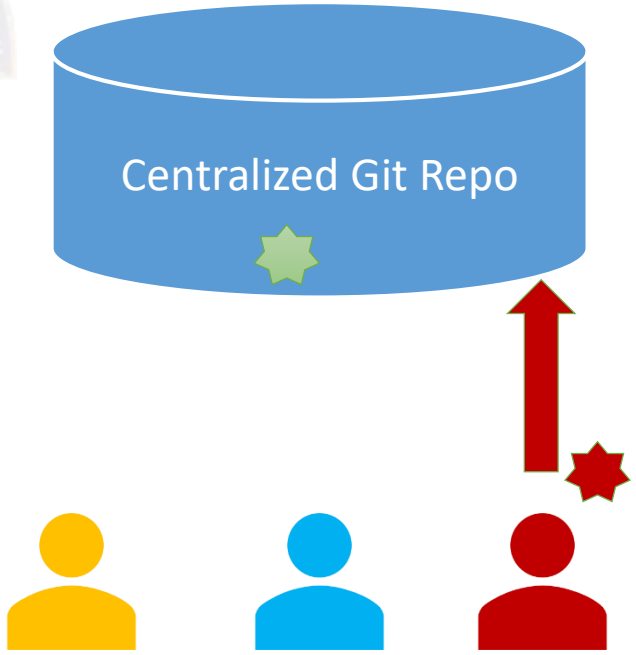
- User Orange has completed his feature development
- And user Orange publishes his feature
- I.e. User Orange should commit his feature development changes to Central repository
- So now these updated commits are available to all users



Git Centralized workflow

Git Centralized workflow: Example Contd..

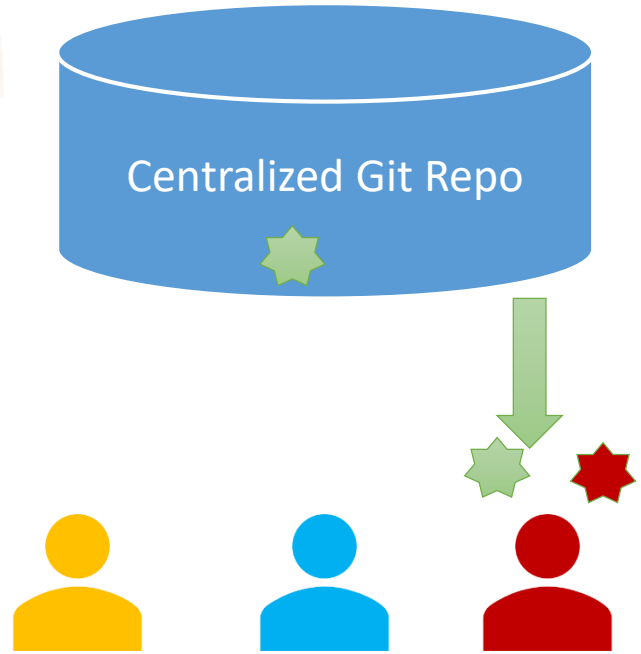
- After the commit of Orange user
- User Red complete his feature development
- And now User Red would like to commit his changes to the Central Repository
- And It Fails



Git Centralized workflow

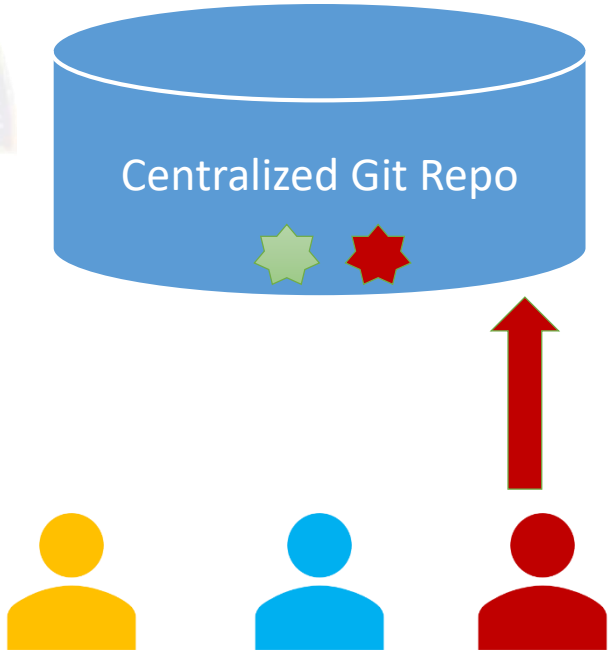
Git Centralized workflow: Example Contd..

- Here user Red need to pull the recent changes done to repository
- Incorporate his changes with recent changes done by User Orange
- I.e. Kind of rebase activity and resolve the conflicts if any



Git Centralized workflow

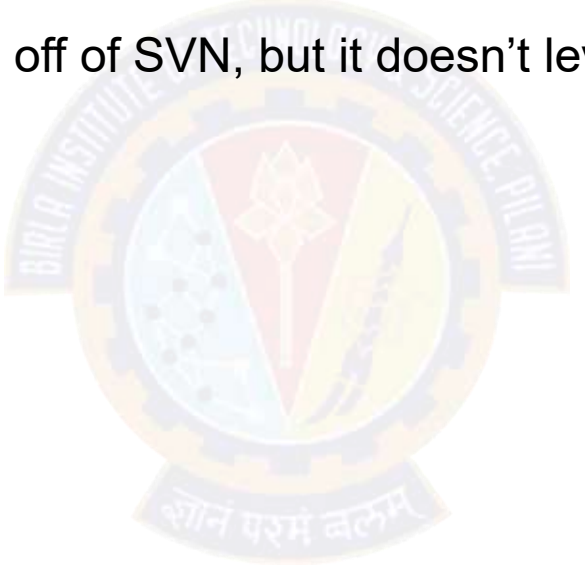
- Now user Red can commit his changes in local repository
- And Publish his feature to the central repository
- Its Successful



Git Centralized workflow

Summary

- The Centralized Workflow is great for small teams
- The conflict resolution process detailed above can form a bottleneck as your team scales in size
- This is great for transitioning teams off of SVN, but it doesn’t leverage the distributed nature of Git



Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Feature workflow

- Introduction
- How it works?

Git Feature Branch Workflow

Introduction

- The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the master branch
- This makes it easy for multiple developers to work on a particular feature without disturbing the main codebase
- It also means the master branch will never contain broken code, which is a huge advantage for continuous integration environments
- Git Feature Branch Workflow is branching model focused, meaning that it is a guiding framework for managing and creating branches

Git Feature Branch Workflow

Git Feature Branch workflow: How it works?

- The Feature Branch Workflow assumes a central repository, and master represents the official project history
- Instead of committing directly on their local master branch, developers create a new branch every time they start work on a new feature
- Feature branches should have descriptive names like web page development #23
- Git makes no technical distinction between the master branch and feature branches, so developers can edit, stage, and commit changes to a feature branch
- Feature branches can be pushed to the central repository
- This makes it possible to share a feature with other developers without touching any official code

Git Feature Branch Workflow

Git Feature Branch workflow: Example

- Lets say the team of three people named Orange, Blue and Red are working on a Centralized Git Repository
- They will be collaborating with Feature Branch Workflow

Git Feature Branch Workflow

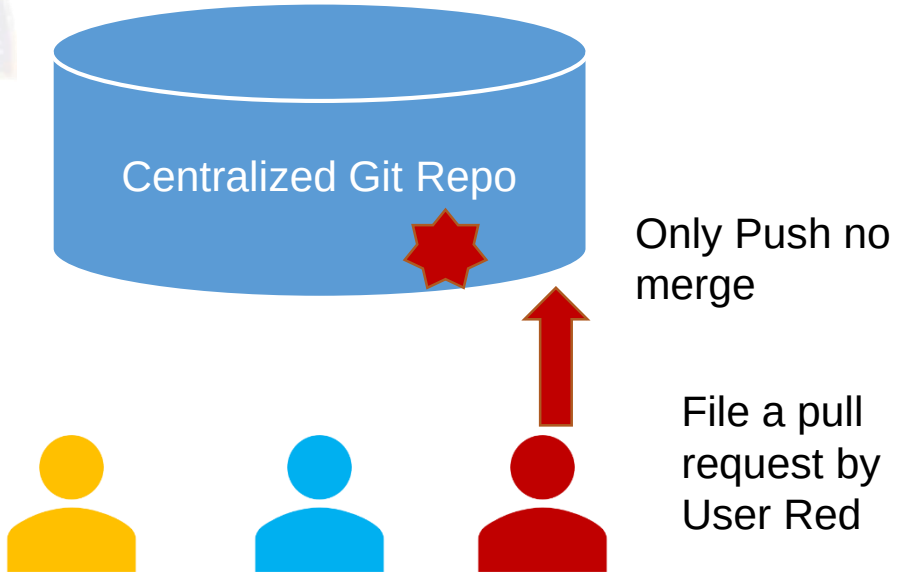
Git Feature Branch workflow: Example Contd..

- Before user Red starts working on a new feature, he request his own new branch
- User Red Starts development of his new feature
- At the end of the day user Red complete 80% of his feature development
- Before leaving for the day user Red push the commit to the Central Repository
- So the new feature work will be available for all team member and also to the member who is collaborating with user Red for this feature development

Git Feature Branch Workflow

Git Feature Branch workflow: Example Contd..

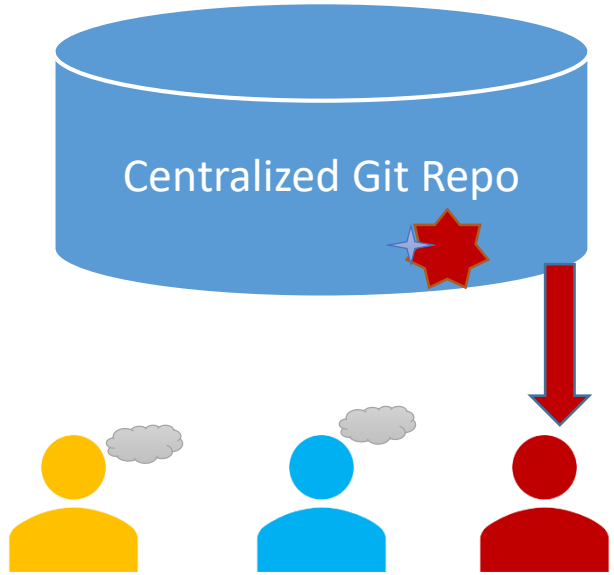
- Next day User Red complete his development
- Now before merging the new Feature to master, User Red would like to let all other team member know he is done with his work by filing a pull request
- Before filing pull request User Red need to make sure the Branch is available on Central Repository



Git Feature Branch Workflow

Git Feature Branch workflow: Example Contd..

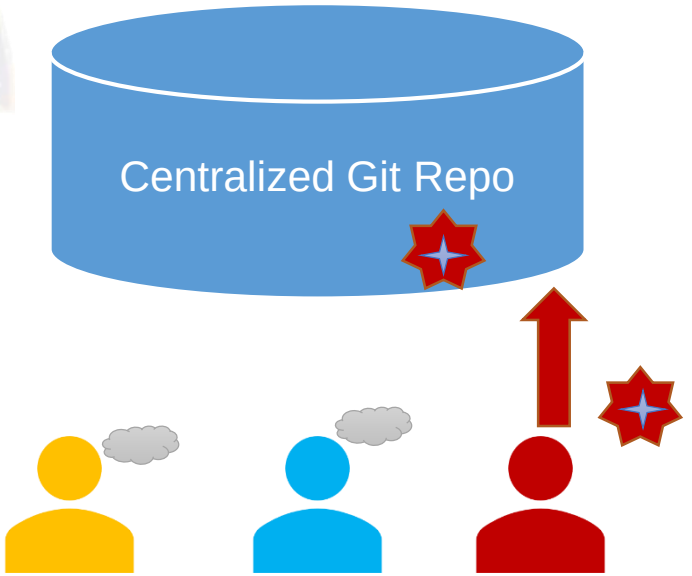
- Once User Red file a pull request Every user gets a notification that User Red is done with his feature development
- User Blue Checks the notification and visit the feature developed by User Red
- User Blue adds some more in to the feature pushed by user Red
- User Blue and User Red has discussed the addition and synced



Git Feature Branch Workflow

Git Feature Branch workflow: Example Contd..

- User Red combined the addition by user Blue and creates final Push
- And finally user Red merges feature with Master
- Even here if wanted User Blue could have copied feature developed by User Red to his local repo and do the changes



Git Feature Branch Workflow

Summary

- This workflow helps organize and track branches that are focused on business domain feature sets
- Feature Branch Workflow are focused on branching patterns
- Feature Branch Workflow promotes collaboration with team members through pull requests and merge reviews
- A feature branching model is a great tool to promote collaboration within a team environment

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

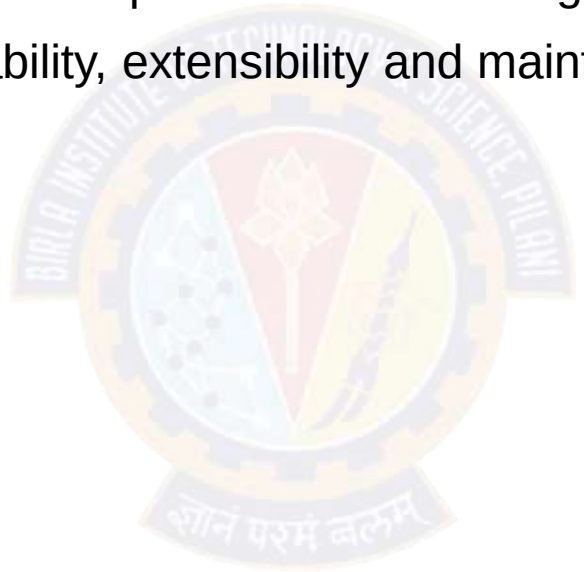
Best Practices of Clean Code

- Clean code
- General Rules

Clean code

Condition for a Clean Code

- If it can be understood easily
- It can be read and enhanced by a developer other than its original author
- If code follows readability, changeability, extensibility and maintainability



Best Practices of Clean Code

General Rules

- Follow standard conventions
- Keep it simple, Simpler is always better, Reduce complexity as much as possible
- Be consistent i.e. If you do something a certain way, do all similar things in the same way
- Use self explanatory variables
- Choose descriptive and unambiguous names
- Keep functions small

Best Practices of Clean Code

General Rules Contd..

- Each Function should do one thing
- Use function names descriptive
- Prefer to have less arguments
- Always try to explain yourself in code; put proper comments
- Declare variables close to their usage
- Keep lines short
- Code should be readable
- Code should be fast
- Code should be generic
- Code should be reusable

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Dependencies

- What is component
- Best Practices for component based design
- Overview of Dependencies

Component

What is component?

- This is a overloaded term in software
- Components, mean a reasonably large-scale code structure within an application, with a well-defined API, that could potentially be swapped out for another implementation
- A component-based software system is distinguished by the fact that the codebase is divided into discrete pieces that provide behavior through well-defined, limited interactions with other components

Component

What is component? Contd..

- Components also refer as “modules”
- In Windows, a component is normally packaged as a DLL
- In UNIX, it may be packaged as an SO (Shared object) file
- In the Java world, it is probably a JAR file
- Benefits of component-based design
- encouraging reuse and good architectural properties such as loose coupling
- It is one of the most efficient ways for large teams of developers to collaborate
- Lets understand how to create and manage build systems for component-based applications

Component

Challenges of component based design

- In large system, components form a series of dependencies, which in turn depend on external libraries
- Each component may have several release branches
- So, we have heard of projects where it takes months
  - finding good versions of each of these components
  - that can be assembled into a system which even compiles is an extremely difficult process
- Which results into ages to just release the system
- To overcome this we need to follow the best practices

Component

Keeping Your Application Releasable

- During development, teams continue an implementation of features, and sometimes need to make major architectural changes
- In such cases the application cant be released, although it will still pass the commit stage of continuous integration
- Usually, before release, teams will stop developing new functionality and focuses only on bug fixing
- When the application is released, a release branch is created in version control, and new development begins again
- However, this process generally results in weeks or months between releases
- The aim of continuous delivery is for the application to always be in a releasable state
- How can we achieve this?

- One approach is to create branches in version control that are merged when work is complete, so that mainline is always releasable (feature workflow)

Component

Strategies to keep your application releasable during change

- Hide new functionality until it is finished
  - problem with continuous deployment of applications is that a feature, or a set of features, can take a long time to develop which might prevent it being released
  - One solution is to put in the new features, but make them inaccessible to users

- Make all changes incrementally as a series of small changes, each of which is releasable
  - Here, analysis plays an important to make large changes as a series of small changes

- Use branch by abstraction to make large-scale changes to the codebase
  - to make a large-scale change to an application
  - Instead of branching, an abstraction layer is created over the piece to be changed
  - A new implementation is then created in parallel with the existing implementation
  - Once it is complete, the original implementation and (optionally) the abstraction layer are removed

Dependencies

Overview of Dependencies

- A dependency occurs whenever one piece of software depends upon another in order to build or run
- In most trivial of applications, there will be some dependencies
- Most software applications have, a dependency on their host operating environment
- Like Java applications depend on the JVM which provides an implementation of the Java SE API, .NET applications on the CLR, Rails applications on Ruby and the Rails framework, C applications on the C standard library etc.

Dependencies

Types of Dependencies

- Build time dependencies and Run time dependencies
- Libraries and components

Libraries	Components
• Software packages that your team does not control, other than choosing which to use • Libraries are Usually updated rarely	• Pieces of software that your application depends upon, but which are also developed by your team, or other teams in your organization • Components are usually updated frequently

- This distinction is important because when designing a build process, there are more things to consider when dealing with components than libraries.
- For example, do you compile your entire application in a single step, or compile each component independently when it changes? How do you manage dependencies between components, avoiding circular dependencies?

Dependencies

Build time dependencies and Run time dependencies

Build-time dependencies must be present when your application is compiled and linked (if necessary)

Runtime dependencies must be present when the application runs, performing its usual function

- Ex. In C and C++, your build-time dependencies are simply header files, while at run time you require a binary to be present in the form of a dynamic-link library (DLL) or shared library (SO)
- Managing dependency can be difficult

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Managing Dependency Problems

- Common dependency problem with libraries at run time
- Managing Libraries
- Managing Component
- Managing Dependency Graphs

Dependency Problems

Common dependency problem with libraries at run time

- Dependency Hell also refer as DLL Hell
- Occurs when an application depends upon one particular version of something, but is deployed with a different version, or with nothing at all
- DLL hell was common problem in earlier versions of Windows,
  - as all shared libraries (DLLs), stored in a system directory (windows\system32) without any versioning; new versions would simply overwrite old ones
  - In versions of Windows prior to XP the COM class table was a singleton, so applications that required a particular COM object would be given whichever version had been loaded first

Dependency Problems

Common dependency problem with libraries at run time contd..

- The introduction of the .NET framework resolves the DLL hell problem by introducing the concept of assemblies that allow different versions of the same library to be distinguished
- Linux avoids dependency hell by using a simple naming convention: It appends an integer to every .so file in the global library directory (/usr/lib), and uses a soft link to determine the canonical system-wide version

Managing Libraries

Implementing Version Control

- One is to check them into version control
  - Is the simplest solution, and will work fine for small projects
  - a lib directory is created in your project's root to put libraries and by adding three further subdirectories: build, test, and run—for build-time, test-time, and runtime dependencies
  - using a naming convention for libraries that includes their version number; you know exactly which versions you're using
- Benefit:
  - Everything you need to build your application is in version control
  - Once you have a local check-out of the project repository, you know you can repeatably build the same packages that everybody else has
- Problems:
  - your checked-in library repository may become large and it may become hard to know which of these libraries are still being used by your application
  - Another problem crops up if your project must run with other projects on the same platform
  - Manually managing transitive dependencies across projects rapidly becomes painful

Managing Libraries

Automated

- Another solution is to declare libraries and use a tool like Maven or Ivy to download libraries from Internet repositories or (preferably) your organization's own artifact repository [Automated]

Managing Component

Overview

- Almost all modern software systems consist of a collection of components
- These components may be DLLs, JAR files, OSGi bundles, Perl modules, or something else
- Components have a relatively long history in the software industry

Managing Component

Components make development processes efficient

- They divide the problem into smaller and more expressive chunks
- Components often represent differences in the rates of change of different parts of the system, and have different lifecycles
- They encourage us to design and maintain software with clear description of responsibilities, which in turn limits the impact of change, and makes understanding and changing the codebase easier
- They can provide us with additional degrees of freedom in optimizing our build and deployment process

Managing Component

Components to be separated from Codebase

- Part of your codebase needs to be deployed independently (for example, a server or a rich client)
- You want to turn a monolithic codebase into a core and a set of plugins, perhaps to replace some part of your system with an alternative implementation, or to provide user extensibility
- The component provides an interface to another system (for example a framework or a service which provides an API)
- It takes too long to compile and link the code
- It takes too long to open the project in the development environment
- Your codebase is too large to be worked on by a single team

Managing Component

Pipelining Components

- Split your system into several different pipelines
- Parts of your application that have a different lifecycle (perhaps you build your own version of an OS kernel as part of your application, but you only need to do this once every few weeks)
- Functionally separate areas of your application that are worked on by different (perhaps distributed) teams may have components specific to those teams
- Components that use different technologies or build processes
- Shared components that are used by several other projects
- Components that are relatively stable and do not change frequently
- It takes too long to build your application, and creating builds for each component will be faster

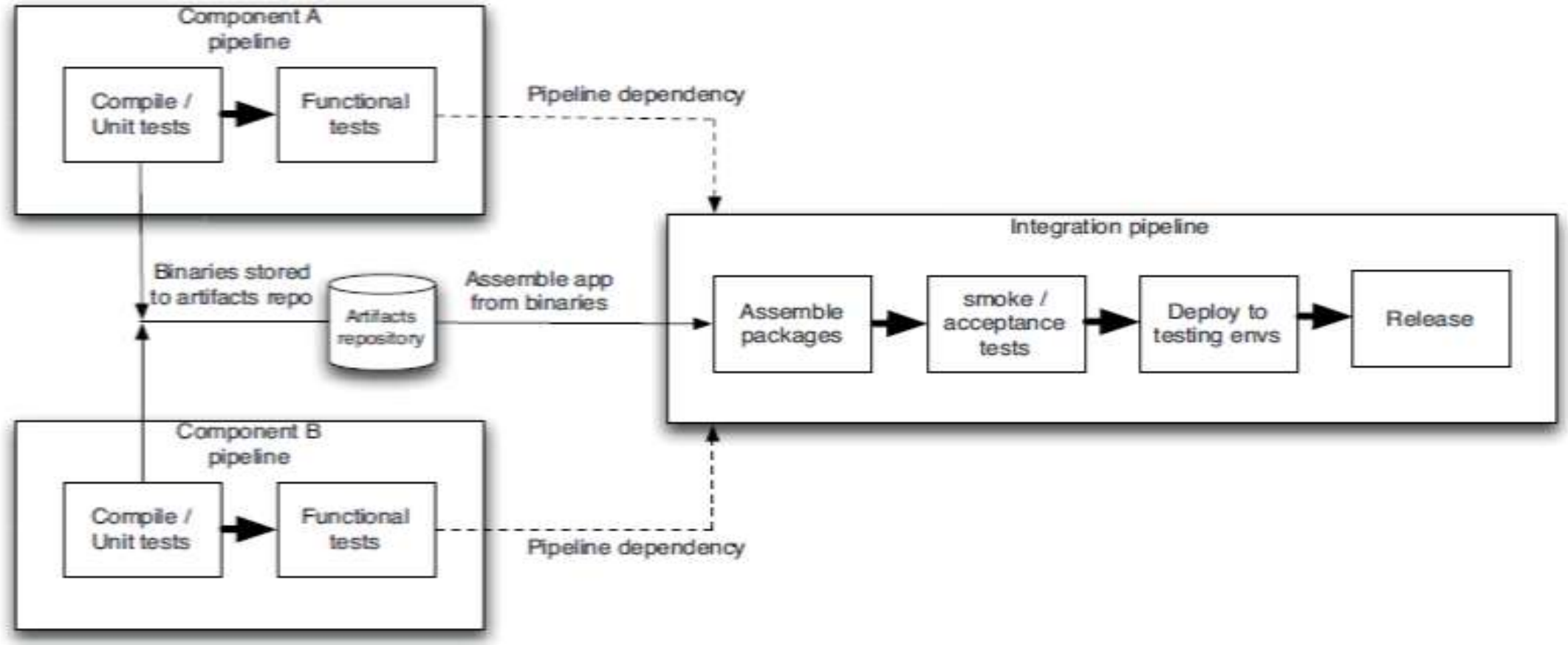
Managing Component

Pipelining Components

- The build for each component or set of components should have its own pipeline to prove that it is fit for release
- This pipeline will perform the following steps
  - Compile the code, if necessary
  - Assemble one or more binaries that are capable of deployment to any environment
  - Run unit tests
  - Run acceptance tests
  - Support manual testing, where appropriate

Managing Component

Integration Pipeline



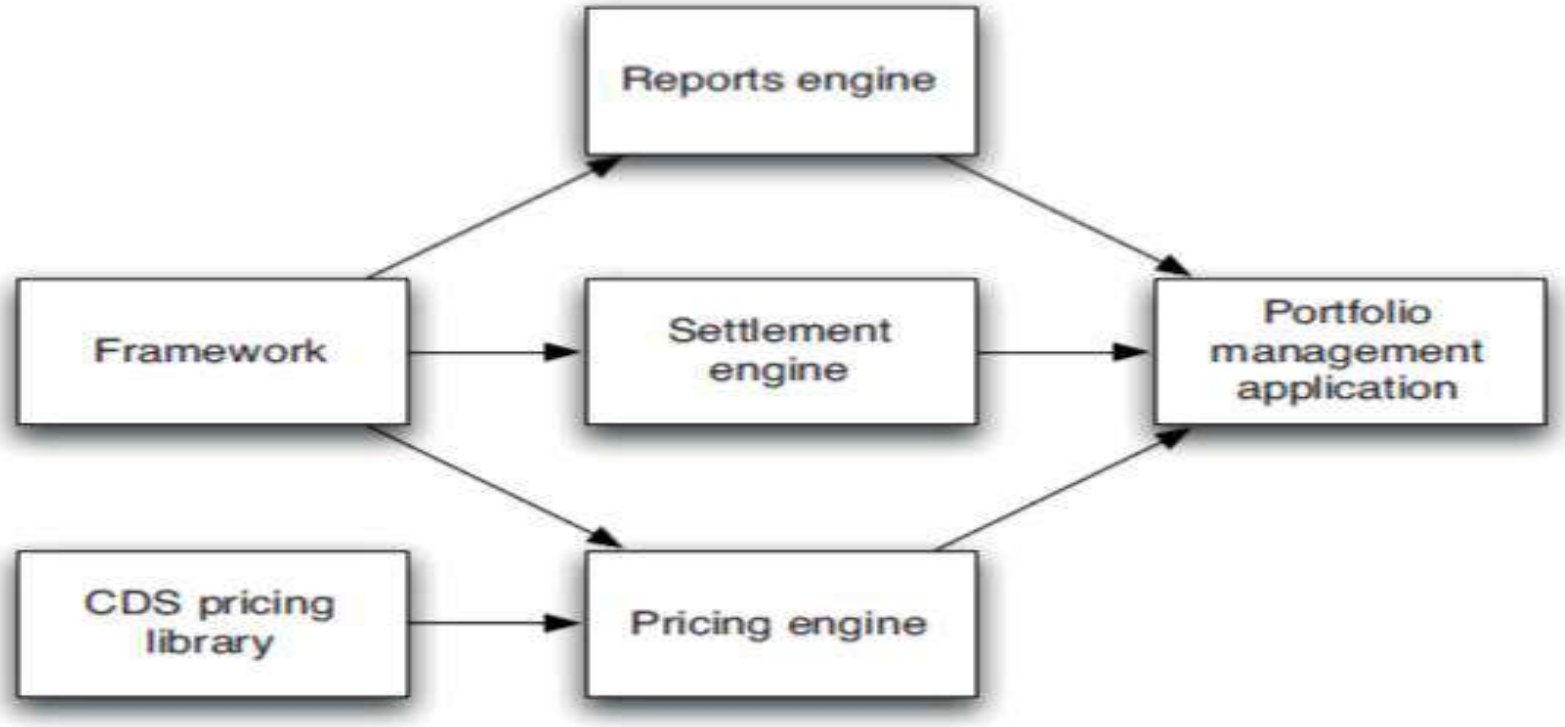
Managing Component

Managing Dependency Graphs

- Having set of components, each with their own pipeline, feeding into an integration pipeline which assembles the application and runs automated and manual tests on the final application
- However, things are often not quite this simple:
- Components can have dependencies on other components, including third-party libraries
- If you draw a diagram of the dependencies between components, it should be a directed acyclic graph (DAG)
- If this is not the case (and in particular, if your graph has cycles) you have a pathological dependency problem

