



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

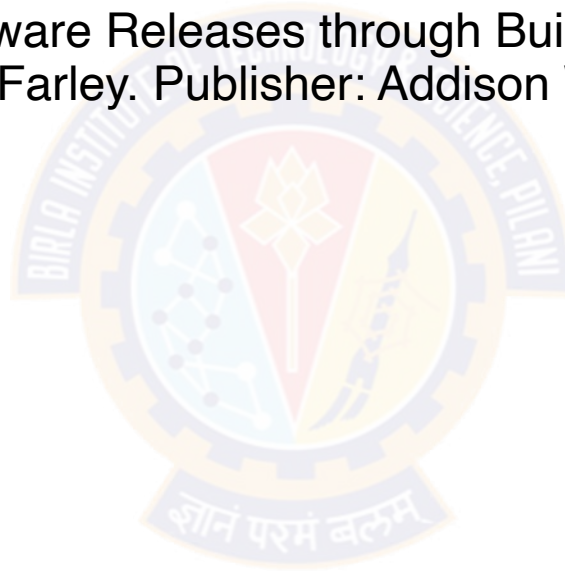
Sonika Rathi

Assistant Professor
BITS Pilani

Text Books

T1 & T2

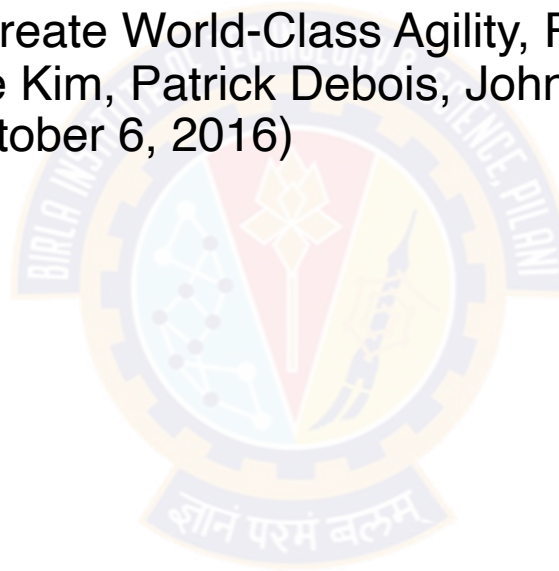
- DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu , Publisher: Addison Wesley (18 May 2015)
- Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation by Jez Humble, David Farley. Publisher: Addison Wesley, 2011



Reference Books

R1 & R2

- Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis , Ryn Daniels. Publisher: O'Reilly Media, June 2016
- The DevOPS Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations by Gene Kim, Patrick Debois, John Willis, Jez Humble, John Allspaw. Publisher: IT Revolution Press (October 6, 2016)



Agenda

Introduction

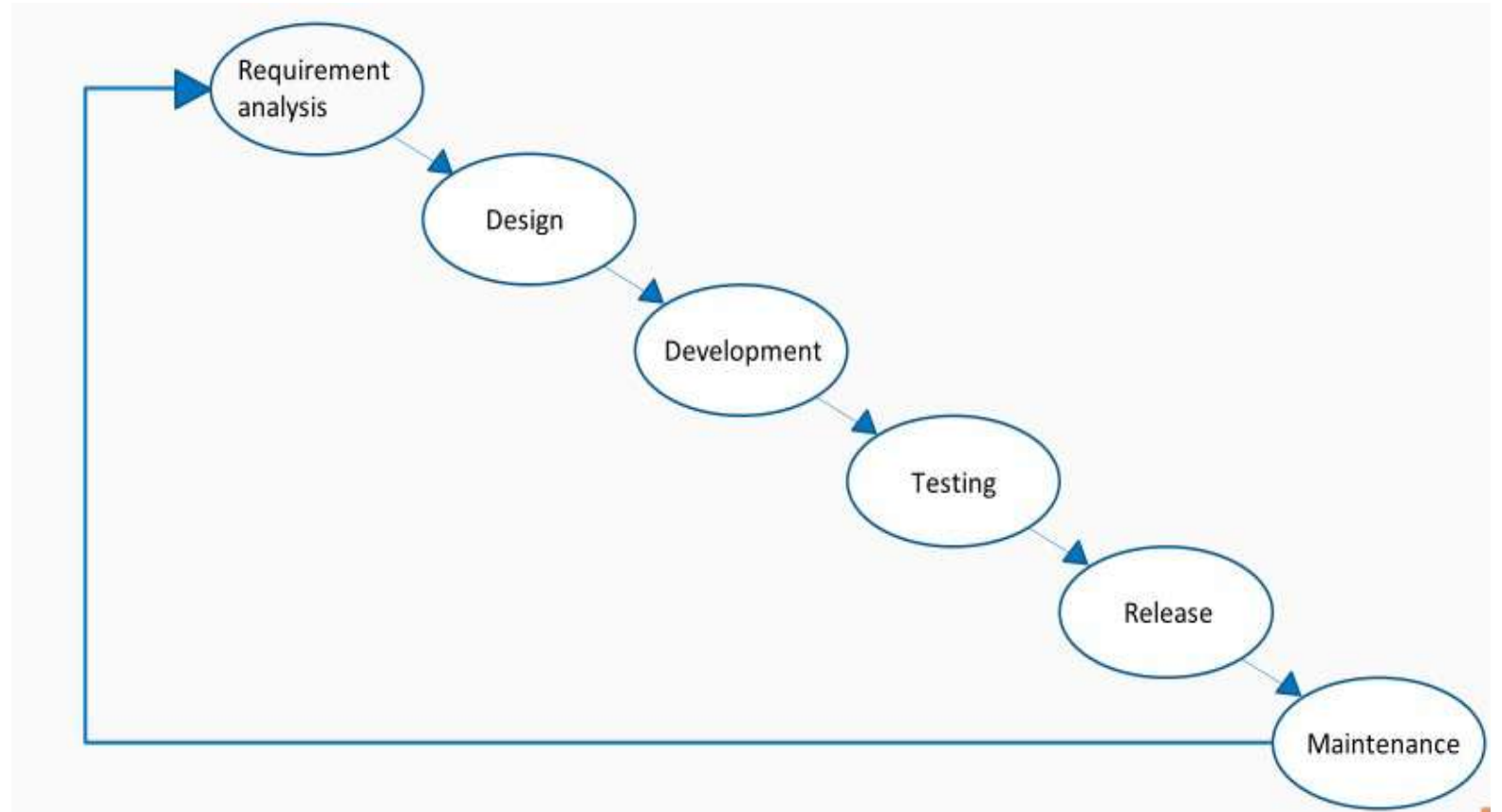
- Software development lifecycle
- The Waterfall approach : Advantages & Disadvantages
- Agile Methodology
- Operational Methodologies: ITIL



Software Development Life Cycle

SDLC Phases

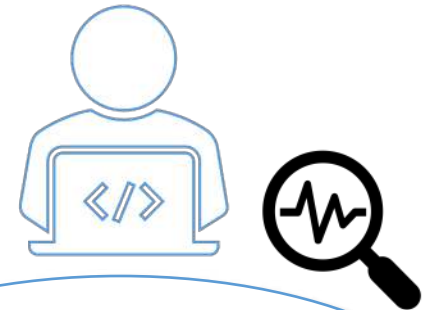
Diagram of our day-to-day activity
as software engineers



Software Development Life Cycle

Requirement Analysis

- Encounter majority of problems
- Find common language between people outside of IT and people in IT
- Leads to different problems around terminology
- Business flow being capture incorrectly
- Iterative approach

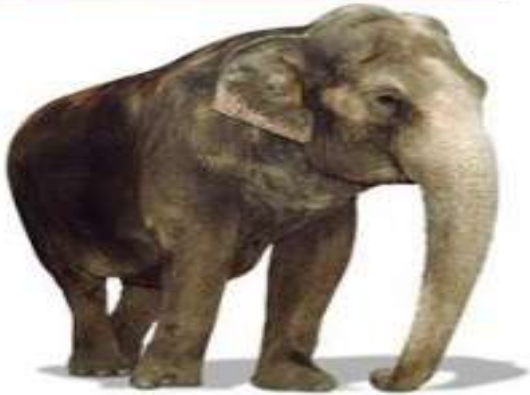


Requirement Analysis

Software Development Life Cycle

Requirement Analysis Contd..

Customer requirement



1. Have one trunk
2. Have four legs
3. Should carry load both passenger & cargo
4. Black in color
5. Should be herbivorous

Our Solution



1. Have one trunk ☒
2. Have four legs ☒
3. Should carry load both passenger & cargo ☒
4. Black in color ☒
5. Should be herbivorous ☒

Our Value add:

Also gives milk 😊

Software Development Life Cycle

Design

- Design our flows in language that IT crowd can understand straight away
- Overlaps with requirement analysis
- Desirable as diagrams are perfect middle language that we are looking for
- Minimal Viable Product

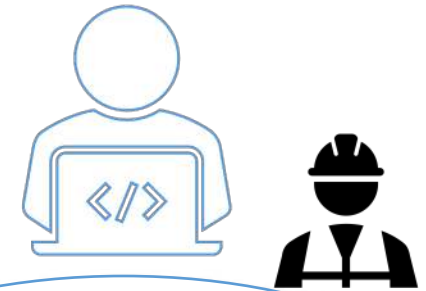
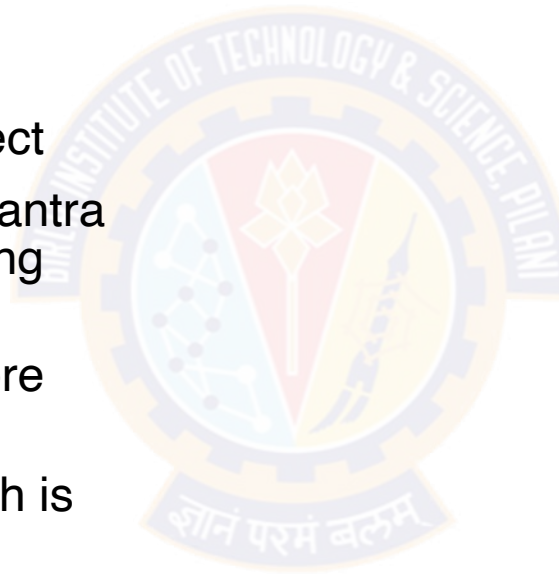


Design

Software Development Life Cycle

Development

- Software is built
- Builds technical artifacts that work and according to potentially flawed specification
- Our software is going to be imperfect
- Deliver early and deliver often is mantra followed to minimize impact of wrong specification
- Stakeholders can test product before problem is too big to be tackled
- Involving stakeholders early enough is good strategy
- No matter what we do, our software has to be modular so we can plug and play modules in order to accommodate new requirements

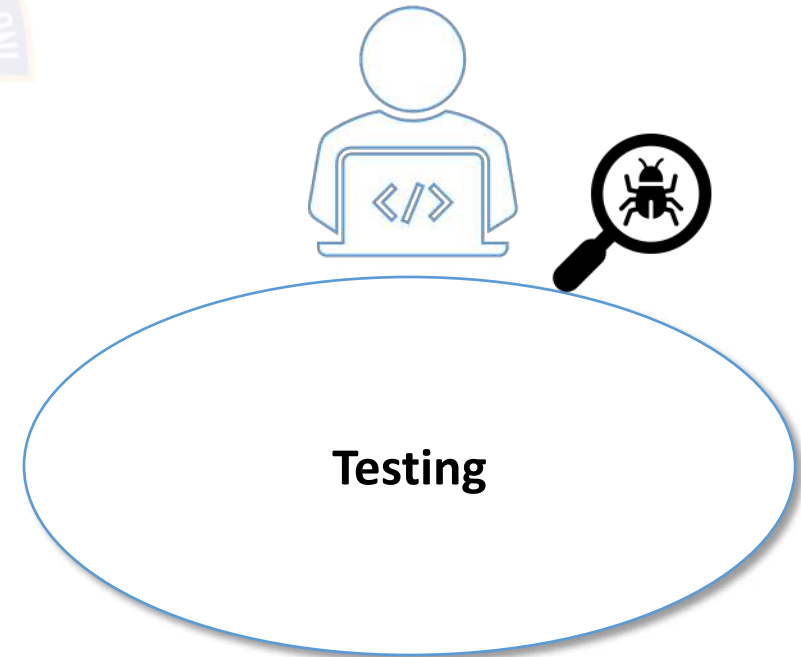


Development

Software Development Life Cycle

Testing

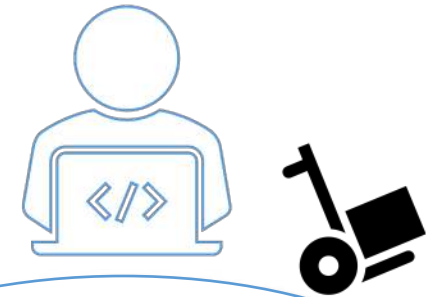
- Continuous Integration server will run testing and inform us about potential problems in application
- Depending on complexity of software, testing can be very extensive
- Continuous integration server focuses on running integration and acceptance



Software Development Life Cycle

Release

- Deliver software to what we call production
- There will be bugs and reason why we planned our software to be able to fix problems quickly
- Create something called as Continuous delivery pipelines
- Enables developers to execute build-test-deploy cycle very quickly
- Deploy = Release

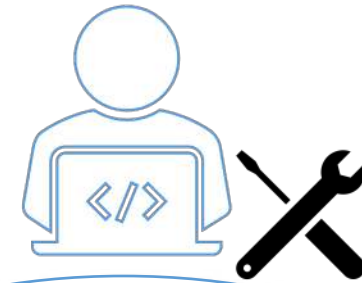


Release

Software Development Life Cycle

Maintenance

- There are two types of maintenance
evolutive and corrective
- Evolutive maintenance – evolve
software by adding new
functionalities or improving business
flows to suit business needs
- Corrective maintenance – One
where we fix bugs and
misconceptions
- Minimize latter but we can not totally
avoid

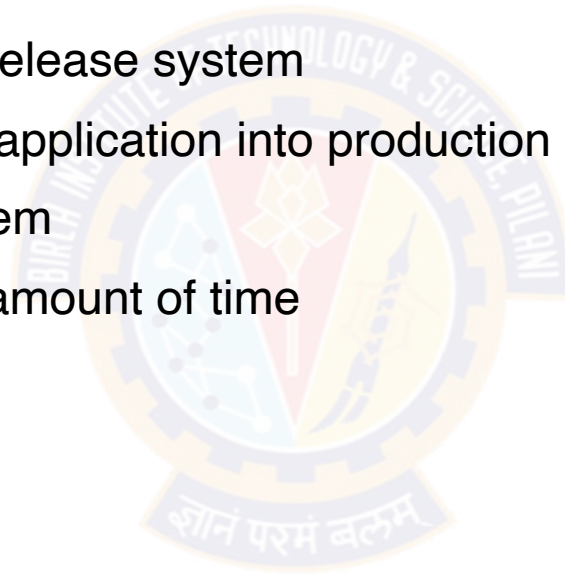


Maintenance

Case Study

The Power of Automated Deployment form Continuous Delivery Book

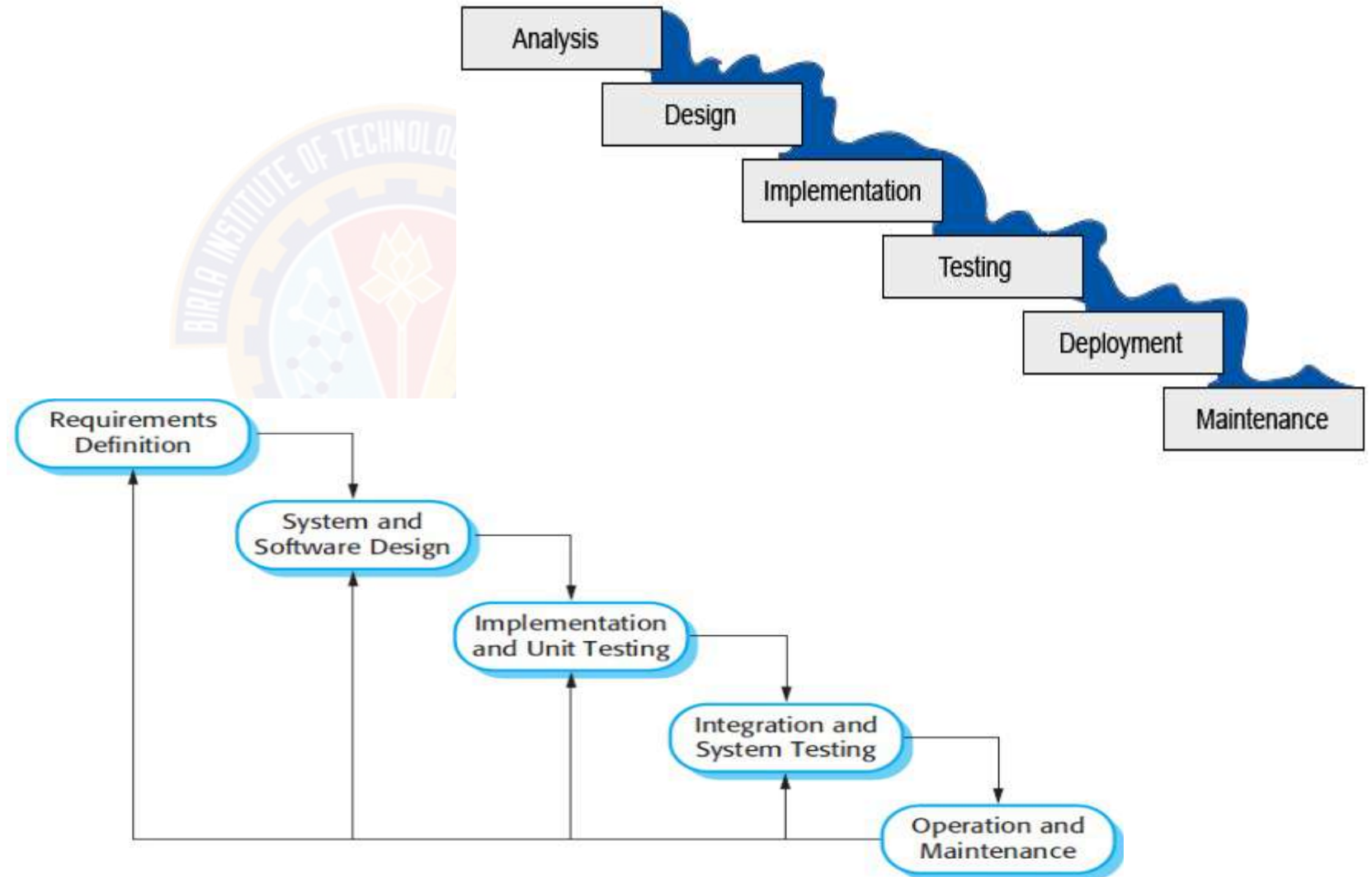
- Large team dedicated to release
- High level of intervention
- Automated build, deploy, test and release system
- Only seven seconds to deploy the application into production
- Successful deployment of the system
- Roll back the change in the same amount of time



Waterfall Model

Waterfall Model and Feedback Amendment in Waterfall Model

- Classical Life cycle /Black Box Model
- Sequential in Nature
- Systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software



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
- Does Waterfall ends????
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites
 - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work

Waterfall Model Contd..

When to apply Waterfall?



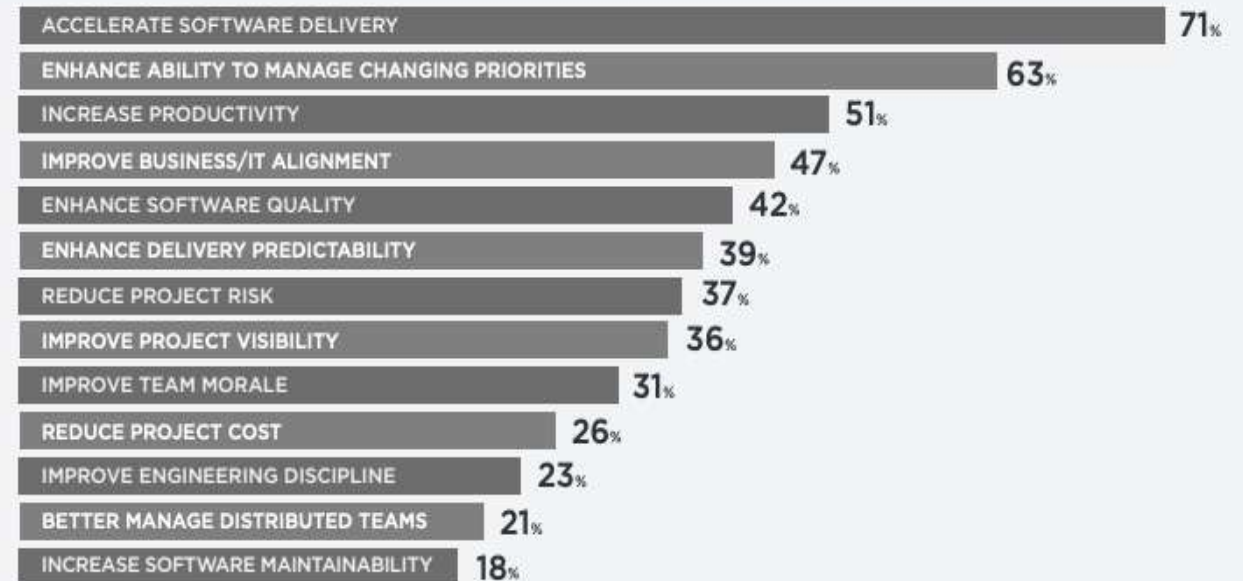
Need of Agile

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

REASONS FOR ADOPTING AGILE

Accelerating software delivery and enhancing ability to manage changing priorities remain the top reasons stated for adopting Agile. Respondents indicated this year that reasons for adoption were less about reducing project cost (26% compared to 41% last year), and more about reducing project risk (37% compared to 28% last year).



*Respondents were able to make multiple selections

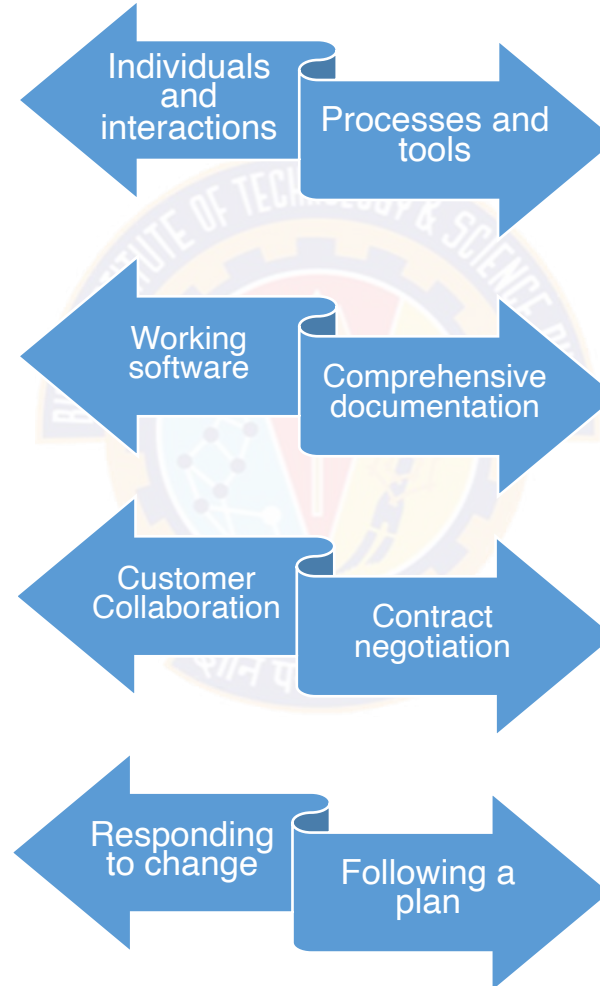
Principles of Agile Methodology

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

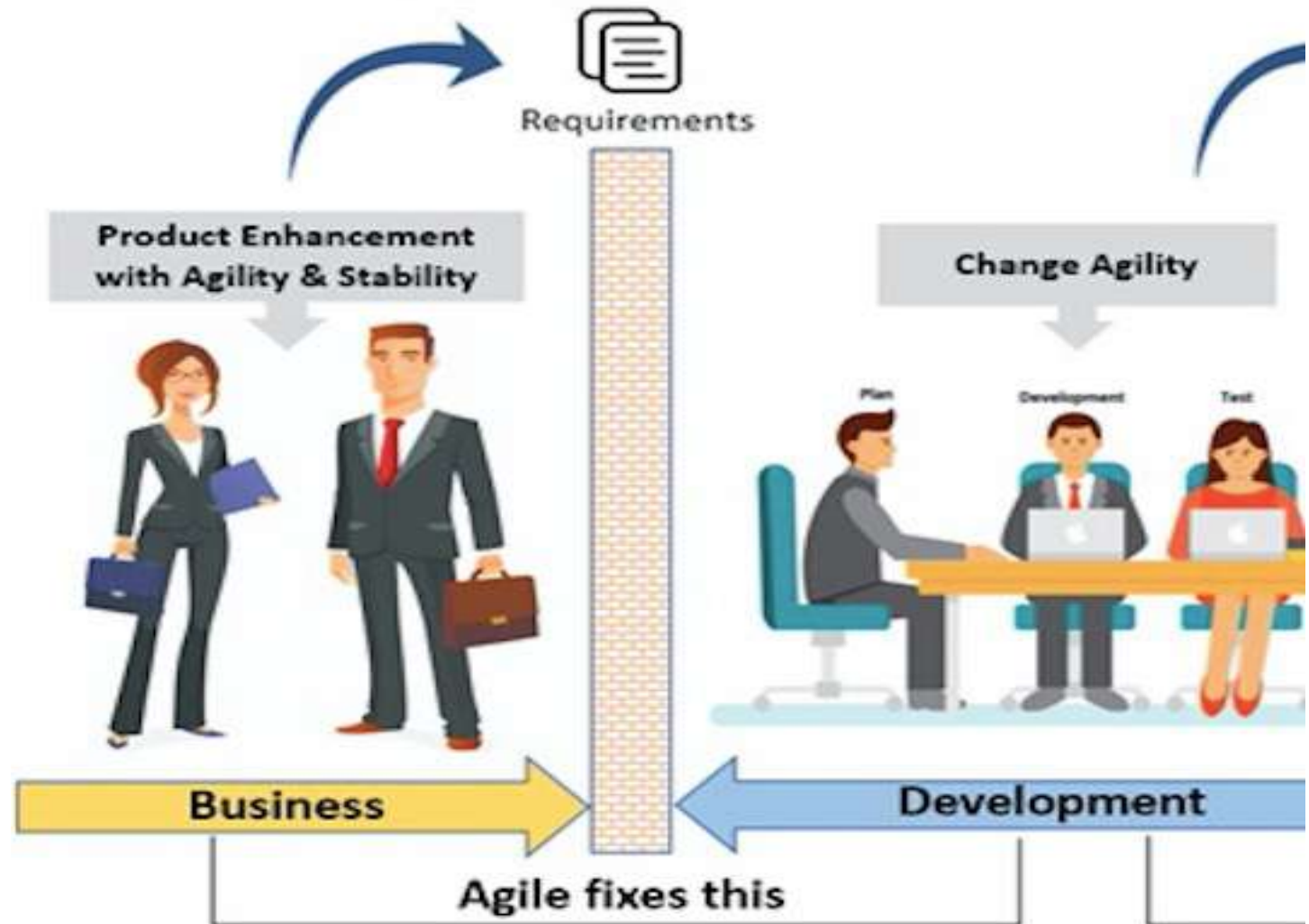
Agile focuses on



Agile Brings What??