Managing Component

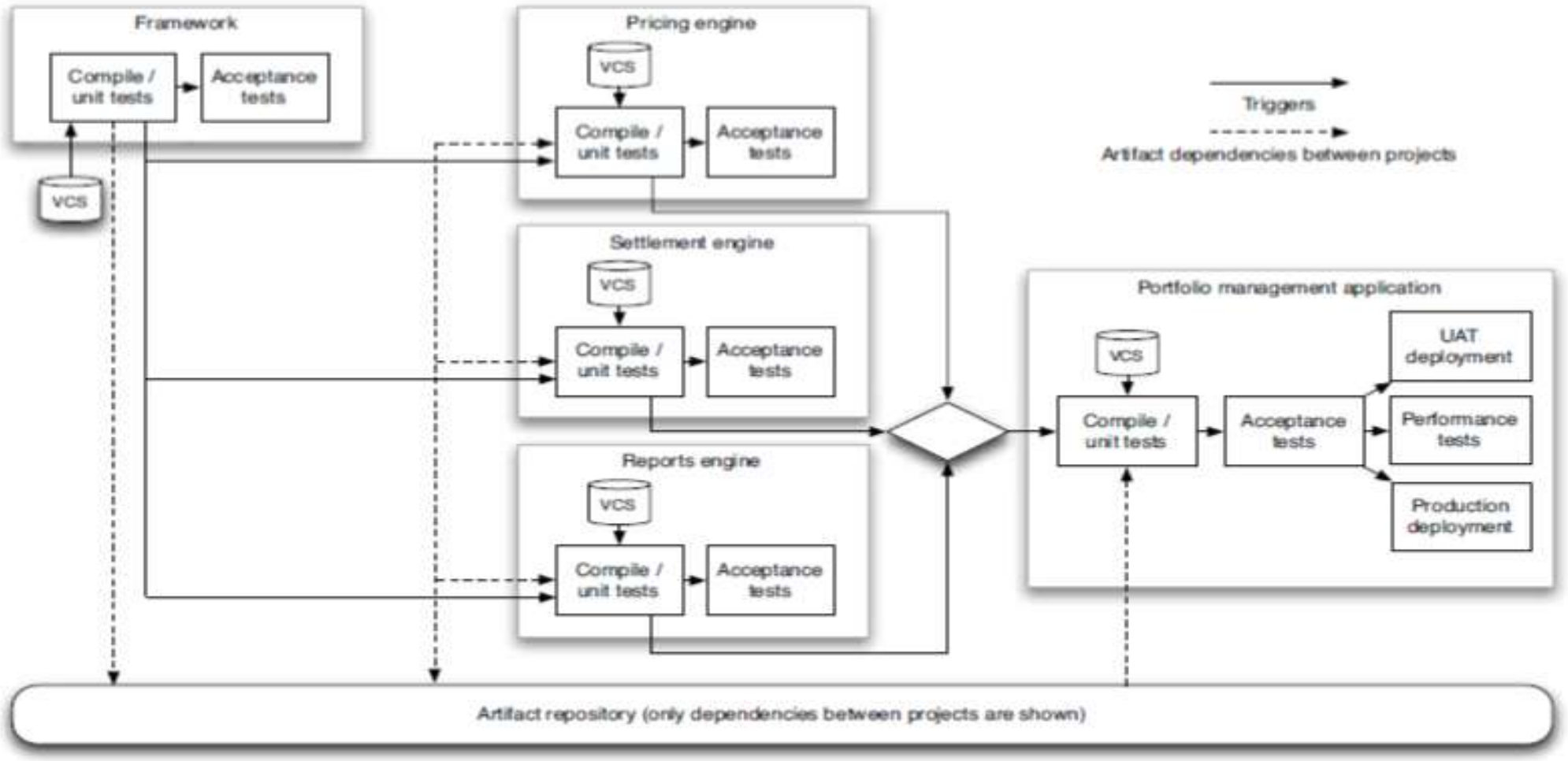
Building Dependency Graphs



credit default swap (CDS) library that is provided by third party

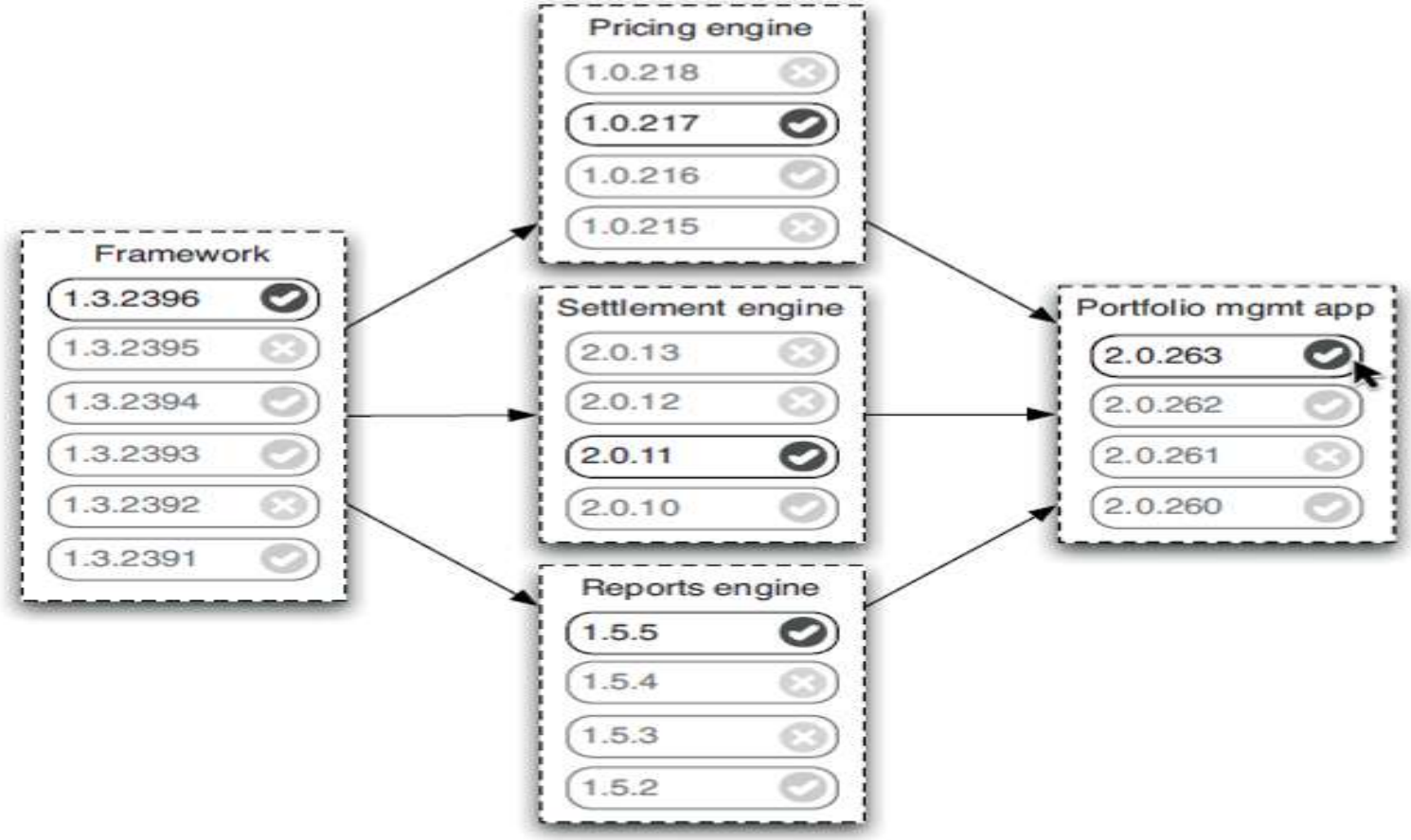
Managing Component

Pipelining Dependency Graphs



Managing Component

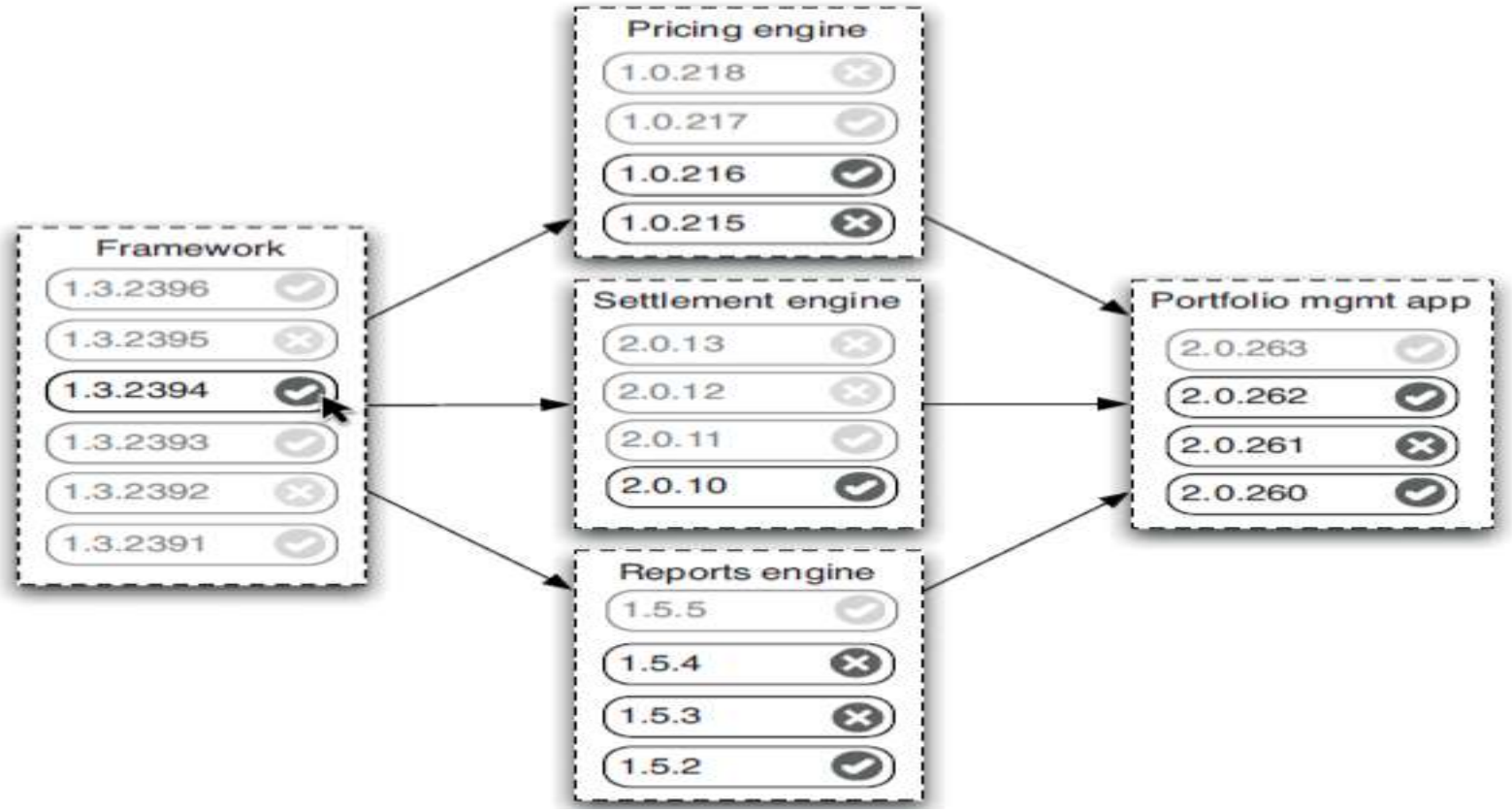
Pipelining Dependency Graphs Contd..



Visualizing upstream dependencies

Managing Component

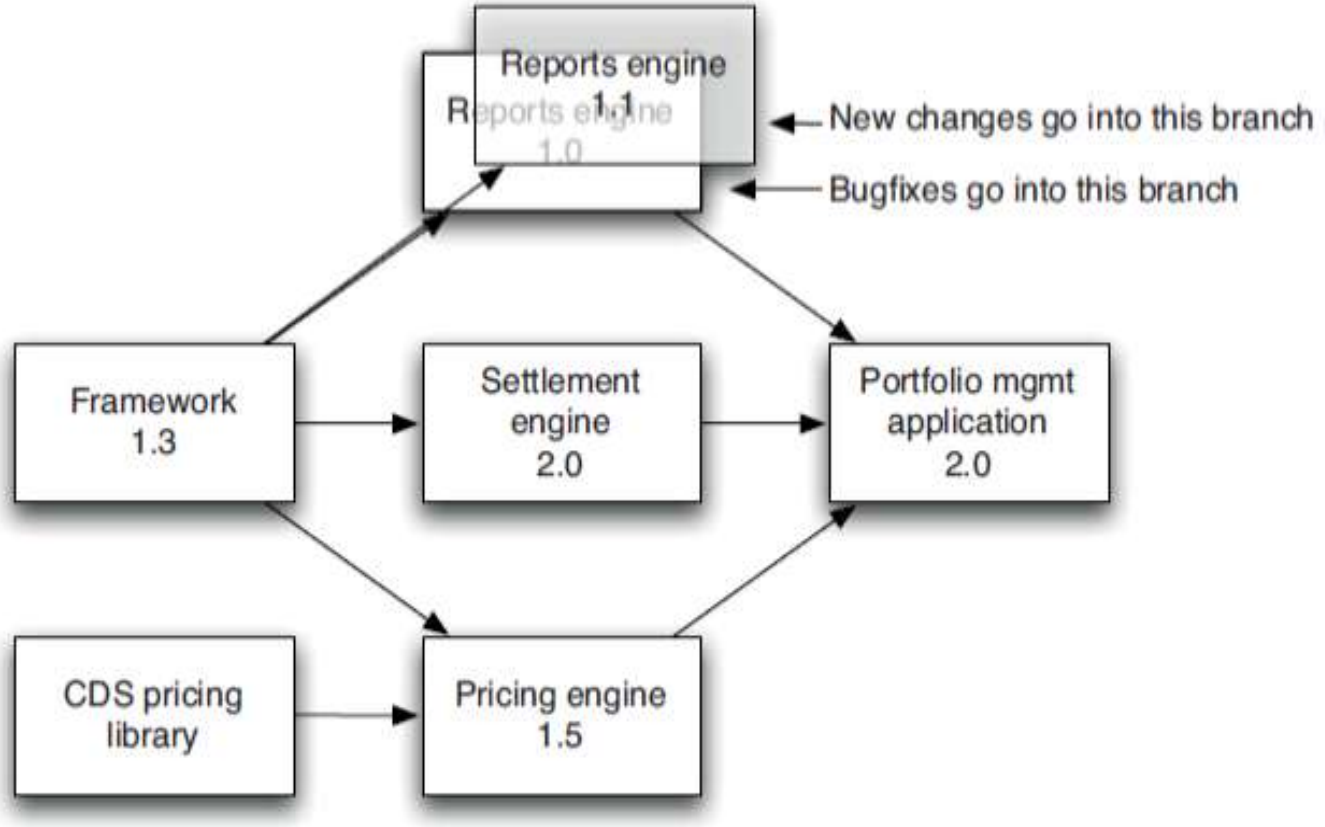
Pipelining Dependency Graphs Contd..



Visualizing downstream dependencies

Managing Component

Pipelining Dependency Graphs Contd..

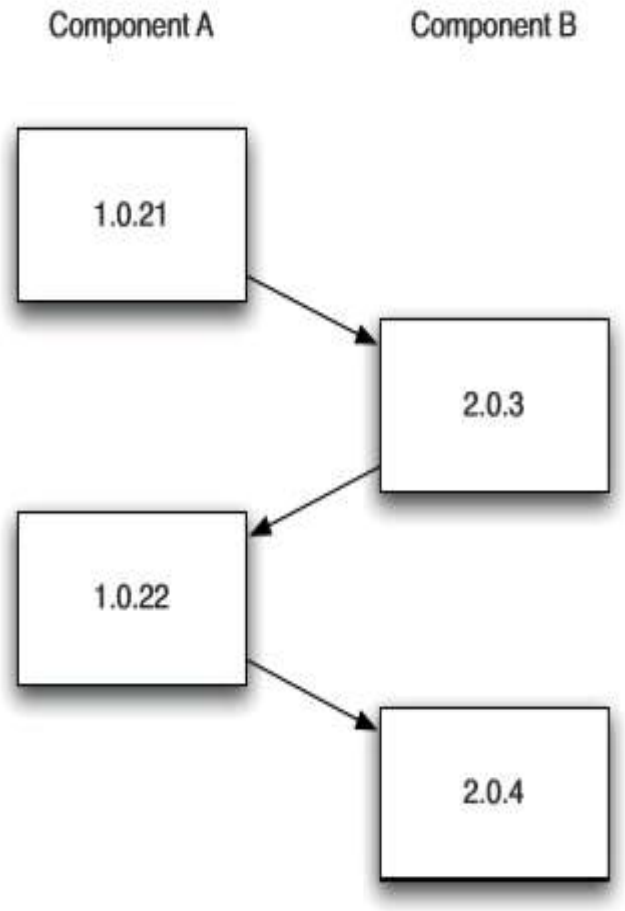


Branching Components

Managing Component

Circular Dependencies

- This occurs when the dependency graph contains cycles
- The simplest example is that you have a component, A, that depends on another component, B, unfortunately component B in turn depends on component A
- to survive this problem so long as there is a version of component A that you can use to build component B then use the new version of B to build the new version of A
- This results in a kind of “build ladder”
- No build system supports such a configuration out of the box, so you have to hack your toolchain to support it.
- also have to be cautious in how the parts of your build interact



Circular dependency build ladder

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Introduction to build

- Build automation and Build tools

Build automation and Build tools

Build tools

- Build tools automate software builds
- Usually, a software build consists of a number of steps that are dependent on each other
- In detail the sequence of events depends, on which programming language is used and for which target platform software is developed
- The following phases are generally part of a build:
  - Compiling the source code to binary code
  - Running and evaluating unit tests
  - Processing existing resource files (for example, configurations)
  - Generating artifacts that can be used later (for example, WAR or EAR files, Debian packages)

Build automation and Build tools

Build tools Contd..

- Additional steps that are often executed in the a build and that can likewise be automated with the employed build tools are:
  - Administering dependencies of, for instance the libraries that are used in the project
  - Running additional tests, such as acceptance and load tests
  - Analyzing code quality and examining defined conventions in the source code (static code analysis)
  - Archiving generated artifacts and packages in a central repository

Build automation and Build tools

Build tools Contd..

Technology	Tool
Rails	Rake
.Net	MsBuild
Java	Ant, Maven, Buildr, Gradle
C,C++	SCons

- Maven
- Gradle

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Build Tools – Maven and Gradle

- Maven
- Gradle

Build Tool - Maven

History

- Maven, a Yiddish word meaning accumulator of knowledge
- Originally started as an attempt to simplify the build processes in the Jakarta Turbine project
- Concept behind building Maven?
  - There were several projects each with their own Ant build files that were all slightly different and JARs were checked into CVS
  - Need of a standard way to build the projects, a clear definition of what the project consisted of, an easy way to publish project information and a way to share JARs across several projects
- The result is a tool that can now be used for building and managing any Java-based project

Build Tool - Maven

History

- With maven, day-to-day work of Java developers get easier and generally it help with the understanding of any Java-based project
- Maven addresses two aspects of building software
  - describes how software is built
  - describes its dependencies
- Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages
- The Maven project is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project
- And Initial Release of Maven was : 13 July 2004; 14 years ago

Build Tool - Maven

Software

- Maven is a software which collects all your dependency find it from all over internet compile your classes and provide you the final jar/war
- JAR: Java Archive Files
- WAR: Web Application Resource or Web Application Achieve Files
- EAR: Enterprise Archive

Build Tool - Maven

Objective

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

Build Tool - Maven

Phases of a standard Maven build

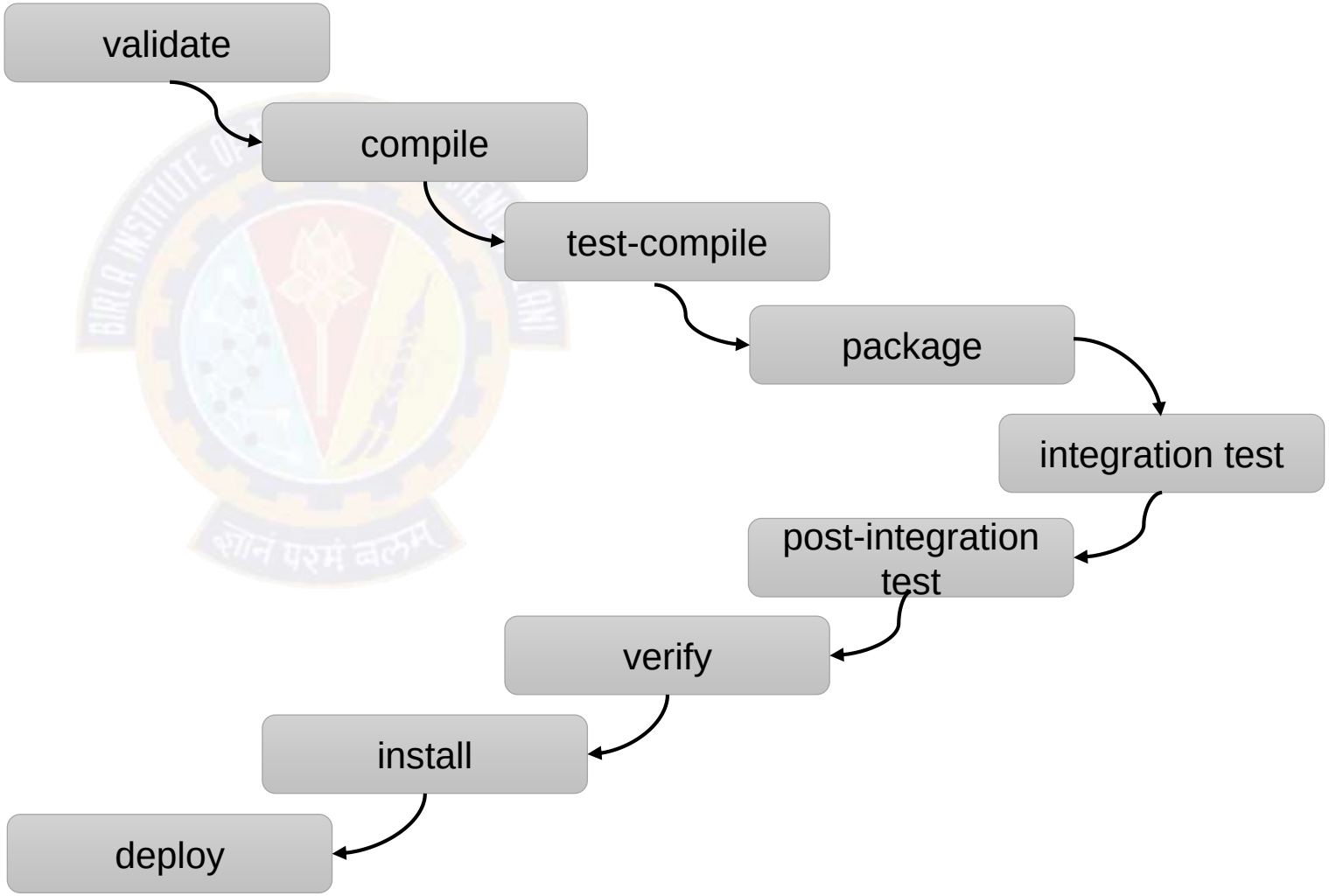
- Currently, Maven is likely the most widely used build tool for Java especially in large enterprises
- Maven default life cycle comprises a number of predefined and fixed phases
  - First, the source files are verified, and the build is initialized
  - Subsequently, the sources and resources are compiled and processed
  - The same happens with test sources and resources
  - Afterwards, the result is packaged, and optionally an integration test is run
  - Finally, the result can be verified, installed, and deployed in an artifact repository

- Not all actions are necessarily executed in each phase
- In this way Maven offers a skeleton along which your own build logic can be oriented

Build Tool - Maven

Phases of a standard Maven build

- Standardized phases, the commands for the compilation and testing of software are the same in each Maven project:
- *mvn package* executes all phases up to packaging and creates a build
- *mvn test* stops at the phase “test.”



Build Tool - Maven

Maven POM

- Stands for Project Object Model
- Describes a project
  - Name and Version
  - Artifact Type
  - Source Code Locations
  - Dependencies
  - Plugins
  - Profiles (Alternate build configurations)
  - Uses XML by Default

Build Tool - Maven

Project Name (GAV)

- Maven uniquely identifies a project using
- groupID: Arbitrary project grouping identifier (no spaces or colons)
  - Usually loosely based on Java package
- artfiactId: Arbitrary name of project (no spaces or colons)
  - version: Version of project
  - Format {Major}.{Minor}.{Maintenance}
  - Add ‘-SNAPSHOT ‘ to identify in development
- GAV Syntax: groupId:artifactId:version

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>Maven_BITS_Training</groupId>
  <artifactId>Maven_BITS_Training</artifactId>
  <version>0.0.1-SNAPSHOT</version>
```

Build Tool - Maven

Packaging

- Build type identified using the “packaging” element
- Tells Maven how to build the project
- Example packaging types: – pom, jar, war, ear, etc.
- Default is jar

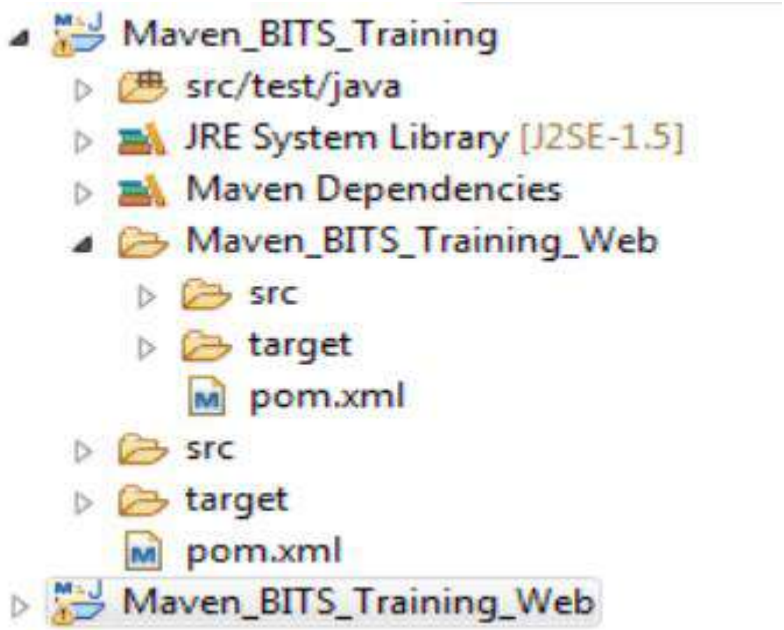
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>Maven_BITS_Training</groupId>
  <artifactId>Maven_BITS_Training</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>
```

Build Tool - Maven

Multi Module Projects

- Maven has 1st class multi-module support
- Each maven project creates 1 primary artifact
- A parent pom is used to group modules



Build Tool - Maven

Example Maven Goals

- *mvn install*
  - Invokes generate* and compile, test, package, integration-test, install
- *mvn clean*
  - Invokes just clean
- *mvn clean compile*
  - Clean old builds and execute generate*, compile
- *mvn compile install*
  - invokes generate*, compile, test, integration-test, package, install
- *mvn test clean*
  - Invokes generate*, compile, test then clean

Build Tool - Gradle

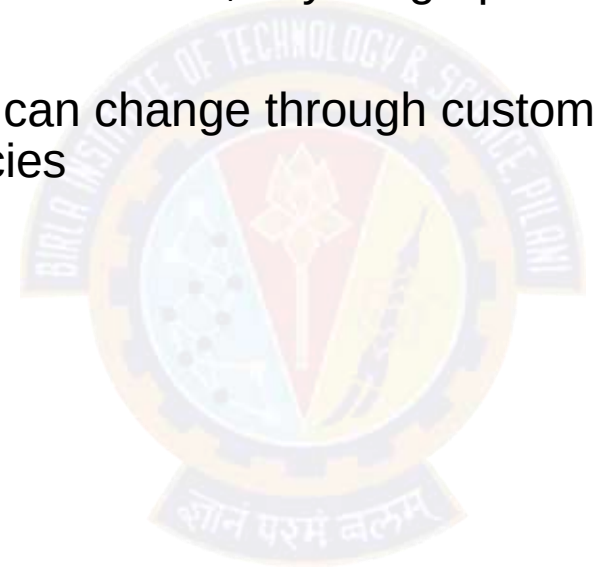
About Gradle

- Gradle is an open-source build automation tool focused on flexibility and performance
- Gradle build scripts are written in Domain Specific Language [DSL], i.e. Groovy
- Groovy resembles Java
- Every Java program is also a valid Groovy program, Groovy offers options for writing programs with substantially easier syntax
- Why Consider Gradle?
  - High performance
    - It is very specific on running only required tasks; which have been changed
    - Build cache helps to reuse tasks outputs from previous run
    - It has ability to have shared build cache within different machine
  - JVM foundation
    - It has JVM Foundation; where using Java Development Kit is Must
    - Benefits to use of standard Java API’s in build logic
    - However it is not limited to Java

Build Tool - Gradle

Gradle concepts

- The core concepts of Gradle are tasks and the dependencies between them
- Based on these, Gradle calculates a directed, acyclic graph to determine which tasks have to be executed in which order
- This is necessary since this graph can change through custom tasks, additional plug-ins, or the modification of existing dependencies



Build Tool - Gradle

Gradle offerings

- Gradle also offers the option to extend its functionality by embedding plug-ins

- This allows the developer to embed support of other programming languages like Groovy, C++, or Objective-C in Gradle

Build Tool - Gradle

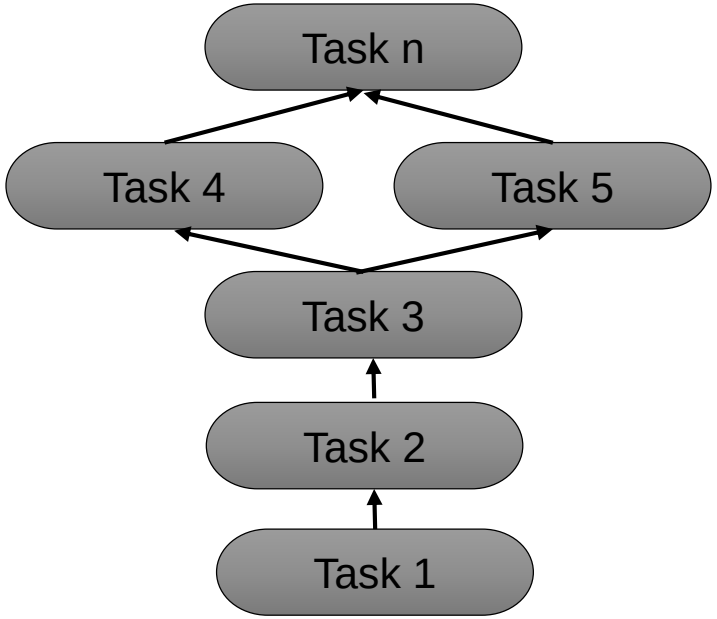
Gradle benefits

- Extensibility
  - You can readily extend Gradle to provide your own task types or even build model
- IDE Support
  - Several major IDEs allow you to import Gradle builds and interact with them
  - Example: Android Studio, IntelliJ IDEA, Eclipse, and NetBeans
- Insight
  - Build scans provide extensive information about a build run that you can use to identify build issues
  - They are particularly good at helping you to identify problems with a build’s performance

Build Tool - Gradle

Gradle DAG

- Gradle is a general-purpose build tool
  - Currently it supports Maven- and Ivy-compatible repositories and the filesystem
  - It supports create and publish custom plugins to encapsulate own conventions and build functionality
- The core model is based on tasks
  - Directed Acyclic Graphs (DAGs) of tasks
  - Once the task graph has been created
  - It determines which tasks need to be run in which order and then proceeds to execute them
  - Example of a Graph
  - Tasks Consists of:
    - Action : To perform something
    - Input : values to action
    - Output : generated by



Build Tool - Gradle

Summary

- Gradle has several fixed build phases
  - It evaluates and executes build scripts in Three Phases
  - Initialization : Set-up environment & Identify which Projects will be part of it
  - Configuration: Construct Task graph and then Identify task to be executed in defined order
  - Execution: Run the task
- Gradle is extensible in more ways than one
  - It allows you to Custom Task Types – Define your own Task Types
  - Custom Task Actions
  - Extra Properties on Projects and Tasks
  - Custom Conventions – User friendly

Build Tool

Gradle vs Maven

Gradle	Maven
Flexibility: • Flexibility on Conventions • User Friendly & Customized	**Flexibility:** • No flexibility on Conventions • It is rigid
Performance: • It process only the files has been changed • Reusability by working with Build Cache • Shipping is faster	**Performance:** • It process the complete build • No Build Cache concept • Shipping is slow as compared to Gradle
User Experience: • IDE Support : Is in evolving Stage • CLI : Modern CLI	**User Experience:** • IDE Support : Is mature • CLI solution is classic in comparison with Gradle

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Selenium - Test Automation

- Introduction
- Test login page – example
- Selenium Components
- Selenium History
- Selenium Supports
- Selenium Architecture
- Selenium Benefits

Selenium - Test Automation

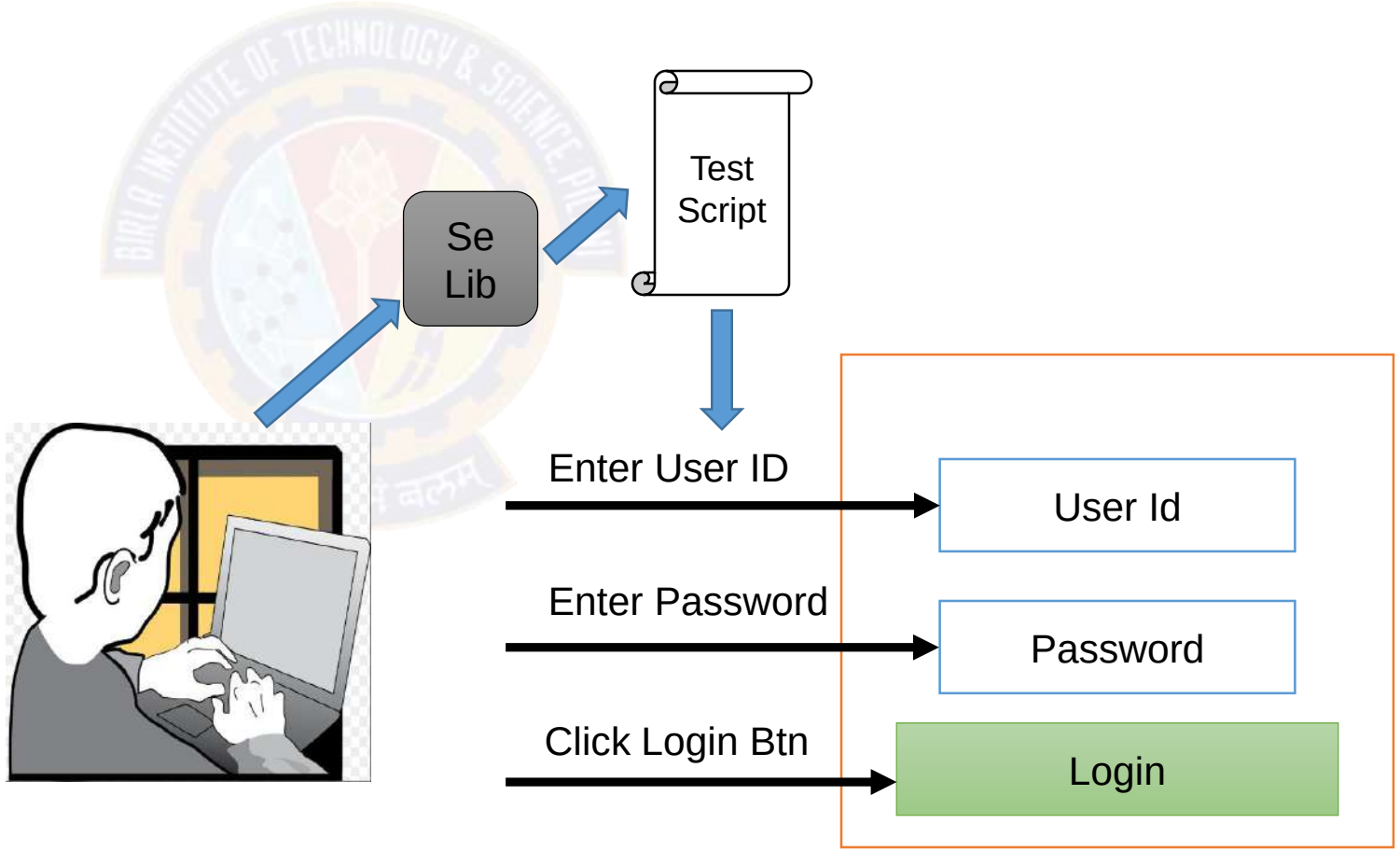
Introduction

- Selenium helps to automate web browser interaction
- Scripts perform the same interactions that any user can perform manually
- Selenium can perform any sort of automated interaction; however it was originally designed for automated web application testing
- Common benefits of Test Automation:
  - Frequent regression testing
  - Rapid feedback to developers
  - Virtually unlimited iterations of test case execution
  - Support for Agile and extreme development methodologies
  - Disciplined documentation of test cases
  - Customized defect reporting
  - Finding defects missed by manual testing

Selenium - Test Automation

Example

Test Login Page

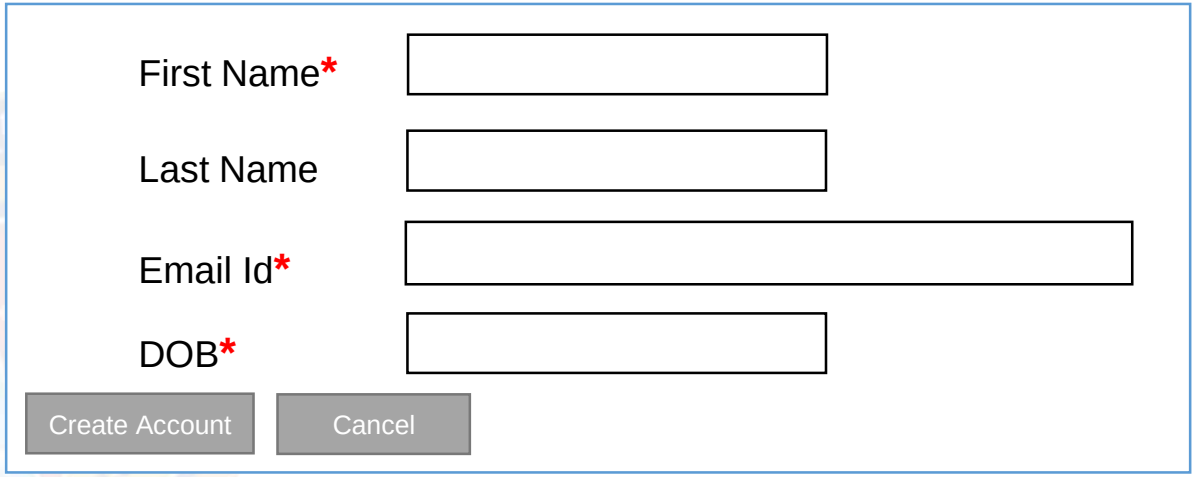


Selenium - Test Automation

Working with Selenium

- At a high level you will be doing three things with Salenium

1. Recognize web elements



2. Add Actions (Script using preferred lang)

- Input
- Select
- click

Test Data

3. Run the Test

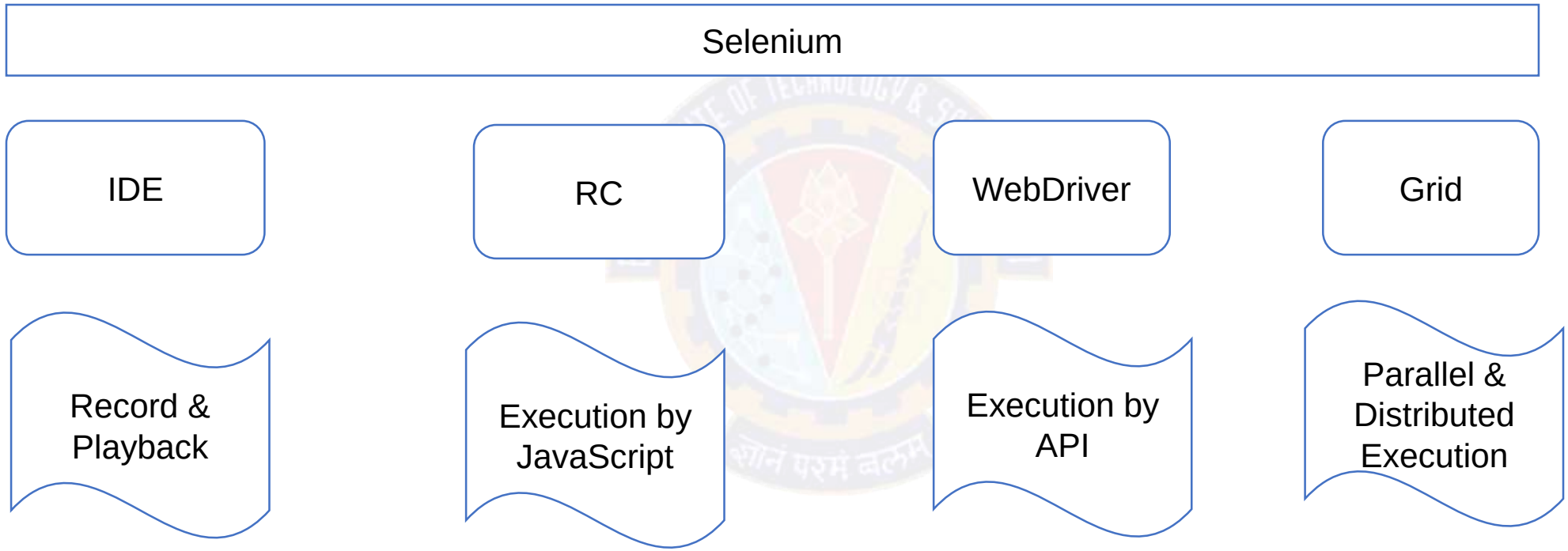
Selenium - Test Automation

Selenium Components

- Selenium IDE
  - A Record and playback plugin for Firefox add-on
  - Prototype testing
- Selenium RC (Remote Control)
  - Also known as selenium 1
  - Used to execute scripts (written in any language) using Javascript
  - Now Selenium 1 is deprecated and is not actively supported
- WebDriver
  - Most actively used component today
  - An API used to interact directly with the web browser
  - Is a successor to Selenium 1 / Selenium RC
  - Selenium RC and WebDriver are merged to form Selenium 2
- Selenium Grid
  - A tool to run tests in parallel across different machines and different browser simultaneously
  - Used to minimize the execution time

Selenium - Test Automation

Selenium Components Contd..



Selenium - Test Automation

Selenium History

- Selenium is a set of different software tools each with a different approach to support test automation
- Selenium support executing one’s tests on multiple browser platforms
- Selenium first came to life in 2004 when Jason Huggins was testing an internal application at ThoughtWorks
- It started with a Javascript library
- In 2006 at Google, Simon Stewart started work on a project WebDriver
- The merging of Selenium and WebDriver provided a common set of features for all users and brought some of the brightest minds in test automation under one roof in 2009

Selenium - Test Automation

Selenium - Supports

- Browsers
- OS
- Language

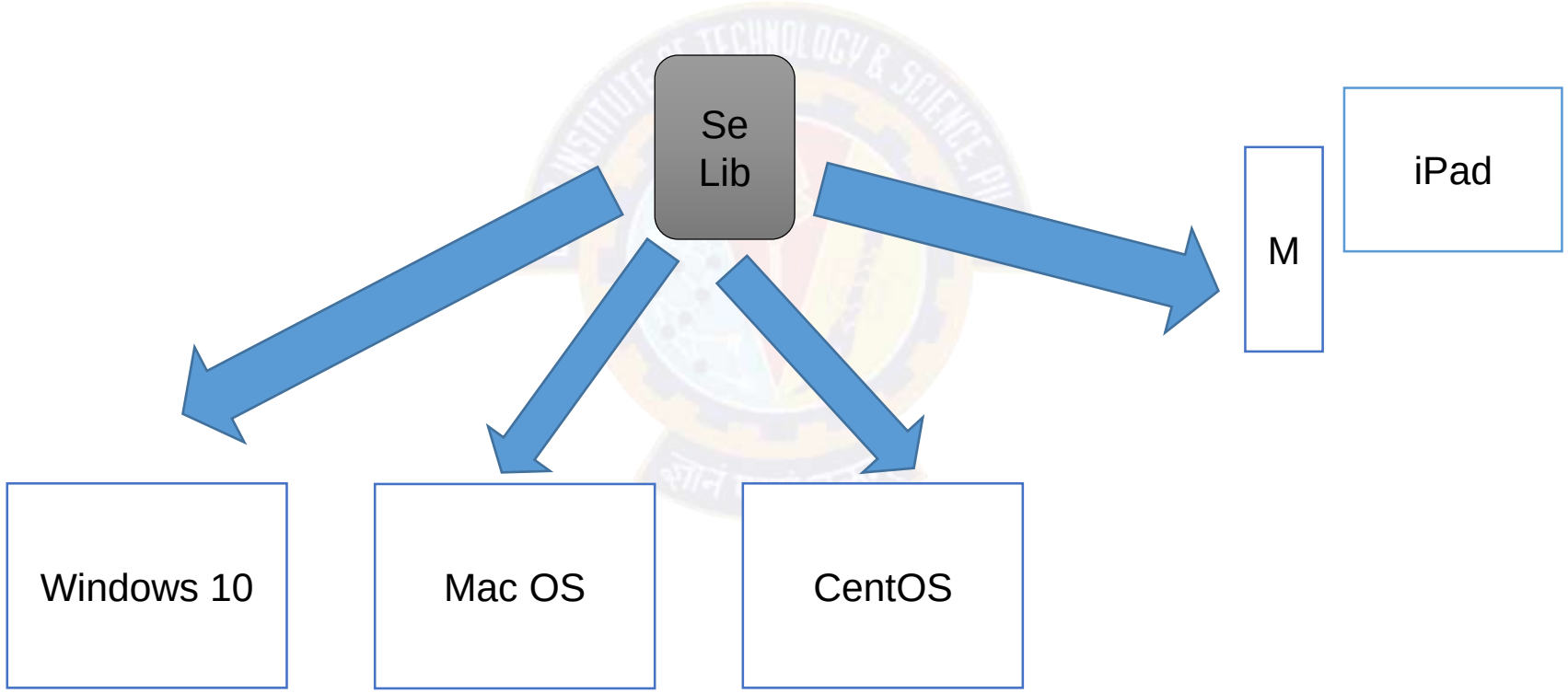
Selenium - Test Automation

Selenium client-server architecture

- Selenium Client includes:
  - The WebDriver API, which is used to develop test scripts to interact with page and application elements
  - The RemoteWebDriver class, which communicates with a remote Selenium server
- Selenium Server includes:
  - A server component, to receive requests from Selenium Client 's RemoteWebDriver class
  - The WebDriver API, to run tests against web browsers on a server machine
  - Selenium Grid, implemented by Selenium Server in command-line options for grid features

Selenium - Test Automation

Selenium Grid



Selenium - Test Automation

Benefits

- Faster Feedback
  - It enhances communication between product owners, developers and designers
  - Quick fix of malfunctions & bugs in real-time
- Accelerated Results
  - Testing can be executed on a repetitive basis, which in turn will deliver speedier results every time with minimized time and reduced effort
- Reduced Business Expenses
  - Time needed to execute tests will decrease dramatically
  - Larger quantity of work can be undertaken in the same amount of time with greatly increased levels of accuracy
  - This will directly reduce project costs

Selenium - Test Automation

Benefits Contd..

- Testing Efficiency Improvement
  - Testing occupies a considerable part of the whole Software Development Lifecycle
  - So even the smallest improvement in the overall efficiency can result in a massive difference to the overall project timeline
- Reusability of Automated Tests
- Earlier Detection of Defects
  - Early defect detection leads to cost-effective code rework
- Faster Time-to-Market
  - Time to market is a critical factor in product planning and success with regard to software applications
  - Efficiency of test automation will reduce the time of project; which results you to hit market early

References

- https://www.seleniumhq.org/docs/index.jsp/

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

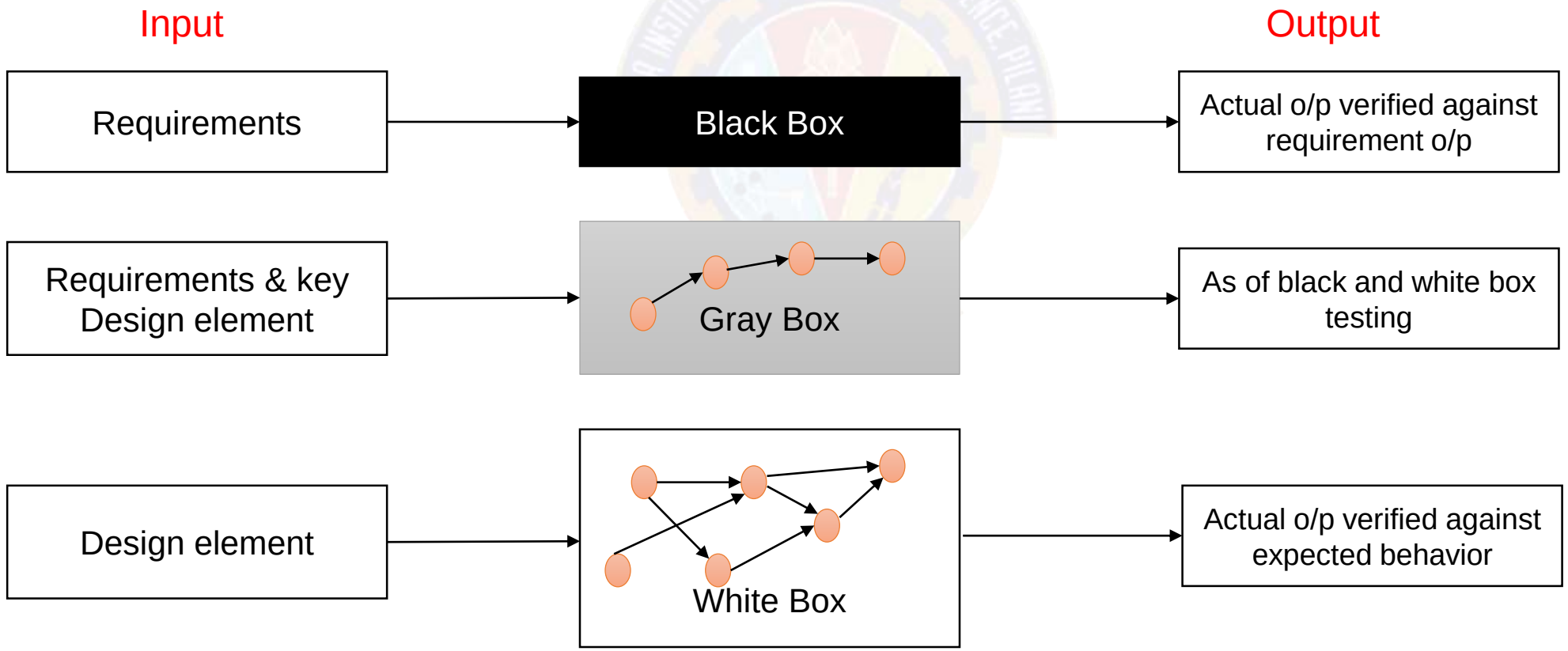
Unit Testing

- Types of Testing
- Traditional Vs. Unit Testing

Testing

Types of Testing

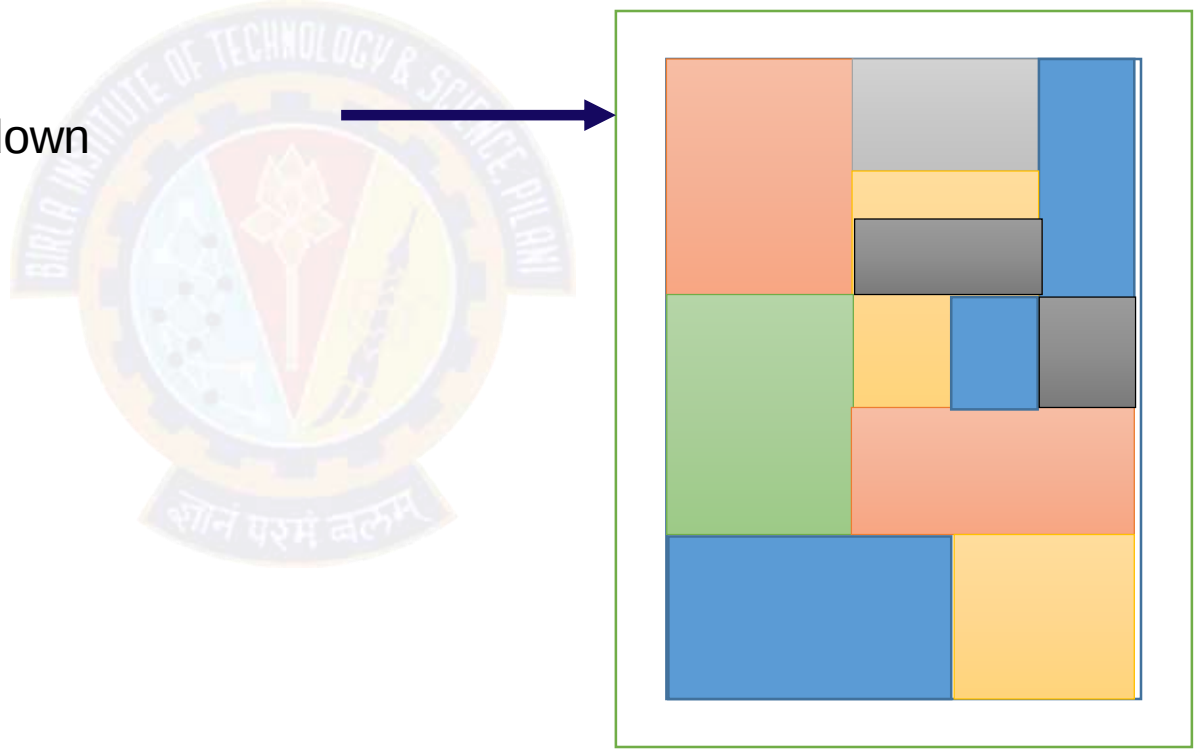
- Black box testing – (application interface, internal module interface and input/output description)
- White box testing- function executed and checked
- Gray box testing - test cases, risks assessments and test methods



Testing

Traditional Testing

- Test the system as a whole
- Individual components rarely tested
- Errors go undetected
- Isolation of errors difficult to track down



Testing

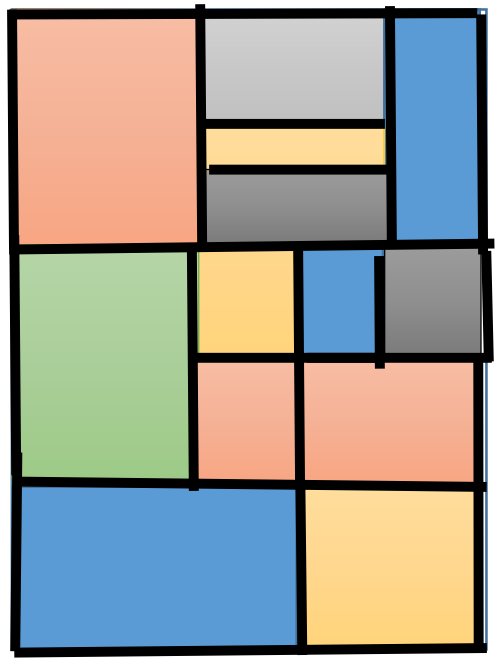
Traditional Testing Strategies

- Print Statements
- Use of Debugger
- Debugger Expressions
- Test Scripts

Testing

Unit Testing

- Is a level of the software testing process where individual units/components of a software/system are tested
- The purpose is to validate that each unit of the software performs as designed
- Each part tested individually
- All components tested at least once
- Errors picked up earlier
- Scope is smaller, easier to fix errors



Testing

Unit Testing Contd..

- A method by which individual units of source code are tested to determine if they are fit for use
- Concerned with functional correctness and completeness of individual program units
- Typically written and run by software developers to ensure that code meets its design and behaves as intended
- Its goal is to isolate each part of the program and show that the individual parts are correct

Unit Testing

What is Unit Testing

- Concerned with
- Functional correctness and completeness
- Error handling
- Checking input values (parameter)
- Correctness of output data (return values)
- Optimizing algorithm and performance

Unit Testing

Why Unit Testing?

- Faster Debugging
- Faster Development
- Better Design
- Excellent Regression Tool
- Reduce Future Cost

Unit Testing

Benefits

- Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly
- By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier
- Unit testing provides a sort of living documentation of the system

Unit Testing

Guidelines

- Keep unit tests small and fast
  - Ideally the entire test suite should be executed before every code check in Keeping the tests fast reduce the development turnaround time
- Unit tests should be fully automated and non-interactive
  - The test suite is normally executed on a regular basis and must be fully automated to be useful. If the results require manual inspection the tests are not proper unit tests
- Make unit tests simple to run
  - Configure the development environment so that single tests and test suites can be run by a single command or a one button click.
- Measure the tests
  - Apply coverage analysis to the test runs so that it is possible to read the exact execution coverage and investigate which parts of the code is executed and not

Unit Testing

Guidelines

- Fix failing tests immediately
  - Each developer should be responsible for making sure a new test runs successfully upon check in, and that all existing tests runs successfully upon code check in. If a test fails as part of a regular test execution the entire team should drop what they are currently doing and make sure the problem gets fixed
- Keep testing at unit level
  - Unit testing is about testing classes
  - There should be one test class per ordinary class and the class behavior should be tested in isolation. Avoid the temptation to test an entire work-flow using a unit testing framework, as such tests are slow and hard to maintain. Work-flow testing may have its place, but it is not unit testing and it must be set up and executed independently

Unit Testing

Guidelines

- Start off simple
  - One simple test is infinitely better than no tests at all. A simple test class will establish the target class test framework, it will verify the presence and correctness of both the build environment, the unit testing environment, the execution environment and the coverage analysis tool, and it will prove that the target class is part of the assembly and that it can be accessed
- Keep tests independent
  - To ensure testing robustness and simplify maintenance, tests should never rely on other tests nor should they depend on the ordering in which tests are executed
- Name tests properly
  - Make sure each test method test one distinct feature of the class being tested and name the test methods accordingly. The typical naming convention is test[what] such As testSaveAs(), testAddListener(), testDeleteProperty() etc.