Being Agile

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed



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

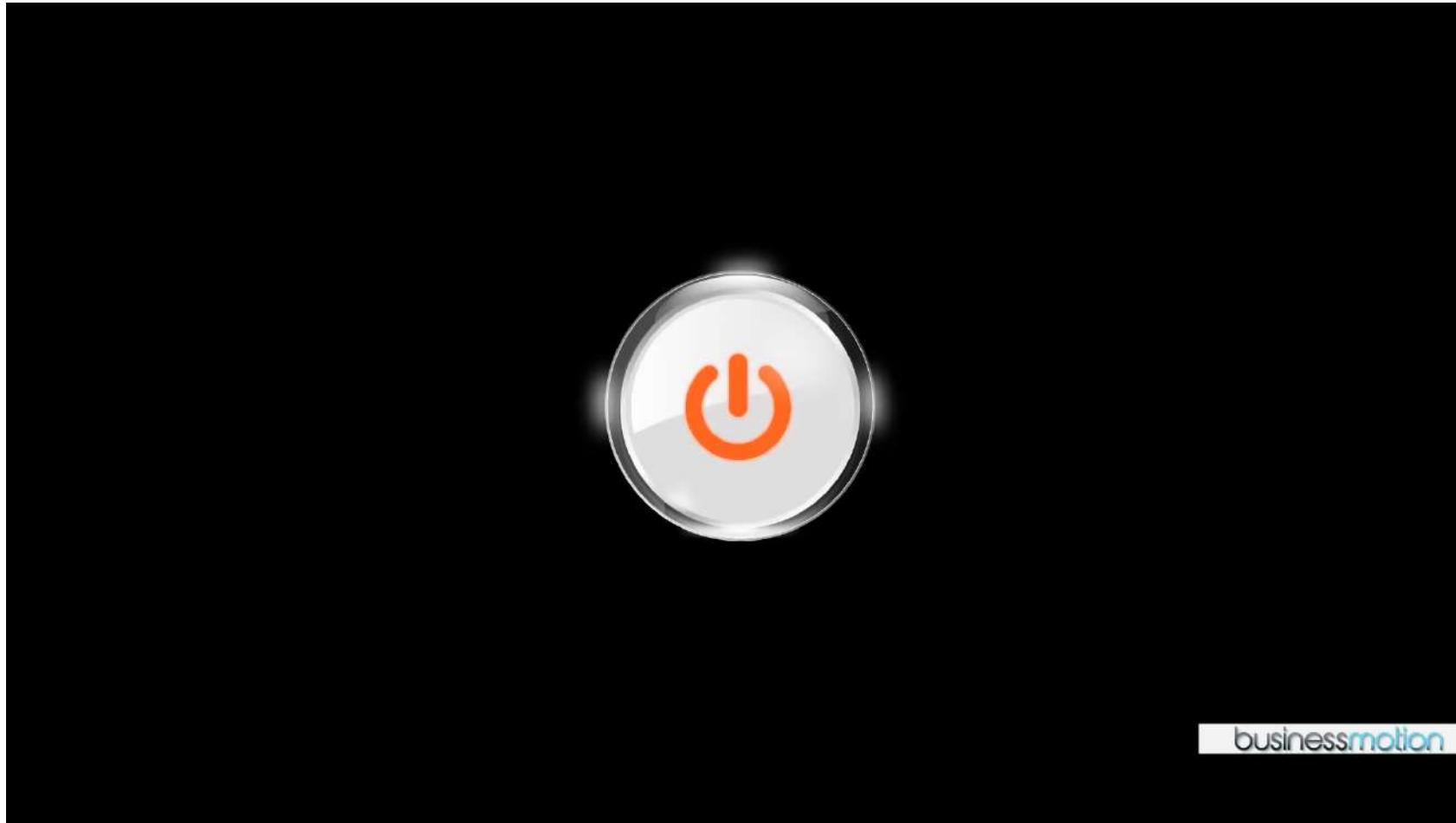
Basic roles involved

- User
- Product Owner
- Software Development Team



Agile Methodology

Summary

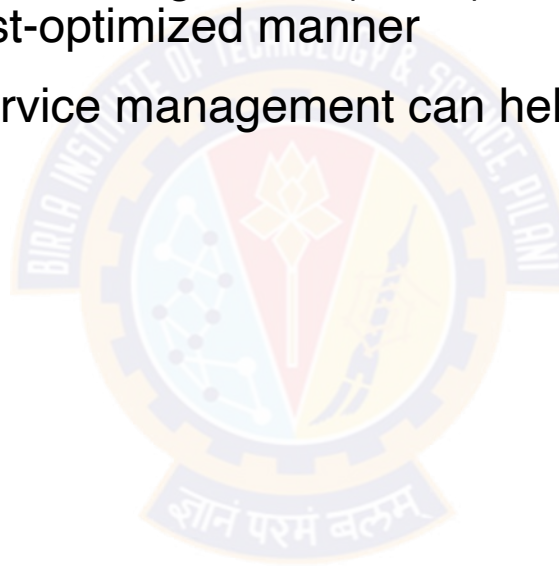


<https://www.youtube.com/watch?v=1iccpf2eN1Q>

Operational Methodology

ITIL

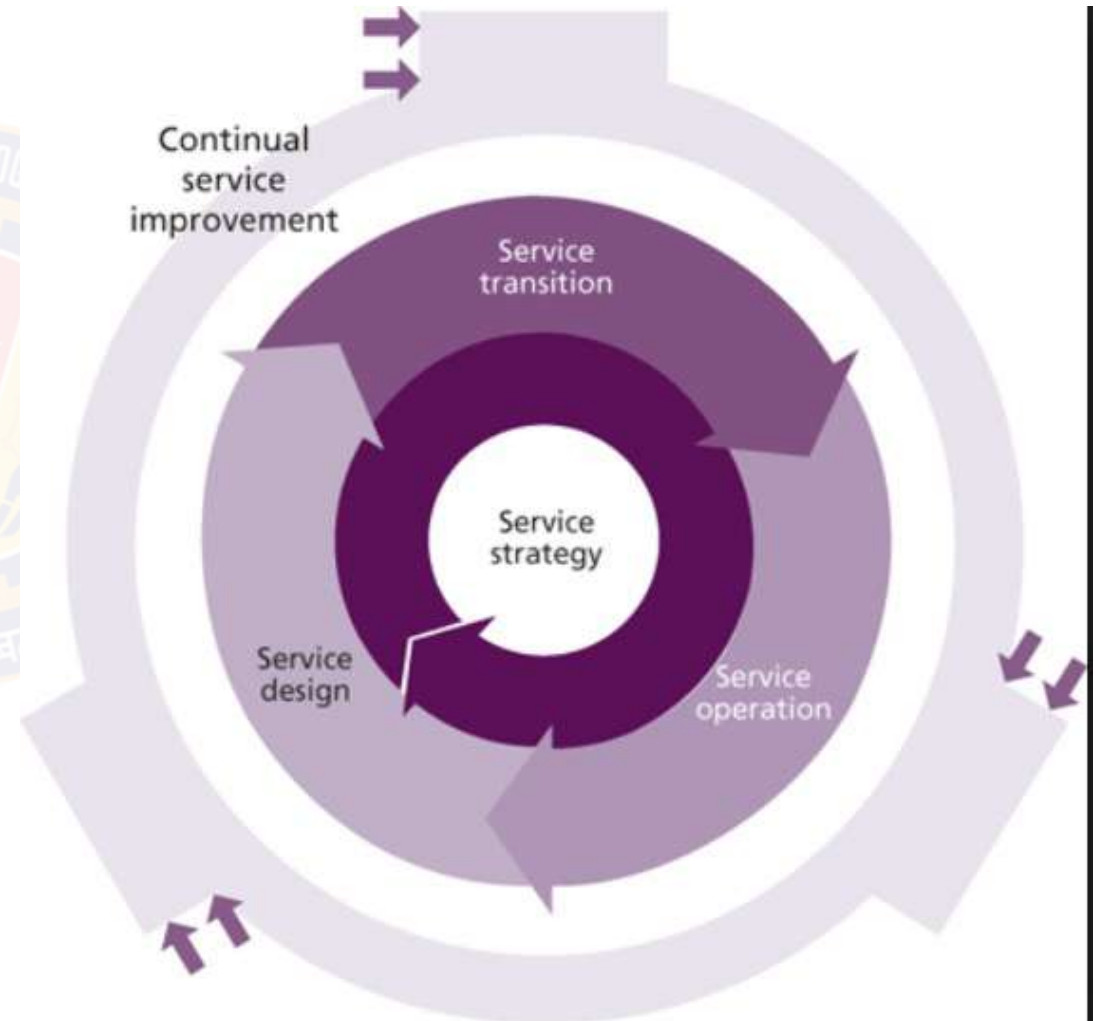
- ITIL is a framework of best practices for delivering IT services
- The ITIL processes within IT Service Management (ITSM) ensure that IT Services are provided in a focused, client-friendly and cost-optimized manner
- ITIL's systematic approach to IT service management can help businesses
 - Manage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



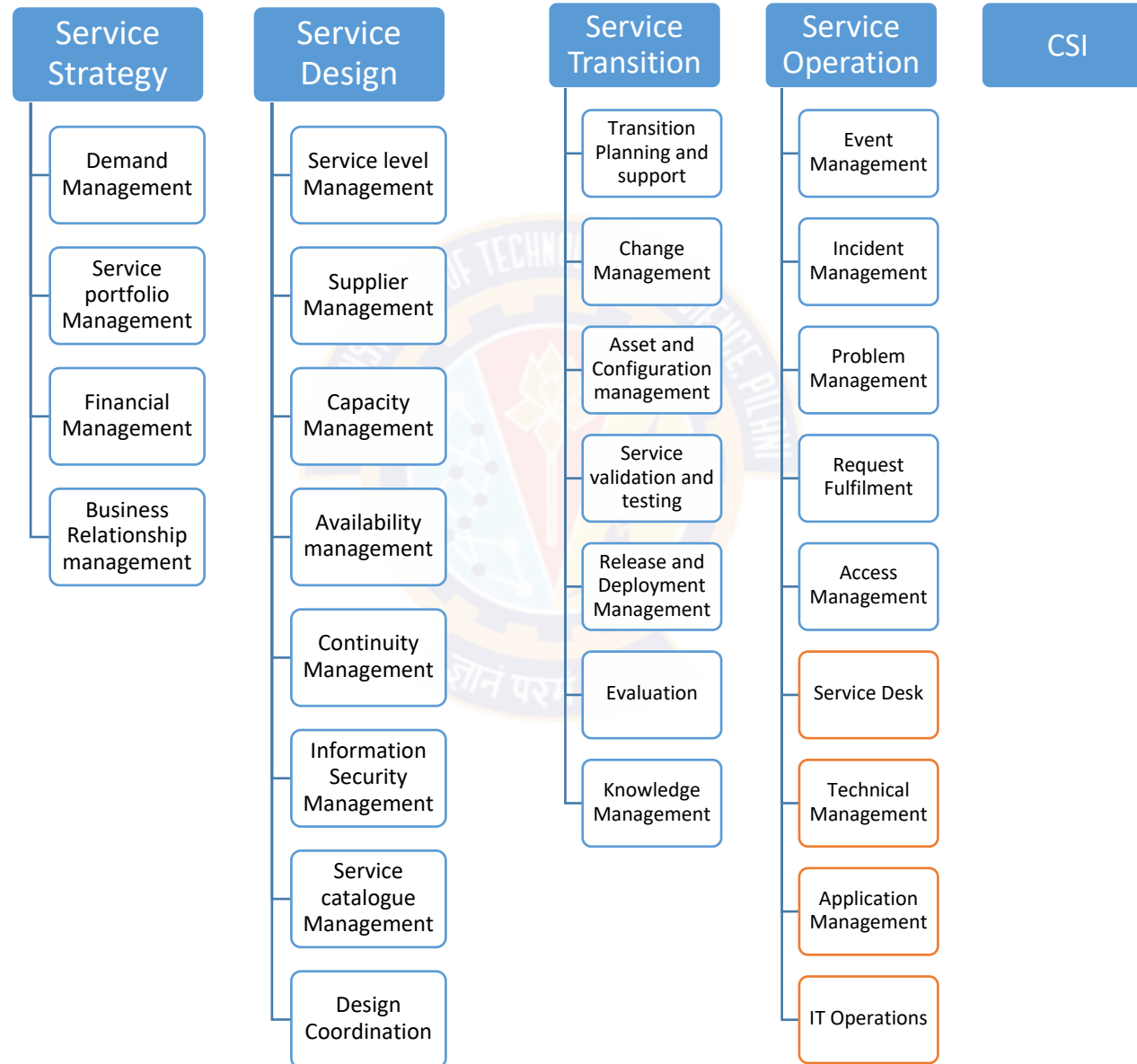
IT Service Management (ITSM)

Lifecycle

- ITIL views ITSM as a lifecycle
- Five Phases:
 - Service Strategy
 - Service Design
 - Service Transition
 - Service Operation
 - Service Improvement [create and maintain value]



Framework



Is a Project???

- ITIL is not a “Project”
- ITIL is ongoing journey
- Benefits of ITIL:
 - Pink Elephant reports that Procter and Gamble saved about \$500 million over four years by reducing help desk calls and improving operating procedures
 - Nationwide Insurance achieved a 40 percent reduction in system outages and estimates a \$4.3 million ROI over three years
 - Capital One reduced its "business critical" incidents by 92 percent over two years

Lets Review



<https://www.youtube.com/watch?v=FSfgovmPHO8>



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Introduction

- About DevOps
- Problems of Delivering Software
- Principles of Software Delivery
- Need for DevOps
- DevOps Practices in Organization
- The Continuous DevOps Life Cycle Process
- Evolution of DevOps
- Case studies



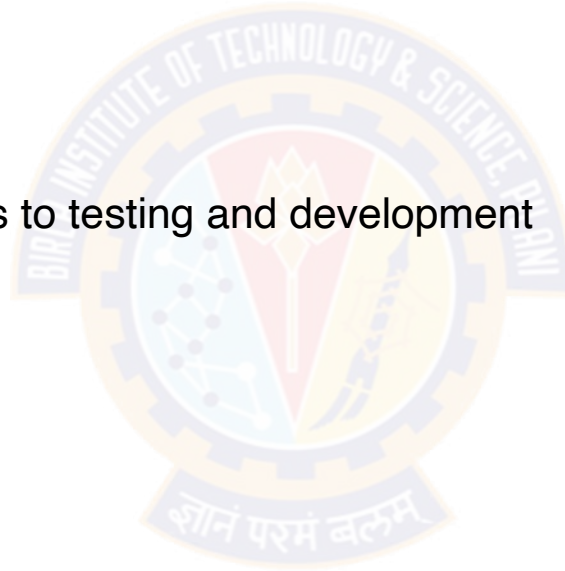
DevOps

Definition

- “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”

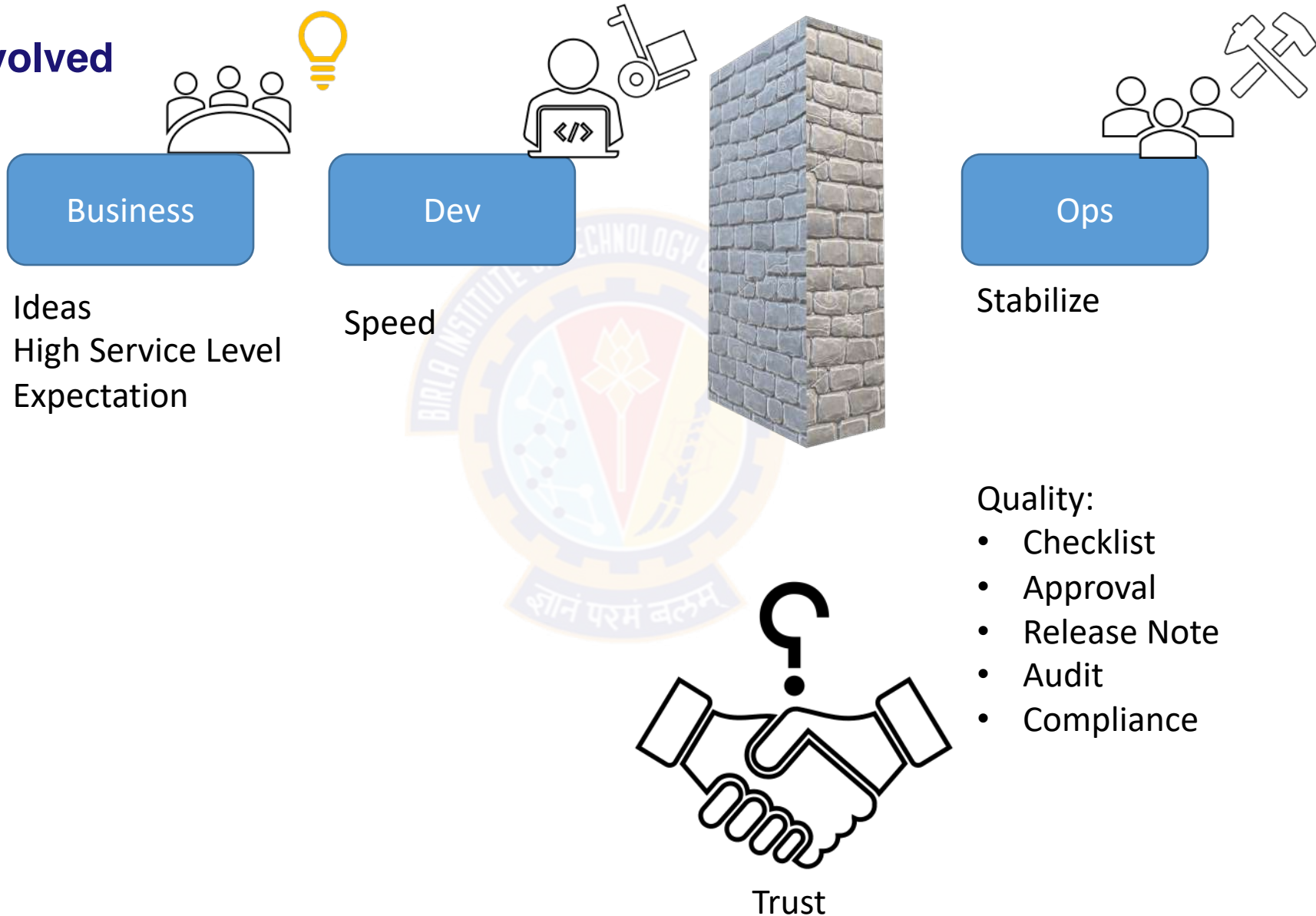
Implications of this definition

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



DevOps

Perspective involved



Problems of Delivering Software

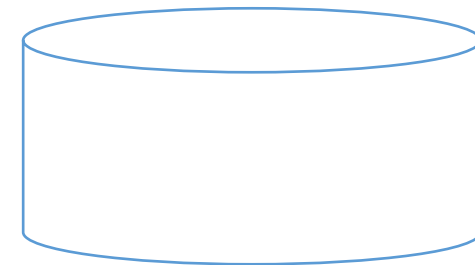
- Converting Idea to Product / Service?
- Reliable, rapid, low-risk software releases
- Ideal Environment
- Generic Methodologies for Software Methodologies
- More focus on Requirement Gathering
- Understanding the Value Stream Map

1

2

3

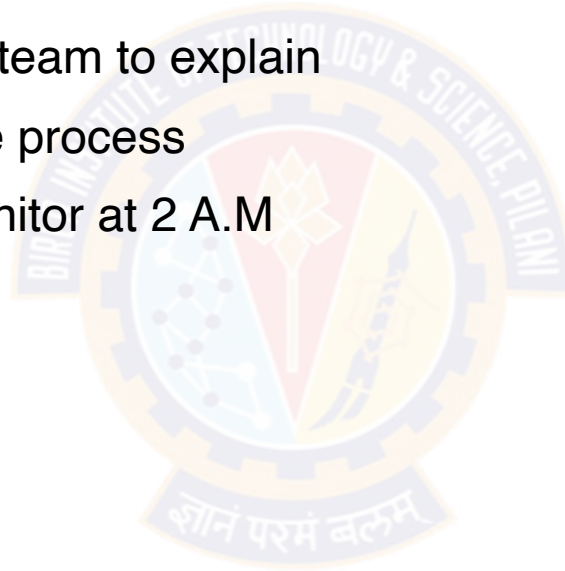
4



Common Release Antipatterns

Deploying Software Manually

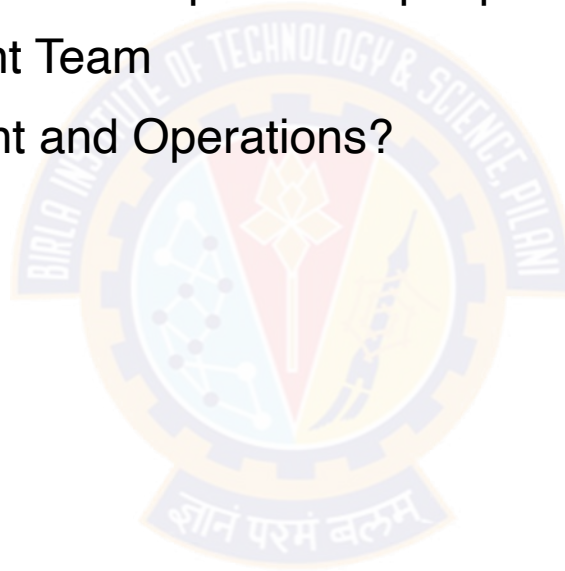
- Extensive and detailed documentation
- Reliance on manual testing
- Frequent calls to the development team to explain
- Frequent corrections to the release process
- Sitting bleary-eyed in front of a monitor at 2 A.M



Common Release Antipatterns

Deploying to a Production-like Environment Only after Development Is Complete

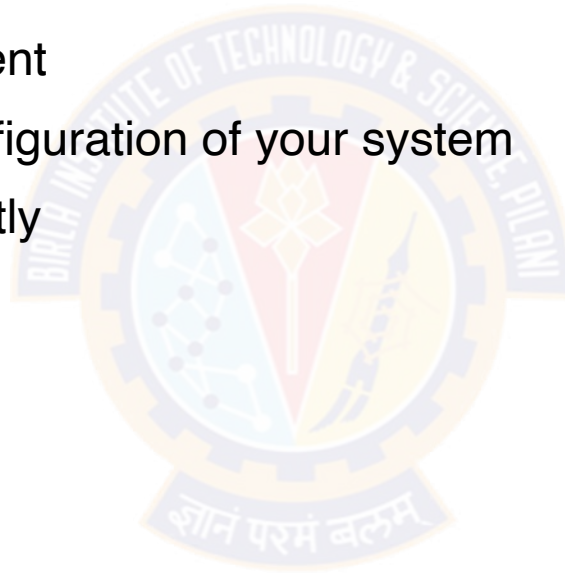
- Tester tested the system on development machines
- Releasing into staging is the first time that operations people interact with the new release
- Who Assembles? The Development Team
- Collaboration between development and Operations?