Unit Testing

Guidelines

- Keep tests close to the class being tested
  - If the class to test is Foo the test class should be called FooTest (not TestFoo) and kept in the same package (directory) as Foo. Keeping test classes in separate directory trees makes them harder to access and maintain. Make sure the build environment is configured so that the test classes doesn't make its way into production libraries or executables
- Test public API
  - Unit testing can be defined as testing classes through their public API. Some testing tools makes it possible to test private content of a class, but this should be avoided as it makes the test more verbose and much harder to maintain. If there is private content that seems to need explicit testing, consider refactoring it into public methods in utility classes instead. But do this to improve the general design, not to aid testing

Unit Testing

Guidelines

- Think black-box
  - Act as a 3rd party class consumer, and test if the class fulfills its requirements. And try to tear it apart
- Think white-box
  - After all, the test programmer also wrote the class being tested, and extra effort should be put into testing the most complex logic
- Test the trivial cases too
  - It is sometimes recommended that all non-trivial cases should be tested and that trivial methods like simple setters and getters can be omitted. However, there are several reasons why trivial cases should be tested too:
    - Trivial is hard to define. It may mean different things to different people
    - From a black-box perspective there is no way to know which part of the code is trivial
    - The trivial cases can contain errors too, often as a result of copy-paste operations

Unit Testing

Guidelines

- Provide negative tests
  - Negative tests intentionally misuse the code and verify robustness and appropriate error handling
- Design code with testing in mind
  - Writing and maintaining unit tests are costly, and minimizing public API and reducing cyclomatic complexity in the code are ways to reduce this cost and make high-coverage test code faster to write and easier to maintain
- Don't connect to predefined external resources
  - Unit tests should be written without explicit knowledge of the environment context in which they are executed so that they can be run anywhere at anytime. In order to provide required resources for a test these resources should instead be made available by the test itself

Unit Testing

Guidelines

- Prioritize testing
  - Unit testing is a typical bottom-up process, and if there is not enough resources to test all parts of a system priority should be put on the lower levels first
- Prepare test code for failures
  - If the first assertion is false, the code crashes in the subsequent statement and none of the remaining tests will be executed. Always prepare for test failure so that the failure of a single test doesn't bring down the entire test suite execution
- Write tests to reproduce bugs
  - When a bug is reported, write a test to reproduce the bug (i.e. a failing test) and use this test as a success criteria when fixing the code.
- Know the limitations
  - Unit tests can never prove the correctness of code

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Continuous Code Inspection

- Code quality
- Code Inspection Measures
- How to Improve Your Code Inspection Process

Continuous Code Inspection

Code quality

- Continuous code inspection = Constantly scanning code
- Continuous code inspection is to Identify if any defects
- It is a process of code review; however the goal is to prevent defects
- Its been proved 90% of defects can be addressed using code inspections tools
- Code Inspection & Testing?
  - Testing verifies functionality and improves software quality
  - Testing is expensive if you have to go through it over and over again
  - By minimizing defects during code inspections, you can make your testing efforts more efficient
  - If your code is complex, it may not be fully testable
  - A code inspection, however, can find defects at the code level

Note: Even with automated testing, it takes time to verify functionality; by resolving defects at the code level, you'll be able to test functionality faster

Continuous Code Inspection

Code Inspection Measures

- Code inspections must be well-defined as per requirements:
  - Functional requirements : User Needs : Cosmetic
  - Structural requirements : System Needs : Re-engineering
- Run Time Defects:
  - Identify run time errors before program run
  - Examples: Initialization (using the value of unset data), Arithmetic Operations (operations on signed data resulting in overflow) & Array and pointers (array out of bounds, dereferencing NULL pointers), etc.,
- Preventative Practices:
  - This help you avoid error-prone or confusing code
  - Example: Declarations (function default arguments, access protection), Code Structure (analysis of switch statements) & Safe Typing (warnings on type casting, assignments, operations), etc.,
- Style:
  - In-house coding standards are often just style, layout, or naming rules and guidelines
  - Instead using a proven coding standard is better for improving quality

Continuous Code Inspection

How to Improve Your Code Inspection Process

- Involve Stakeholders
  - Developer, Management & Customer
- Collaborate
  - Collaboration — both in coding and in code inspections
- Recognize Exceptions
  - Sometimes there are exceptions to the rule
  - In an ideal world, code is 100% compliant to every rule in a coding standard
  - The reality is different
- Document Traceability
  - Traceability is important for audits
  - Capture the history of software quality

- What to Look For in Code Inspection Tools
- Automated inspection
- Collaboration system

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Continuous Integration

- Introduction
- History
- Continuous Integration Pre-requisites
- Continuous Integration System
- Continuous Integration Best Practices

Continuous Integration

Introduction

- Continuous integration (CI) is the process of integrating new code written by developers with a mainline or “master” branch frequently throughout the day
- Base Challenge : “Nobody is interested in trying to run the whole application until it is finished”
- Results:
  - For many software projects during the development process, its been observed the application is not in a working state
  - In fact, most software developed by large teams spends a significant proportion of its development time in an unusable state
  - This is doubly true in projects that use long-lived branches or defer acceptance testing until the end
  - Many such projects schedule lengthy integration phases at the end of development to allow the development team time to get the branches merged and the application working so it can be acceptance-tested
  - Even worse, some projects find that when they get to this phase, their software is not fit for purpose

Continuous Integration

Continuous Integration on other way

- Its been Observed projects that spend at most a few minutes in a state where their application is not working with the latest changes
- The difference is the use of continuous integration
- Continuous integration requires:
  - every time somebody commits any change, the entire application is built and a comprehensive set of automated tests is run against it
  - If the build or test process fails, the development team stops whatever they are doing and fixes the problem immediately
  - The goal of continuous integration is that the software is in a working state all the time
- In simple words we can say “Finish First Before Start”

Continuous Integration

History

- Continuous integration was first written about in Kent Beck's book Extreme Programming Explained (first published in 1999)
- The idea behind continuous integration was "if regular integration of your codebase is good, why not do it all the time?"
- Here all the time means "every single time somebody commits any change to the version control system"
- Facts:
  - Without continuous integration:
    - Your software is broken until somebody proves it works usually during a testing or integration stage
    - Your software delivery is slow
    - Your software may have more bugs
    - High chances to find the bugs in later stage where your project time would cost more
  - With continuous integration:
    - Your software is proven to work with every new change and you know the moment it breaks and can fix it immediately
    - You will able to deliver software much faster, and with fewer bugs
    - Bugs are caught much earlier in the delivery process when they are cheaper to fix

Continuous Integration

Pre-requisites

- Continuous Integration systems will usually run a series of tests automatically upon merging in new changes
- The outcome of these tests is often visualized, where “green” means the tests passed and the newly integrated build is considered clean, and failing or “red” tests means the build is broken and needs to be fixed
- There are three things that we need before we can start with continuous integration
  - Version Control
  - An Automated Build
  - Agreement of the Team

Continuous Integration Pre-requisites

Version Control

- Everything in a project must be checked in to a single version control repository: code, tests, database scripts, build and deployment scripts, and anything else needed to create, install, run, and test your application
- Kill the mindset of not considering the Version Control in reference to the size of Project
- There is belief, no single project is small enough to not to use version control
- There are several Version Control Systems Available
  - Git
  - Apache Subversion etc

Continuous Integration Pre-requisites

An Automated Build

- An Ideal IDE (Integrated Development Environment)
  - Must be able to start your build from the command line
  - It should support build, test, and deployment process in an automated fashion via the command line
  - It doesn't mean we should not have our build scripts [Do not create dependencies]
- Facts:
  - Build process must be automated in reference to audit for bugs / issues through your continuous integration
  - Build Scripts must be like codebase (they must be tested and refactored constantly); which makes it neat and easy to understand
  - Build process gets more and more important, the more complex the project becomes

Continuous Integration Pre-requisites

Agreement of the Team

- This is more about People & Culture
- Continuous integration is a practice, not a tool
- It requires a degree of commitment and discipline from development team or people involved
- As said “Fix first before Proceed”: Need everyone to check in small incremental changes frequently to mainline and agree that the highest priority task on the project is to fix any change that breaks the application
- If people don’t adopt the discipline necessary for it to work, attempts at continuous integration will not lead to the improvement in quality that you hope for

Continuous Integration Pre-requisites

A Basic Continuous Integration System

- A Basic Continuous Integration System:
  - Misconception: There is need of a continuous integration software
  - You don’t need a continuous integration software in order to do continuous integration—as we say, it is a practice, not a tool
- List of CI Tools:
  - Open Source :
  - Hudson
  - Cruise Control
- Commercial:
  - ThoughtWorks Studios
  - TeamCity by JetBrains
  - Bamboo by Atlassians
  - BuildForge by IBM

Continuous Integration

Continuous Integration System

- Let your CI tool find:
  - Your source control repository
  - What script to run in order to compile
  - Run the automated commit tests for your application
  - How to tell you if the last set of changes broke the software
- Using CI Server, process to be followed:
  - Check to see if the build is already running; If so, wait for it to finish
  - If it fails, make it green before you check in
  - Once it has finished and the tests have passed, update the code in your development environment
  - Run the build script and tests on development machine to make sure that everything still works correctly on local computer
  - If local build passes, check in the code into version control
  - Wait for your CI tool to run the build with changes
  - If it fails, stop current activity and fix the problem immediately
  - If the build passes, move on to next task

Note: Every time if it fails, fix first then move to the next step; this makes our code stable and working always

Continuous Integration Best Practices

Check In Regularly

- The most important practice for continuous integration to work properly is frequent check-ins to mainline
- A standard frequency should be checking in code at least a couple of times a day
- Benefits:
  - It makes changes smaller and thus less likely to break the build
  - Easy to revert in case a mistake
  - Disciplined about refactoring
  - It ensure that changes altering a lot of files are less likely to conflict with other team members work
  - It helps developers trying out ideas and discarding them by reverting back to the last committed version
  - If there is accidental deleting happens; we wont lost too much work
- Why Check In to Mainline?
  - Even though we have distributed branching supported by various Version Control Tools; It is complex task to do continuous integration while using branches
  - Branch makes code separate from other team members
  - Integration of long-lived branches are again a crazy task, much of manual task to resolve the conflicts

Note: If you try to Check In Regularly to Mainline; means you are in control

Continuous Integration Best Practices

Create a Comprehensive Automated Test Suite

- Testing plays an important role to verify application is working or not
  - it's essential to have some level of automated testing to provide confidence that application is actually working
- Types of tests involved in continuous integration build
  - Unit tests :
    - These are written to test the behavior of small pieces of application in isolation
    - These tests can usually be run without starting the whole application
    - These tests don't require application to be running in a production-like environment
    - Unit tests should run very fast
- Component tests:
  - These tests, test the behavior of several components of an application
  - These tests also don't always require starting the whole application
  - Component tests typically take longer to run
- Acceptance tests:
  - Acceptance tests test that the application meets the acceptance criteria decided by the business / customer
  - Including both the functionality provided by the application and its characteristics such as capacity, availability and security etc…
  - These tests run against the whole application in a production-like environment

Note: These three sets of tests, combined, should provide an extremely high level of confidence that any introduced change has not broken existing functionality

Continuous Integration Best Practices

Keep the Build and Test Process Short

- Why it should be short?
  - If it is too long to build the code and run the unit tests, you will run into the following problems:
    - People will stop doing a full build and running the tests before they check in
    - If multiple commits have taken place; if the build fails, its difficult to identify which check in has caused this
    - People will check in less often because they have to sit around for ages waiting for the software to build and the tests to run
- Ideally, the compile and test process should take no more than a few minutes
- Standard Recommendation:
  - 10 min is Good
  - 5 min is Better
  - 90 Sec is IDEAL
- Techniques to reduce the build time
  - Making your tests run faster
  - Refactoring the code for fast processing
  - Split test process on the basis of
  - Compile the software
  - Create a deployable binary
  - Run acceptance tests
  - Integration tests
  - Performance tests

Continuous Integration Best Practices

Managing Development Workspace

- It is important for development team that their development environment is carefully managed
- Development team should be able to run the build, execute the automated tests, and deploy the application in an environment under their control
- If feasible the development workspace should be on their own local machine; Only in exceptional circumstances they should use shared environments for development
- The local Development Workspace must be replica of Production
- Manage configuration as same as in production; better to use a configuration management tools
- All automated tests should be configured to run from local Developer Machine

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

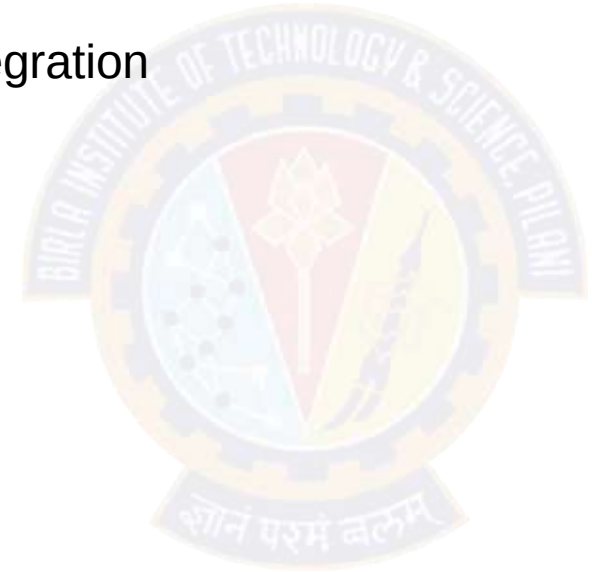
Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Using Continuous Integration Software

- Basic Functionality of Continuous Integration Software
- Basic Operations
- How it was before Continuous Integration
- Prevention to be adhere



Continuous Integration Software

Basic functionality of continuous integration software

- Poll version control system
- Check if any commits have occurred
- If so, check out the latest version of the software
- Trigger / Run build script to compile the software
- Run the tests
- And then notify you of the results
- There are many products on the market that can provide the infrastructure for automated build and test process

Continuous Integration Software

Basic Operation

- Continuous Integration server software has two components
  - The first is a long-running process which can execute a simple workflow at regular intervals
    - The usual CI workflow polls revision / version control system at regular intervals
    - If it detects any change, it will check out a copy of project to a directory on the server
    - It will then execute the commands (these commands build an application and run the relevant automated tests)
  - The second provides a view of the results of the processes that have been run, notifies for the success or failure of build and test runs, and provides access to test reports, installers, and so on
    - Most CI servers include a web server that shows you a list of builds that have run
    - Allows look at the reports that define the success or failure of each build

Continuous Integration Software

Bells and Whistles

- Apart from basic operations we can get the status of the most recent build sent to an external device
- Organizations use red and green lava lamps to show the status of the last build
- Even few organization go with all in one health board for all the tools used; where you can see health of each build, git commits and configuration management results
- Some Advanced experiments:
  - Flashing lights and sirens for build progress
  - Text-to-speech to read out the name of the person who broke the build
  - Display the status of the build, along with the avatars of the people who checked in

Note: These are just a great way to allow everyone to see the status of the build at a glance and Visibility is one of the most important benefits of using a CI server

How it was before Continuous Integration

Nightly Build

- It was a common practice for many years
- If I break it; I will monitor until next break by someone
- The idea is that a batch process will compile and integrate the codebase every night when everybody goes home
- If it is failed again during the night run, can do changes to fix it however need to wait for next night run
- So verification of System Integration will be on hold until next run

Note: this strategy will not be a good Idea when you have a geographically dispersed team working on a common codebase from different time zones

Prevention to be adhere

Don’t Check In on a Broken Build

- Per process if a build breaks, developer team need to fix it ASAP
- If any check-in breaks the build then try to fix it before moving ahead
- At the Broken Stage; nobody should be allowed to check-in
- What if we do Check In on Broken Build:
  - If any new check-in or build trigger during broken state will take much longer time to fix
  - Frequent broken build will encourage team not to care much about working condition

Note: The CI System we are building is to make sure green build and working software

Prevention to be adhere

Commit locally than direct to Production

- Per process a commit triggers the creation of a release candidate
- Try to keep Check-in lightweight
- Generally regular check-in with 20 min interval is a achievable state
- To achieve this we need some commitment from Development Team
  - Commit locally
    - Always keep refreshed local copy
    - If feasible run local build and commit tests
    - If successful; then push to Production
- Why Check-in locally and Commit test run?
  - If there are check-in's before our last update; high chance of conflict :: So if we check out and run tests locally, we have chance to identify problem without breaking the build
  - If new artifacts inclusion has been missed in the Check In, it will result to common errors :: running locally will help to Identify missing configuration

Prevention to be adhere

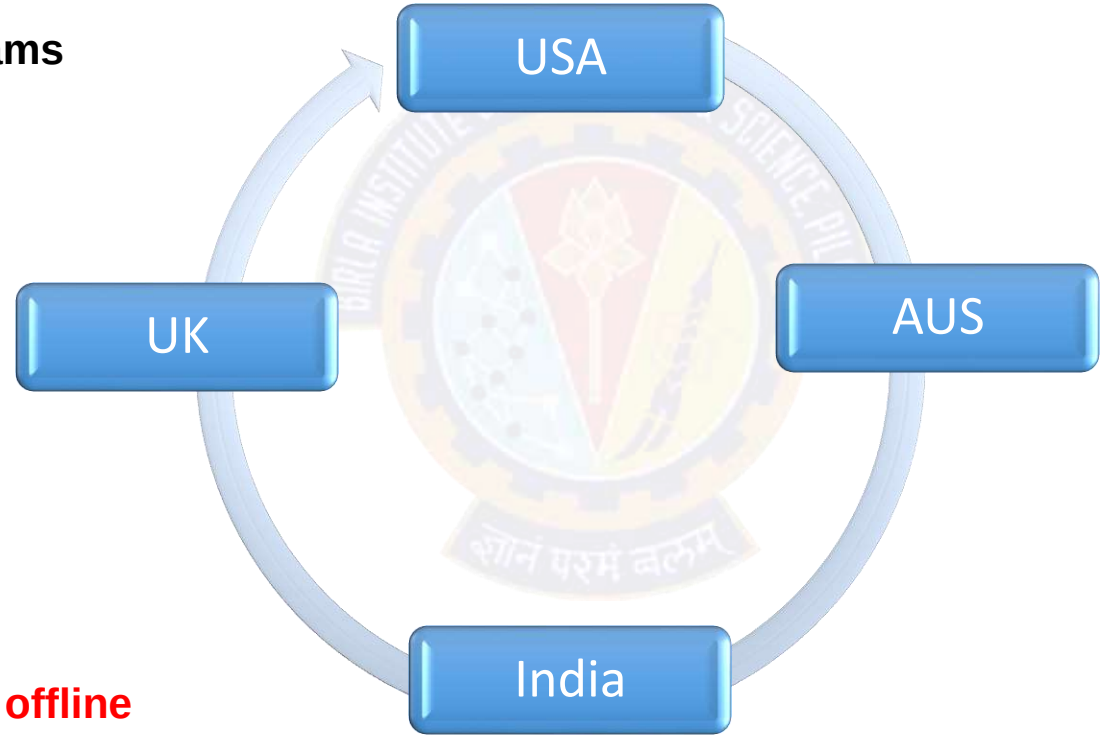
Wait for Commit Tests to Pass before Moving On

- The CI System is a Shared resource
- At time of Check-in, we should monitor the build progress – Its Developers default duty
- Until the check-in is compiled and passed commit tests, Do not start any new task
- Its not good to say however “Developer should not even leave out for Lunch or Meeting”
- If the commit succeeds the developers are then and only then free to move
- If it fails “Either fix it or Revert it to previous working version”
- Never go home with broken build

Prevention to be adhere

Wait for Commit Tests to Pass before Moving On Contd..

Geo – Distributed Teams Working on CI



Hardly it will be 3 hrs offline time to fix build

Prevention to be adhere

Always Be Prepared to Revert to the Previous Revision

- In a scenario of failed commit stage, it is important that you get everything working again quickly
- If you can’t fix the problem quickly, for whatever reason, you should revert to the previous change-set held in revision control
- And remedy the problem in our local environment
- Example:
  - Airplane pilots are taught that every time they land, they should assume that something will go wrong
  - So they should be ready to abort the landing attempt and “go around” to make another try
  - Use the same mindset when checking in
  - Assume that it may break something that will take more than a few minutes, and know what to do to revert the changes and get back to the known-good revision in version control

Prevention to be adhere

Always Be Prepared to Revert to the Previous Revision Contd..

- If you try to revert every time then; how can you make progress?
- Time Boxing
  - Establish a team rule
  - When the build breaks on check-in, try to fix it for ten minutes or any approximate time environment can bare
  - However it should not be so long
  - If, after ten minutes, you aren’t finished with the solution, revert to the previous version from your version control system
- Take Responsibility for All Breakages That Result from committed Changes

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Continuous Integration Software: Jenkins

- Introduction
- History
- From Hudson to Jenkins

Jenkins

Introduction

- It is an open-source CI tool written in Java
- It is a cross-platform CI tool
- It offers configuration both through GUI interface and console commands
- Its plugin list is very comprehensive
- It also flexible to allow you to add your own plugin
- Jenkins prides itself on distributing builds and test loads on multiple machines
- It is published under MIT license so it is free to use and distribute

- Note: Jenkins is one of the most popular solution in Market

Jenkins

History

- The Jenkins project was started in 2004 (originally called Hudson) by Kohsuke Kawaguch
- It started when Kohsuke was part of Sun Microsystems
- He created Jenkins CI as a way to perform continuous integration – that is, to test his code before he did an actual commit to the repository
- He open sourced it and soon it became famous
- Today Jenkins orchestrates the entire software delivery pipeline – called continuous delivery
- Continuous delivery (CD), coupled with a DevOps culture, dramatically accelerates the delivery of software

Jenkins

From Hudson to Jenkins

- Jenkins has been started as a project under the name of Hudson in late 2004 at Sun Microsystem
- As Hudson evolved over the years, it was adopted by more and more teams within Sun for their own projects
- By early 2008, Sun recognized the quality and value of the tool, and ask Kohsuke to work on Hudson full-time
- In 2009, Oracle acquired Sun
- Most of the core Hudson developers, led by Kohsuke, preferred to continue with the open, flexible, and fast-paced community-focused model that had worked so well for Hudson in the past

Jenkins

From Hudson to Jenkins

- In January 2011, the Hudson developer community decisively voted to rename the project to Jenkins
- They subsequently migrated the original Hudson code base to a new GitHub project
- The vast majority of core and plugin developers upped camp and followed Kohsuke Kawaguchi and other core contributors to the Jenkins camp
- After the fork, a majority of users also followed the Jenkins developer community and switched to Jenkins

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Jenkins

- Jenkins Prerequisites
- Prepare Jenkins environment
- Jenkins Installation
- Post-installation setup wizard
- Customizing Jenkins with plugins

Jenkins

Prerequisites

- Minimum hardware requirements:
  - 256 MB of RAM
  - 1 GB of drive space
- Recommended hardware configuration for a small team:
  - 1 GB+ of RAM
  - 50 GB+ of drive space
- Software requirements:
  - Java 8 - either a Java Runtime Environment (JRE) or a Java Development Kit (JDK) is fine

- Jenkins is typically run as a standalone application in its own process with the built-in Java servlet container/application server (Jetty)
- Jenkins can also be run as a servlet in different Java servlet containers such as Apache Tomcat or GlassFish

Jenkins

Prepare Jenkins environment

- The most basic function of any Continuous Integration tool is to monitor source code in a version control system and to fetch and build the latest version of your source code whenever any changes are committed
  - The first thing needed is to install Java
  - Need a version control system like Git
  - Open an GitHub Account

Jenkins

Prepare Jenkins environment Contd..

- Set-up the Java Runtime Environment [JRE]; A recent version of Java
  - Check Java Version on System with
    - $java -version
- Install and configure Git
  - On Debian Linux Machine such as Ubuntu
    - $ sudo apt-get install git-core
  - In Fedora
    - $sudo yum install git-core
  - Post installation we can check the version with
    - $git –version
  - Windows
  - Get installer from https://git-scm.com/downloads

- Set-up a GitHub Account

Jenkins

Prepare Jenkins environment Contd..

- Configure SSH Key:
  - GitHub uses SSH keys to establish a secure connection between your computer and the GitHub servers [Linux]
- Create a Sample Repository:
  - Login to the your GitHub Account
  - Then create a repository
  - Once the Repository is created; get this cloned to your local machine
    - $git clone <Path of the Repository>

Jenkins

Installation: macOS Platform

- To install from the website, using a package:
  - Download the latest package [https://jenkins.io/download/]
  - Open the package and follow the instructions
- Jenkins can also be installed using brew:
  - Install the latest release version
    - $ brew install jenkins
  - Install the LTS [Long Term Support] version
    - $ brew install jenkins-lts

Jenkins

Installation: Debian/Ubuntu Platform:

- install Jenkins through apt :
  - $ sudo apt-get install jenkins
  - Setup Jenkins as a daemon launched on start in /etc/init.d/jenkins
  - Create a ‘jenkins’ user to run this service
  - Log file location incase of troubleshooting : /var/log/jenkins/jenkins.log
  - Populate /etc/default/jenkins with configuration parameters for the launch
  - Set Jenkins to listen on port 8080

- Note: if the port has been occupied already i.e. 8080 then edit the /etc/default/jenkins to replace the line HTTP_PORT=8080 with another port available

Jenkins

Installation: Windows Platform

- To install from the website, using the installer:
  - Download the latest package [https://jenkins.io/download/]
  - Open the package and follow the instructions

Jenkins

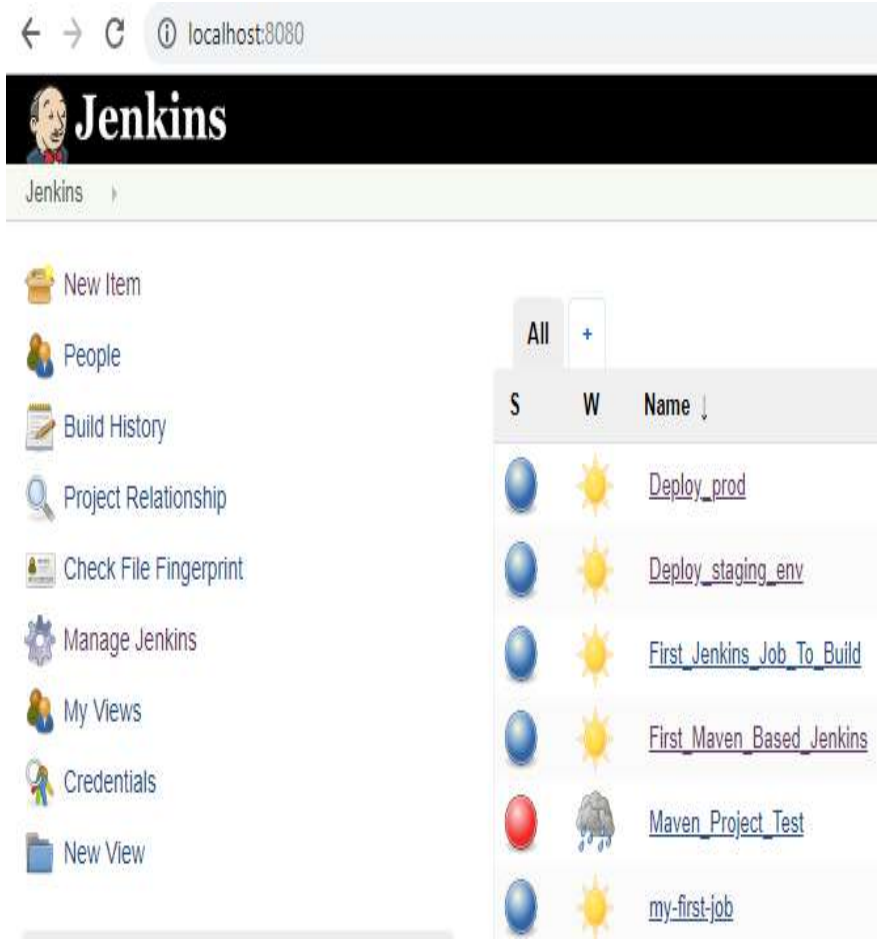
Post-installation setup wizard

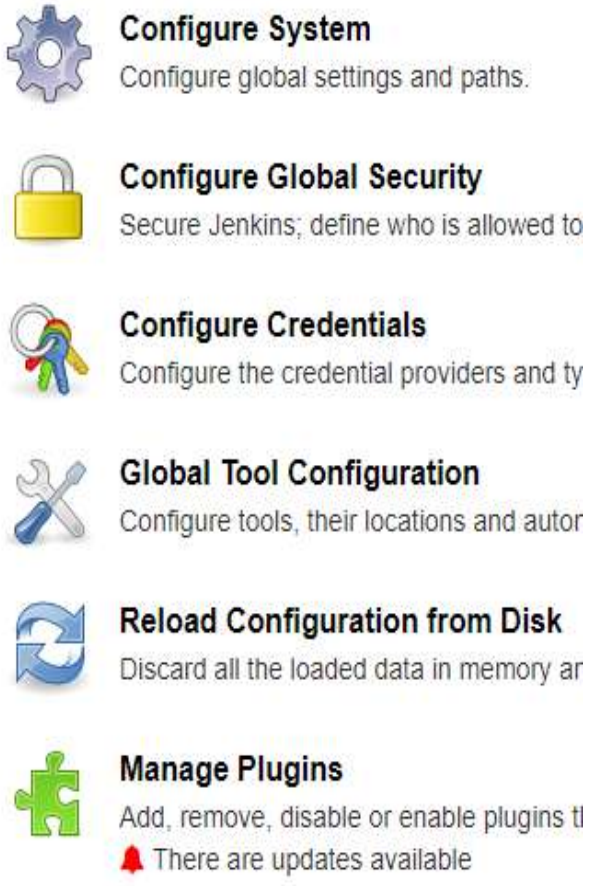
- Unlocking Jenkins: [Linux]
  - first access a new Jenkins instance, will asked to unlock it using an automatically generated password
- Step1:
  - Browse to localhost:8080 [or] localhost:<port-number>, whichever port configured for Jenkins when installing it
  - And wait until the Unlock Jenkins page appears
- Step2:
  - Go to the Jenkins console log output
  - Copy the automatically-generated alphanumeric password [between the 2 sets of asterisks]
- Step3:
  - On the Unlock Jenkins page, paste this password into the Administrator password field
  - Click Continue
- Unlocking Window:
  - Copy the password from initialAdminPassword.txt present in secrets folder of installed Jenkins directory

Jenkins

Customizing Jenkins with plugins

- At Customize Jenkins page we can install any number of useful plugins
- Install suggested plugins - to install the recommended set of plugins, which are based on most common use cases
- We can install (or remove) additional Jenkins plugins at a later point in time via the Manage Jenkins > Manage Plugins page in Jenkins





Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Artifact Management

- Introduction
- Objective of Artifact Management
- Importance of Artifact Management
- Benefits of Artifact Management

Artifact Management

Introduction

- An artifact is the output of any step in the software development process
- Artifacts can be a number of things like JARs , WARs, Libraries, Assets and application
- Generally, an artifact repository can serve as:
  - A central point for management of binaries and dependencies;
  - A configurable proxy between organization and public repositories; and
  - An integrated depot for build promotions of internally developed software

Artifact Management

Objective of Artifact Management

- An artifact repository should be:
  - secure;
  - trusted;
  - stable;
  - accessible; and
  - Versioned
- Artifact Management Tools:
  - JFrog
  - Nexus
  - Maven
  - HelixCore

Artifact Management

Importance of Artifact Management

- Having an artifact repository allows us to treat dependencies statically
- Artifact repositories allow to store artifacts the way we use them
- Some repository systems only store a single version of a package at a time
- Ideally, local development environment has the same access to your internal artifact repository as the other build and deploy mechanisms in your environment
- This helps minimize the “it works on my laptop” syndrome because the same packages and dependencies used in production are now used in the local development environment
- If you don’t have access to the internet within your environment; artifact management helps to have your own universe
- Relying on the internet for your dependencies means that somebody else ultimately owns the availability and consistency of your builds, something that many organizations hope to avoid
- An artifact repository organizes and manages binary files and their metadata in a central place

Artifact Management

Benefits

- The Binary Repository Manager: Artifact repositories allow teams to easily find the correct version of a given artifact
- This means developers can push to, and pull from, the artifact repository as part of the DevOps workflow
- Single Source of Truth: Artifact repositories improve the way teams work together by ensuring everyone is using the correct files
- It helps CI / CD to be more reliable

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Continuous Deployment

- What Is a Deployment?
- Deployment Consideration
- Challenges in Deployment
- New Approach of Deployment

Continuous Deployment

What Is a Deployment?

- “Deployment is the process of placing a version of service into Production”
- The initial deployment of a service can be viewed as going from no version of the service to the initial version of the service
- Initial deployment happens only once
- New version deployment is a frequent process
- “The overall goal of a deployment is to place an upgraded version of the service into production with minimal impact to the users of the system”
- Why we change Service?
  - To Fix an Error
  - To Improve some quality of the service
  - To add a new feature

Deployment Consideration

Compatibility

- Backward Compatibility: “A service is backward compatible if the new version of the service behaves as the old version”
  - For requests that are known to the old version of a service, the new version provides the same behavior
  - The external interface provided by new version of a service are a superset of the external interface provided by old version of that service.
- Forward Compatibility: “It means that a client deals gracefully with error responses indicating an incorrect method call”
  - Suppose a client wishes to utilize a method that will be available in new version of a service but the same method is not present in old version
  - Then if the service returns an error code indicating it does not recognize the method call, the client can conclude that it has reached Old version of the Service

Deployment Consideration

Canary Testing

- A new version is deployed in to production after having been tested in a staging environment, which is close to a production environment
- There is still possibility of errors existing in the new version
- It could be any functional error or a quality issue
- Performing an additional step of testing in a real production environment is the purpose of canary testing
- “A Canary Test is conceptually similar to a beta test
- Who should perform the Canary Test?
  - Selected Development Team
  - End Customers
  - Mixed Stake Holders

Deployment Consideration

Rollback

- In every deployment the consideration would be for some period the new version of service deployed will be in probation
- Even it has gone through testing of a variety of forms but it still is not fully trusted
- Here comes the request for Rollback
- Rolling back means reverting to a prior release
- It is even possible to roll forward i.e. this includes correcting the error and generate a new release with the error fixed
- Here rolling forward is essentially just an instance of upgrading
- In Rollback it is highly recommended to have a human interference to decide “Whether the error is serious enough to justify discontinuing the current deployment?” then “decide whether to roll back or roll forward?”

Deployment Consideration

Tools

- A large number of tools exist to manage deployment
- For an Example: If a VM image contains all the required software including the new version of service, then you can replace whole VM of the old version with VM of new version
- Or other approach will be change the internals of a VM so as to deploy the new version by replacing the old version without terminating the VM
- Configuration Management Tools like Puppet and Chef manages the item inside of a VM, they can replace a version of piece of software; inline of configuration specified

Challenges in Deployment

Deployment Process Waste

- Build and operations teams waiting for documentation or fixes
- Testers waiting for “good” builds of the software
- Development teams receiving bug reports weeks after the team has moved on to new functionality
- Discovering, towards the end of the development process, that the application’s architecture will not support the system’s nonfunctional requirements

Note: This leads to software that is undeployable because it has taken so long to get it into a production-like environment, and buggy because the feedback cycle between the development team and the testing and operations team is so long

New Approach of Deployment

Expectation to Overcome

- End-to-End approach to delivering software
- Deployment of Application should be easy & one click to go
- A powerful feedback loop; rapid feedback on both the code and the deployment process
- Lowering the risk of a release and transferring knowledge of the deployment process to the development team
- Testing teams deploy builds into testing environments themselves, at the push of a button
- Operations can deploy builds into staging and production environments at the push of a button
- Developers can see which builds have been through which stages in the release process, and what problems were found
- Managers can watch such key metrics as cycle time, throughput, and code quality
- Automate Build, Deploy, Test and Release System

Note: As a result, everybody in the delivery process gets two things: access to the things they need when they need them, and visibility into the release process to improve feedback so that bottlenecks can be identified, optimized, and removed

This leads to a delivery process which is not only faster but also safer

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Deployment Pipeline

- What Is a Deployment Pipeline?
- Value Stream Map of a Product Creation
- Structure of Deployment Pipeline
- Basic Deployment Pipeline
- Deployment Pipeline Practices
- Preparing to Release

Deployment Pipeline

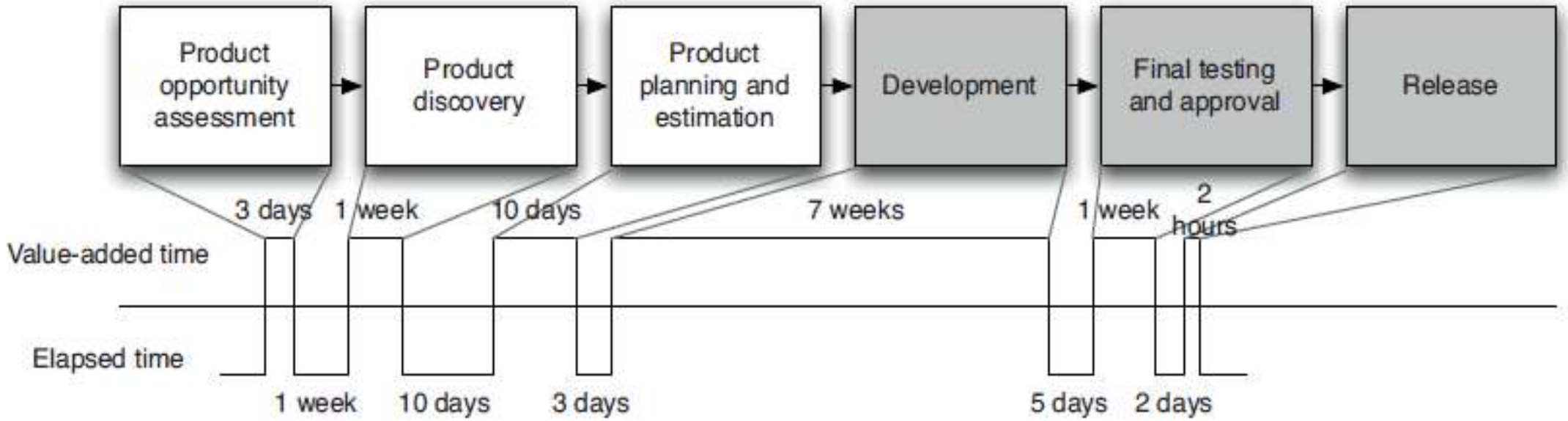
Deployment Pipeline

- “At an abstract level, a deployment pipeline is an automated manifestation of your process for getting software from version control into the hands of your users”
- Every change to your software goes through a complex process on its way to being released
- That process involves building the software, followed by the progress of these builds through multiple stages of testing and deployment
- This requires collaboration between many individuals, and perhaps several teams
- The Reason
  - The deployment pipeline models the above process, and its manifestation in a continuous integration and release management tool is what allows you to see and control the progress of each change as it moves from version control through various sets of tests and deployments to release to users

Deployment Pipeline

Value Stream Map of a Product Creation

- A high-level value stream map for the creation of a new product is shown in Figure

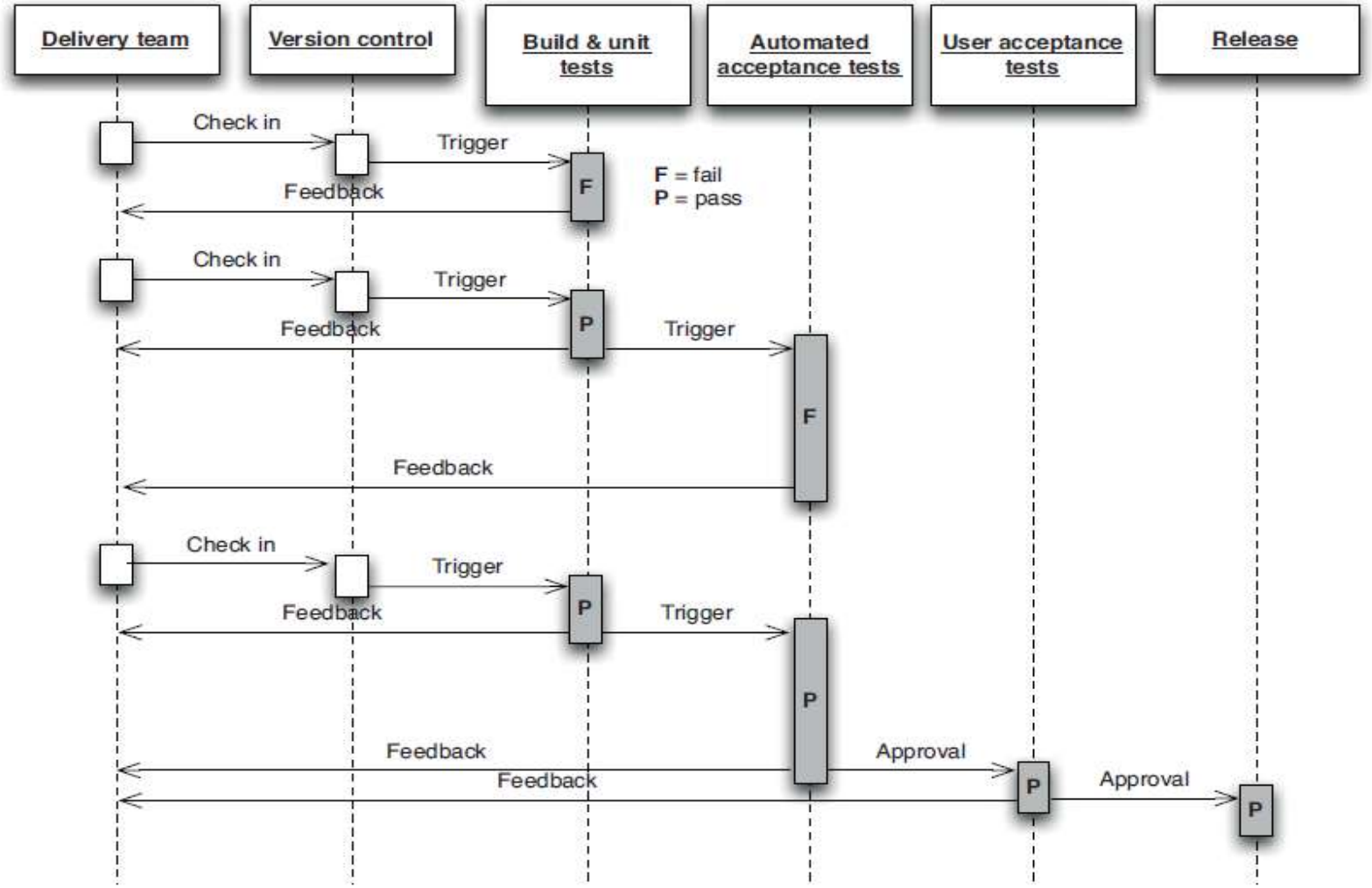


Structure of Deployment Pipeline

Expected steps to be followed

- The input to the pipeline is a particular revision in version control
- Every change creates a build
- It pass through a sequence of tests and challenges to, its viability as a production release
- As the build passes each test of its fitness, confidence in it increases
- The objective is to eliminate unfit release candidates as early in the process as we can and get feedback on the root cause of failure to the team as rapidly as possible
- Any build that fails a stage in the process will not generally be promoted to the next

Structure of Deployment Pipeline



Structure of Deployment Pipeline

Consequences

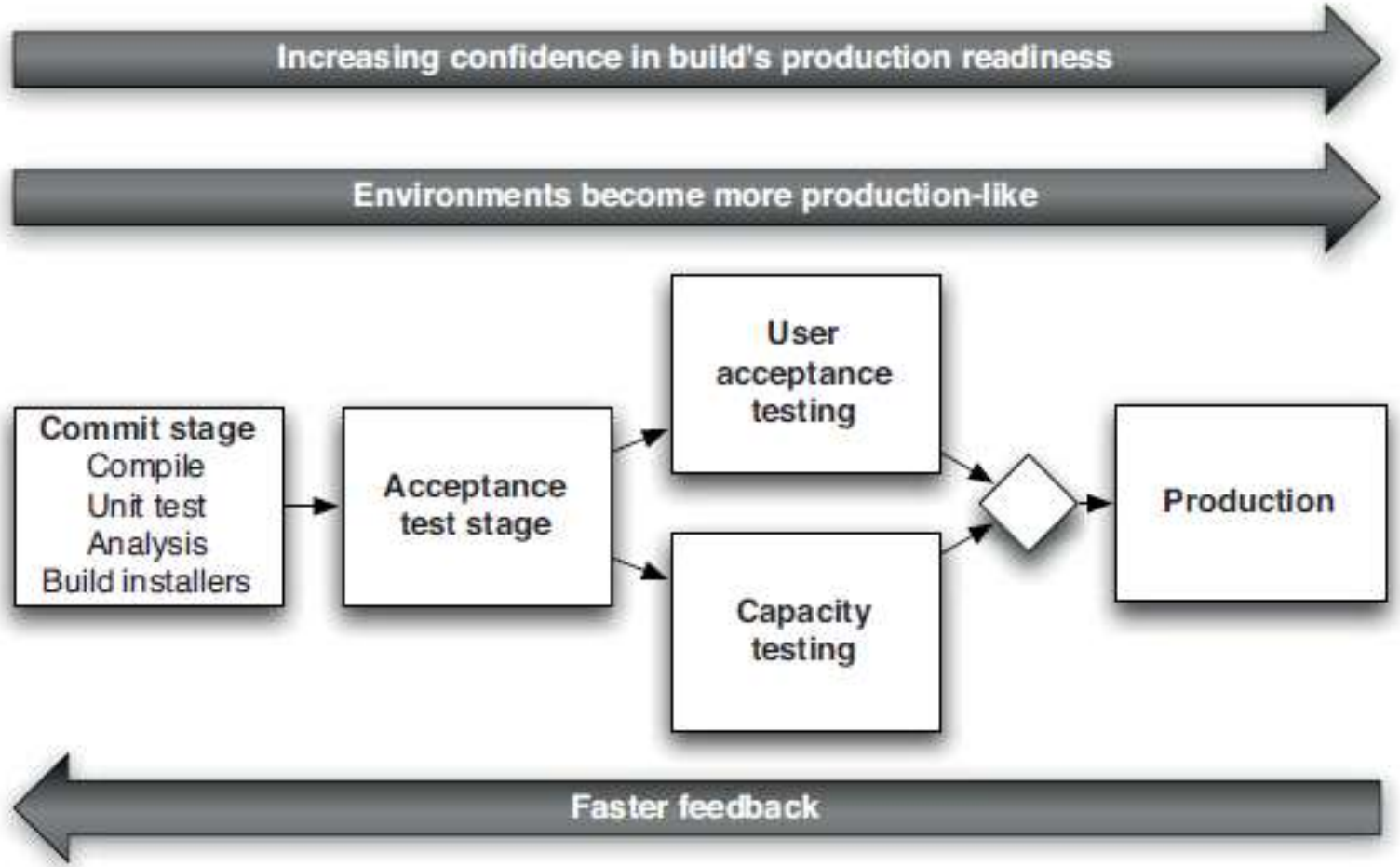
- It is also extremely common for newly released software to break down due to some unforeseen interaction between the components of the system and its environment, for example due to a new network topology or a slight difference in the configuration of a production server
- The other worst that can happen is that after finding a critical bug revert to an earlier version that doesn’t contain the bug while you fix the new release offline

Structure of Deployment Pipeline

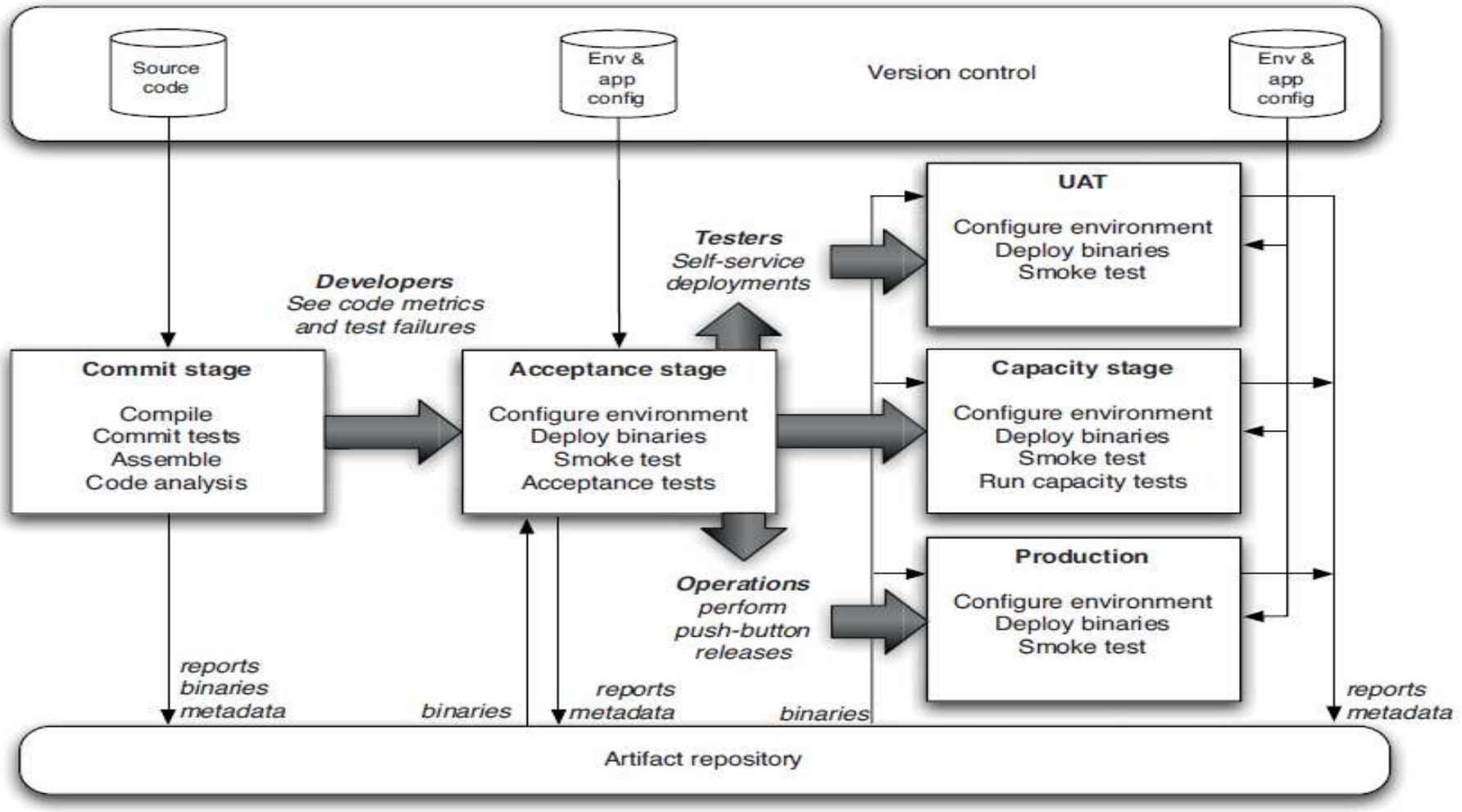
Importance of Automated Release Management

- Must automate a suite of tests that prove that our release candidates are fit for their purpose
- The commit stage asserts that the system works at the technical level : It compiles, passes a suite of automated tests, and runs code analysis
- Automated acceptance test stages assert that the system works at the functional and nonfunctional level; that behaviorally it meets the needs of its users and the specifications of the customer
- Manual test stages assert that the system is usable and fulfills its requirements; detect any defects not caught by automated tests, and verify that it provides value to its users, these stages might typically include exploratory testing environments, integration environments, and UAT (user acceptance testing)
- Release stage delivers the system to users, either as packaged software or by deploying it into a production or staging environment

Structure of Deployment Pipeline



Basic Deployment Pipeline



Basic Deployment Pipeline

Outcome

- The ultimate purpose of all this is to get feedback as fast as possible
- To make the feedback cycle fast
- To be able to see which build is deployed into which environment and
- Which stages in your pipeline each build has passed

Note: It means that if we see a problem in the acceptance tests, can immediately find out which changes were checked into version control that resulted in the acceptance tests failing

Antipatterns

Antipatterns of dealing with Binaries

- For convenience, we will refer to the collections of executable code as binaries, although if you don’t need to compile your code these “binaries” may be just collections of source files
- Examples : Jars, .NET assemblies, and .so files
- The source code will be compiled repeatedly in different contexts: during the commit process, again at acceptance test time, again for capacity testing, and often once for each separate deployment target
- Every time you compile the code, you run the risk of introducing some difference
- The version of the compiler installed in the later stages may be different from the version that you used for your commit tests
- You may pick up a different version of some third-party library that you didn’t intend
- Even the configuration of the compiler may change the behavior of the application

Antipatterns

Antipatterns of dealing with Binaries

- The above antipatterns violates two Principles:
  - Efficiency of Deployment pipeline:
    - Recompiling violates this principle because it takes time, especially in large systems
    - Delayed feedback
  - Always build upon foundations:
    - The binaries that get deployed into production should be exactly the same as those that went through the acceptance test process
    - Recreation of binaries violates this principle
- Scenario: Some organizations insist that compilation and assembly, or packaging in the case of interpreted languages, occurs in a special environment that cannot be accessed by anyone except senior personnel; however this also carries a low amount of risk

Note: If we re-create binaries, we run the risk that some change will be introduced between the creation of the binaries and their release

Deployment Pipeline Practices

Only Build Your Binaries Once

- Should only build binaries once, during the commit stage of the build
- These binaries should be stored on a filesystem somewhere
- Most CI servers will handle this for us, and will also perform the crucial task of allowing you to trace back to the version control check-in which was used to create them
- It isn't worth spending a lot of time and effort ensuring binaries are backed up
- It should be possible to exactly re-create them by running your automated build process from the correct revision in version control
- One important result of this principle is that it must be possible to deploy these binaries to every environment

Note: If your build creates binaries that only run on specific machines, start planning how to restructure them now

Deployment Pipeline Practices

Deploy the Same Way to Every Environment:

- It is essential to use the same process to deploy to every environment
- Example: A developer or analyst's workstation, a testing environment, or production
- In order to ensure that the build and deployment process is tested effectively
- The environment you deploy to least frequently (production) is the most important
- Every environment is different in some way:
  - It will have a unique IP address
  - operating system and middleware configuration settings,
  - the location of databases
  - And external services
- This does not mean we should use a different deployment script for every environment; Instead, keep the settings that are unique for each environment separate
- Use properties files to hold configuration information

Note: Using the same script to deploy to production that you use to deploy to development environments is a fantastic way to prevent the "it works on my machine" syndrome

Deployment Pipeline Practices

Smoke-Test Your Deployments

- When we deploy an application, we should have an automated script that does a smoke test to make sure that it is up and running
- Example: This could be as simple as launching the application and checking to make sure that the main screen comes up with the expected content
- Smoke test should also check that any services your application depends on are up and running—such as a database, or external service
- It gives you the confidence that your application actually runs
- If it doesn’t run, smoke test should be able to give some basic diagnostics as to whether an application is down because something it depends on is not working

Deployment Pipeline Practices

Deploy into a Copy of Production

- The other main problem many teams experience going live is that their production environment is significantly different from their testing and development environments
- To get a good level of confidence that going live, you need to do your testing and continuous integration on environments that are as similar as possible to your production environment
- Ideally, if production environment is simple or we have a sufficiently large budget, we can have exact copies of production to run our manual and automated tests on
- Making sure that two environments are the same requires a certain amount of discipline to apply good configuration management practices:
  - infrastructure, such as network topology and firewall configuration, is the same
  - operating system configuration, including patches, is the same
  - application stack is the same
  - application's data is in a known, valid state

Deployment Pipeline Practices

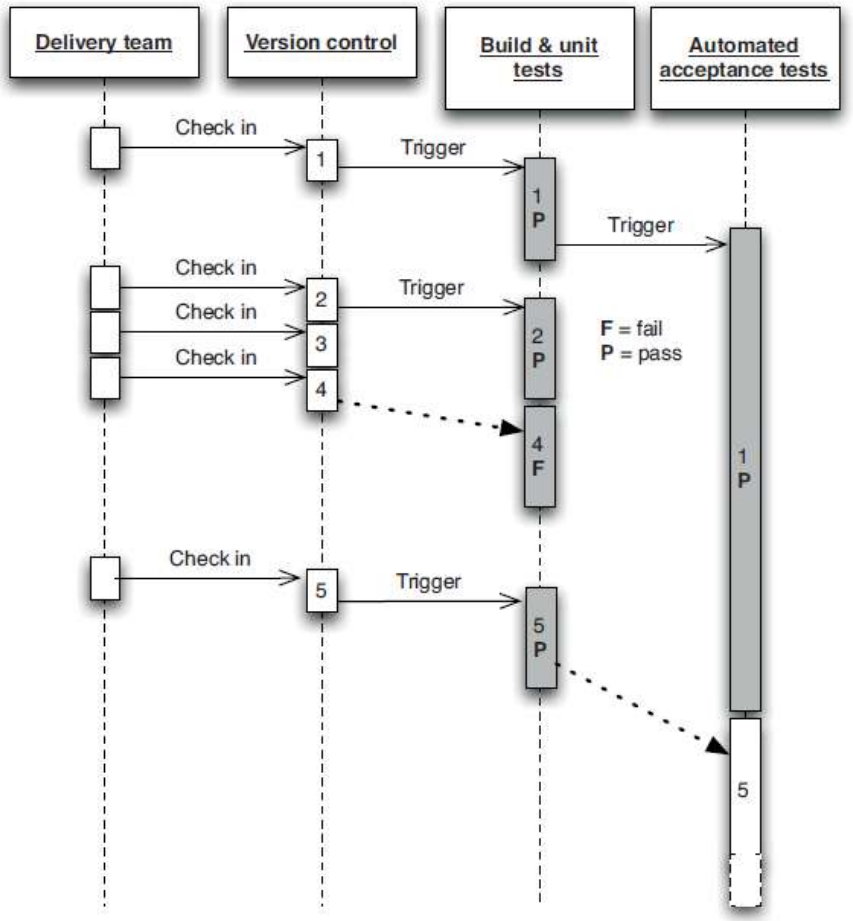
Each Change Should Propagate through the Pipeline Instantly

- Before continuous integration was introduced, many projects ran various parts of their process off a schedule
- For example, builds might run hourly, acceptance tests nightly, and capacity tests over the weekend
- The deployment pipeline takes a different approach:
  - The first stage should be triggered upon every check-in
  - And each stage should trigger the next one immediately upon successful completion
- However this is not always possible when developers especially on large teams are checking in very frequently, given that the stages in your process can take a not insignificant amount of time

Deployment Pipeline Practices

Each Change Should Propagate through the Pipeline Instantly Contd..