Common Release Antipatterns

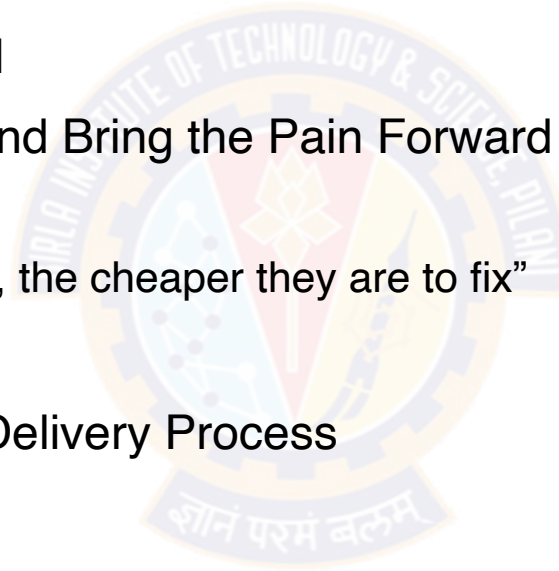
Manual Configuration Management of Production Environments

- 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 your system
- Modification to Configuration Directly



Principles of Software Delivery

- 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



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

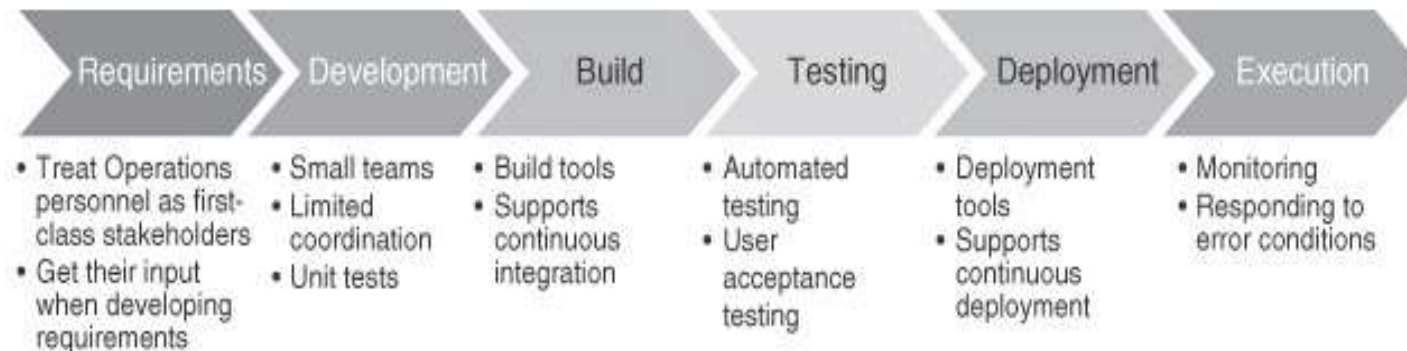
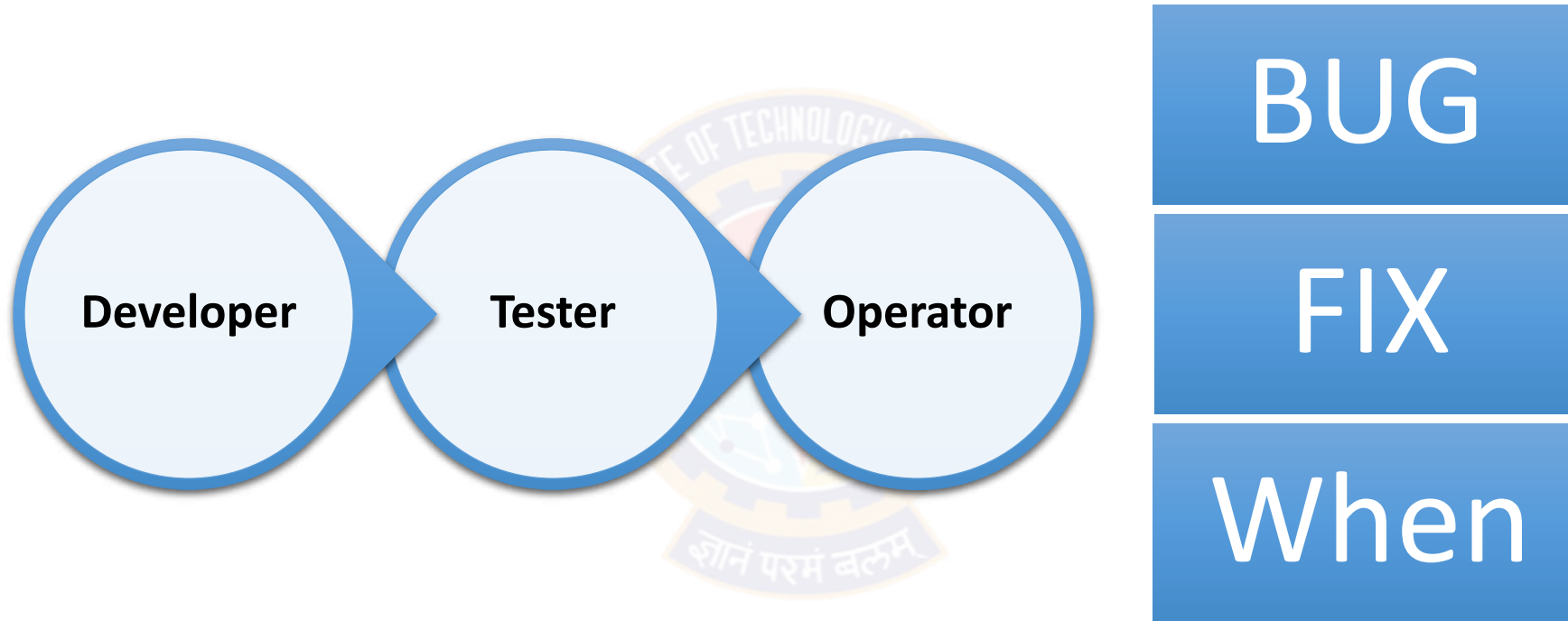


FIGURE 1.1 DevOps life cycle processes [Notation: Porter's Value Chain]

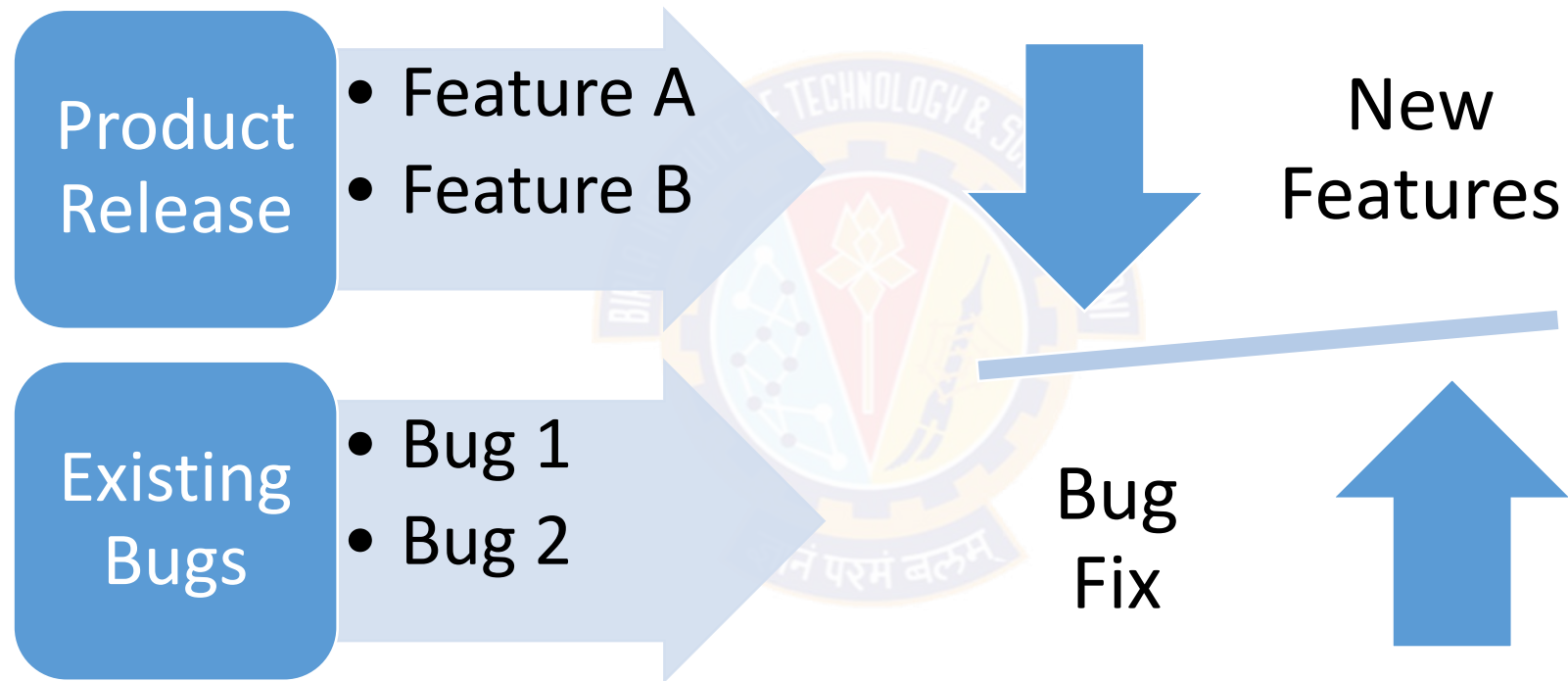
Need for DevOps

Timelines



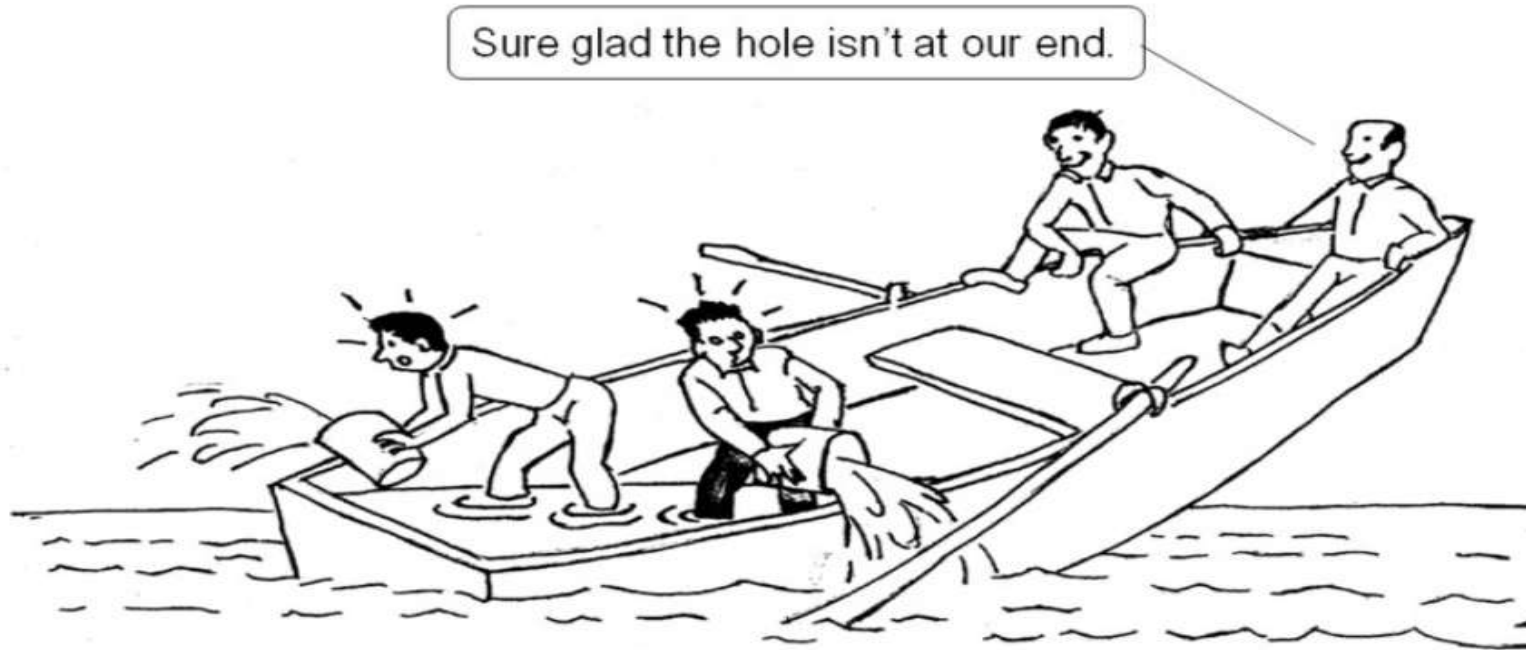
Need for DevOps

Imbalance



Need for DevOps

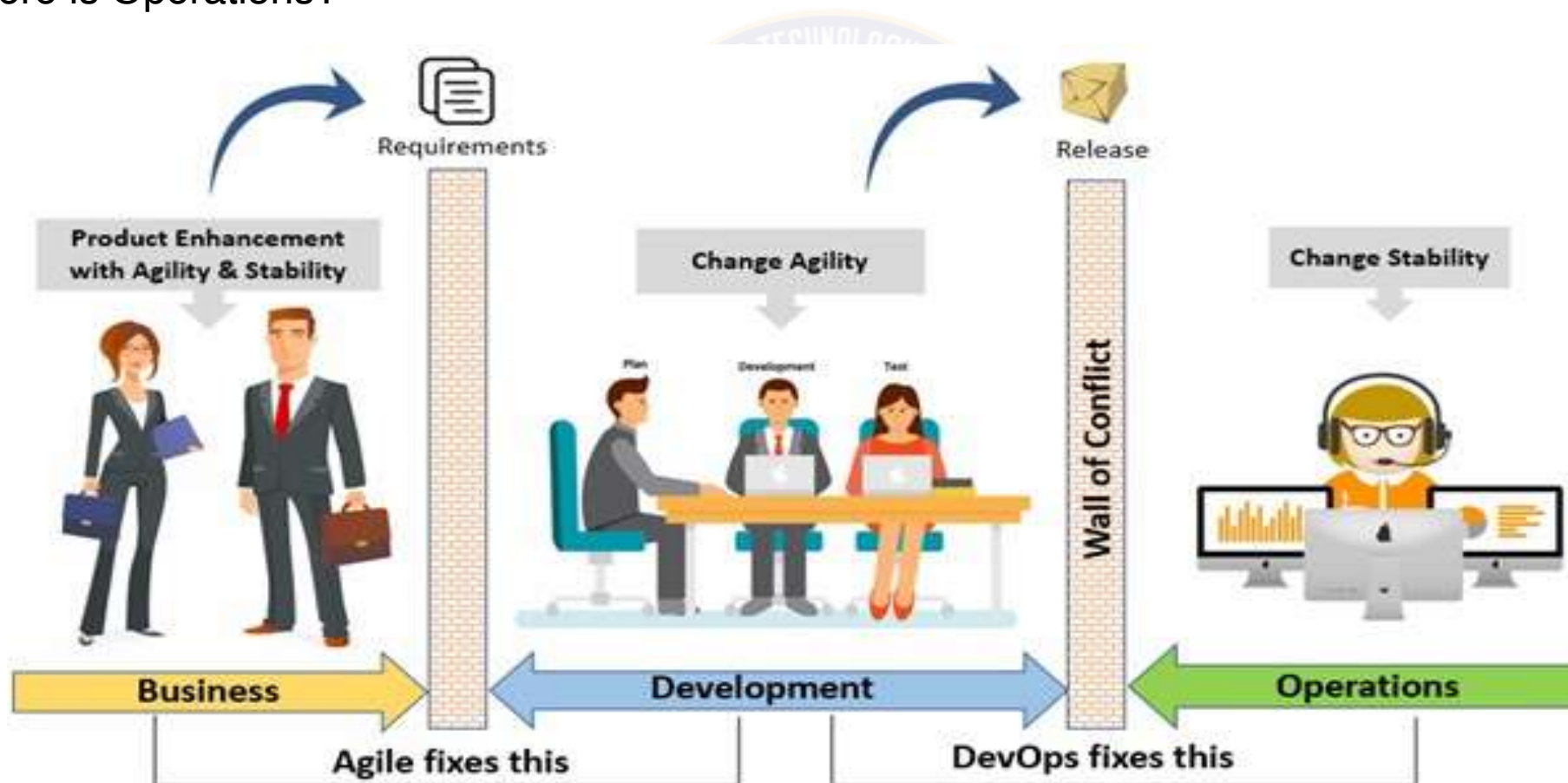
Blame Game



Need for DevOps

Where is Operations?

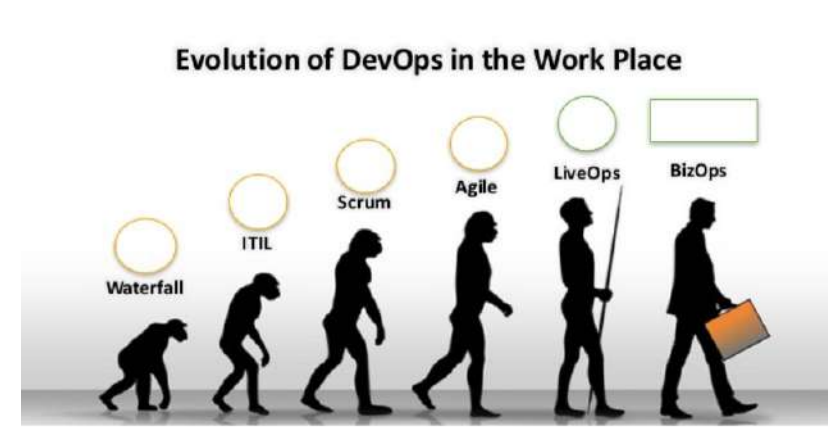
- Development is All Well (Waterfall, Agile)
- Where is Operations?



The evolution of DevOps

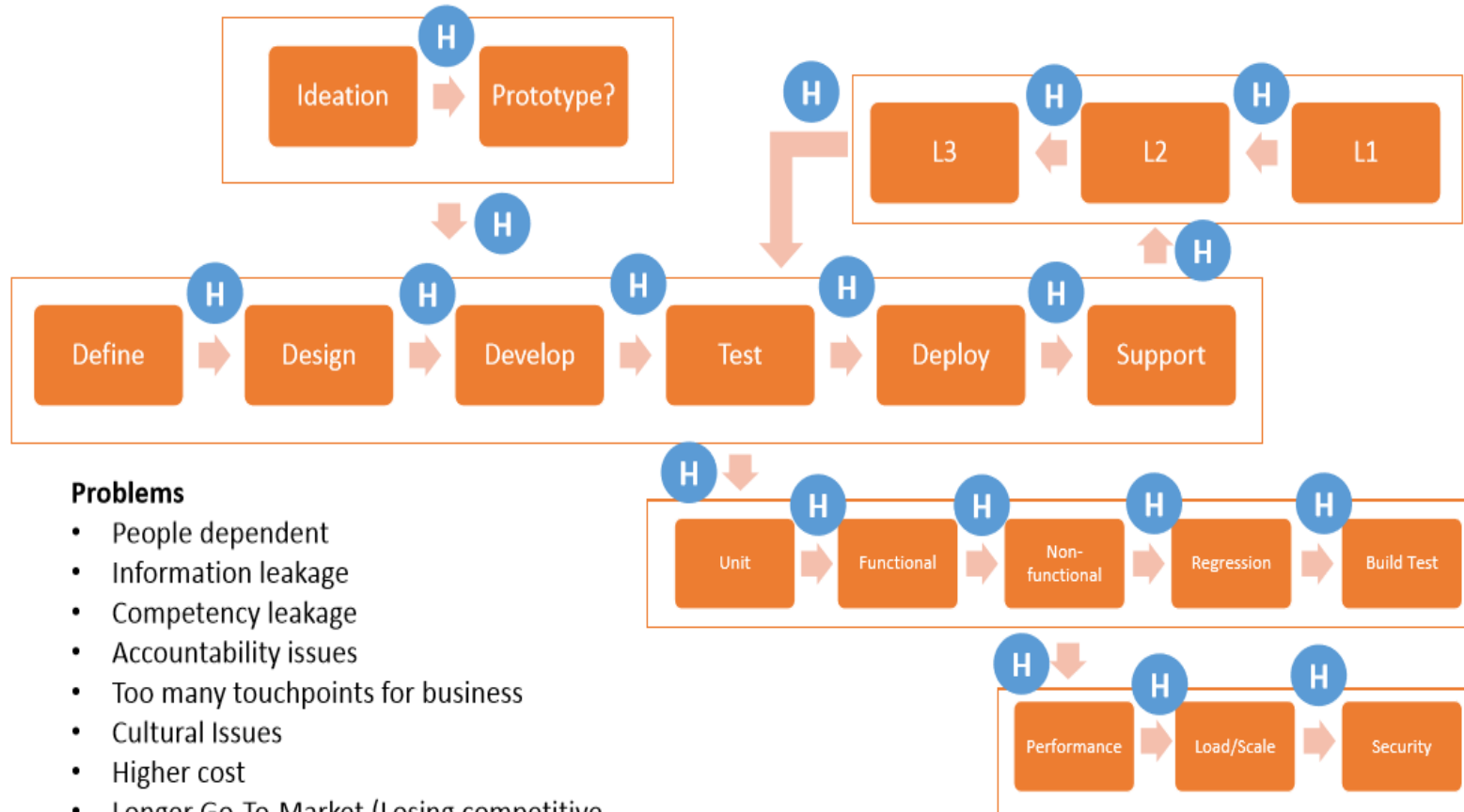
History

- Back in 2007
- Patrick Debois [Belgian Engineer]
- Initially it was Agile Infrastructure but later coined the phrase DevOps
- Velocity conference in 2008
- And if you see you may come across more tangential DevOps Initiative
 - WinOps
 - DevSecOps
 - BizDevOps



The old world before DevOps

More Handshakes

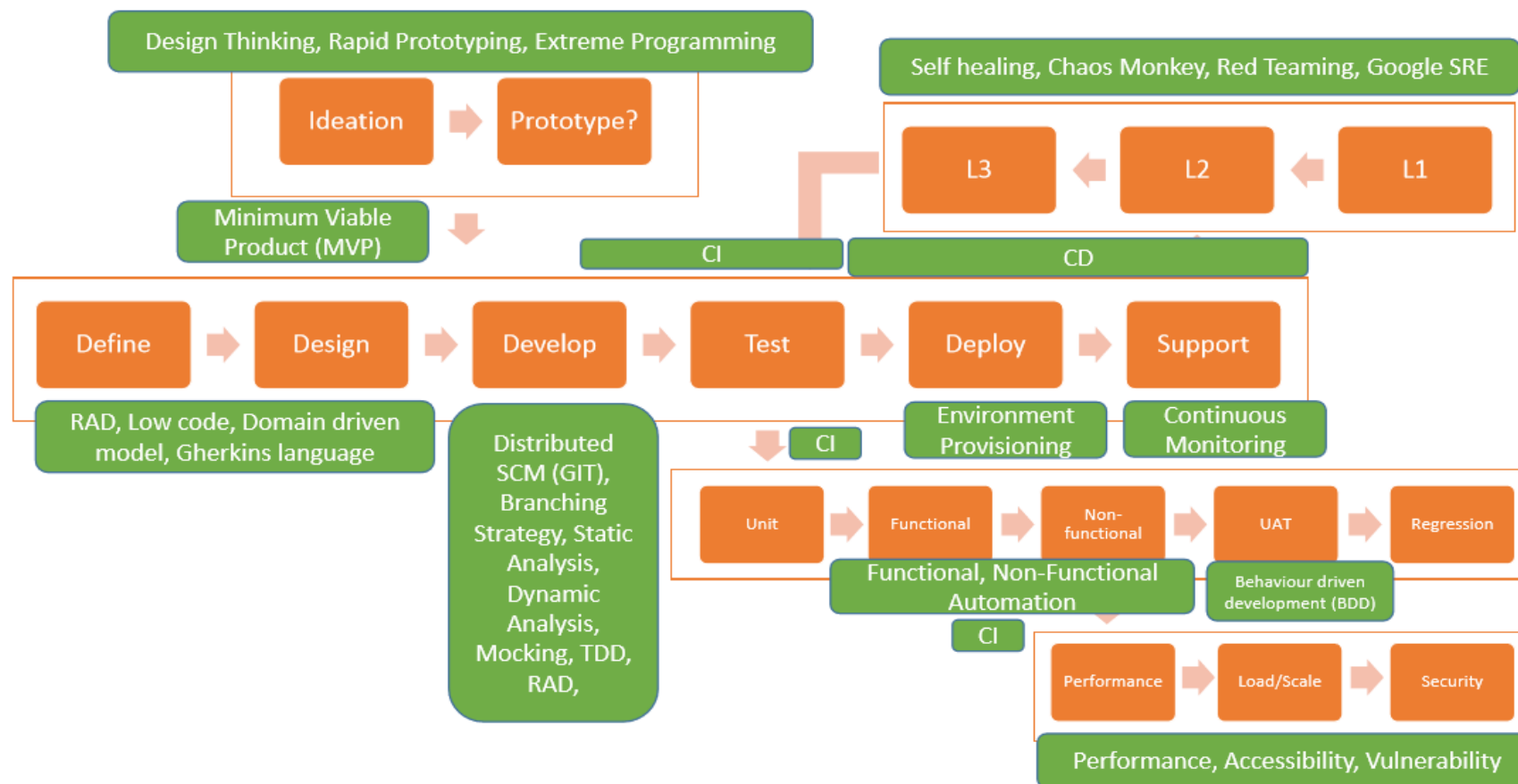


Problems

- People dependent
- Information leakage
- Competency leakage
- Accountability issues
- Too many touchpoints for business
- Cultural Issues
- Higher cost
- Longer Go-To-Market (Losing competitive advantage, Opportunity loss)

Evolution of DevOps :: New world

No More Handshakes



Case Study

Flickr / Yahoo

- Flickr was able to reach
 - 10+ Deploy per day after adopting DevOps
- In 2009
- You May Also Refer:
- YouTube Link: <https://www.youtube.com/watch?v=LdOe18KhtT4>



flickr

Case Study

Netflix

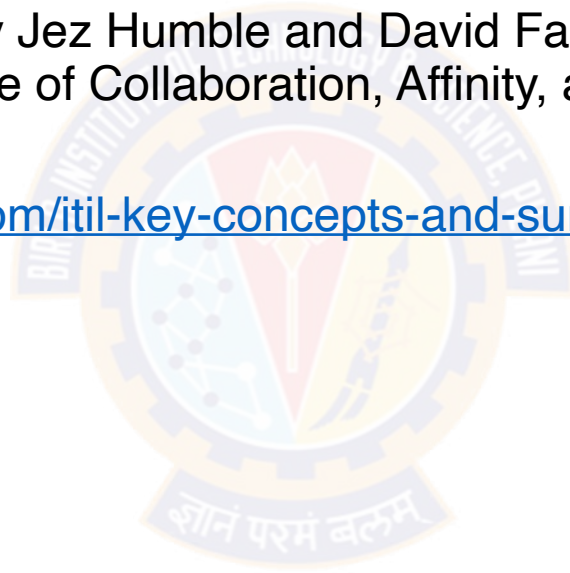
- Netflix's streaming service is a large distributed system hosted on Amazon Web Services (AWS)
- So many components that have to work together to provide reliable video streams to customers across a wide range of devices
- Netflix engineers needed to focus heavily on the quality attributes of reliability and robustness for both server- and client-side components
- Achieved this with DevOps by introducing a tool called Chaos Monkey
- Chaos Monkey is basically a script that runs continually in all Netflix environments, causing chaos by randomly shutting down server instances
- Thus, while writing code, Netflix developers are constantly operating in an environment of unreliable services and unexpected outages
- Unique opportunity to test their software in unexpected failure conditions

NETFLIX

References

CS 1 & 2

- 4th chapter from Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis and Katherine Daniels
- 1st chapter Continuous Delivery by Jez Humble and David Farley and 5th Chapter from Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis and Katherine Daniels
- For ITIL: <https://www.simplilearn.com/itil-key-concepts-and-summary-article>,