- Intelligent scheduling is crucial to implementing a deployment pipeline
- Make sure your CI server ensure that changes propagate immediately so that you don’t have to run stages off a fixed schedule
- This only applies to stages that are fully automated, such as those containing automated tests



Deployment Pipeline Practices

If Any Part of the Pipeline Fails, Stop the Line

- As we said in the “Implementing Continuous Integration” that every time team check code into version control, it will successfully build and pass every test
- This applies to the entire deployment pipeline
- If a deployment to an environment fails, the whole team owns that failure
- They should stop and fix it before doing anything else

Deployment Pipeline

Preparing to Release

- There is a business risk associated with every release of a production system
- At Best if there is a serious problem at the point of release, it may delay the introduction of valuable new capabilities
- At worst, if there is no sensible back-out plan in place, it may leave the business without mission-critical resources because they had to be decommissioned as part of the release of the new system
- The mitigation of these problems is very simple when we view the release step as a natural outcome of our deployment pipeline

Deployment Pipeline

Release Plan

- Have a release plan that is created and maintained by everybody involved in delivering the software, including developers and testers, as well as operations, infrastructure, and support personnel
- Minimize the effect of people making mistakes by automating as much of the process as possible
- Practice the procedure often in production-like environments, so you can debug the process and the technology supporting it
- Have the ability to back out a release if things don’t go according to plan
- Have a strategy for migrating configuration and production data as part of the upgrade and rollback processes

Note: Goal is a completely automated release process, Releasing should be as simple as choosing a version of the application to release and pressing a button and Backing out should be just as simple

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

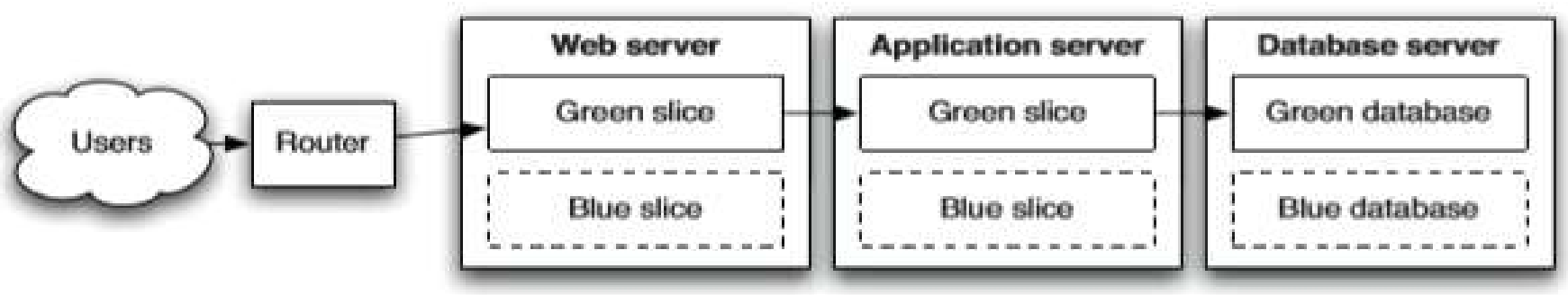
Deployment Strategies

- Blue Green Deployment
- Canary Releasing
- Rolling Upgrade

Blue-Green Deployment

Introduction

- This is one of the most powerful techniques we know for managing releases
- The idea is to have two identical versions of your production environment, which we'll call blue and green



Blue-Green Deployment

Challenges

- It is usually not possible to switch over directly from the green database to the blue database because it takes time to migrate the data from one release to the next if there are schema changes
- Solutions:
- Put the application into read-only mode shortly before switchover
- Another approach is to design your application so that you can migrate the database independently of the upgrade process

Blue-Green Deployment

Setup

- If you can only afford a single production environment, you can still use blue-green deployments
- First approach
  - Simply have two copies of your application running side by side on the same environment
  - Each copy has its own resources: its own ports, its own root on the filesystem, and so forth
  - So they can both be running simultaneously without interfering with each other. You can deploy to each environment independently
- Another approach would be to use virtualization, although you should first test the effect of virtualization on the capacity of your application

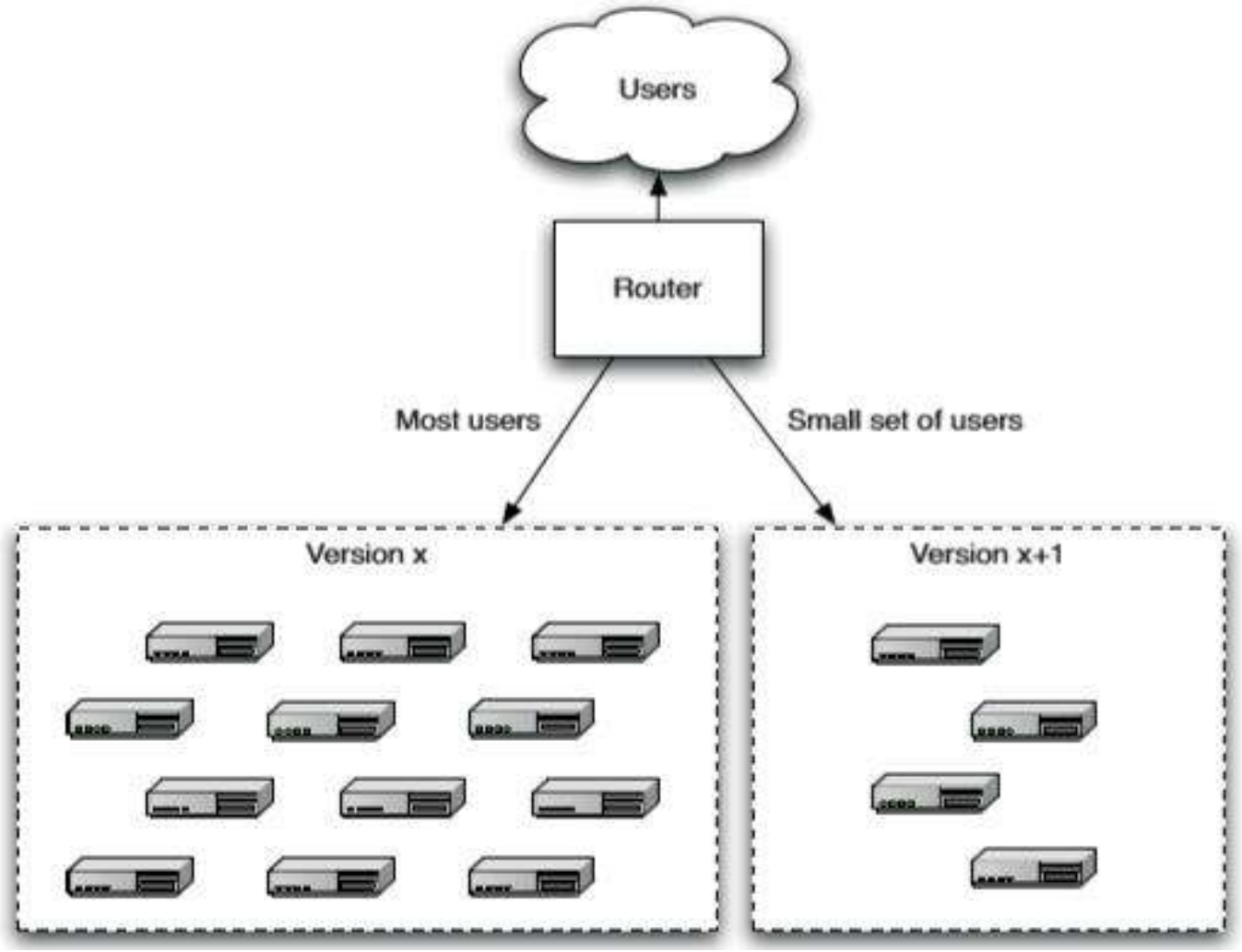
Canary Releasing

Introduction

- Safe assumption that to only have one version of a software in production at a time, this makes it much easier to manage bugfixes, and indeed infrastructure in general
- It also presents an weakness to testing a software, defects pop up in production
- In an extremely large production environment, it’s impossible to create a meaningful capacity testing environment (unless an application’s architecture employs end-to-end sharing)
- How to ensure a new version of an application won’t perform poorly?
- Canary releasing aims to address these challenges
- It Involves rolling out a new version of an application to a subset of the production servers to get fast feedback
- Like a canary in a coal mine, this quickly uncovers any problems with the new version without impacting the majority of users

Canary Releasing

Introduction Contd



Canary Releasing

Benefits

1. It makes rolling back easy: Just stop routing users to the bad version, and you can investigate the logs at your freedom
2. One can use it for A/B testing by routing some users to the new version and some to the old version
   - Some companies measure the usage of new features, and kill them if not enough people use them
   - Others measure the actual revenue generated by the new version, and roll back if the revenue for the new version is lower.
3. You can check if the application meets capacity requirements by gradually ramping up the load, slowly routing more and more users to the application and measuring the application's response time and metrics like CPU usage, I/O, and memory usage, and watching for exceptions in logs

Rolling Upgrade

Introduction

- A rolling upgrade consists of deploying a small number of new version systems at a time directly to the production environment
- In this deployment simultaneously you turn off the old version system
- Before you remove the original system; make sure the new version system is serving the purpose
- Benefits:
  - Cost Effective
  - Risk Effective

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Monitoring

- Introduction to Monitoring
- Consideration for Monitoring
- Goals of Monitoring
- Agentless Monitoring
- Monitoring Operation Activities
- Monitoring Data

Monitoring

Introduction

- Monitoring has a long history in software development and operation
- The earliest monitors were hardware devices like oscilloscopes, and such hardware devices still exist in the monitoring ecosystem
- The term monitoring is referred for the process of observing and recording system state changes and data flows
- State changes can be expressed by direct measurement of the state or by logs recording updates that impact part of the state
- Data flows can be captured by logging requests and responses between both internal components and external systems

Monitoring

Introduction Contd..

- As per Richard Hamming: “The purpose of computing is insight, not numbers”
- The insights available from monitoring fall into five different categories
  1. Identifying failures and the associated faults both at runtime and during postmortems held after a failure has occurred
  2. Identifying performance problems of both individual systems and collections of interacting systems
  3. Characterizing workload for both short- and long-term capacity planning and billing purposes
  4. Measuring user reactions to various types of interfaces or business offerings
  5. Detecting intruders who are attempting to break into the system

Monitoring

Consideration for Monitoring

- The data to be monitored for the most part comes from the various levels of the stack

Goal of Monitoring	Source of Data
Failure Detection	Application and Infrastructure
Performance Degradation Detection	Application and Infrastructure
Capacity Planning	Application and Infrastructure
User reaction to business offerings	Application
Intruder detection	Application and Infrastructure

Goals of Monitoring

Failure detection

- Failures of any element in physical infrastructure is possible; the cause can be anything from overheating to mice eating the cables
- It could be total failure or partial failure
- The total failures are relatively easy to detect: No data is flowing where data used to flow
- The partial failures that are difficult to detect
  - Example:
    - A cable is not firmly seated and degrades performance
    - Before a machine totally fails because of overheating it experiences intermittent failure
- Hardware Failure:
  - Detecting failure of the physical infrastructure is the datacenter provider’s problem
  - Instrumenting the operating system or its virtual equivalent will provide the data for the datacenter
- Software Failure:
  - Dependency Software failure
  - Software Misconfiguration

Goals of Monitoring

Performance Degradation Detection

- It is the most common use of monitoring data
- Degraded performance can be observed by comparing current performance to historical data—or by complaints from clients or end users
- Ideally your monitoring system catches performance degradation before users are impacted at a notable strength
- Performance measures include:
  - Latency
  - Throughput
  - And Utilization

Goals of Monitoring

Capacity Planning

- Types:
  - Long-term Capacity Planning : This capacity planning is intended to match hardware needs, whether real or virtualized, with workload requirements
  - Short-term Capacity Planning : In this capacity planning the context of a virtualized environment such as the cloud means creating a new virtual machine (VM) for an application or deleting an existing VM
- Long-term capacity planning involves humans and has a time frame on the order of days, weeks, months, or even years
- Short-term capacity planning is performed automatically and has a time frame on the order of minutes

Goals of Monitoring

User Interaction

- User satisfaction is an important element of a business
- User satisfaction depends on four elements:
  - The latency of a user request:
    - Users expect decent response times
    - Example: Google reports that delaying a search results page by 100ms to 400ms has a measurable impact on the number of searches that users perform
  - The reliability of the system with which the user is interacting:
    - This is more related to failure and failure detection
  - User interface modification
  - The organization’s particular set of metrics:
    - Every organization has a set of metrics that it uses to determine the effectiveness of their offerings and their support services
    - Example: For a photo gallery website, you may be interested specifically in metrics like photo upload rates, photo sizes, photo processing times, photo popularity, advertisement click-through rates, and levels of user activity

Goals of Monitoring

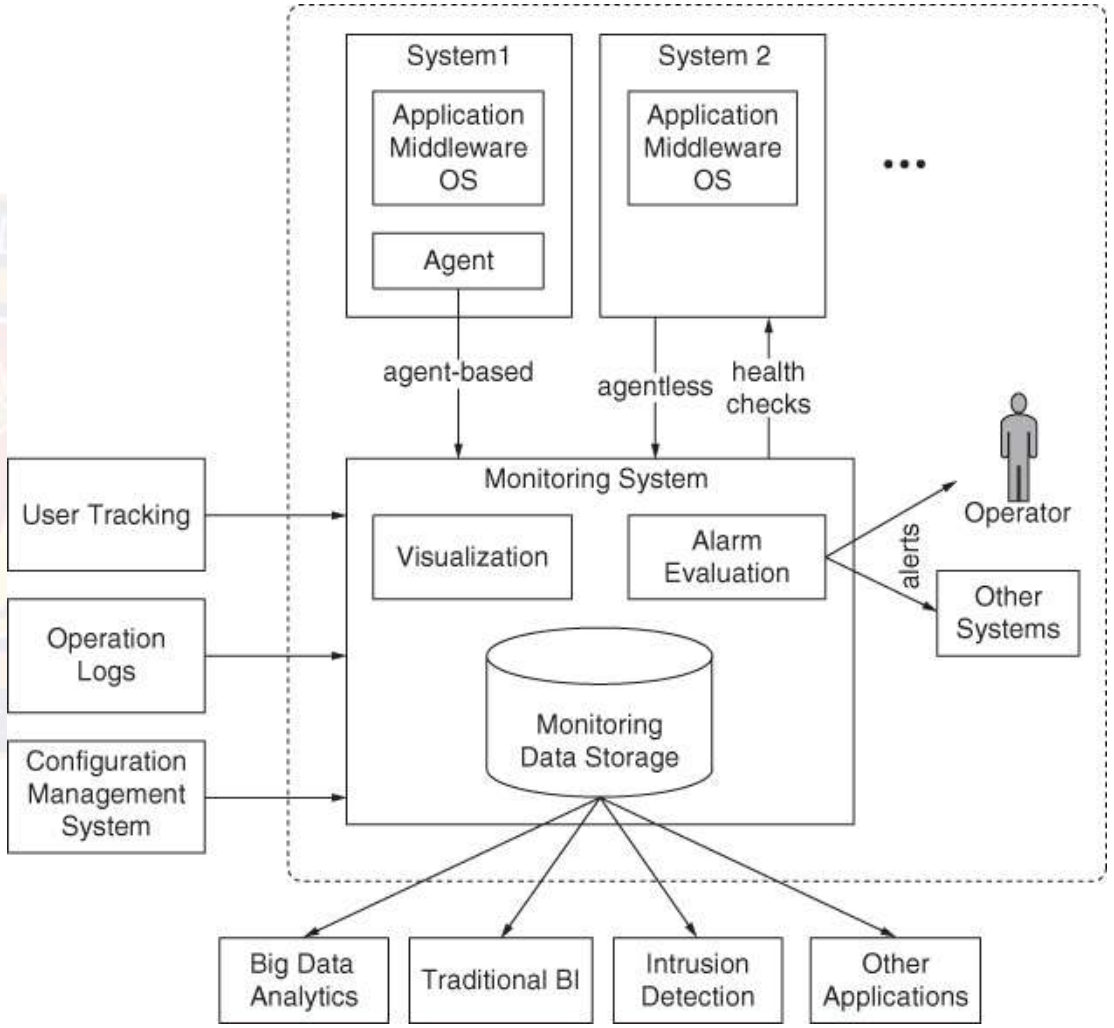
Intrusion Detection

- Intruders can break into a system by disrupting an application
  - Example: Through incorrect authorization
- Applications can monitor users and their activities to determine whether the activities are consistent with the users' role in the organization
  - For Example: For instance, if user John has a mobile phone using the application, and the phone is currently in Australia, any log-in attempts from, say, Nigeria should be seen as suspicious
- An intrusion detector is a software application that monitors network traffic by looking for abnormalities
- Intrusion detectors use a variety of different techniques to identify attacks:
  - They use historical data from an organization's network to understand what is normal
  - They use libraries that contain the network traffic patterns observed during various attacks
  - Example: Current traffic on network vs Expected traffic in historical data
    - The organization may have a policy disallowing external traffic on particular ports

Monitoring

Agent based and Agentless Monitoring

- If the system is actively contributing to the data being monitored (the arrow labeled “agentless”) then the monitoring is intrusive and affects the system design
- If the system is not actively contributing to the data being monitored (the arrow labeled “agent-based”) then the monitoring is nonintrusive and does not affect the system design
- Health Check: External systems can also monitor system or application-level states through health checks, performance-related requests, or transaction monitoring



Monitoring

Agentless Monitoring

- Agentless monitoring is particularly useful when you cannot install agents, and it can simplify the deployment of your monitoring system
- It is especially useful on network equipment because that equipment often comes as a closed system and you cannot install monitoring agents
- You can use protocols like Secure Shell (SSH) to remotely access a system and retrieve available data

Monitoring Operation Activities

Monitoring Operations

- Operations tools monitor resources such as configuration settings to determine whether they conform to pre-specified settings and monitor resource specification files to identify any changes
- Both of these types of monitoring are best done by agents that periodically sample the actual values and the files that specify those values
- There are different Operation Tools like Chef, Puppet, Saltstack and Ansible etc
- The offerings of different configuration management tools now available with both Agent Based and AgentLess

Monitoring Operation Activities

Collection and Storage

- The core of monitoring is recoding and analyzing time series data, namely, a sequence of time-stamped data points
- The data collected is acquired at successive intervals in time and represent certain state or state changes
- The system being monitored will generate time-stamped event notifications at various levels of severity
- The monitoring system can conduct direct measurement or collect existing data, statistics, or logs and then turn them into metrics
- The data is then transferred to a repository using a predefined protocol
- The incoming data need to be processed into a time series and stored in a time series database

Monitoring Data

Log

- A log is a sequence of records ordered by time
- Logs usually record the actions performed that may result in a state change of the system; however the changed value itself may not be included in the log
- Logs play an important role in monitoring
- Application Logging: programmers are familiar with application logging, where they print out system states and actions to assist their development, testing, and debugging activities
- Most logging will then be turned off or removed for production deployment, so that only warnings and critical information will be logged and displayed
- The sources of logs are:
  - Applications
  - Web servers
  - Database systems
  - DevOps pipeline
  - Another Logs by
    - Operations tools
    - An upgrade tool
    - Migration tool
    - Configuration management tool

Monitoring Data

Log Contd..

- Logs are used:
  - During operations to detect and diagnose problems
  - During debugging to detect errors
  - During post-problem forensics to understand the sequence that led to a particular problem
- General rules about writing logs:
  - Logs should have a consistent format
  - Logs should include an explanation for why this particular log message was produced
  - Log entries should include information to support tracking the log entry such as:
    - Source of the log entry within the code
    - Process ID for the process executing when the log entry was produced
    - Request ID for the request that caused that process to execute this log producer
    - VM ID for the VM that produced this message
  - Logs should provide screening information

Monitoring Data

Graphing and Display

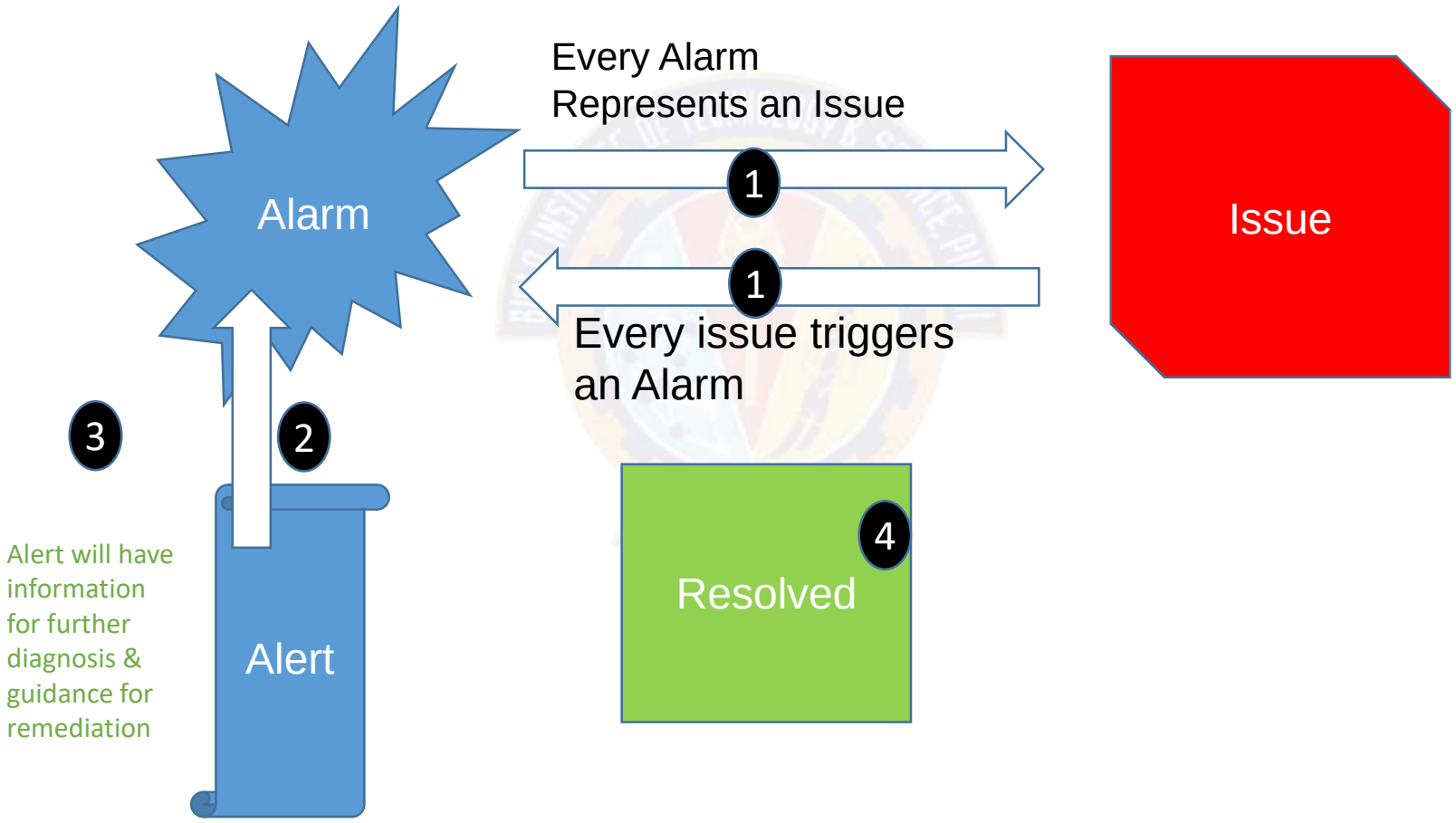
- It is useful to visualize all relevant data collected by monitoring system
- Most monitoring data is time series data, which is agreeable to plotting
- A flexible system should allow you to have full control over what to plot and how
  - Example: Graphite is a specialized systems for visualization and querying, which support real-time graphing of large amounts of data
- Monitoring systems should have strong visualization capabilities embedded
  - A dashboard showing important real-time aspects of your system and its components at an aggregated level
  - Allow interactive navigation through history when you detect an issue
  - Should provide visual patterns of graphs to determine problems
  - The graphs may show spikes, bursts, cyclic variation, steadily trending up/down, or sparse events, all of which need to be understood in the context of characteristics of the state being monitored

Monitoring Data

Alarms and Alerts

- Monitoring systems inform the operator of significant events in the form of either an alarm or an alert
- Alerts:
  - Alerts are raised for purposes of informing
  - Alerts are raised in advance of an alarm
  - Example: The datacenter temperature is rising
- Alarms:
  - Alarms require action by the operator
  - Or
  - Alarms require action by another system
  - Example: The datacenter is on fire
- Alarms and alerts are generated based on configurations set by the operators

Alarms and Alert Contd..



Monitoring Data

Diagnosis and Reaction

- Monitoring systems are not designed for interactive or automated diagnosis
- Operator’s use monitoring systems to diagnose the causes and observe the progress of mitigation and recovery
- So, Operator’s, in ad hoc ways, will try to correlate events, dive into details and execute queries, and examine logs
- Operator’s trigger more diagnostic tests and recovery actions and observe their effects from the monitoring system
- Damon Edwards lists five things that are important to monitor:
  - A business metric
  - Cycle time
  - Mean time to detect errors
  - Mean time to report errors
  - Amount of scrap (rework)

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Monitoring in DevOps

- Challenges in Monitoring by DevOps & Probable Solutions

Monitoring in DevOps

Challenges in Monitoring by DevOps

- Challenge 1: Monitoring Under Continuous Changes
- Challenge 2: Bottom-Up vs. Top-Down and Monitoring in the Cloud
- Challenge 3: Monitoring a Microservice Architecture
- Challenge 4: Dealing with Large Volumes of Distributed (Log) Data

Monitoring in DevOps

Challenges in Monitoring by DevOps Contd..

- **Challenge 1: Monitoring Under Continuous Changes:**
- The new challenges come from both cloud elasticity, making infrastructure resources more volatile, and the automated DevOps operations, which trigger various periodic operations (such as upgrade, reconfiguration, or backups)
- Periodic operations and continuous deployment and deployment practices make software changes more frequent
- As we have seen, deploying a new version into production multiple times a day is becoming a common practice
- However the fact is:
  - Each deployment is a change to the system and may impact monitoring
  - And these changes may be happening simultaneously in different portions of an application or the infrastructure

Monitoring in DevOps

Challenges in Monitoring by DevOps Contd..

- Challenge:
- To what extent can you use the past monitoring data of your system to do performance management, capacity planning, anomaly detection, and error diagnosis for the new system?

- As a solution to the above problem “In practice, operators may turn off monitoring during scheduled maintenance and upgrades as a work-around to reduce false positive alerts triggered by those changes, When change is the norm (Usual & Expected), this can lead to no monitoring—for example, flying blind”
- Alternate technique is to carefully identify the non-changing portions of the data
  - For example, use dimensionless data (i.e., ratios)
  - We may find that although individual variables change frequently, the ratio of two variables is relatively constant
- Also try to focus monitoring on those things that have changed

Monitoring in DevOps

Challenges in Monitoring by DevOps Contd..

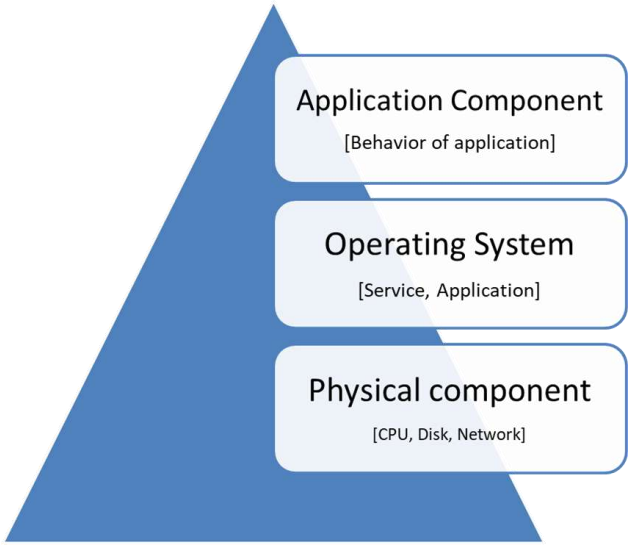
- Challenge:
- Continuous changes in the system infrastructure and the system itself complicate the setting of monitoring parameters?

- Here the solution is to automate the configuration of alarms, alerts, and thresholds as much as possible; the monitoring configuration process is just another DevOps process that can and should be automated
- Example:
  - When you provision a new server, a part of the job is to register this server in the monitoring system automatically
  - When a server is terminated, a de-registration process should happen automatically
  - For example, the monitoring results during canary testing for a small set of servers can be the new baseline for the full system and populated automatically

Monitoring in DevOps

Challenges in Monitoring by DevOps Contd..

- **Challenge 2: Bottom-Up vs. Top-Down and Monitoring in the Cloud**
- One major goal of monitoring is to detect faults, errors, or small-scale failures as soon as possible, so you can react to them early
- To fulfil this goal, it is natural to monitor in a bottom-up fashion



Monitoring in DevOps

Challenges in Monitoring by DevOps Contd..

- Challenge:
- Scenario1: We may have a single application that is spanned across several components deployed on hundreds of servers, which are in turn supported by multiple networks and storage components; It can be very tricky to identify the root cause in a real-world environment
- Scenario2: A second challenge is related to continuous change caused by cloud elasticity and automation; Consider in the cloud, servers come and go for both termination for preventing server drifts, scaling out/in, and rolling upgrades as well as for reasons like instance failures or resource sharing uncertainty, In this case it will be difficult to identify the reason for failures

- Adopting a more top-down approach for monitoring cloud-based and highly complex systems is an attempt to solve these problems
  - You monitor the top level or aggregated data and only dive into the lower-level data in a smart way if you notice issues at the top level
  - The lower-level data must still be collected but not systematically monitored for errors
- Risk : By above solution you are sacrificing the opportunity to notice issues earlier; and it might already be too late to prevent a bigger impact once you notice that something is wrong at the top level

Note: There is no easy solution, bottom-up and top-down monitoring are both important and should be combined in practice

Monitoring in DevOps

Challenges in Monitoring by DevOps Contd..

- **Challenge 3: Monitoring a Microservice Architecture**
- Adoption of a microservice architecture enables having an independent team for each microservice
- Every external request may potentially travel through a large number of internal services before an answer is returned
- In a large-scale system, one part or another may experience some slowdown at any given time, which may consequently lead to a negative impact on an unacceptable portion of the overall requests
- Challenge:
- Micropartitions and selective replication enable easier migration, which can be used to move services away from problematic parts of the network, so monitoring multiple requests for the same service and determining that only one response is necessary becomes quite a challenge
- Need of intelligent monitor systems; one can monitor at microservice level

Monitoring in DevOps

Challenges in Monitoring by DevOps Contd..

- **Challenge 4: Dealing with Large Volumes of Distributed (Log) Data**
- In a large-scale system, monitoring everything will incur a considerable overhead in terms of performance, transmission, and storage
- A large-scale system can easily generate millions of events, metric measurements, and log lines per minute
- Challenge:
- The performance overhead of collecting metrics at a small time interval might be significant
- Operators should use varied and changeable intervals rather than fixed ones, depending on the current situation of the system
  - If there are initial signs of an anomaly or when a sporadic operation is starting, set finer-grained monitoring
  - Return to bigger time intervals when the situation is resolved or the operation completed
- Use a modern distributed logging or messaging system for data collection
  - A distributed logging system such as Logstash can collect all kinds of logs and conduct a lot of local processing before shipping the data off
  - This type of system allows you to reduce performance overhead, remove noise, and even identify errors locally
- *Note: LinkedIn developed Kafka, a high-performance distributed messaging system, largely for log aggregation and monitoring data collection*

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

ELK

- Introduction to ELK
- Elasticsearch
- Logstash
- Kibana
- ELK: Features & Benefits

ELK

Introduction to ELK

- ELK is the acronym for three open source projects
- Elasticsearch, Logstash, and Kibana
- Elasticsearch is a search and analytics engine
- Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch
- Kibana lets users visualize data with charts and graphs in Elasticsearch

ELK

Elasticsearch

- ELK is started with Elasticsearch
- The open source, distributed, RESTful, JSON-based [Key Pair value] search engine
- Easy to use, scalable and flexible, it earned hyper-popularity among users and a company formed around it
- Elasticsearch lets you perform and combine many types of searches — structured, unstructured, and geometric etc
- Elasticsearch aggregations let you zoom out to explore trends and patterns in your data
- Elasticsearch is a NoSQL database that is based on the Lucene search engine

ELK

Logstash

- Logstash is a critical element in ELK Stack
- Logstash can be used to collect, enrich and transform a wide array of different data types
- There are over 200 different plugins for Logstash, with a vast community making use of its extensible features
- Logstash works in three stages collection, processing, and dispatching

ELK

Kibana

- Kibana lets visualize our Elasticsearch data and navigate the Elastic Stack
- Kibana gives us the freedom to select the way we give shape to our data
- Kibana core ships with the classics: histograms, line graphs, pie charts, sunbursts, and more

ELK

Features and Benefits

- Security:
  - Protects Elasticsearch data in a robust and granular way
- Alerting:
  - Get notifications about changes in data
- Monitoring:
  - Maintain a pulse on Elastic Stack to keep it firing on all cylinders
- Reporting:
  - Create and share reports of Kibana charts and dashboards
- Graph:
  - Explore meaningful relationships in data
- Machine Learning:
  - Automate anomaly detection on Elasticsearch data

Reference

- Elastic [https://www.elastic.co]

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Infrastructure as Code [IaC]

- Introduction to Infrastructure as Code
- Before IaC and DevOps
- Infrastructure as Code for DevOps
- Achievement using DevOps and IaC
- IaC Stand
- Benefits of IaC
- Risk Involved in IaC

Infrastructure as Code [IaC]

Introduction to Infrastructure as Code

- Infrastructure as Code, or IaC
- It is a method of writing and deploying machine-readable definition files that generate service components
- IaC helps IT operations teams manage and provision IT infrastructure automatically through code without relying on manual processes
- IaC is often described as “programmable infrastructure”
- Infrastructure as Code is especially critical for cloud computing, Infrastructure as a Services (IaaS), and DevOps
- DevOps requires agile work processes which can be achieved through automated workflows for IT Infrastructure

Infrastructure as Code [IaC]

Before IaC and DevOps

System Admin Team
- Create / Setup Environment
- Try to mirror the live environment

DB Admin Team
- Setup / create Database Environment
- Handover the machine to Test Team

Development Team
- Deliver the code in above Environment and handover it to Testing Team

Test Team
- Test team run several operational and compliance tests
- After all successful process code will hit to Live Production Environment

Infrastructure as Code [IaC]

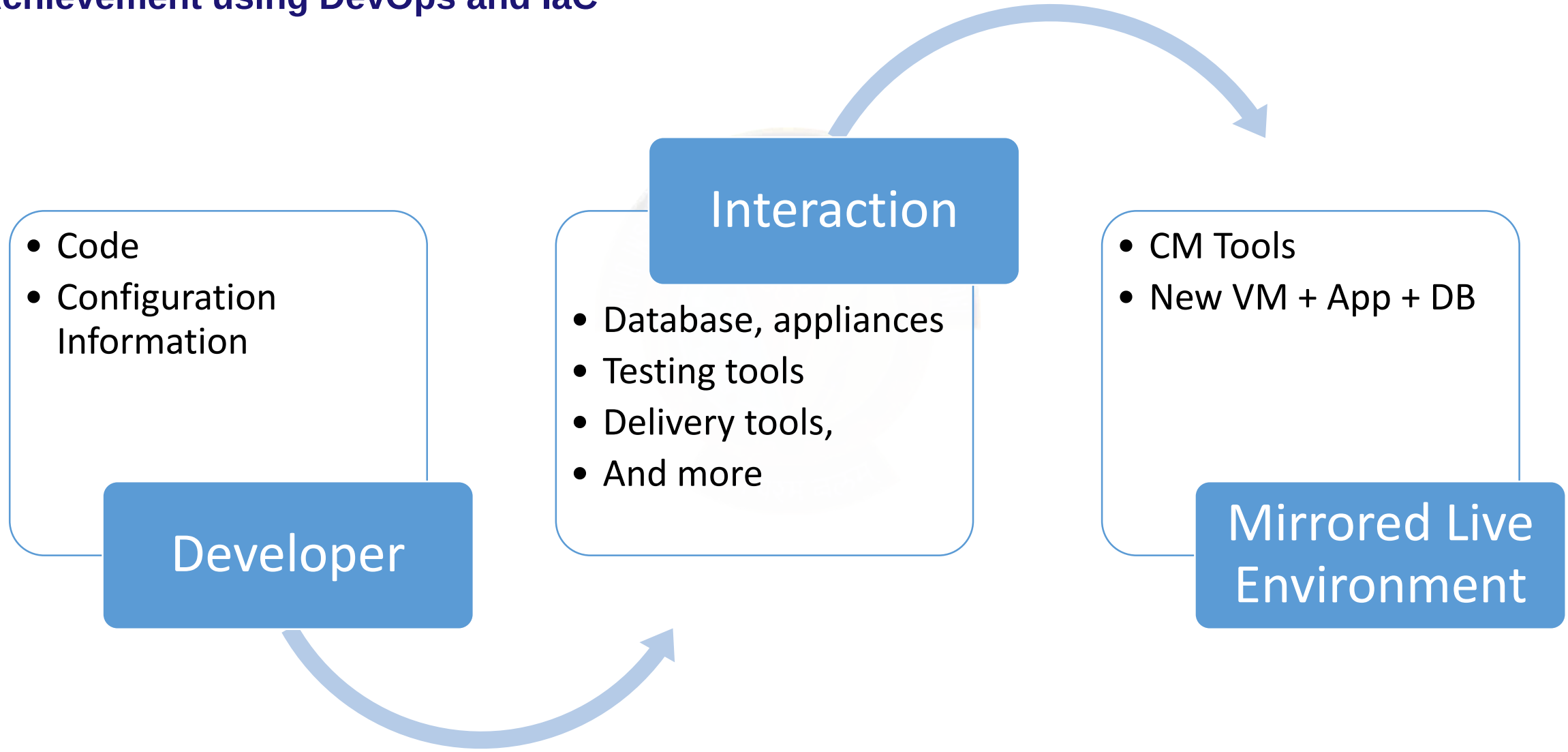
IaC for DevOps

- In generic software development, a fundamental constraint is the need for the environment
- i.e. Testing or Staging environment = Live Environment
- Note: This is the only way of assuring that the new code will not collide with existing code definitions – generating errors or conflicts that may compromise the entire system

- Manual recreation of a live environment leaves doors open to a multitude of most likely minor but potentially quite important human errors, e.g. OS version, patch level, time zone etc
- Where as clone created with the help of IaC can provide absolute guarantee of mirror of Live Production; now imagine a Software Delivery process involving DEV, UAT and Production environments using IaC

Infrastructure as Code [IaC]

Achievement using DevOps and IaC



Infrastructure as Code [IaC]

IaC Stand

- The general IaC concept, as well as available tools, has reached a very mature state with a lot of organizations having defined their roadmaps for adopting it
- There are now a number of tools available to adopt Infrastructure as Code
  - Puppet
  - Chef
  - Ansible
  - Terraform
  - Cloud formation etc

Infrastructure as Code [IaC]

Benefits of IaC

- Reducing Shadow IT:
  - Shadow IT within organizations is due to the inability of IT departments to provide satisfactory and timely answers to operational areas concerning IT infrastructure and systems enhancements
  - This will result to potential unforeseen costs for the organization
  - IaC assures security and compliance with corporate IT standards and also it is helpful with budgeting and cost allocation
- Improving Customer Satisfaction:
  - Being able to deliver a quality service component within a short period of time contributes to customer satisfaction
- Operating Expenses [OPEX]:
  - Save in work time and avoiding human error will help to reduce financial risk
- Capital Expenditure [CAPEX]:
  - Reduce funds used by a company to acquire, upgrade, and maintain physical assets i.e. Single engineer with automation can perform task od multiple people
- Standardization:
  - Assurance of a consistent set of instructions and standardization

Infrastructure as Code [IaC]

Benefits of IaC Contd..

- Safer Change Management:
  - Standardization assurance enables safer changes to take place, with lower risk
- Application of Software Delivery Principles:
  - Ability to promote and use Software Delivery principles, such as version control, peer programming, and code reviews
- Scalable and Absolute Infrastructure

Infrastructure as Code [IaC]

Risk Involved in IaC

- Missing proper planning:
  - Once a company decides to move towards having an IaC; there is the mandatory need to define Infrastructure that will allow the implementation, configuration, and operation of IaC tools
- IaC requires new skills:
  - Most existing IaC tools require expertise to be handled, and reaching such levels requires significant time in learning and training
- Error replication:
  - Since the initial code is developed by humans, there is always the chance that it contains minor errors; several machines may have been automatically created where such errors exist
- Configuration Drift:
  - Once a machine is created via an IaC workflow; Manual or external updates (even if just security patching) may result in configuration drift
- Accidental Destruction:
  - Some IaC tools that maintain state have the ability to automatically destroy resources
  - IaC in an automation pipeline can sometimes have undesired outcomes

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Importance of Configuration Management

- Introduction to Configuration Management
- Traditional Configuration Management
- Configuration Management in DevOps
- Configuration as a Code in DevOps
- Benefits of Configuration Management

Configuration Management

Introduction to Configuration Management

- Configuration management (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life
- There are several components in a configuration management system
- Managed systems can include:
  - Servers
  - Storage
  - Networking
  - And Software
- The above list are the targets of the configuration management system

Configuration Management

Traditional Configuration management

- A Team used to manages all configurations
- via formal documentation
- Where each of the configurations are recorded in the documents
- A configuration team or manager handles the version control of these documents
- And “as and when it undergoes changes”, Change Manager take the responsibility of setting up the environment and managing the configurations manually

Configuration Management

Configuration management in DevOps

- All these configuration management processes are well automated
- The configurations are encapsulated in the form of code or scripts via formal documentation
- Where each of the configurations are recorded in the documents
- All Configuration Changes are controlled through the version control tool

Configuration Management

Configuration management in DevOps Contd..

- Configurations are Version controlled
- Automated and standardized
- Removes dependency
- Error-free infra setups
- Boosts collaboration between Operations and Development team
- Correcting configuration drift
- Treating infrastructure as a flexible resource
- Automated scaling of infrastructure
- Maintaining consistency in the setups

Configuration Management

Configuration management in DevOps Contd..

- The key highlight of configuration management in DevOps is delivering
- Infrastructure as a code : “It is defining the entire environment definition as a code or a script instead of recording in a formal document”
- Configuration as a code : “It is nothing but defining all the configurations of the servers or any other resources as a code or script and checking them into version control”

Configuration Management

Configuration as a code in DevOps

- Defining all the configurations of the servers or any other resources as a code or script
- These configuration scripts need to be checked into the version control
- And are scheduled to run as a part of the deployment pipeline in order to set up the infrastructure
- Defining configurations includes parameters that define the recommended settings for the software to run successfully
- Several tools are available to carry out the infrastructure automation in the market
- Chef
- Puppet
- Terraform
- Ansible

Configuration Management

Benefits of Configuration Management

- The primary benefit of configuration management is consistency of systems and software
- With configuration management, we no longer guess or hope that a configuration is current; It is correct because the configuration management system ensures that it is correct
- When combined with automation, configuration management can improve efficiency because manual configuration processes are replaced with automated processes
- Configuration management is important because it enables the ability to scale infrastructure and software systems without having to correspondingly scale administrative staff to manage those systems

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

On-demand Infrastructure Management

- Introduction to On-demand Infrastructure Management
- On-Demand Infrastructure Architect
- Security still a Major Concern?
- Service Providers
- Benefits of On-demand Infrastructure
- Auto Scaling
- Auto Scaling Approach
- Auto Scaling Providers

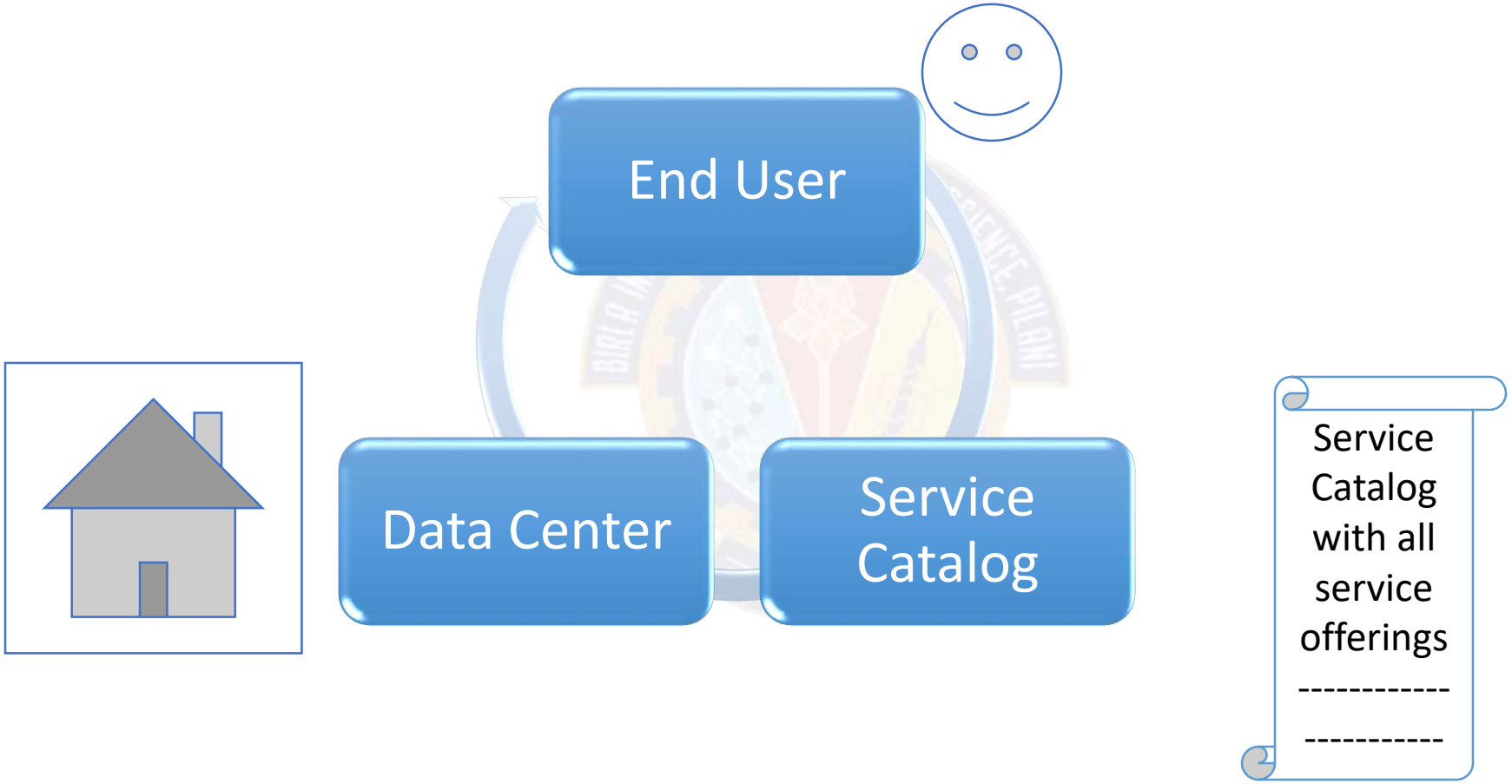
On-demand Infrastructure Management

Introduction to On-demand Infrastructure Management

- New IaaS technology is making it for on-demand infrastructure
- IaaS option provides users with on-demand access, while still giving them a system that is predictable and easy to control
- With IaaS, small businesses no longer have to build infrastructure for data storage internally
- And with IaaS they can pay for what they need as they grow
- Companies like Titanium aims to not only replace cloud storage, but also to provide processor power and memory, so that resource-intensive projects such as artificial intelligence or virtual reality can be accomplished by anyone in the world, at an affordable price

On-demand Infrastructure Management

On-demand Infrastructure Architect



On-demand Infrastructure Management

Security Still A Major Concern?

- According to a recent release by Delta Risk, a cyber security and risk management firm, 90 percent of organizations are currently using cloud services to store data
- Delta Risk found that the top three cloud security concerns are data loss, data privacy and confidentiality breaches
- These are just a few of the companies offering IaaS solutions that address the data access, storage and security needs of small businesses and research indicates these solutions are hitting the mark

On-demand Infrastructure Management

Service Providers

- IaaS Providers:
  - bmc
  - DellEMC
  - Dropbox
  - Cisco
  - Juniper
  - Vmware
- PaaS Providers:
  - Redhat
  - Azure
  - at & t
  - Google
  - Verizon
- SaaS Providers:
  - Salesfroce
  - Netapp
  - Coupa
  - liveops

On-demand Infrastructure Management

Benefits of On-demand Infrastructure

- Cost savings
- Scalability and flexibility
- Faster time to market
- Support for DR and high availability
- Focus on business growth

On-demand Infrastructure Management

Auto Scaling

- Auto scaling, also spelled auto scaling or auto-scaling, and sometimes also called automatic scaling
- It is a method used in cloud computing
- Auto Scaling Offers:
  - For companies running their own web server infrastructure
  - Auto Scaling typically means allowing some servers to go to sleep during times of low load
  - Saving on Power and Energy
  - For companies using infrastructure hosted in the cloud, Auto Scaling can mean lower bills, because most cloud providers charge based on total usage rather than maximum capacity
  - Auto Scaling can help by allowing the company to run less time-sensitive workloads on machines that get freed up by Auto Scaling during times of low traffic
  - Auto Scaling can offer greater uptime and more availability in cases where production workloads are variable and unpredictable

On-demand Infrastructure Management

Auto Scaling Approaches

- Scheduled Auto Scaling Approach:
  - This is an approach to Auto Scaling where changes are made to the minimum size, maximum size, or desired capacity of the Auto Scaling group at specific times of day
  - Scheduled scaling is useful, for instance, if there is a known traffic load increase or decrease at specific times of the day
  - E.g. E-commerce sites: Flipkart Big Billion Day, Amazon's Great India Sale
- Predictive Auto Scaling:
  - The idea is to combine recent usage trends with historical usage data as well as other kinds of data to predict usage in the future, and Auto Scale based on these predictions
  - Identifying huge spikes in demand in the near future and getting capacity ready a little in advance
  - E.g. Online Social and Media Hosting

On-demand Infrastructure Management

Auto Scale Providers

- Amazon Web Services (AWS) : In 2009, Amazon launched its own Auto Scaling feature along with Elastic Load Balancing
  - Netflix is the well know consumer of Auto Scaling at AWS
- Microsoft's Windows Azure : Around 2013, Microsoft announced that Auto Scaling support to its Windows Azure cloud computing platform
- Oracle Cloud Platform: It allows server instances to automatically scale a cluster in or out by defining an Auto Scaling rule
- Google Cloud Platform : In 2015 Google Compute Engine announced a public beta of its Auto Scaling feature for use

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Configuration Management Tools

- Introduction to Configuration Management Tools
- Chef
- Puppet

Configuration Management Tools

Introduction to Configuration Management Tools

- What they Offer in common:
  - Quick Provisioning of New Servers
  - Quick Recovery from Critical Events
  - No More Snowflake Servers
  - Version Control for the Server Environment
  - Replicated Environments
- What they Offer in common for Servers:
  - Automation Framework
  - Idempotent Behavior
  - Templating System
  - Extensibility

Chef

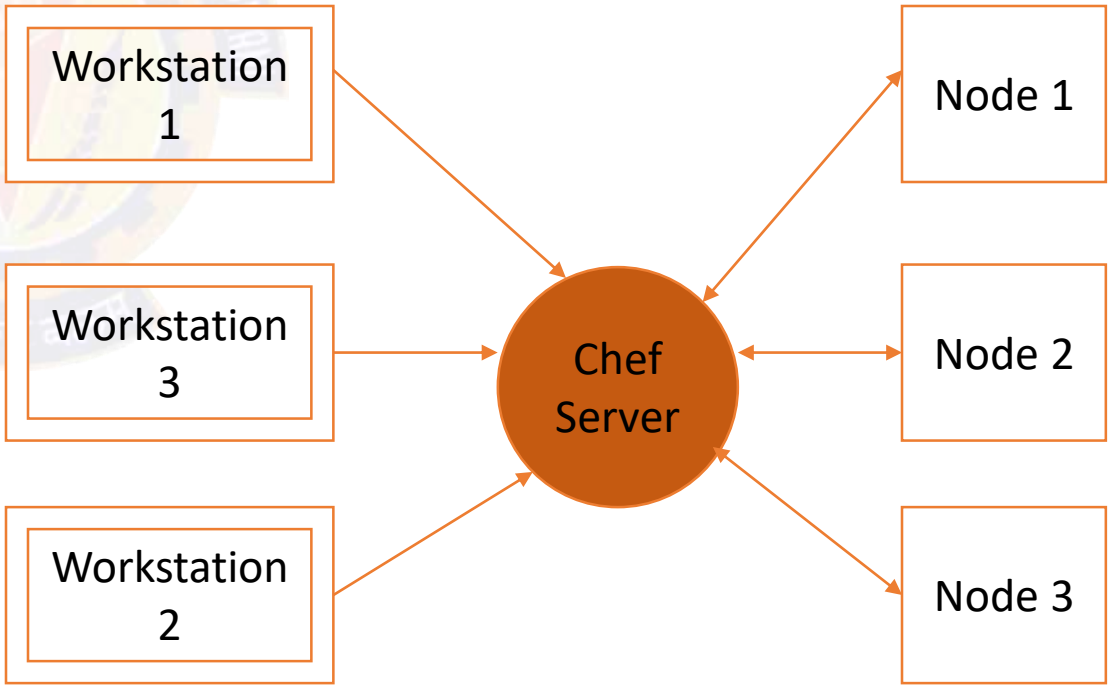
Introduction to Chef

- Chef is a configuration management tool for dealing with machine setup on physical servers, virtual machines and in the cloud
- Many companies use Chef software to control and manage their infrastructure including:
- Facebook
- Etsy
- Cheezburger
- And Indiegogo

Chef

Chef CM Tool

- Chef helps to build infrastructure as code
- Chef Components:
  - The Chef process consists of three core components that interact with one another:
  - Chef server
  - Actual servers called nodes
  - And Chef workstation



Chef

Cookbooks

- A cookbook is the fundamental unit of configuration and policy distribution
- A cookbook defines a scenario and contains everything that is required to support that scenario
- The chef-client uses Ruby as its reference language for creating cookbooks and defining recipes, with an extended DSL for specific resources

Chef

Benefits of Chef CM Tool

- Accelerating Software Delivery
- Increasing Service Resiliency
- Improving Risk Management
- Accelerating Cloud Adoption
- Managing Both Data Center and Cloud Environments
- Delivering All Your Infrastructure – Any App, Everywhere, Continuously