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Understanding DevOps

- DevOps Misconception
- DevOps Anti-Patterns
- Three Dimensions of DevOps
 - Process
 - People
 - Tools



DevOps Misconception

DevOps Myths & Misconception

- DevOps is a Team
- CI/CD is all about DevOps
- Non-Compliant to Industry Standards



✓ An additional team is likely to cause more communication issues

✓ Scalability
Consistency
Reliability

✓ Regulations implement control in order to be compliant
Controls reduce the risk that may affect the confidentiality, integrity, and availability of information

DevOps Misconception

Improve the partnership b/w dev test & Ops

DevOps Myths & Misconception

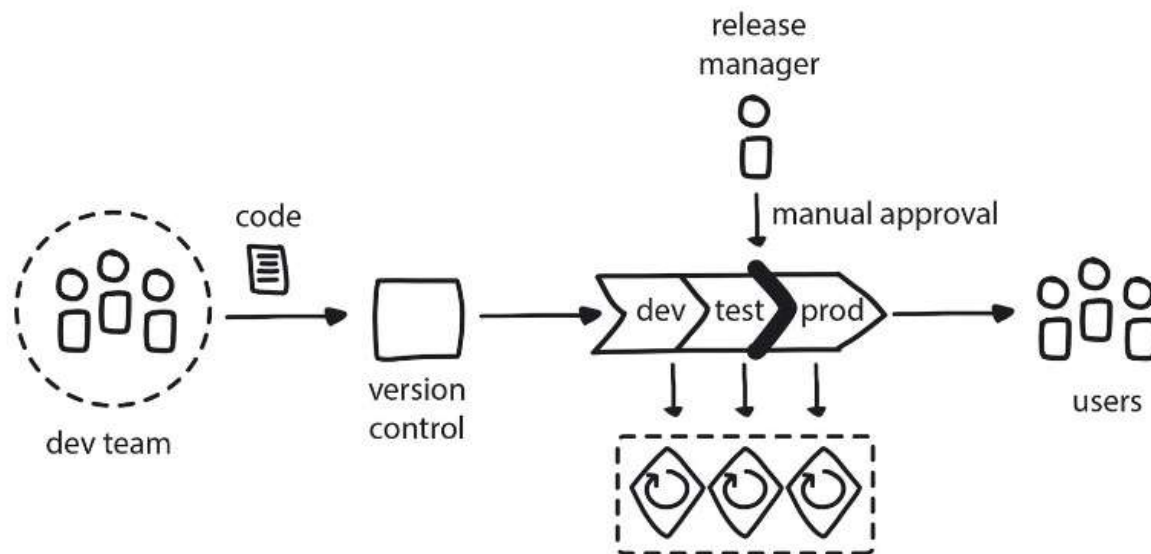
- Automation eliminates ALL bottlenecks
- “Universal” Continuous Delivery pipeline
- All about Tools
- Release as fast as Amazon / Facebook

It increases feedback loops
However Hand off processes can often add to process wait states

Processes to fit organizational and business needs — not vice-versa

Mandating tools developers can use.
Prioritizing tools over people

Empowered smaller self-sufficient teams and Continuous Delivery Engineer



DevOps Anti-Patterns

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

Lessons of History

Example for Silos

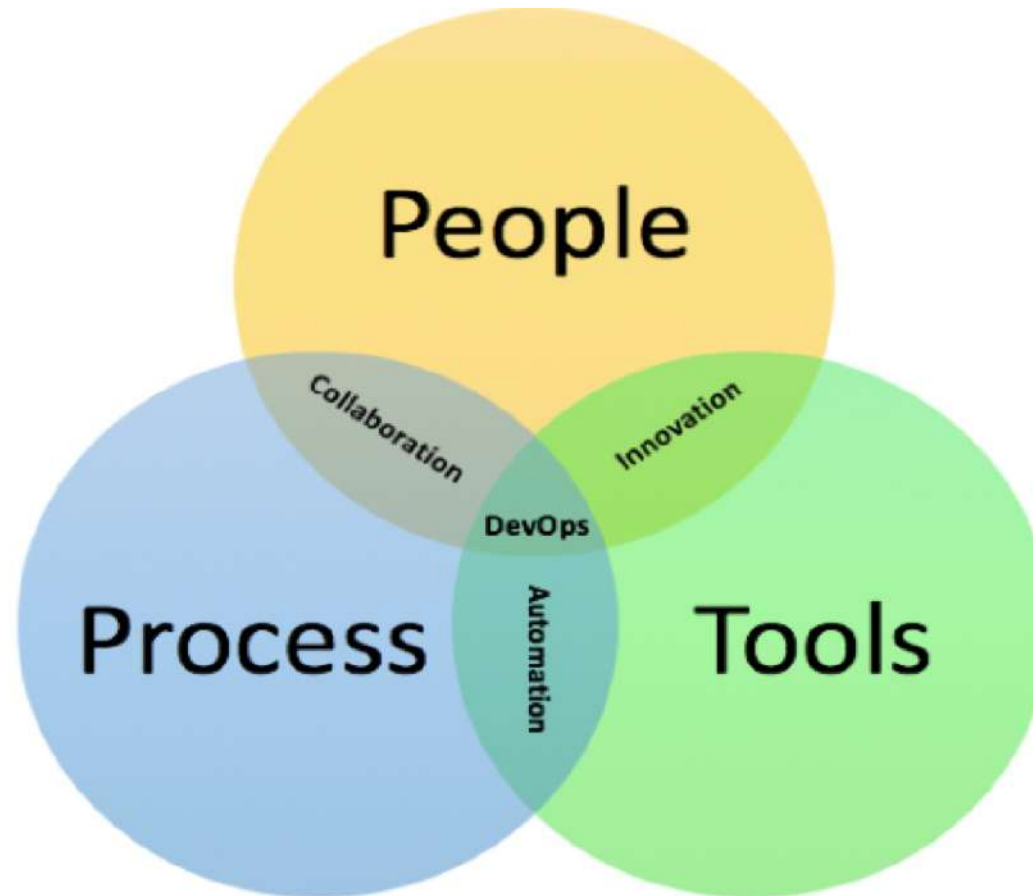
- TVS India
 - T. V. Sundaram Iyengar
 - Founded in 1978



DevOps Dimensions

Three dimensions of DevOps

- People
- Process
- Tools / Technology



DevOps - Process

DevOps and Agile

- 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
- 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

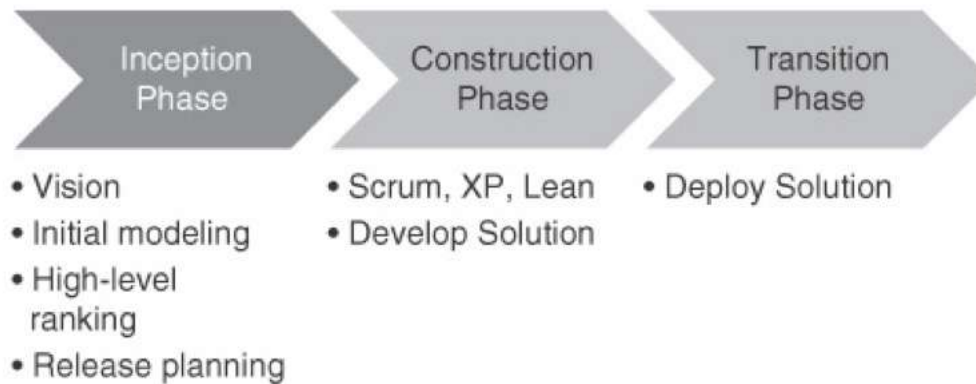


FIGURE 1.2 Disciplined Agile Delivery phases for each release. (Adapted from *Disciplined Agile Delivery: A Practitioner's Guide* by Ambler and Lines) [Notation: Porter's Value Chain]

DevOps and Agile

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



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

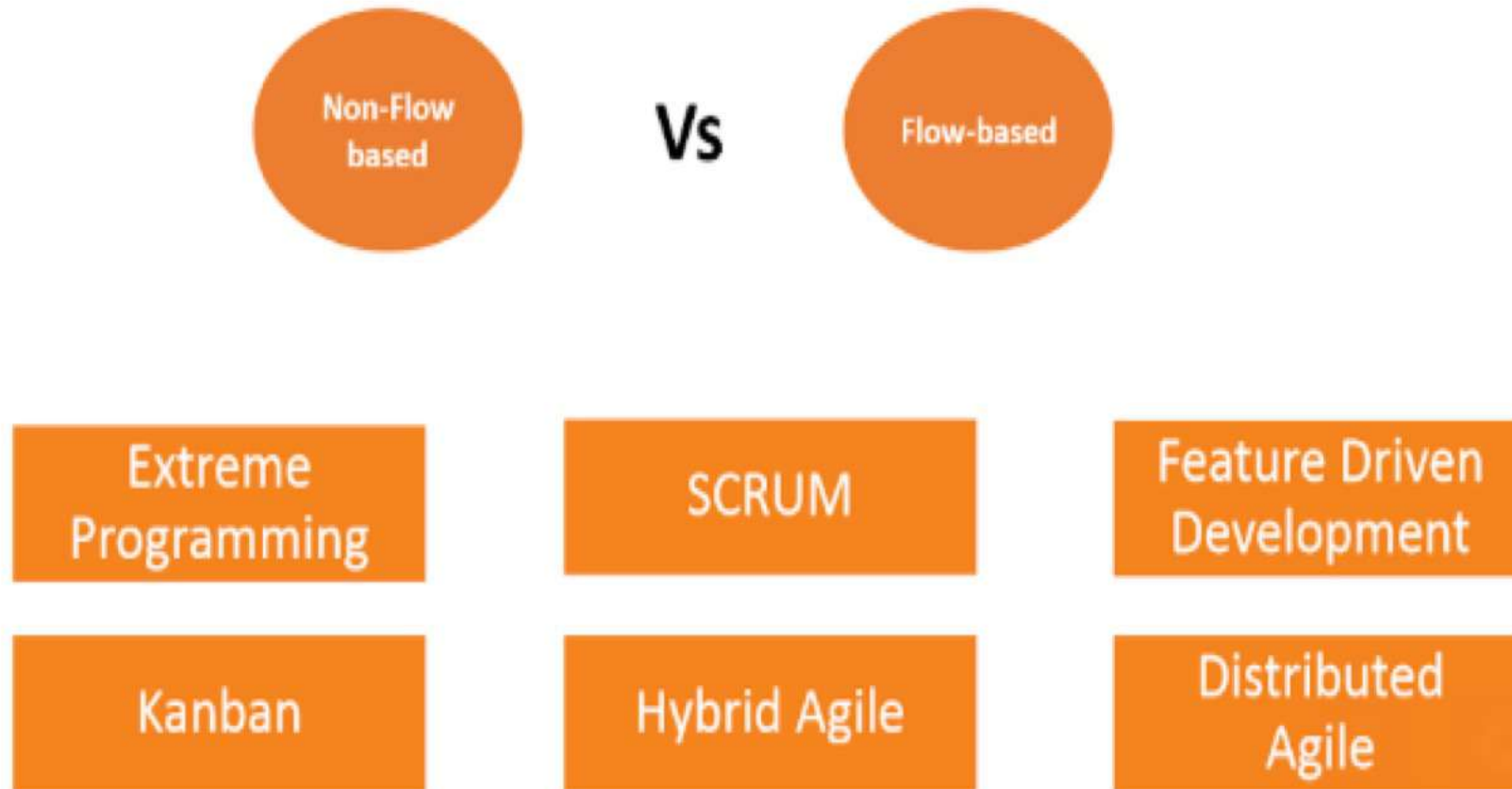
DevOps : Process Dimension

- DevOps and Agile
- Team Structure
- Agile Methodology
 - TDD
 - BDD
 - FDD



DevOps – Process

Process



DevOps – Process

Team Structure

- Choosing appropriate team structure (Resource planning)

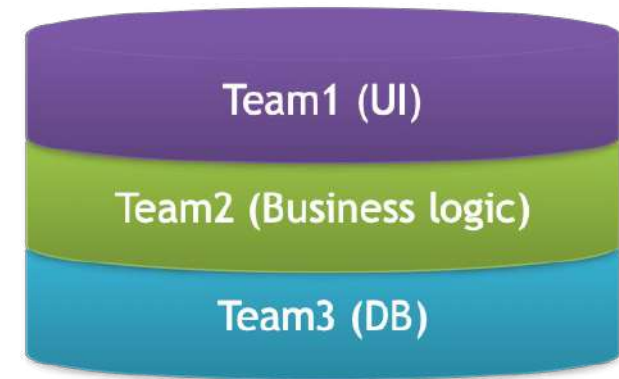
Component Team
Structure

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
- 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

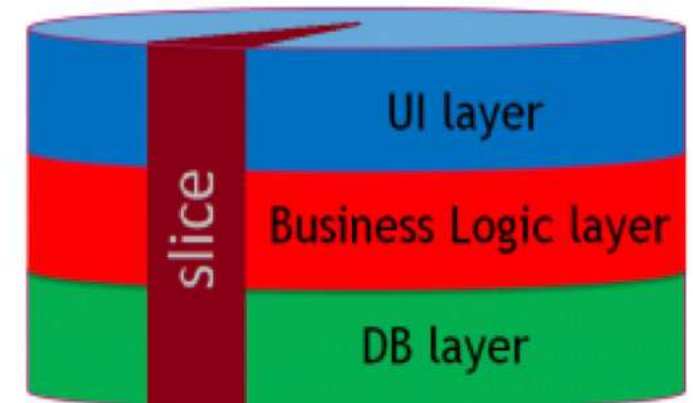
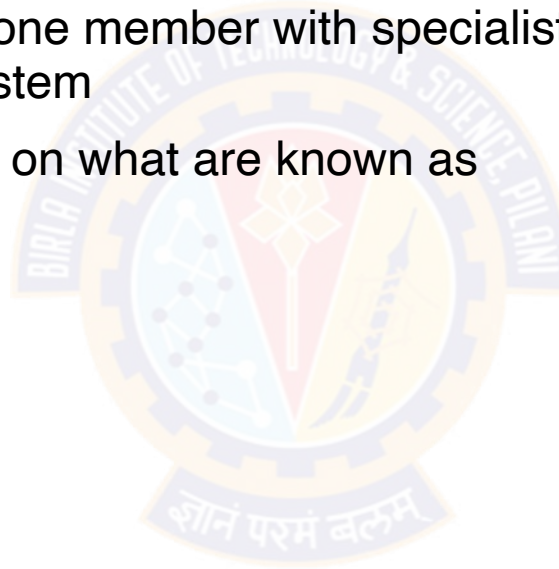


Architecture diagram

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

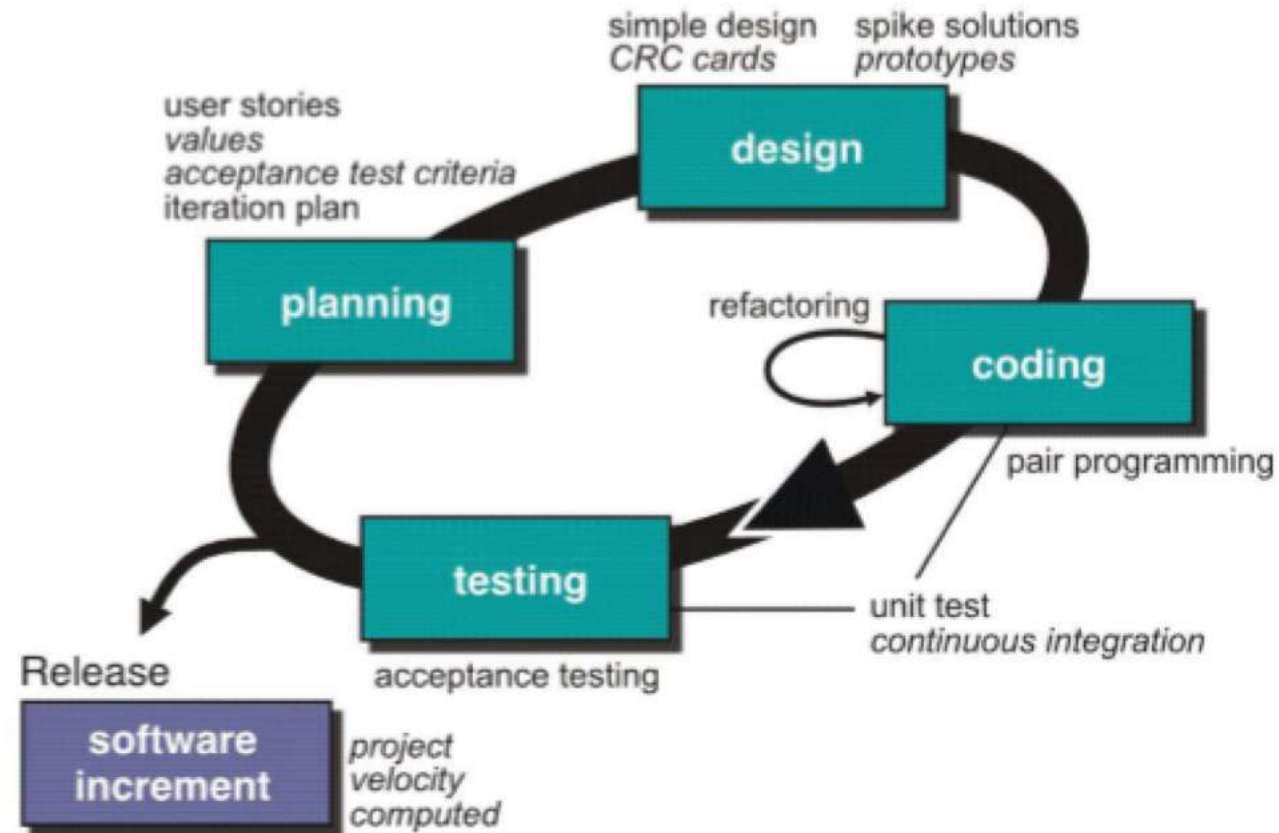
Feature teams

- The characteristics of a feature team are listed below:
 - long-lived—the team stays together so that they can ‘jell’ for higher performance; they take on new features over time
 - 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
- Disadvantages
 - 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

TDD

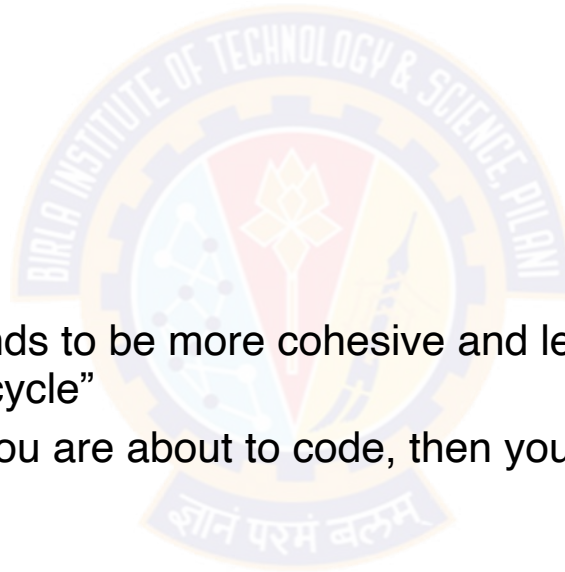
- Prerequisite :: XP



TDD

Understanding

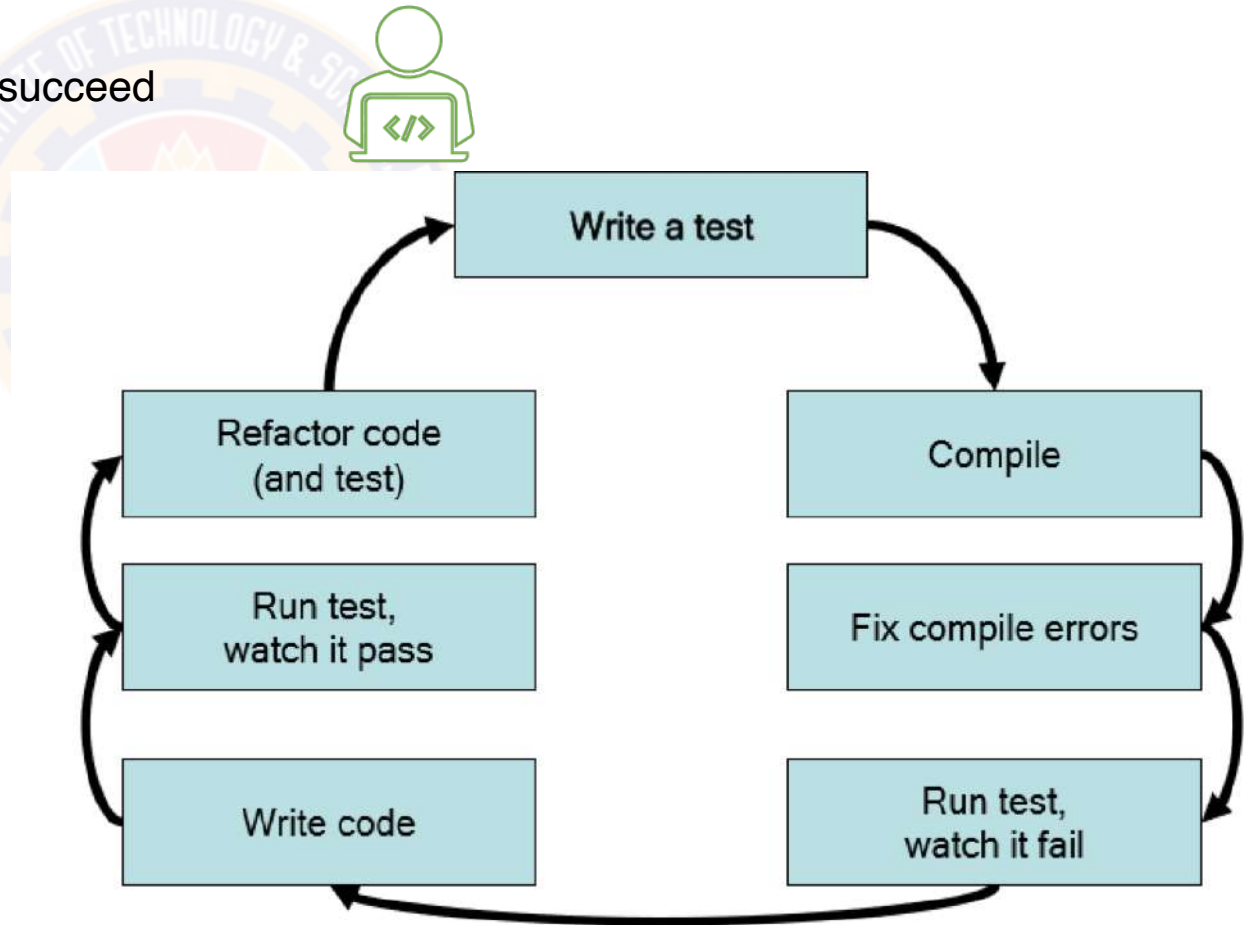
- Many Names
 - Test driven development
 - Test drive design
 - Emergent design
 - Test first development
- TDD Quotes
 - Kent Beck said “Test-first code tends to be more cohesive and less coupled than code in which testing isn’t a part of the intimate coding cycle”
 - “If you can’t write a test for 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

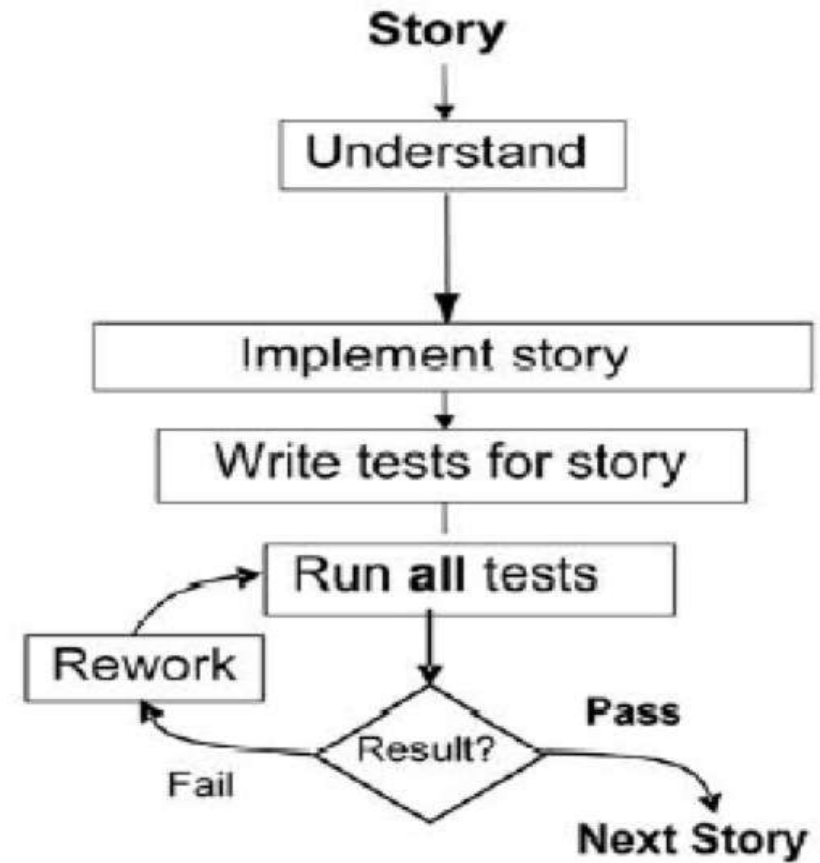
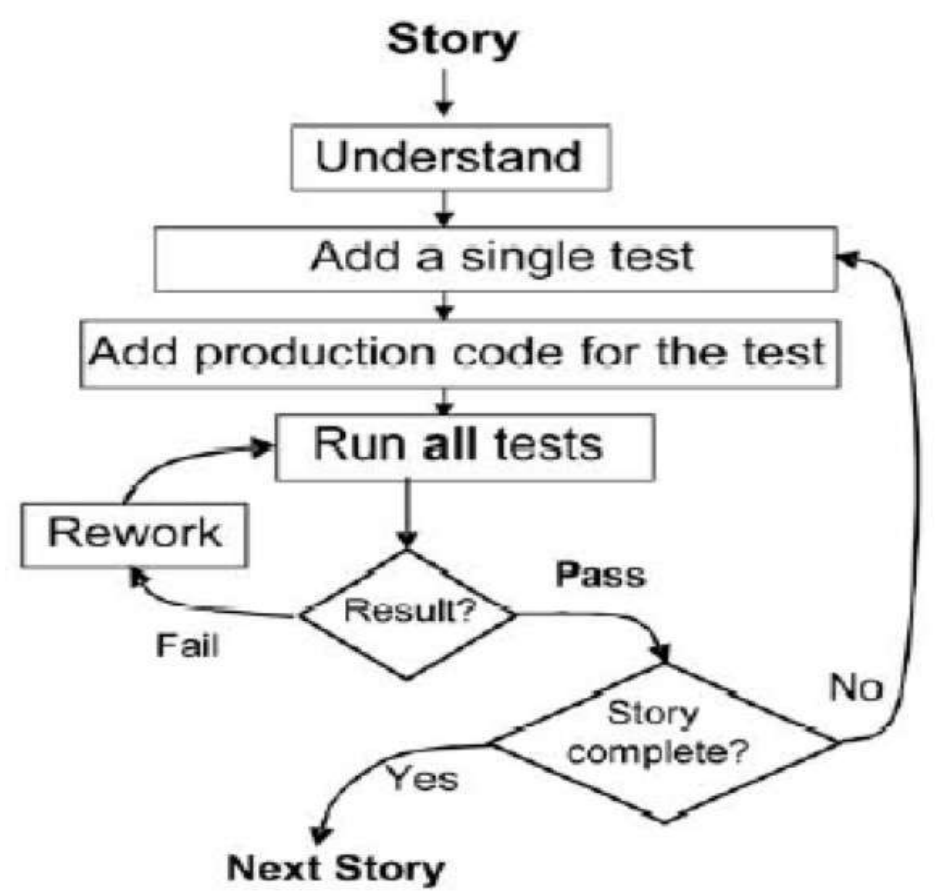
Why TDD

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



TDD

Test First vs. Test Last



TDD Cycle

Red, Green and Refactor

- Example (payment gateway task)

1. Choose a small task
2. Write a failing test (**RED** Test)
3. Write simplest code to make the test pass (**GREEN** Test)
4. **REFACTOR**
5. **Repeat**

Implementation Code

```
payment_gateway = function () {  
    credit_card ();  
    netbanking ();  
    upi_payment ();  
}
```

Run all
test

Tests

.....
.....
.....



```
new_test= function () {  
    payment_gateway_includes_upi_option ();  
}
```

TDD

Lets Have TDD Overview



<https://www.youtube.com/watch?v=uGaNkTahrIw&t=132s>

DevOps: Process

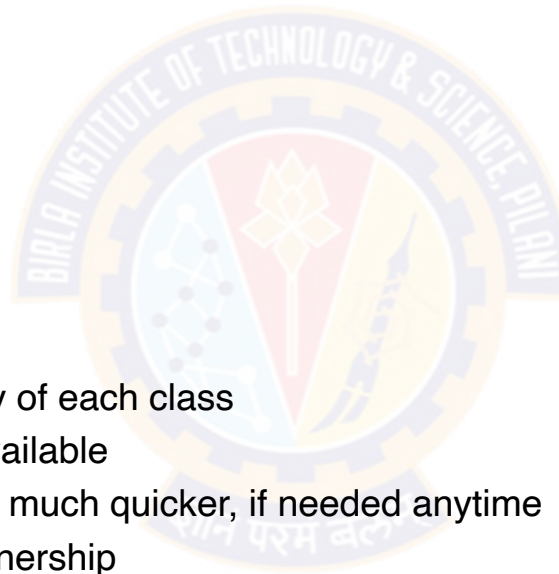
FDD

- What is a Feature?
 - Definition: small function expressed in client-valued terms
 - FDD's form of a customer requirement



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 Primary Roles

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

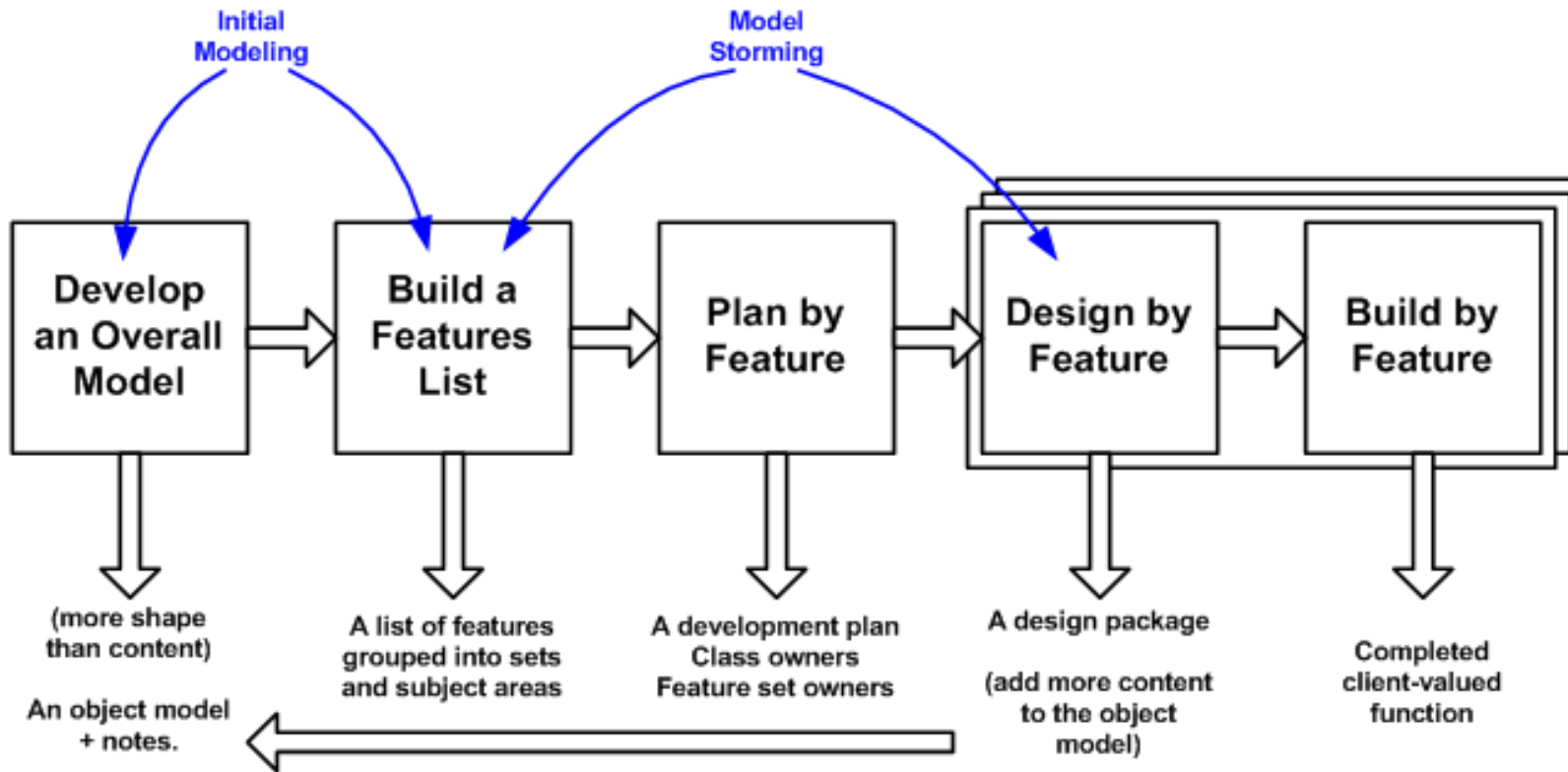


Feature Driven Development Process

- 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

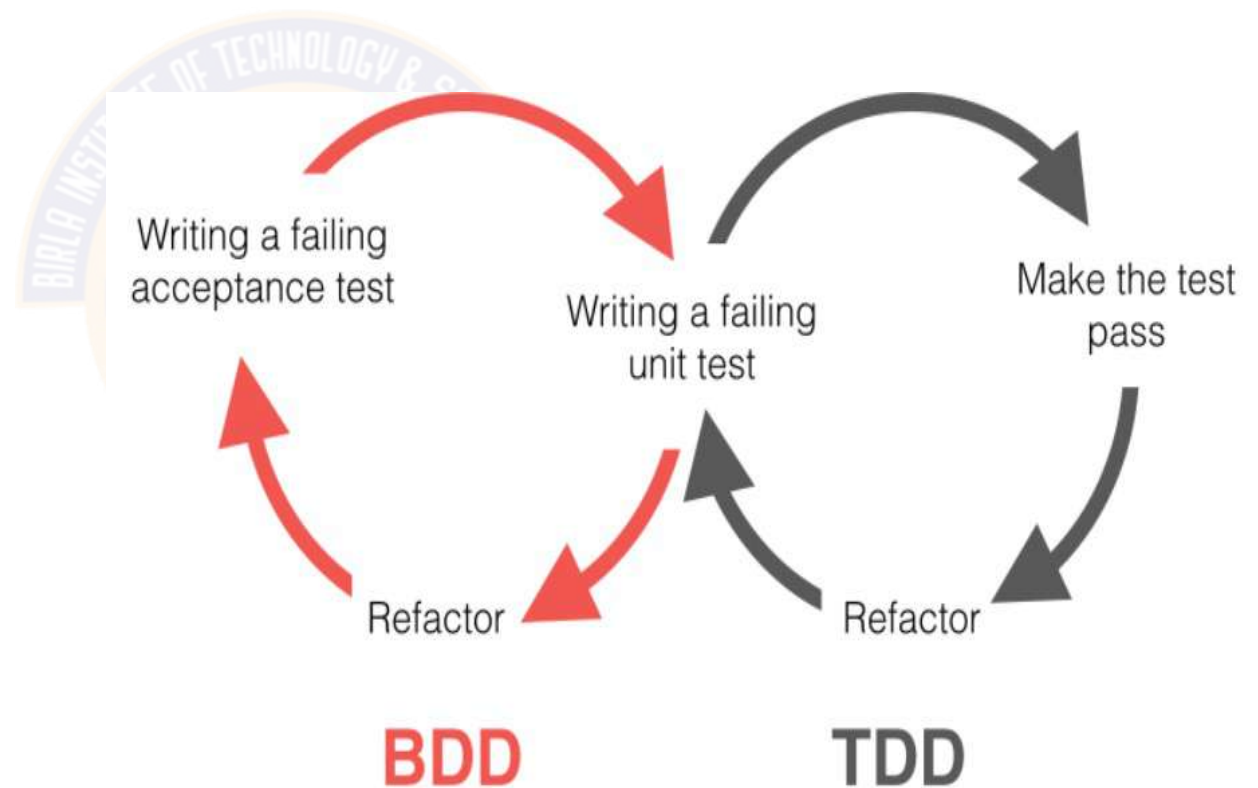
Feature Driven Development Process



DevOps: Process

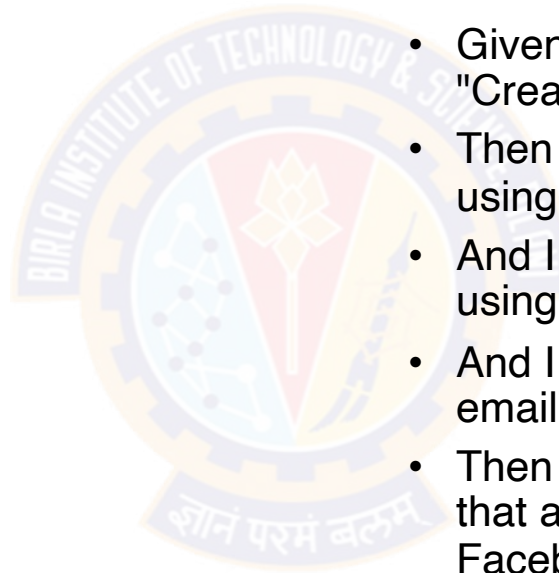
BDD

- What is BDD?:
 - General Technique of TDD
 - Follows same principle as TDD
 - Shared tools
 - Shared Process



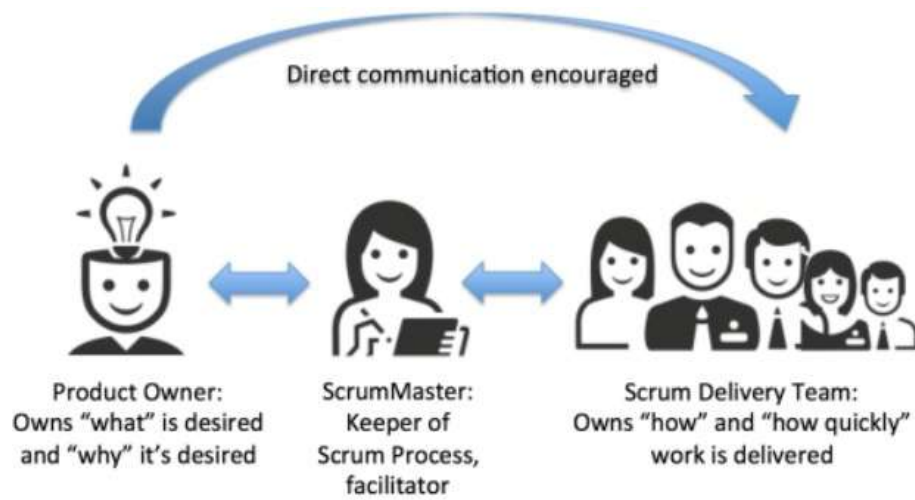
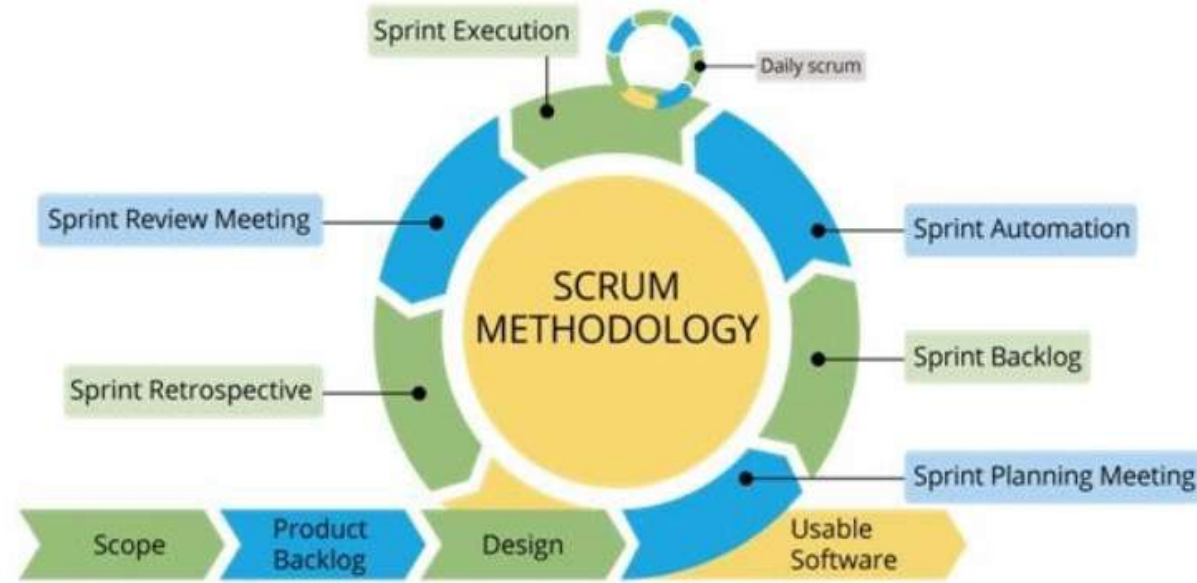
BDD Basic structure : Example

- User story:
 - As someone interested in using the Mobile app, I want to sign up in from the app so that I can enjoy my 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



DevOps: Process

SCRUM



SCRUM

SCRUM Overview

- You can refer below YouTube video for understanding the structure of organization to be agile.
- This Video is of First Bank in world to adopt Agile Scrum
- <https://www.youtube.com/watch?v=uXg6hG6FrG0>



References

CS 2 & 3

- Chapter 5 from Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis and Katherine Daniels
-
- For BDD and FDD: <https://www.agilealliance.org>





Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

DevOps : People & Tools Dimension

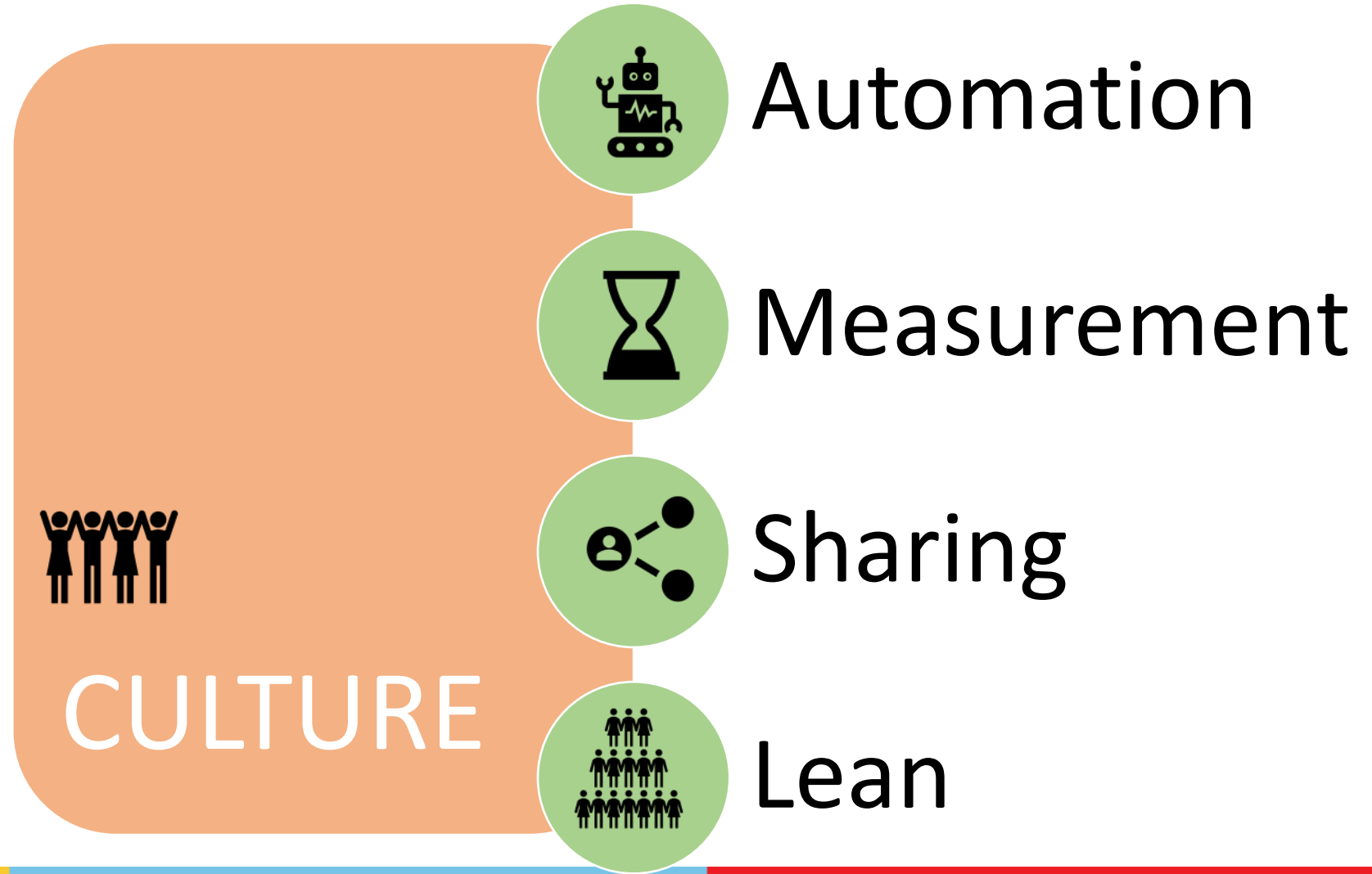
- Transformation to Enterprise DevOps culture
- DevOps - People
- DevOps – Tools



Transformation to Enterprise DevOps culture

DevOps Values

More than anything else, DevOps is a culture movement based on human and technical interaction to improve relationships and results



Transformation to Enterprise DevOps culture

Initial Planning for Enterprise Readiness

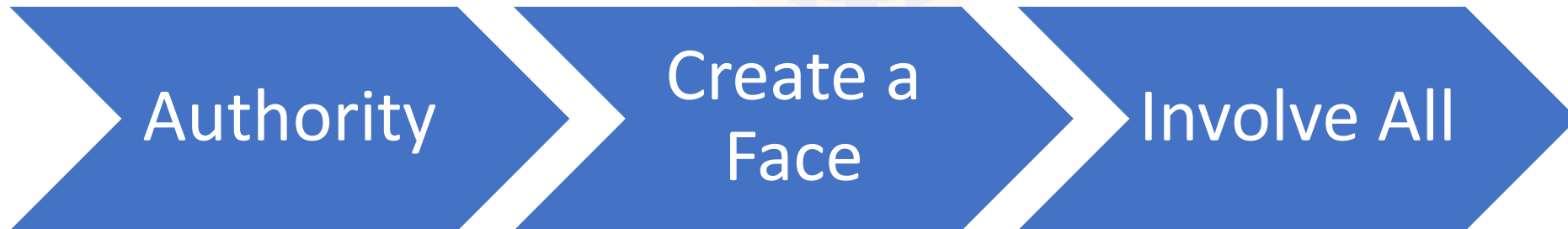
- Participants should absolutely include all the towers that make up the solution delivery
- Design Thinking : It is a great method; it leverages the expertise of all stakeholders, enables them to come to a common understanding
- High Level Output



Transformation to Enterprise DevOps culture

Establish a DevOps Centre of Excellence

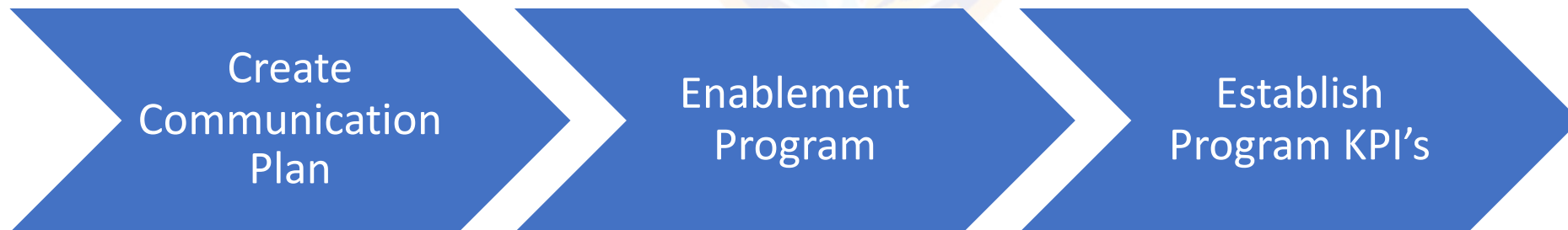
- At the right organizational level and enterprise authority
- Create a Face: It has to be led by an enterprise leader who has the support and buy-in from all the towers
- Active Participant from All Towers of Delivery & Participants must be chosen wisely
- These phases help concrete the vision and strategies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



Transformation to Enterprise DevOps culture

Establish Program Governance

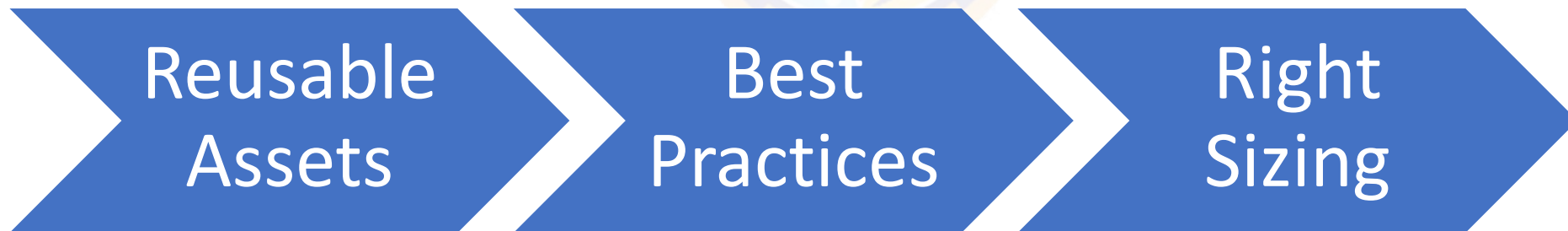
- Agile and DevOps, practitioners' roles and responsibilities will change
- Need awareness, enablement and empowerment to succeed
- KPIs must shift from individual metrics to holistic customer business outcomes



Transformation to Enterprise DevOps culture

Establish Project In-take Process

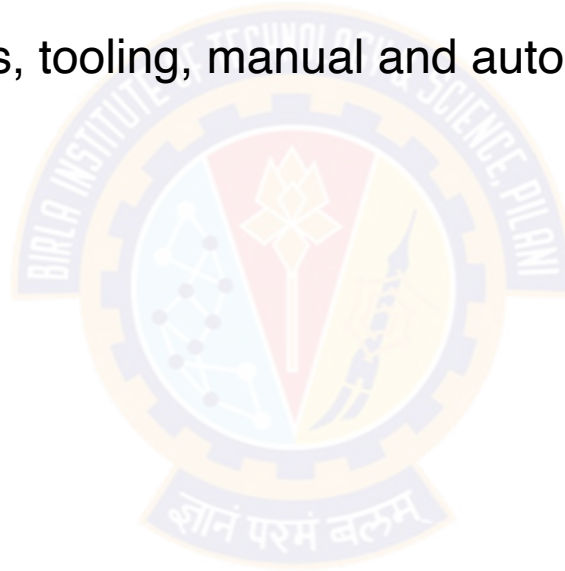
- DevOps SME's conduct in-take workshops for Scrum Teams
- Automation scripts, Infrastructure assets
- Test Automation, Branching and Merging & Lessons Learned
- Fit for purpose tool selection and Application Criticality



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



DevOps - People

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

DevOps – People

Managing People, Not Resources

“Managing teams would be easy if it wasn’t for the people you have to deal with”

- Calling people as resources 
- A resource is something you can manage without much tailoring
- 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



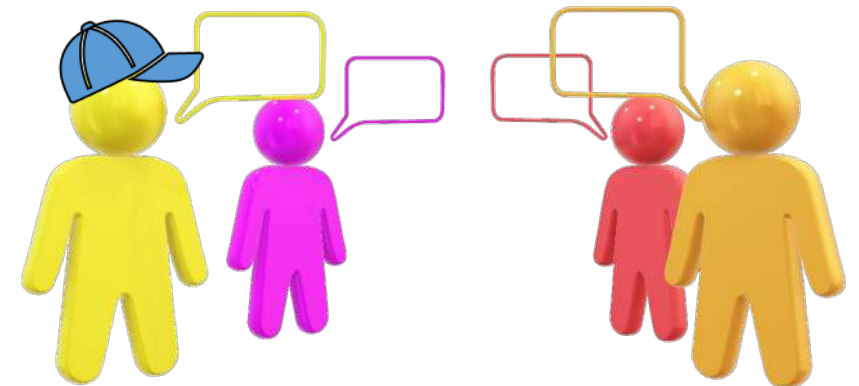
Vs



DevOps – People

Don't change Organization; Change your self

- 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
- Ex. “Jack Welch” he was the CEO of General Electric till 2001, his cabin was of all transparent glass; And this cabin did not have any door. So, it was an open-door cabin. And the label on the door was “PLEASE DISTURB ME”.



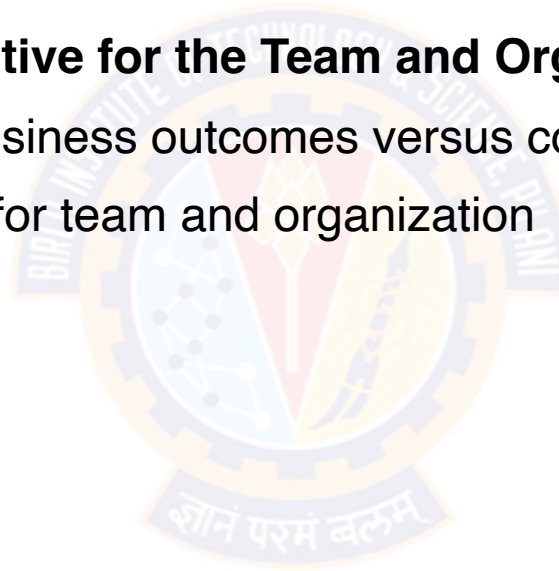
DevOps – People

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

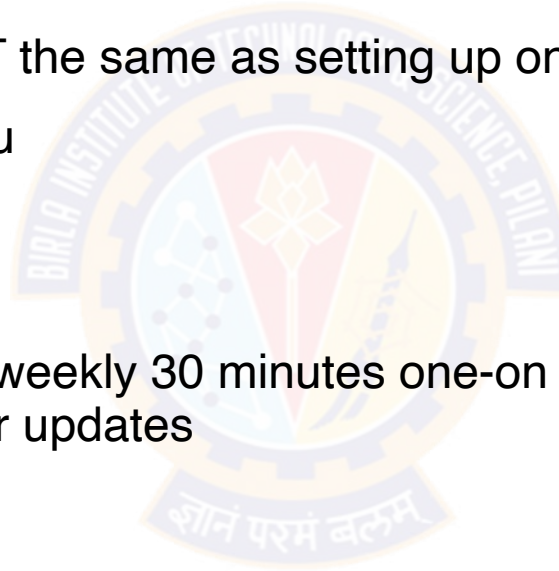


DevOps isn't the goal. It helps you reach your goals

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?



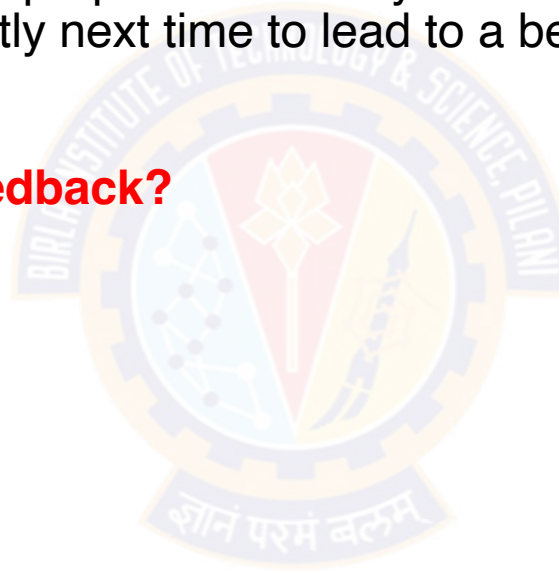
Effective Management of People

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? “

What You hear in your Feedback?

- Focus on feedback?



Effective Management of People

Delegation

- Feel on delegation of JOB to others?

Managerial Economics 101



Effective Management of People

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

Effective Management of People

Measuring Your Organizational Culture

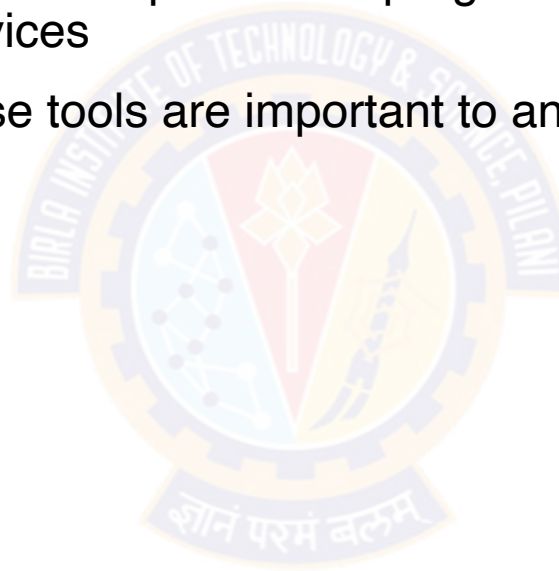
- There are a few different ways to measure culture
- The Westrum survey measures
 - On my team, information is actively sought
 - On my team, failures are learning opportunities, and messengers of them are not punished
 - On my team, responsibilities are shared
 - On my team, cross-functional collaboration 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 – 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 your 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

DevOps Tools

Version Control

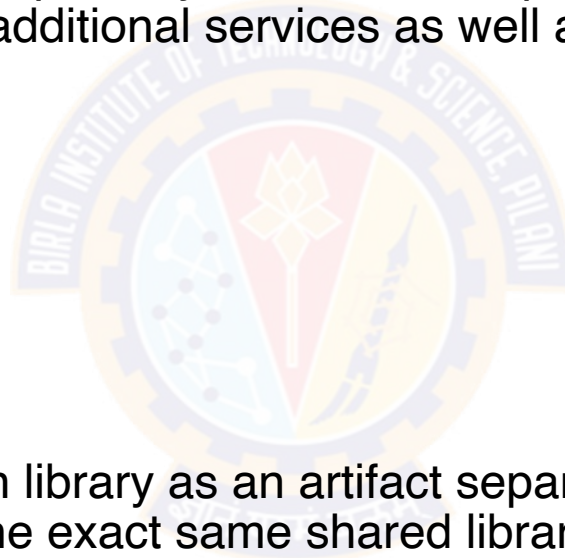
- Having the ability to commit, compare, merge, and restore past revisions of objects to the repository allows for richer cooperation and collaboration within and between teams
- Version control enables teams to deal with conflicts that result from having multiple people working on the same file or project 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;
 - Contributions to your own repositories;
 - Defining processes for contributing; and
 - Sharing commit rights



DevOps Tools

Artifact Management

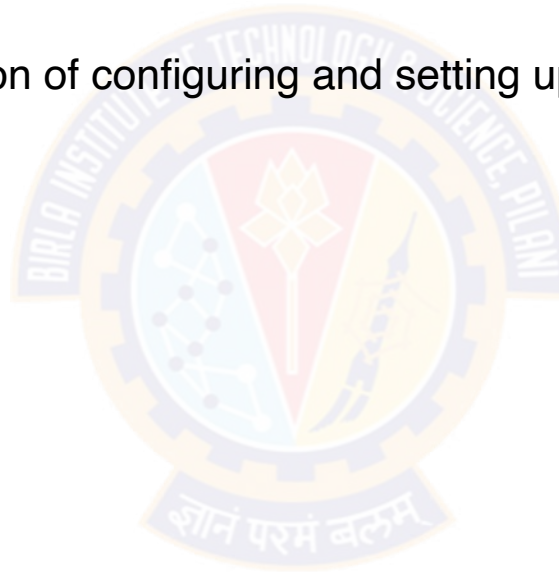
- An artifact is the output of any step in the software development process
- When choosing between a simple repository and more complex feature-full repository, understand the cost of supporting additional services as well as inherent security concerns
- An artifact repository should be:
 - Secure;
 - 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



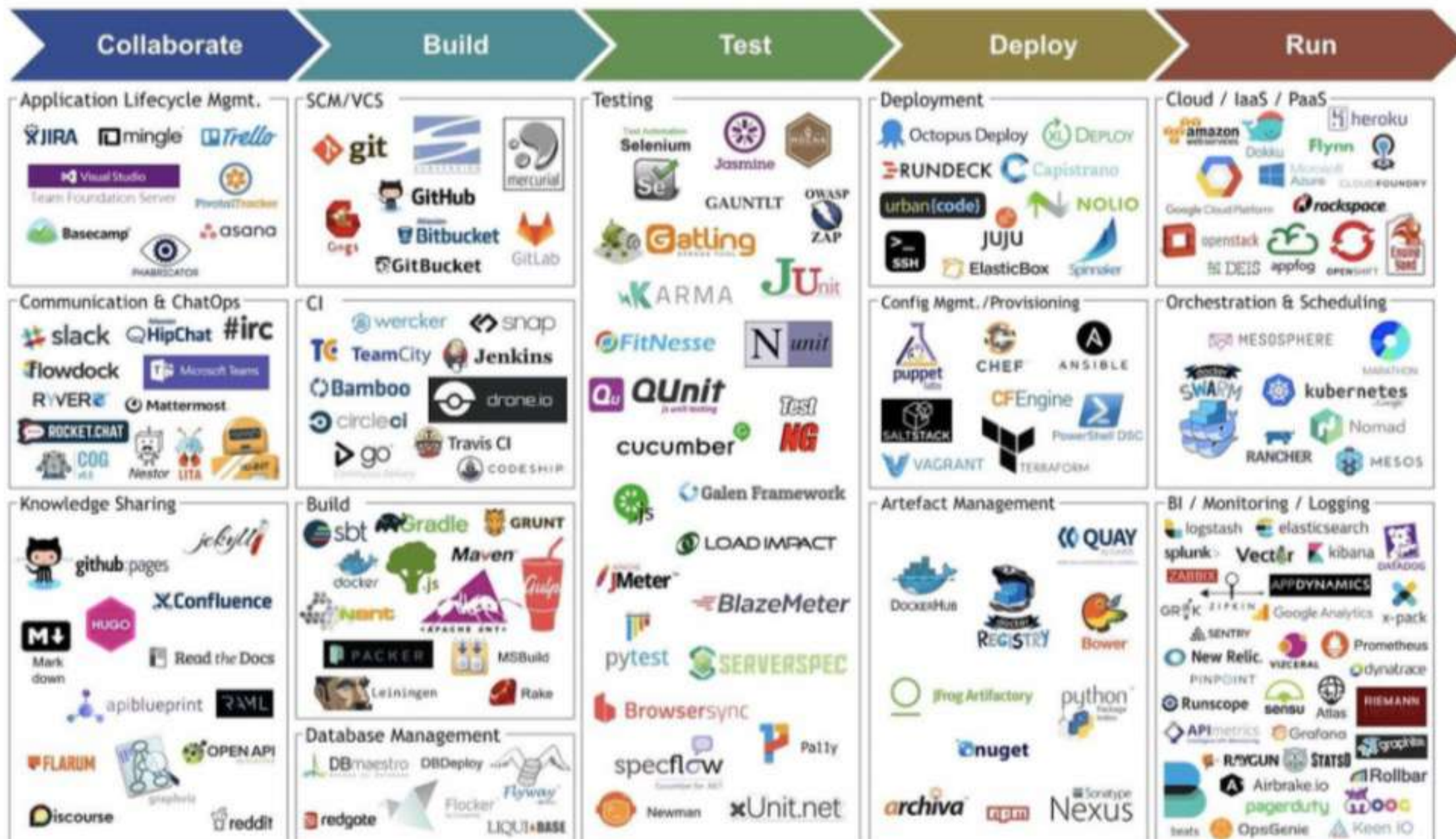
DevOps Tools

Other Automation Tools

- 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 Ecosystem



PERIODIC TABLE OF DEVOPS TOOLS (V2) (V1)

EMBED DOWNLOAD ADD

Os Open Source
Fr Free
Fm Freemium
Pd Paid
En Enterprise

SCM
CI
Deployment
Cloud / IaaS / PaaS
BI / Monitoring

Database Mgmt
Repo Mgmt
Config / Provisioning
Release Mgmt
Logging

Build
Testing
Containerization
Collaboration
Security

3	En	4	En	5	En	6	En	7	En	8	En	9	En	10	En
Ch		Pu		An		Sl		Dk		Az					
Chef		Puppet		Ansible		Salt		Docker		Azure					
11	En	12	En	13	En	14	En	15	En	16	En	17	En	18	En
Ot		Bl		Va		Tf		Rk		Gc					
Octo		BladeLogic		Vagrant		Terraform		Red Hat		Google Cloud Platform					
19	En	20	En	21	En	22	En	23	En	24	En	25	En	26	En
Gd		Sf		Cn		Bc		Lxc		Rs					
Deployment Manager		SmartFrog		Consul		Redfish		Linux Containers		Rackspace					
27	En	28	En	29	En	30	En	31	En	32	En	33	En	34	En
Cp		Ju		Rd		Cf		Ds		Op					
Capistrano		JUnit		Rundeck		Cucumber		Docker Swarm		OpenStack					
35	En	36	En	37	En	38	En	39	En	40	En	41	En	42	En
Ry		Cy		Oc		No		Kb		Bx					
RapidDeploy		CodeDeploy		Octopus Deploy		CA Notify		Kubernetes		Bluemix					
43	En	44	En	45	En	46	En	47	En	48	En	49	En	50	En
Xld		EB		Dp		Ud		Fl		Os					
XL Deploy		ElasticBox		Deploybot		UrbanCode Deploy		Fleet		OpenShift					



Q&A



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Could for DevOps & Version Control System

- Cloud as a catalyst for DevOps
- Evolution of Version Control
- Version Control System Types
 - Centralized Version Control Systems
 - Distributed Version Control Systems
- Introduction to GIT
- GIT Basics commands
- Creating Repositories, Clone, Push, Commit, Review
- Git Branching
- Git Managing Conflicts
- Git Tagging
- Git workflow
 - Centralized Workflow
 - Feature Branch Workflow
- Best Practices- clean code



DevOps Tools

Cloud as a Catalyst for DevOps

- 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)

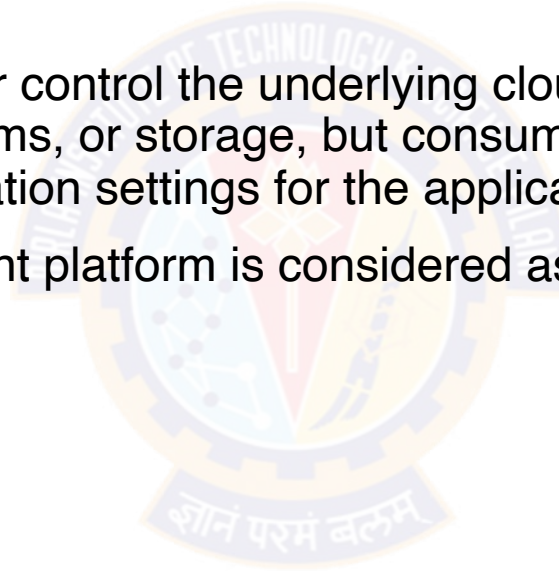
- In this the consumer is provided the capability to use the provider's applications running on a cloud infrastructure
- The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based e-mail) or an application interface
- The 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

ज्ञानं परमं बलम्

Cloud as a Catalyst for DevOps

Platform as a Service (PaaS)

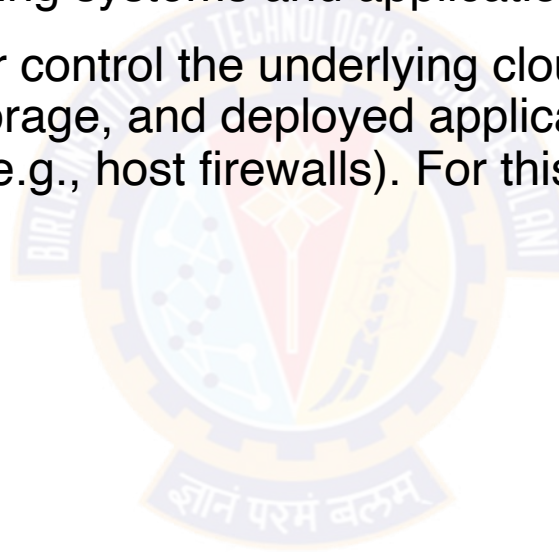
- The consumer is provided the capability to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider
- The consumer does not manage or control the underlying cloud infrastructure including 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)

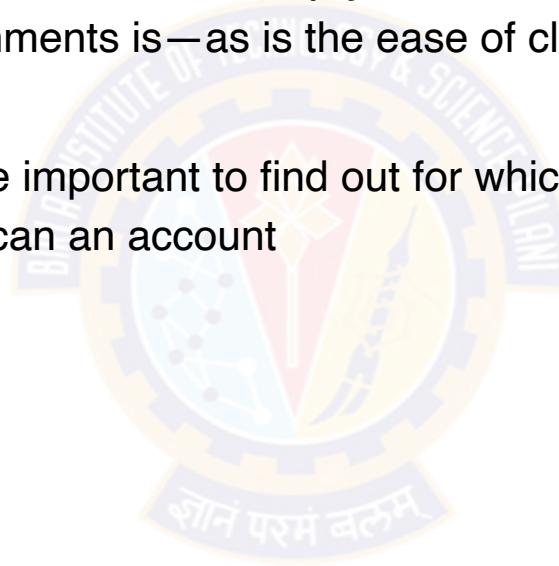
- The consumer is provided the capability to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications
- The consumer does not manage or control the underlying 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

Three of the unique aspects of the cloud that impact DevOps

- Three of the unique aspects of the cloud that impact DevOps:
- The ability to create and switch environments simply
 - Simply create and migrate environments is—as is the ease of cloning new instances
- The ability to create VMs easily
 - Administering the running VMs are important to find out for which VM we are paying but not using it
 - Tool such as, Janitor Monkey to scan an account
- The management of databases



Evolution of Version Control

Generations of VCS

- What is “version control”, and why should you care?
- Definition: 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

Version Control System

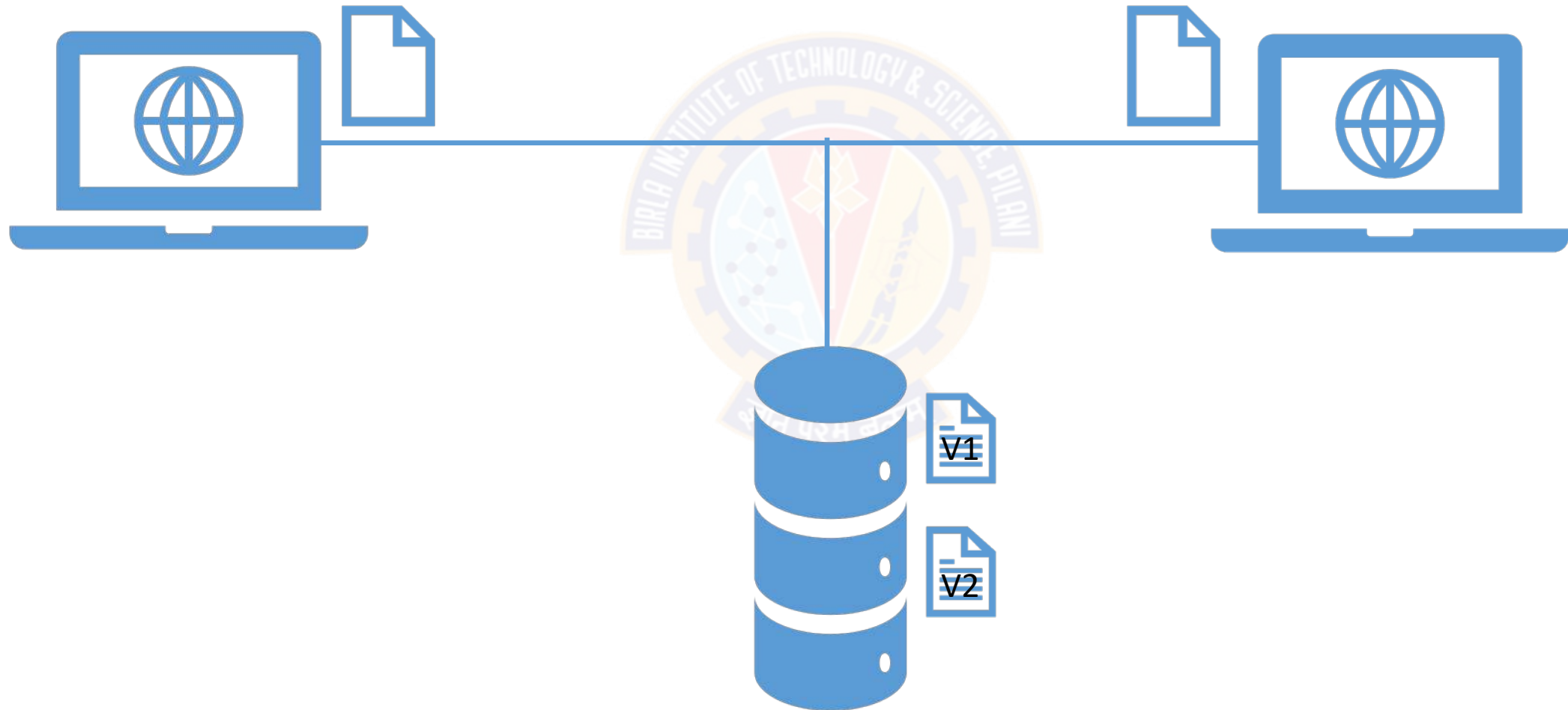
Benefits

- Change history
- Concurrent working (Collaboration)
- Traceability
- Backup & Restoration



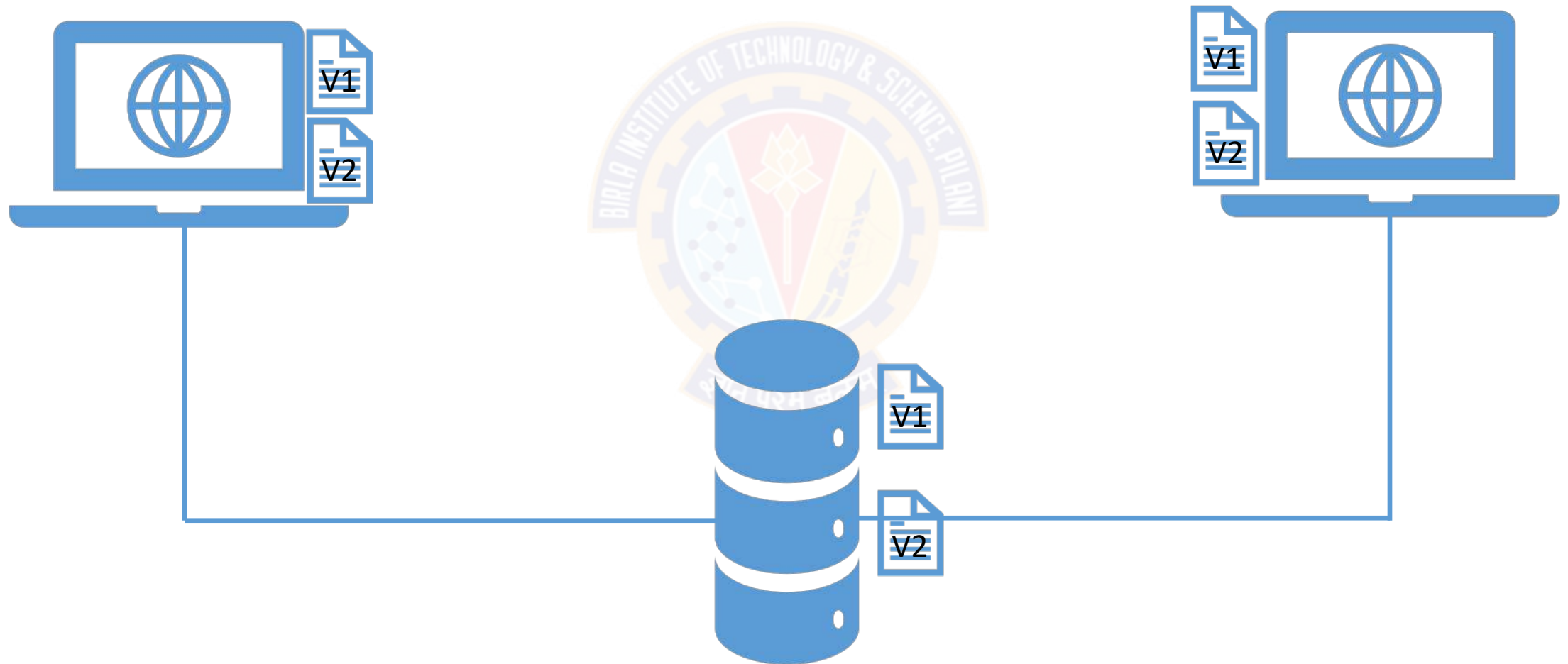
Version Control System Types

Centralized source code management System



Version Control System Types

Distributed source code management System



CVCS Vs. DVCS

Lets discuss con's

CVCS

- Single point of failure
- Remote commits are slow
- Continuous connection

DVCS

- Need more space
- Bandwidth for large project



Available Tools

CVCS

- Open source:
 - Subversion (SVN)
 - Concurrent Versions System (CVS)
 - Vesta
 - OpenCVS
- Commercial:
 - AccuRev
 - Helix Core
 - IBM Rational ClearCase
 - Team Foundation Server (TFS)

DVCS

- Open source:
 - Git
 - Bazaar
 - Mercurial
- Commercial:
 - Visual Studio: Team Services
 - Sun WorkShop: TeamWare
 - Plastic SCM – by Codice Software, Inc
 - Code Co-op



Git & GitHub

What we will learn in this session

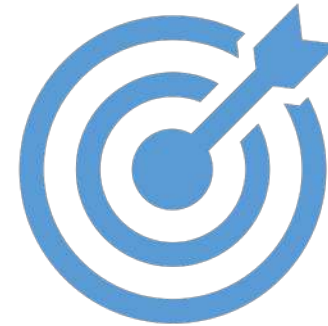
- Git & GitHub relationship
- Prerequisite
- Foundation of Git



Concept of Git



Understanding GitHub



Beyond the basics

Git

What is Git



Popular Source Control system



Distributed system



Free (Open Source tool)



Git

Why use Git

Fast

Disconnected

Powerful yet easy

Branching

Pull Requests



GitHub

What is GitHub?



Hosting service on Git



More than just source control for your code

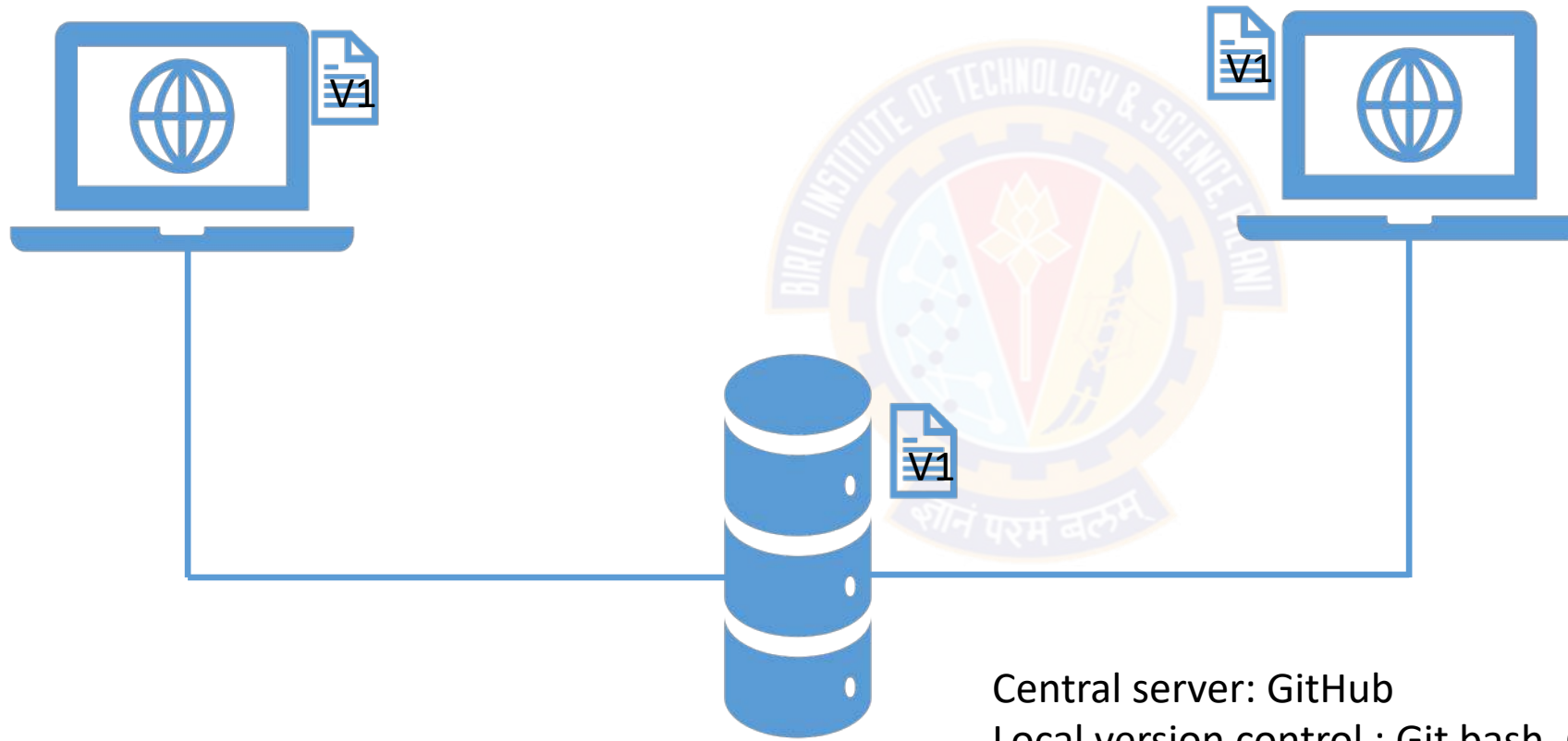
- Issue management, Working with Teams, etc.,



Free & Paid options

Git & GitHub

Relationship



Git

Working with Git



Console

We will use this



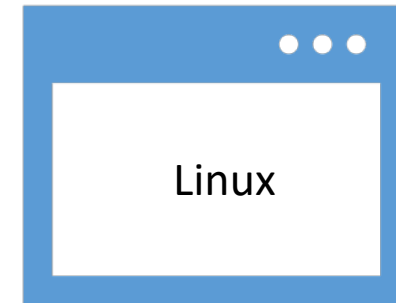
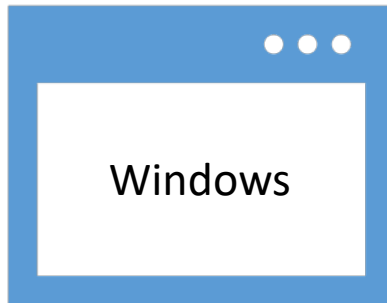
GUI

GitHub Desktop
Source tree

Git & GitHub

Getting your system ready

- Install appropriate Git bash form official site
- Command line (Git bash)
- Server account (GitHub account)



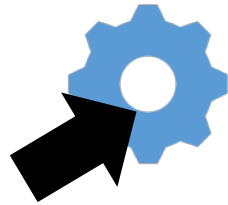
Support for all platform

Git Foundation

The 3 State of Git



Modified



Staged

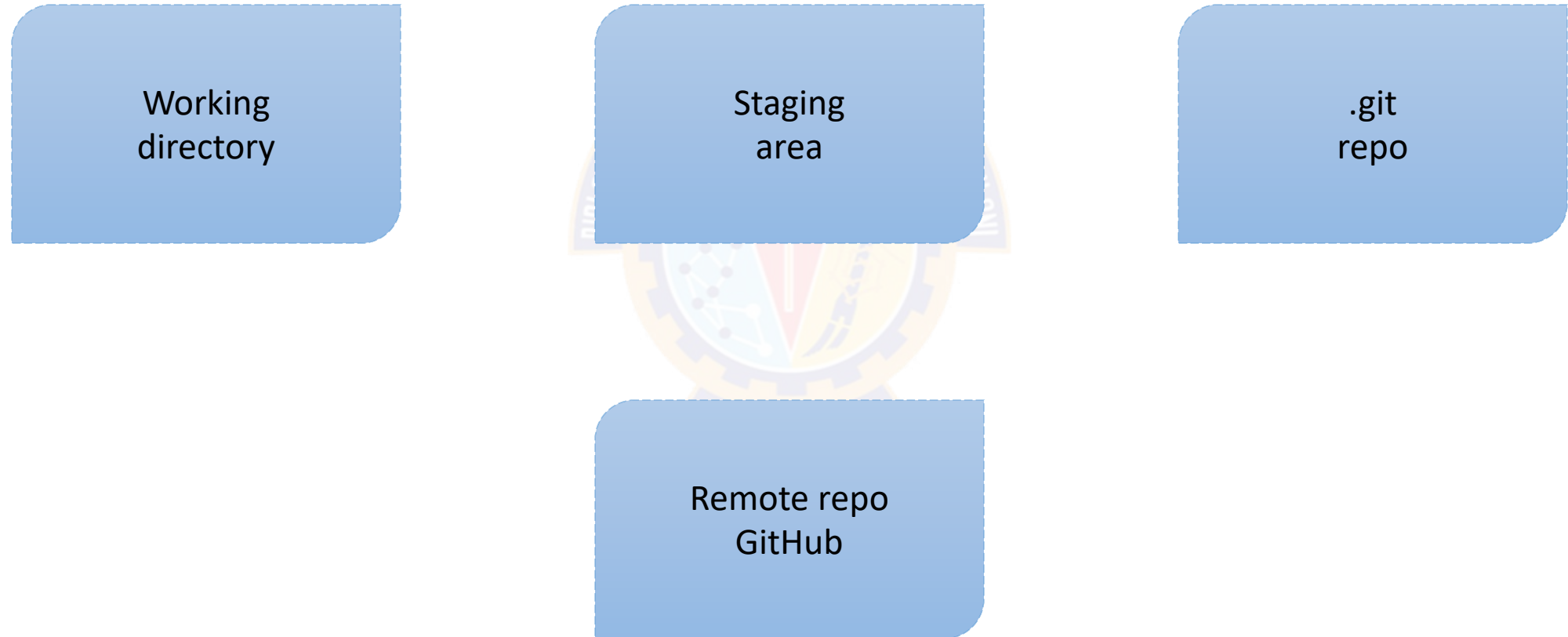


Committed



Git Foundation

The 3 areas of Git



Git Foundation

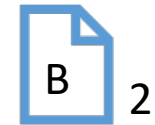
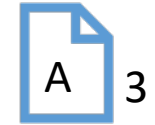
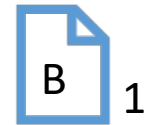
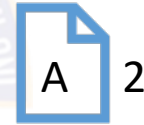
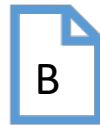
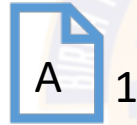
Concepts of Snapshots in Git & GitHub

First
version

Version 2

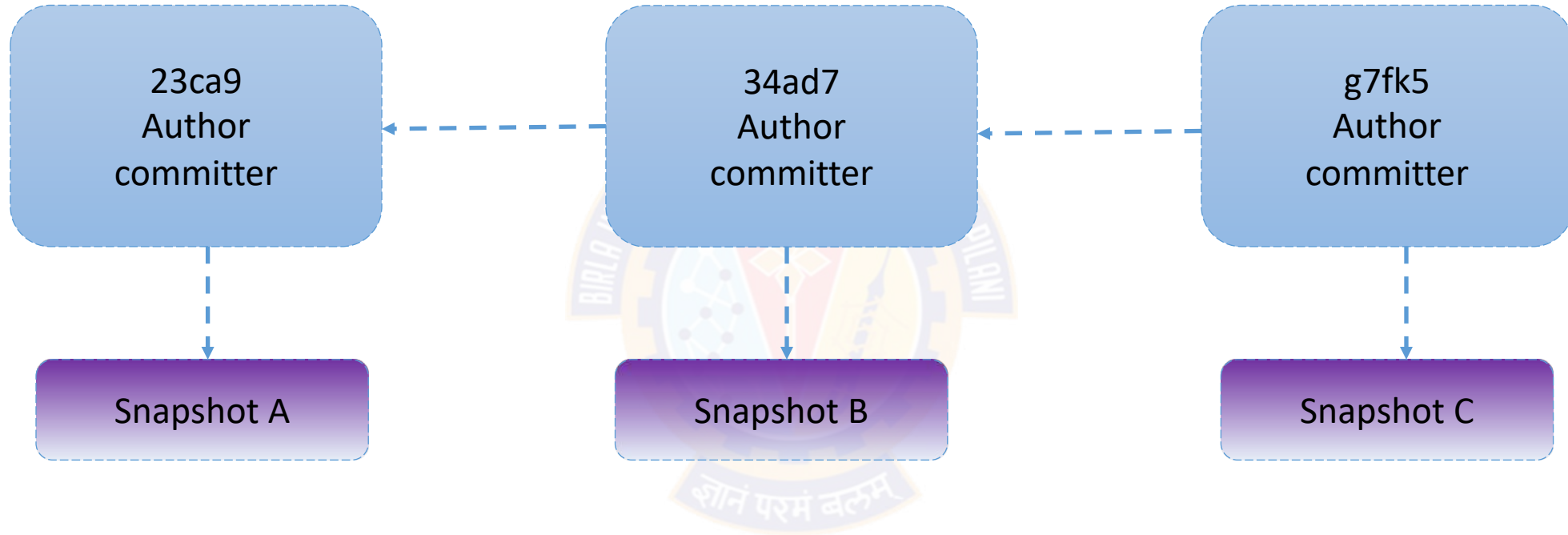
Version 3

Version 4



Git Foundation

Commits in Git



Git

Basic Commands

\$ git	\$ git push
\$ git config	\$ git fetch
\$ git init	\$ git merge
\$ git clone	\$ git pull
\$ git status	\$ git log
\$ git add	\$ git reset
\$ git commit	\$ git revert
\$ git branch	
\$ git checkout	

GitHub

GitHub's main Features



Code management



Pull requests



Issues



CI/CD



Global search

GitHub

Connecting with local machine

HTTPS

Requires user name & password

SSH

Easier to work with



GitHub

Working with repository



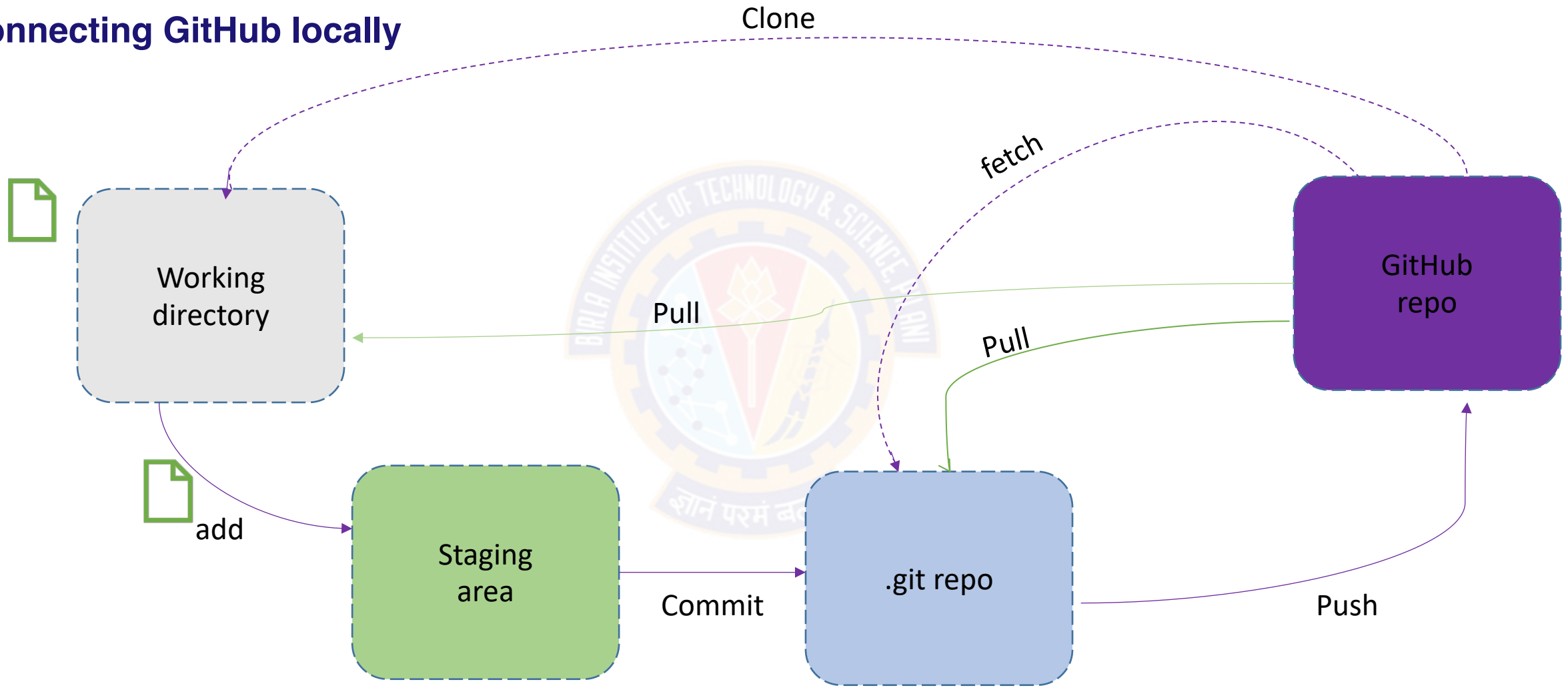
Repositories are building block of GitHub

“Folder” for your project

Public or Private

GitHub

Connecting GitHub locally



GitHub

Working with special files



README is special file known by GitHub

Rendered automatically on landing page

Typically written in markdown(.md)

Other files:

LICENSE

CHANGELOG

CODE_OF_CONDUCT

CONTRIBUTORS

SUPPORT

CODEOWNERS

Repository Feature

TOPICS

ISSUES

Insight

PROJECTS

Pull Requests

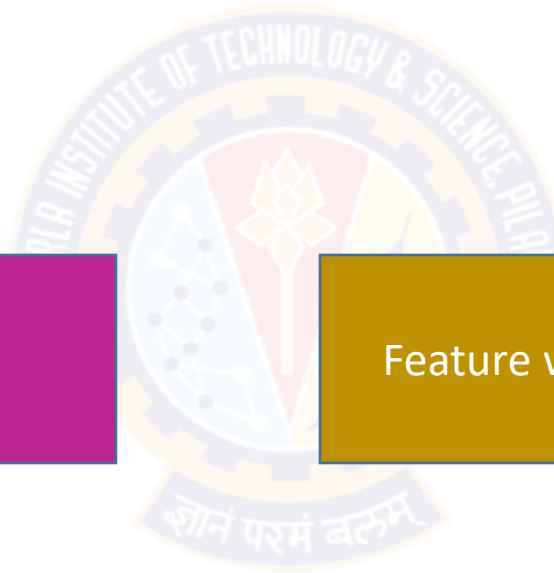
Settings

Git

Workflow

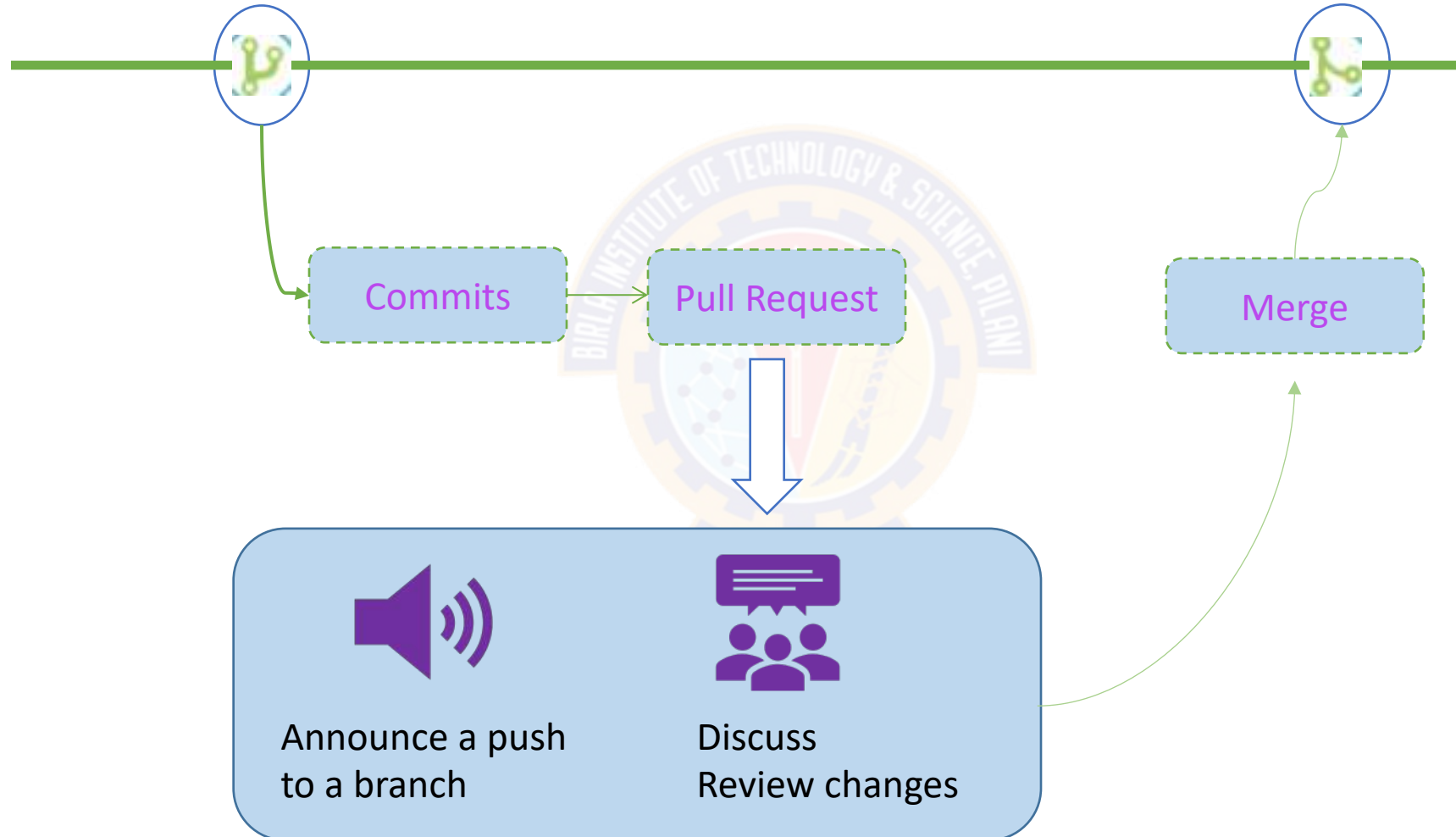
Centralized

Feature workflow



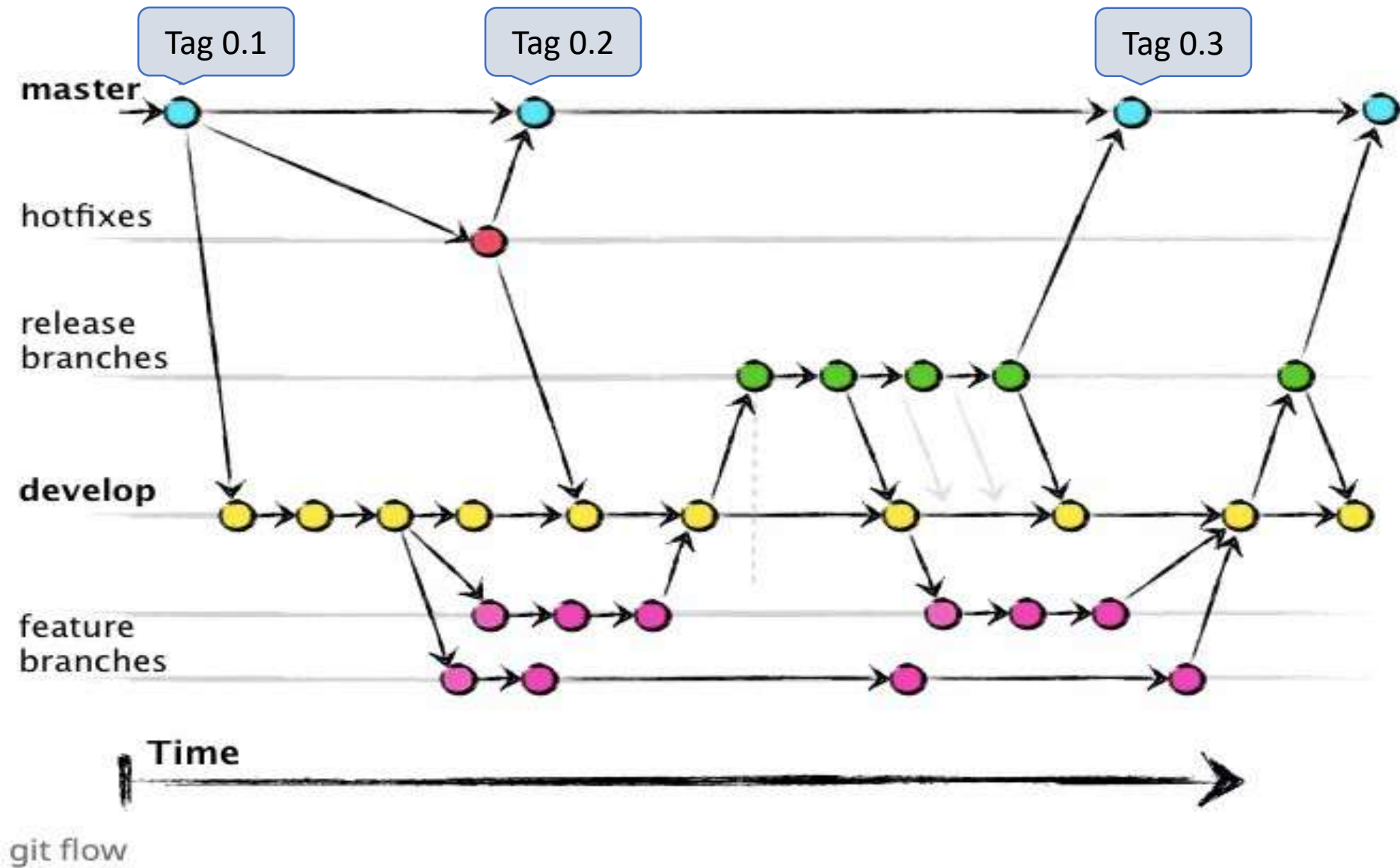
GitHub

Branching



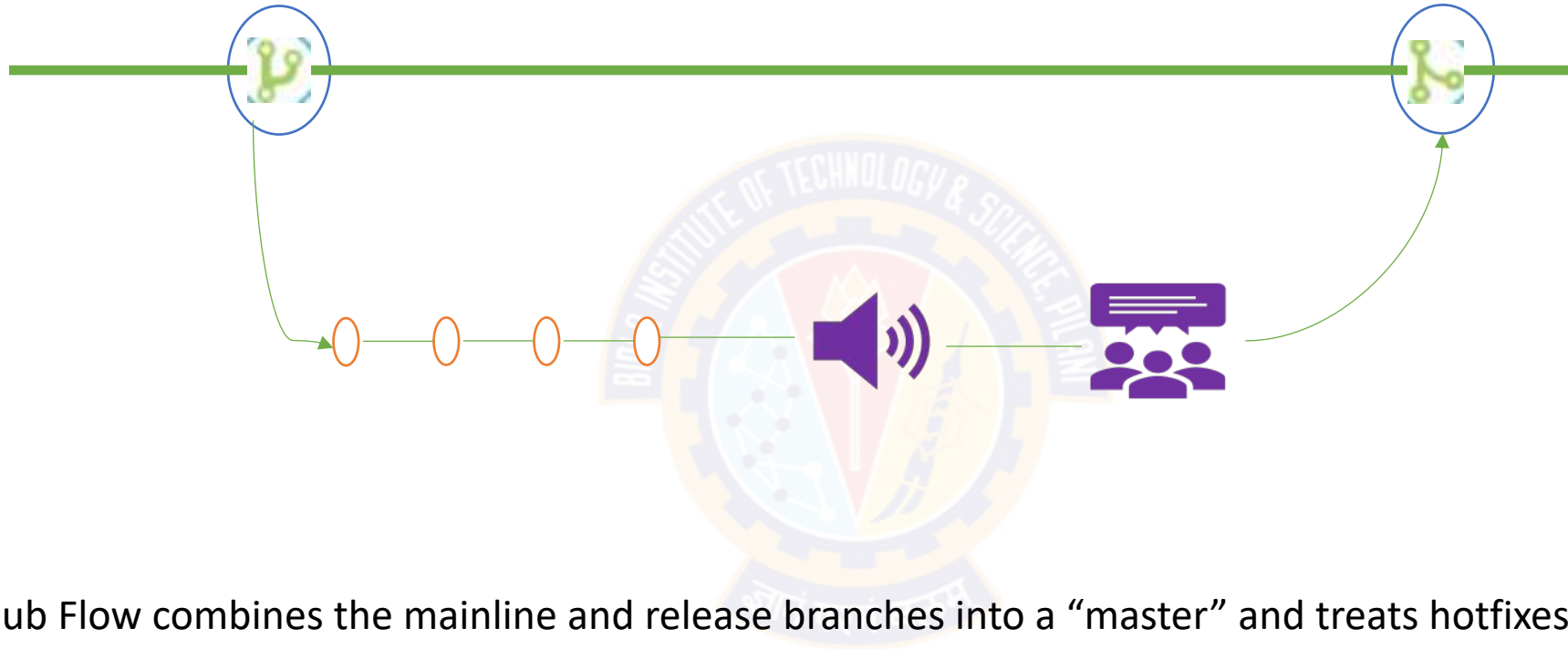
GitHub

Git Flow



GitHub

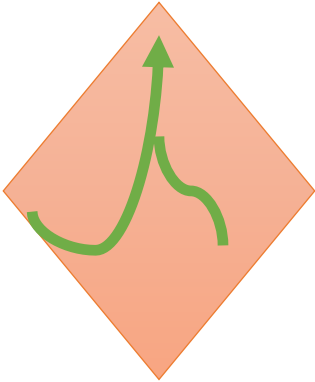
GitHub Flow



GitHub Flow combines the mainline and release branches into a “master” and treats hotfixes just like feature branches.

Git & GitHub

Merging with conflicts



Typical conflicts:

- Editing on the same line

- Editing on already deleted file

Merge conflicts needs to be solved before merge happen (Manual intervention)

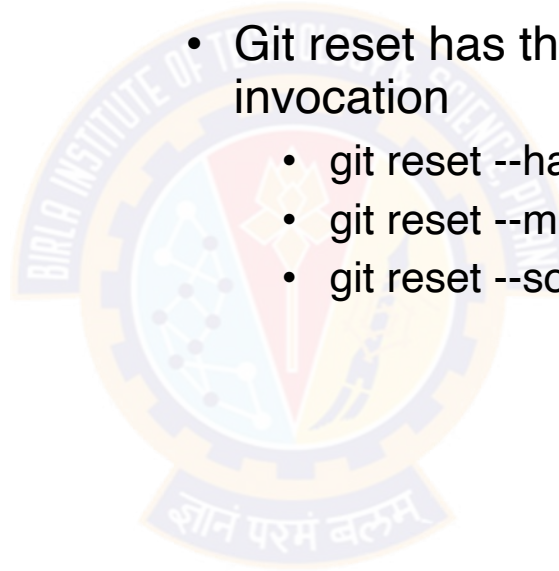
Git Command

Revert

- Git revert will create a new commit
- Git revert undoes a single commit
- It is a safe way if undo

Reset

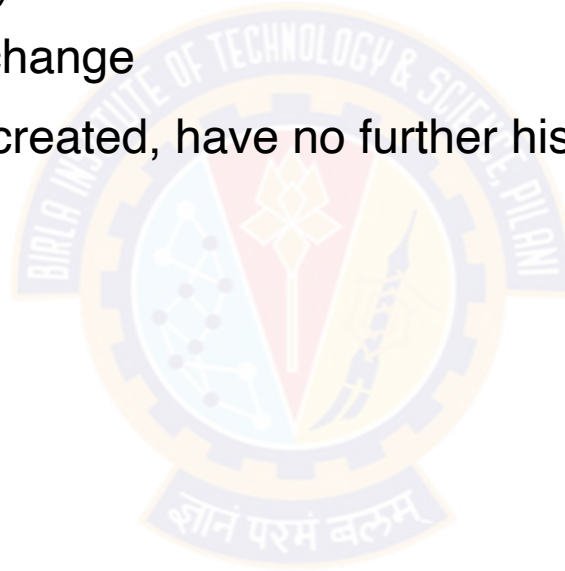
- Git reset command is a complex
- Dangerous
- Git reset has three primary form of invocation
 - `git reset --hard HEAD`
 - `git reset --mixed HEAD`
 - `git reset --soft HEAD`



Git command

Git Tag

- Tags are ref's that point to specific points in Git history
- 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
- Common Tag operations:
 - Create tag
 - List tags
 - Delete tag
 - Sharing tag



Clean Code

- 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
- 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



Q&A



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

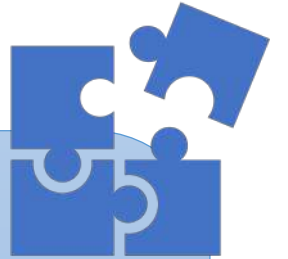
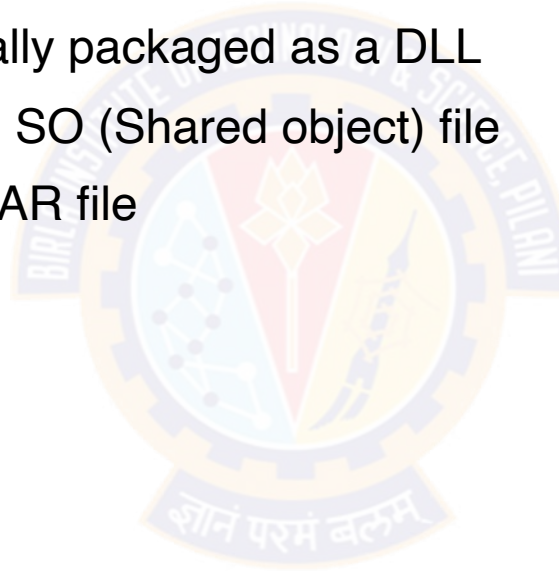
- Manage Dependencies
- Automate the process of assembling software components with build tools
- Use of Build Tools
 - Maven
 - Gradle



Component

Why Component based design

- large-scale code structure within an application
- 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

- encouraging reuse and good architectural (loose coupling)
- efficient ways for large teams of developers to collaborate

Component

Challenges of component based design

- Components form a series of dependencies, which in turn depend on external libraries
- Each component may have several release branches



To overcome this we need to follow the best practices

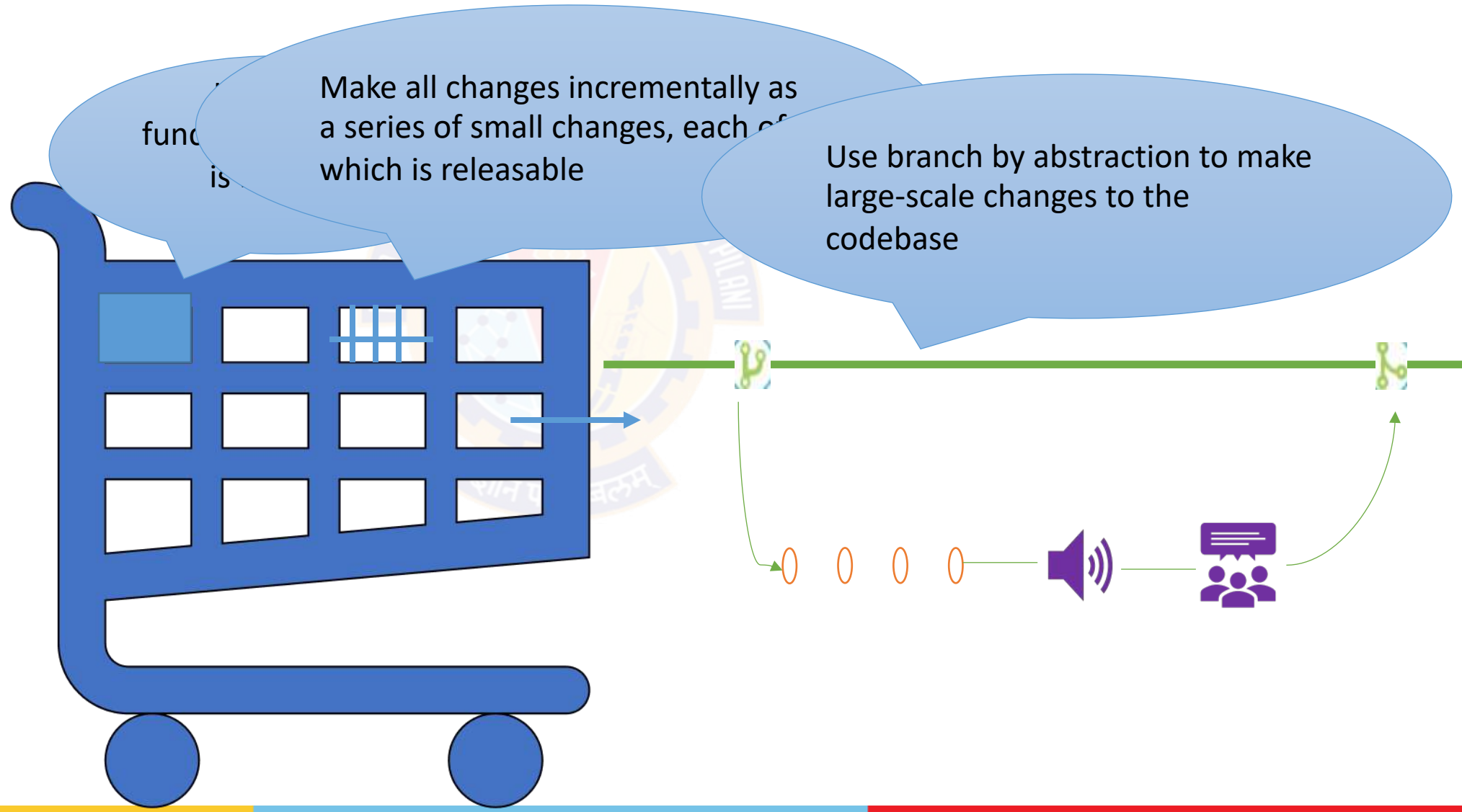


Delay in release

- finding good versions of each of these components
- Then assembled them into a system which even compiles is an extremely difficult process

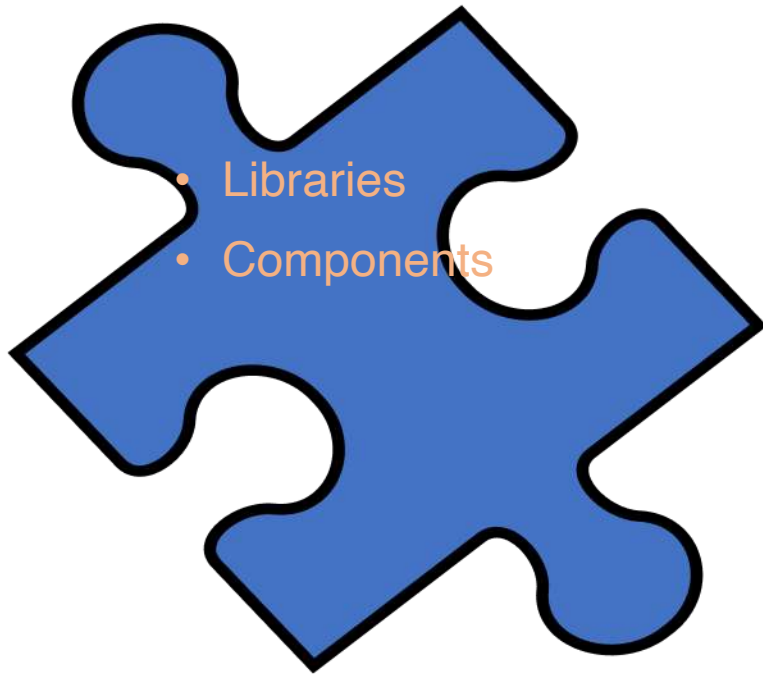
Best Practices for component based design

Keeping Your Application Releasable



Best Practices for component based design

Manage your applications dependencies



Dependencies

What is dependencies

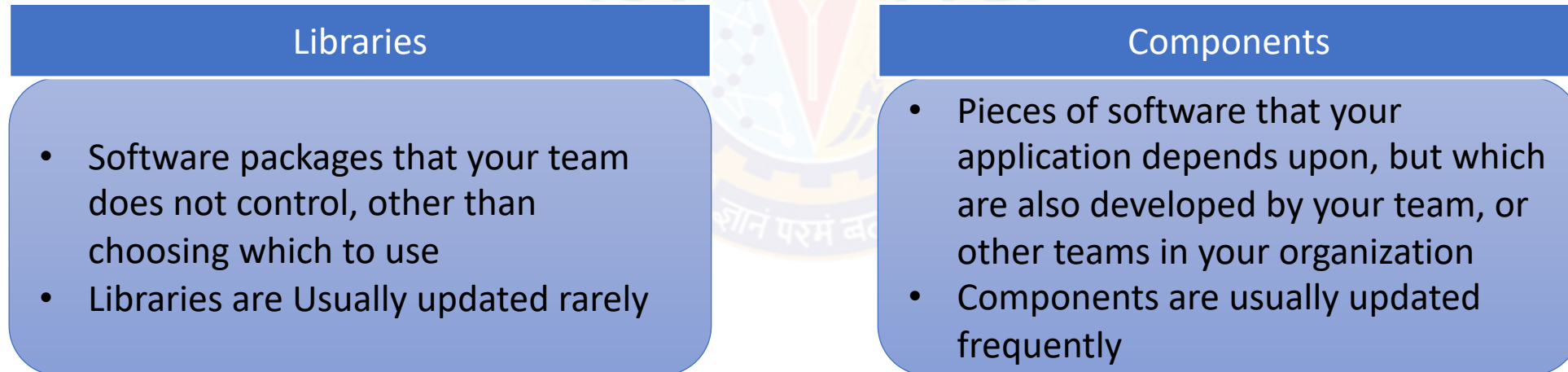
- Dependency occurs when one piece of software depends upon another in order to build or run
- Software applications → host operating environment
- Like:
 - Java applications. → JVM
 - .NET applications → CLR,
 - Rails applications → Ruby and the Rails framework,
 - C applications → C standard library etc.



Dependencies

Dependencies can be

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

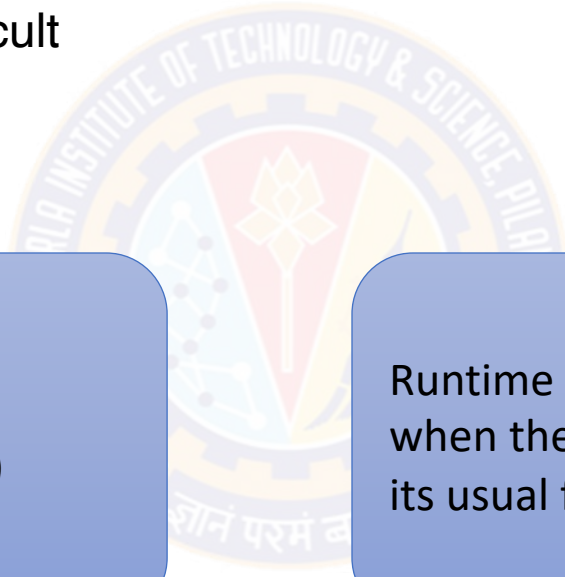


This distinction is important because when designing a build process, there are more things to consider when dealing with components than libraries

Dependencies

Build time dependencies and Run time dependencies

- 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



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

Most 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



Managing Libraries

Implementing Version Control

- Simplest solution, and will work fine for small projects
- Create lib directory
- Use 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

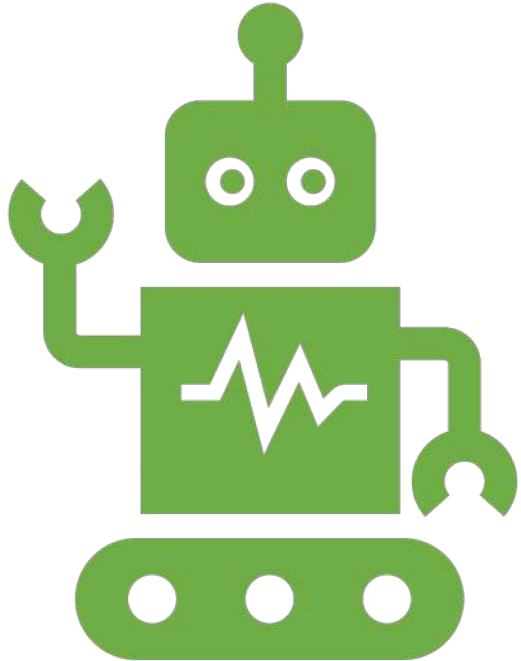
Problems:

- 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

- Declare libraries and use a tool like Maven or Ivy or gradle
- To download libraries from Internet repositories or (preferably) your organization's own artifact repository



Managing Component

Components to be separated from Codebase

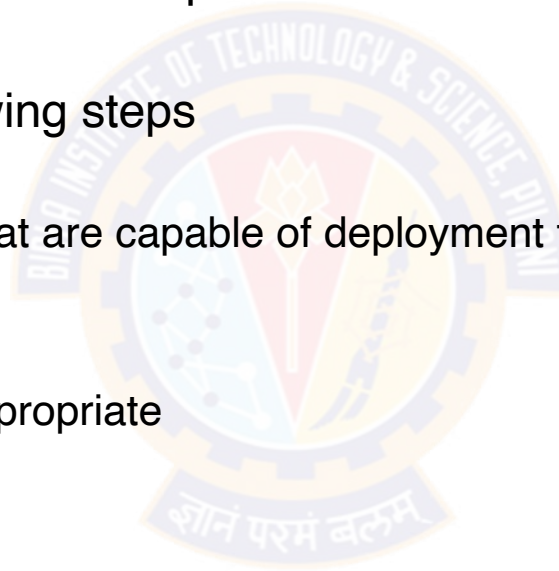
- Part of your codebase needs to be deployed independently (for example, a server or a rich client)
- It takes too long to compile and link the code



Managing Component

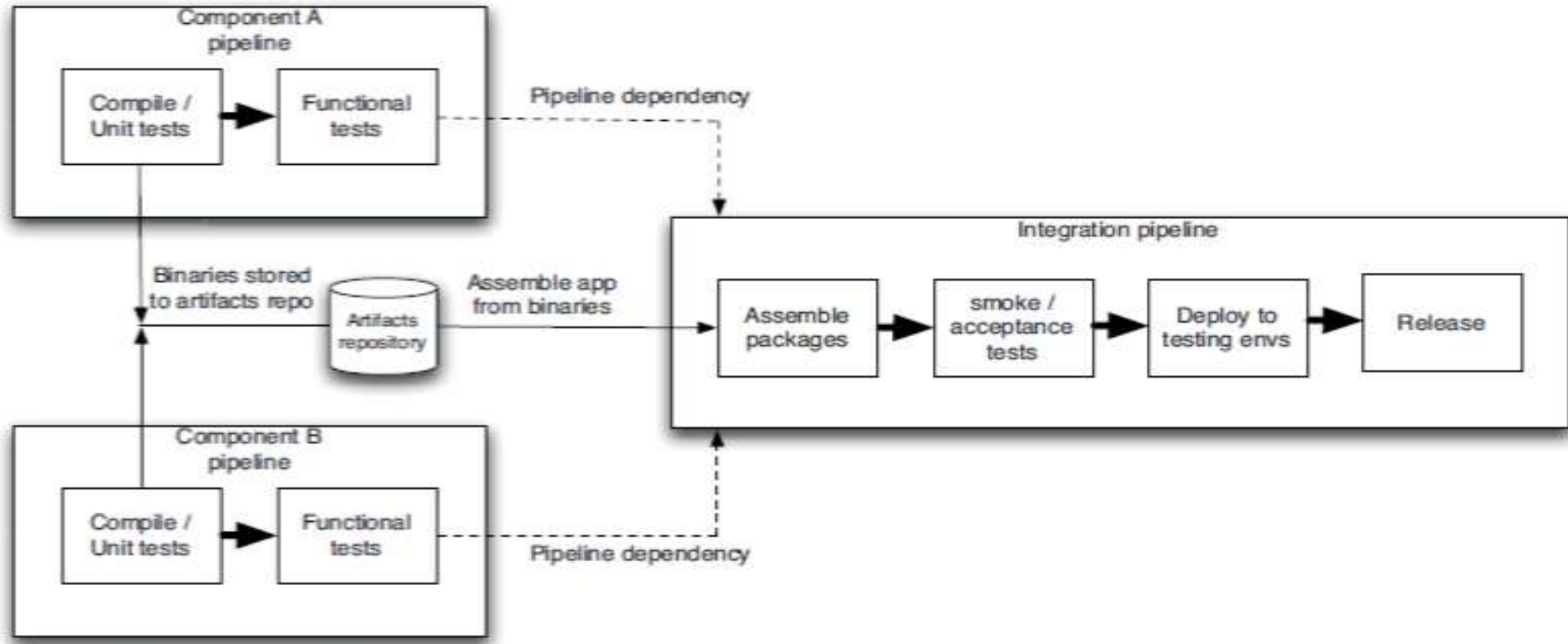
Pipelining Components

- Split your system into several different pipelines
- 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



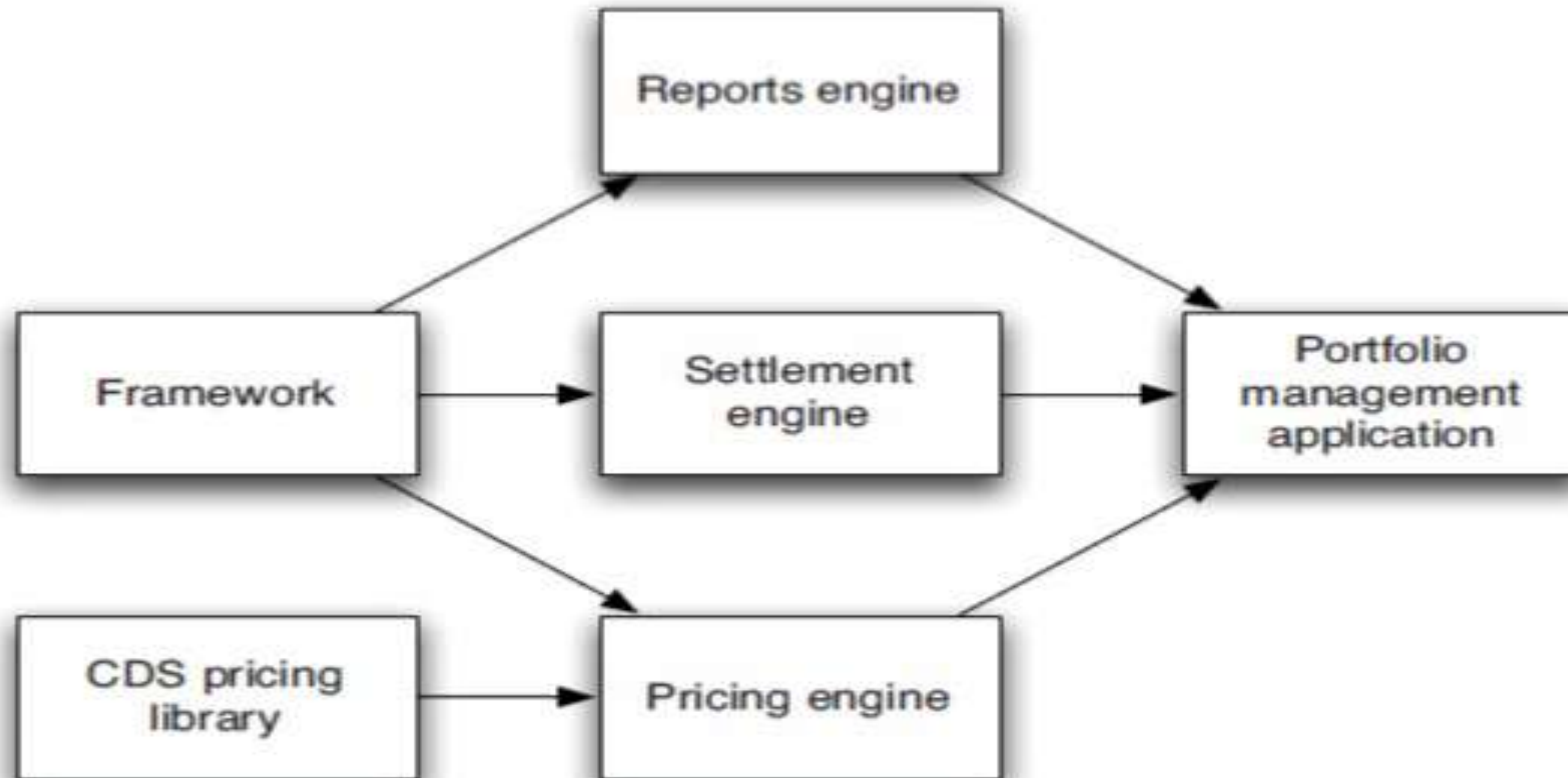
Managing Component

Integration Pipeline



Managing Component