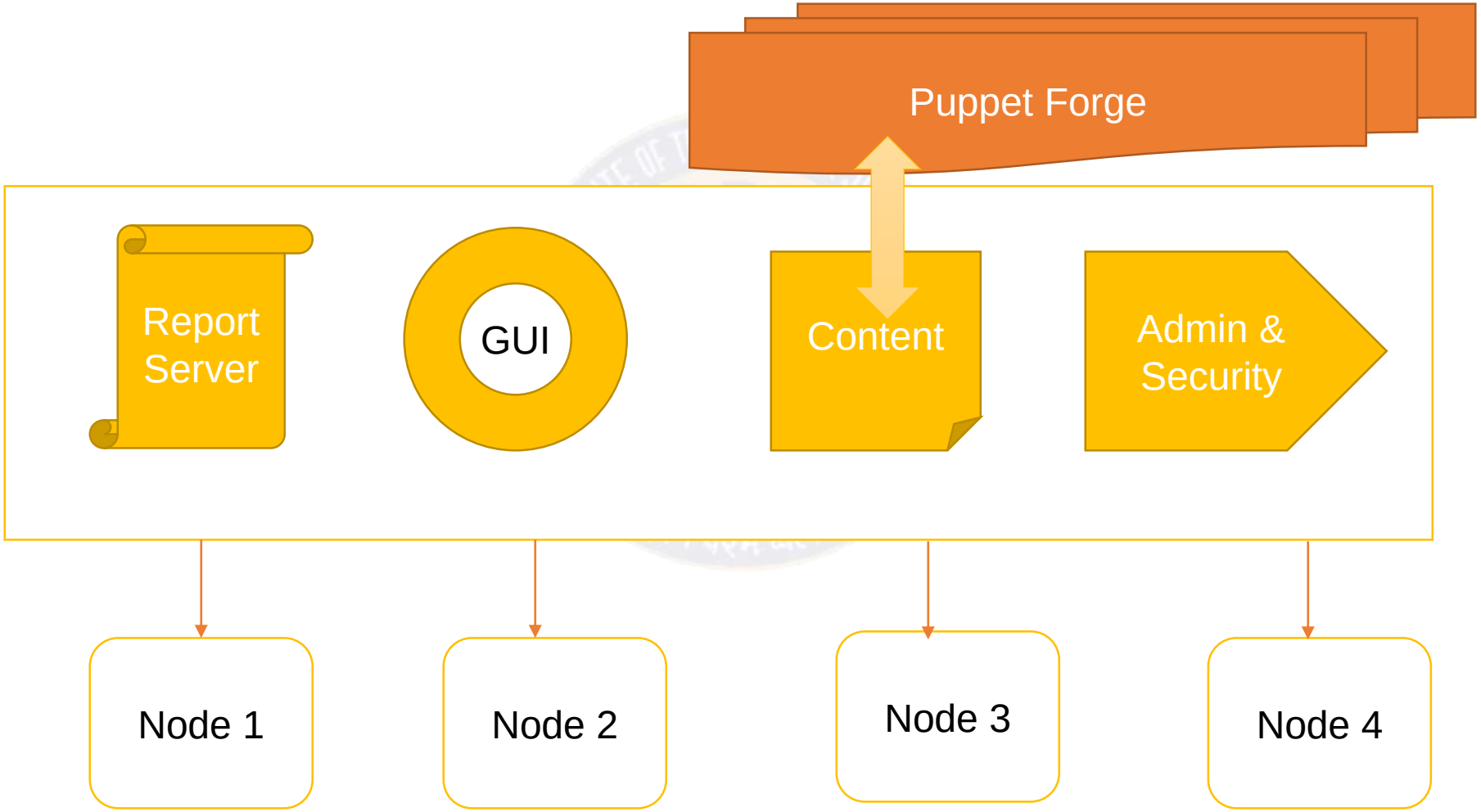
Configuration Management Tool - Puppet

Introduction to Puppet

- Puppet gives you an automatic way to inspect, deliver, operate and future-proof all of your infrastructure and deliver all of your applications faster
- Puppet is an open source IT automation tool that allows IT organizations to encode the configuration of services as a policy
- Puppet Enterprise is proprietary of Puppet Labs

Puppet

Puppet Architecture



Puppet

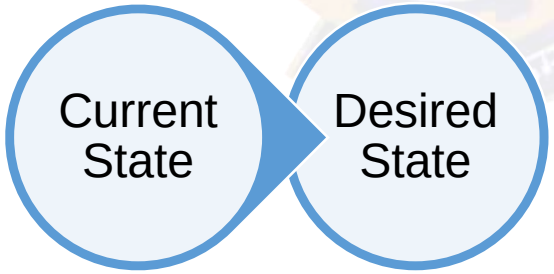
Puppet - Components

- Puppet Server / Master
- Puppet Agent
- Puppet Enterprise Console
- PuppetDB
- Facter
- Puppet Compile Master

Puppet

Puppet Component - Catalog

- The catalog describes the desired state for each resource that should be managed, and may specify dependency information for resources that should be managed in a certain order
- When configuring a node, the agent uses a document called a catalog
- Which it Downloads from Master
- For each resource under management, the catalog describes its desired state
- The Puppet apply command compiles its own catalog



Puppet

Puppet Open Source vs Puppet Enterprise

Function	Puppet	Puppet Enterprise
Graphical User Interface	No	Yes
Configuration management – Discovery	No	Yes
Orchestration –Task automation	No	Yes
Support – option for 24 * 7 * 365	No	Yes
Role Based Access Control	No	Yes

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

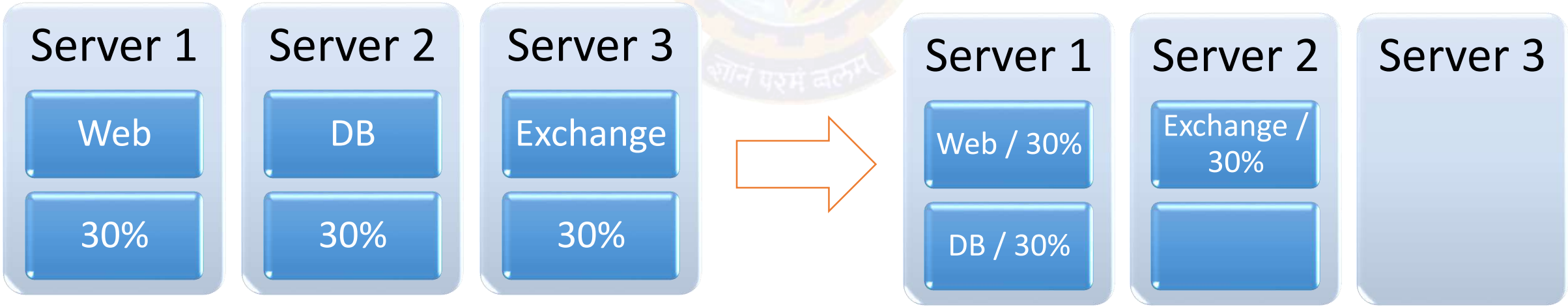
Virtualization

- Introduction to Virtualization
- Virtualization History
- Types of Virtualization
- Hypervisor based Virtualization
- Container based Virtualization
- Service Virtualization as an Enabler of DevOps

Virtualization

Introduction to Virtualization

- Virtualization is technology that lets you create useful IT services using resources that are traditionally bound to hardware
- It allows you to use a physical machine's full capacity by distributing its capabilities among many users or environments



Virtualization

History

- While virtualization technology can be sourced back to the 1960s, it wasn't widely adopted until the early 2000s
- Fast forward to the 1990s. Most enterprises had physical servers and single-vendor IT stacks, which didn't allow legacy apps to run on a different vendor's hardware

Virtualization

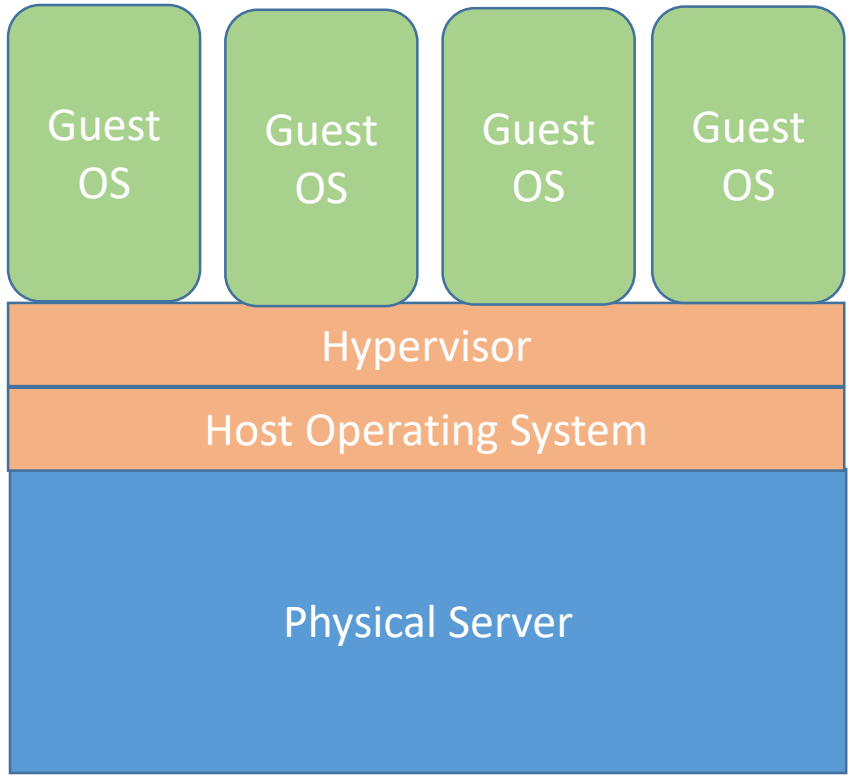
Types of Virtualization

- Data virtualization
- Desktop virtualization
- Server virtualization
- Network functions virtualization

Virtualization

Hypervisor based Virtualization

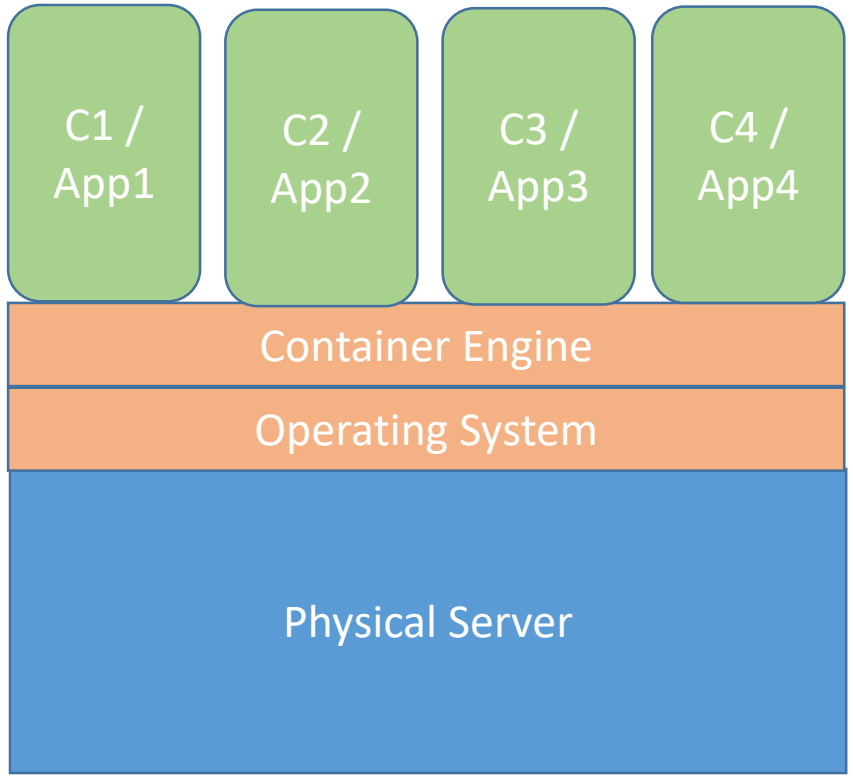
- Benefits:
  - More cost effective
  - Easy to scale
- Limitations:
  - Kernel Resource Duplication
  - Application portability issue



Virtualization

Container based Virtualization

- Benefits:
  - More cost effective
  - Faster deployment speed
  - Great portability



Virtualization

Service Virtualization as an Enabler of DevOps

- DevOps engagements have the primary goal of accelerating the time to market
- Service virtualization and DevOps
  - Service virtualization is the process of simulating the behavior of selected components within an application to enable end-to-end delivery of the application as a whole
  - Application development teams can use virtual services in lieu of the production or real services to conduct integration testing earlier in the development process
  - Virtualized assets look and act like the real thing but may be duplicated and available at times when the real assets are not available

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Containerization

- Introduction to Docker
- Docker Overview
- Docker Engine
- Docker Architecture
- Docker Concept : Swarm and Stack
- Docker File
- Docker cloud
- Docker Store
- Docker Hub
- How is Docker Store different from Docker Hub
- Overview of Docker Editions

Containerization

Introduction to Docker

- Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers
- The use of Linux containers to deploy applications is called containerization
- Containerization is increasingly popular because containers are :
  - Flexible: Even the most complex applications can be containerized
  - Lightweight: Containers leverage and share the host kernel
  - Interchangeable: Can deploy updates and upgrades on-the-fly
  - Portable: Can build locally, deploy to the cloud, and run anywhere
  - Scalable: Can increase and automatically distribute container replicas
  - Stackable: Can stack services vertically and on-the-fly

Containerization

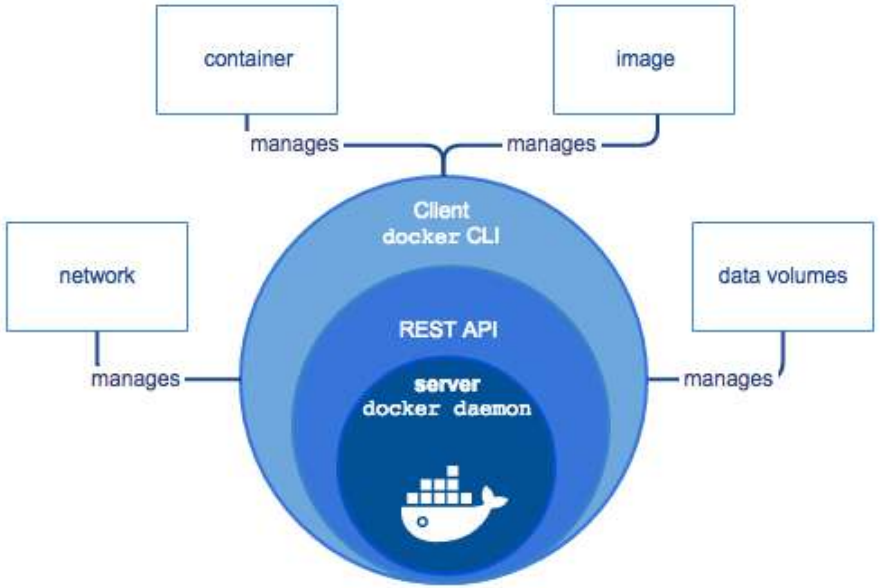
Docker Overview

- Docker is an open platform for developing, shipping, and running applications
- Docker is licensed under the open source Apache 2.0 license
- Docker enables to separate applications from infrastructure so we can deliver software quickly
- With Docker, we can manage our infrastructure in the same ways we manage our applications
- By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, we can significantly reduce the delay between writing code and running it in production

Containerization

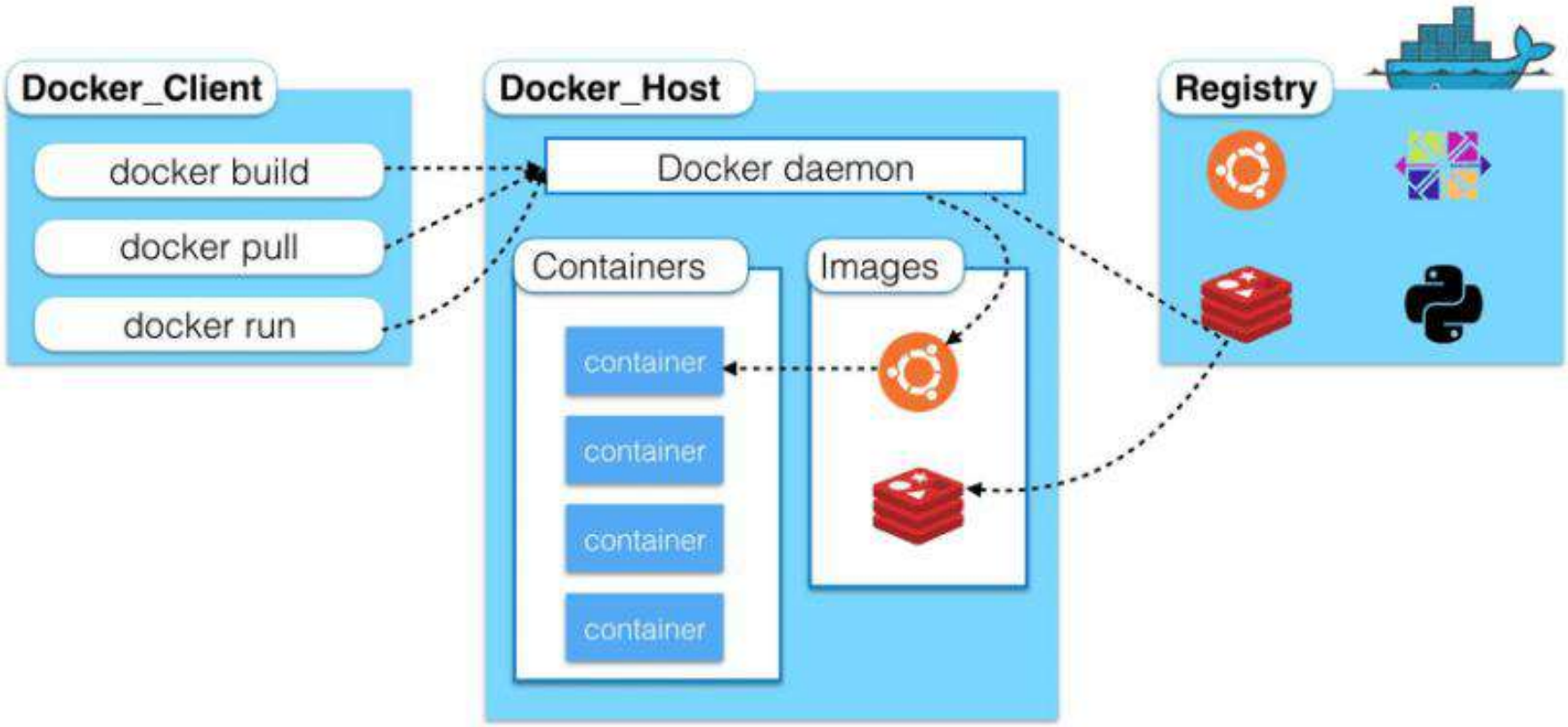
Docker Engine

- Docker Engine is a client-server application
- A server which is a type of long-running program called a daemon process (the dockerd command)
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do
- A command line interface (CLI) client (the docker command)



Containerization

Docker Architecture



Containerization

Docker Concept : Swarm and Stack

- Docker Swarm:
- A swarm is a group of machines that are running Docker and joined into a cluster
- Docker Command then will be executed on a cluster by a swarm manager
- The machines in a swarm can be physical or virtual
- After joining a swarm, machines are referred to as nodes

- Docker Stack:
- A stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together
- A single stack is capable of defining and coordinating the functionality of an entire application

Containerization

Docker File

- Docker file defines what goes on in the environment inside a container
- This file has information about all resources
  - Network interface and
  - Disk drives

Bits Pilani Software installation Docker file:

FROM centos:latest

MAINTAINER “Bits Pilani”

RUN yum install -y bits-pilani-client

EXPOSE 9999

Containerization

Docker Cloud

- Docker Cloud provides a hosted registry service with build and testing facilities for Dockerized application images
- Access to Docker Cloud is managed by Docker ID
- Manage Builds and Images
- Manage Swarms
- Manage Infrastructure
- Manage Nodes and Apps
- Integration with AWS and Microsoft Azure Services

Containerization

Docker Store

- For developers and operators, Docker Store is the best way to discover high-quality Docker content
- Independent Software Vendors (ISVs) can utilize Docker Store to distribute and sell their Dockerized content
- Store Experience:
- Access to Docker’s large and growing customer-base
  - Customers can try or buy your software
  - Use of Docker licensing support
  - Docker handle checkout
  - Seamless updates and upgrades for your customers
  - Become Docker Certified

Containerization

Docker Hub

- Docker Hub is a cloud-based registry service
- Allows link to code repositories
- Build images and test them
- Stores Images
- Links to Docker Cloud to deploy images to hosts
- It is a Centralize Solution for:
  - Container image discovery
  - Distribution and change management
  - User and team collaboration
  - Workflow automation throughout the development pipeline

Containerization

How is Docker Store different from Docker Hub

Docker Hub	Docker Store
Contents by Docker community	Contents submitted for approved by qualified Store Vendor Partners
Anyone can push new images to the community	Contents are published and maintained by Entity
No guarantees around the quality or compatibility	Docker Certified quality assurance
Common Thing: The Docker official Images are Available in both	

Containerization

Overview of Docker Editions

- Docker is available in two editions:
  - Community Edition (CE)
  - Enterprise Edition (EE)
- Docker Community Edition has three types
  - Stable
  - Test
  - Nightly

References

- Docker Documentation [https://docs.docker.com/]

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Overview of Microservices

- Introduction to Microservices
- List of Organizations who are using Microservices
- Common characteristics for Microservices
- Need of Microservices for DevOps
- DevOps and Microservices
- Benefits of DevOps and Microservices

Microservices

Introduction to Microservices

- Microservices are a software development technique
- A type of the service-oriented architecture (SOA); that structures an application as a collection of loosely coupled services
- The benefit of decomposing an application into different smaller services is; it improves modularity
- Microservices enables small autonomous teams to develop, deploy and scale their respective services independently
- It also allows the architecture of an individual service to emerge through continuous refactoring

Microservices

Introduction to Microservices Contd..

- Microservices and their related architectures are gaining power in enterprises of all sizes
- Microservices can be conceptualized as a new way to create corporate applications
- The Idea is to break down applications into smaller, independent services
- That are not dependent upon a specific coding language
- Using Microservices Ideology divide large & complex applications in to smaller building blocks

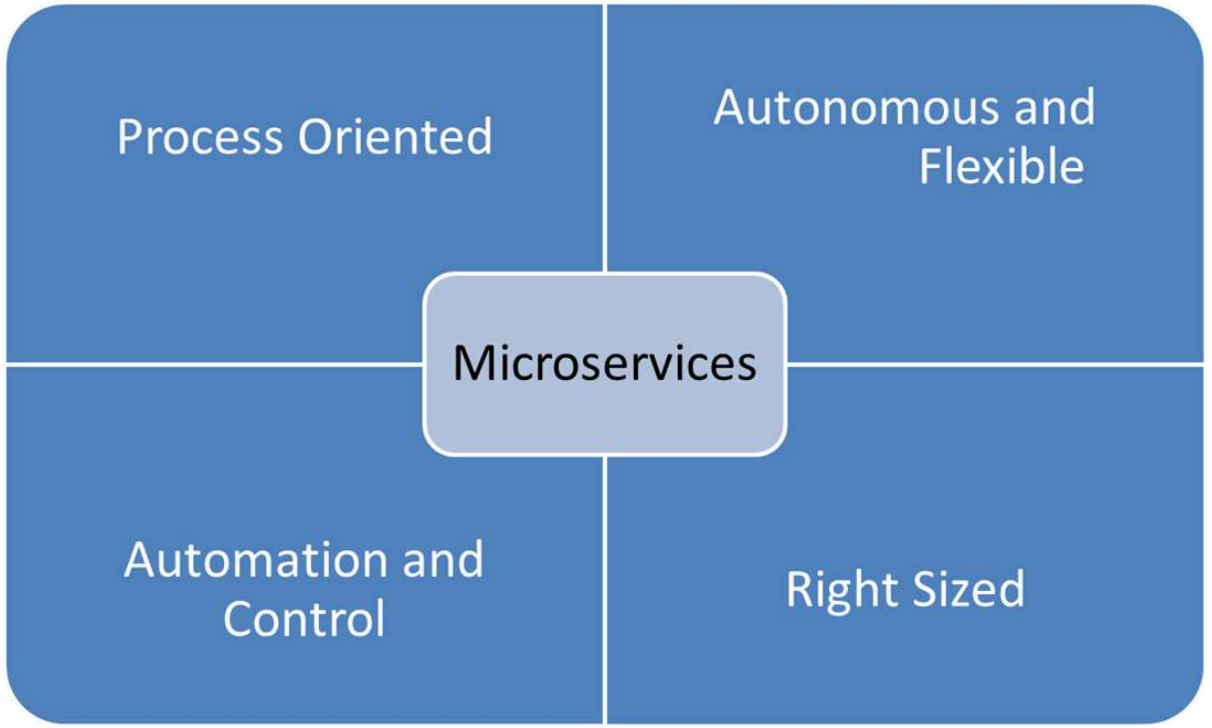
Microservices

List of Organizations who are using Microservices

- Uber
- Netflix
- Amazon
- Ebay
- Gilt
- Tesla

Microservices

Common characteristics for Microservices



Microservices

Need of Microservices for DevOps

- DevOps promotes small, more empowered and autonomous teams, along with automation and measurements
- It has great potential to be a fantastic improvement in the way IT and business work together
- Struggling Area:
- Setup Pipeline for new automation tooling
- The mindset of Architects
- Flow we followed to Emerge:
  - Centralized IT Components and Deliverables – [2005]
  - Cloud and DevOps [2010 – 2013]

Microservices

DevOps and Microservices

- With small autonomous DevOps teams, the world will start producing small autonomous components, called microservices
- It makes sense to produce deployable components for a business function
- This will help in speed improve five to ten times by moving to the cloud and using a high productivity platform
- DevOps and microservices are more or less inseparable, they can hardly exist in separation

Microservices

Benefits of DevOps and Microservices

- Deployability
- Reliability
- Availability
- Scalability
- Modifiability
- Management
- Microservices bring additional productivity to DevOps by embracing a common toolset, which can be used for both development and operations

Reference

- https://www.mendix.com/blog/the-microservices-you-need-for-devops/
- https://www.mulesoft.com/resources/api/what-are-microservices

Thank You!

In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

AWS Lambda

- Introduction to AWS Lambda
- AWS Lambda : Application Support
- Function as a Service [FaaS]
- AWS Lambda : Serverless
- Success Story of The Seattle Times
- Benefits Summary

AWS Lambda

What is AWS Lambda?

- AWS Lambda is a compute service that lets you run code without provisioning or managing servers
- AWS Lambda executes your code only when needed and scales automatically
- You pay only for the compute time you consume - there is no charge when your code is not running
- Often referred to as “serverless” computing. AWS Lambda allows developers to think only about the code, and not have to worry about the layers below
- You can think of PaaS service to co-relate

AWS Lambda

AWS Lambda : Application Support

- Can any application run in Lambda?
- Some applications may need to be modified to run with Lambda
- Serverless is a specific architecture where state and compute are decoupled
- Lambda functions can write state to external data stores via web requests
- External Data store can be of AWS or Any
- AWS Solutions: S3, Dynamo, and Redshift
- Other Solutions: PostgreSQL, Cassandra and Kafka

AWS Lambda

Function as a Service [FaaS]

- Microservices and AWS Lambda
- Microserviecs where applications are broken into much Smaller pieces
- To enable better scalability, more agile development
- Similarly AWS Lambda allows developers to deploy a single function at a time
- Once Deployed, Lambda will manage the operational aspects of that function, from scaling to high availability
- AWS Lambda is referred as Function As A Service

AWS Lambda

Serverless?

- Its an Architecture
- Serverless computing is a cloud-computing execution model in which the cloud provider acts as the server
- Dynamically managing the allocation of machine resources
- Pricing is based on the actual amount of resources consumed by an application
- It is a form of utility computing

AWS Lambda

Success Story of The Seattle Times

- The Challenge:
  - After maintaining on-premises hardware and custom publishing software for nearly two decades, The Seattle Times sought to migrate its website publishing to a contemporary content management platform.
  - To avoid the costs of acquiring and configuring new hardware infrastructure and the required staff to maintain it, the company initially chose a fully managed hosting vendor.
  - But after several months, The Times' software engineering team found it had sacrificed flexibility and agility in exchange for less maintenance responsibility.
  - As the hosted platform struggled with managing traffic under a vastly fluctuating load, The Seattle Times team was constrained in its ability to scale up to meet customer demand.

AWS Lambda

Success Story of The Seattle Times Contd..

- Why Amazon Web Services:
  - To address these core scalability concerns, The Seattle Times engineering team considered several alternative hosting options, including self-hosting on premises, more flexible managed hosting options, and various cloud providers.
  - The team concluded that the available cloud options provided the needed flexibility, appropriate architecture, and desired cost savings. The company ultimately chose Amazon Web Services (AWS), in part because of the maturity of the product offering and, most significantly, the auto-scaling capabilities built into the service.
  - The Seattle Times deployed its new system in just six hours. The website moved to the AWS platform between 11 p.m. and 3 a.m. and final testing was completed by 5 a.m. — in time for the next news day.

AWS Lambda

Success Story of The Seattle Times Contd..

- End Result:
  - With AWS, The Seattle Times can now automatically scale up very rapidly to accommodate spikes in website traffic when big stories break, and scale down during slower traffic periods to reduce costs.
  - “Auto-scaling is really the clincher to this,” Seattle Times says. “With AWS, we can now serve our online readers with speed and efficiency, scaling to meet demand and delivering a better reader experience.’’
  - "AWS Lambda provides us with extremely fast image resizing,” Seattle times says. “Before, if we needed an image resized in 10 different sizes, it would happen serially. With AWS Lambda, all 10 images get created at the same time, so it’s quite a bit faster and it involves no server maintenance.”

Reference Architecture: Sample Code

AWS Lambda

Benefits Summary

- No servers to Manage
- Continuous Scaling
- Subsecond Metering

Reference

- https://aws.amazon.com/lambda

Thank You!

In our next session: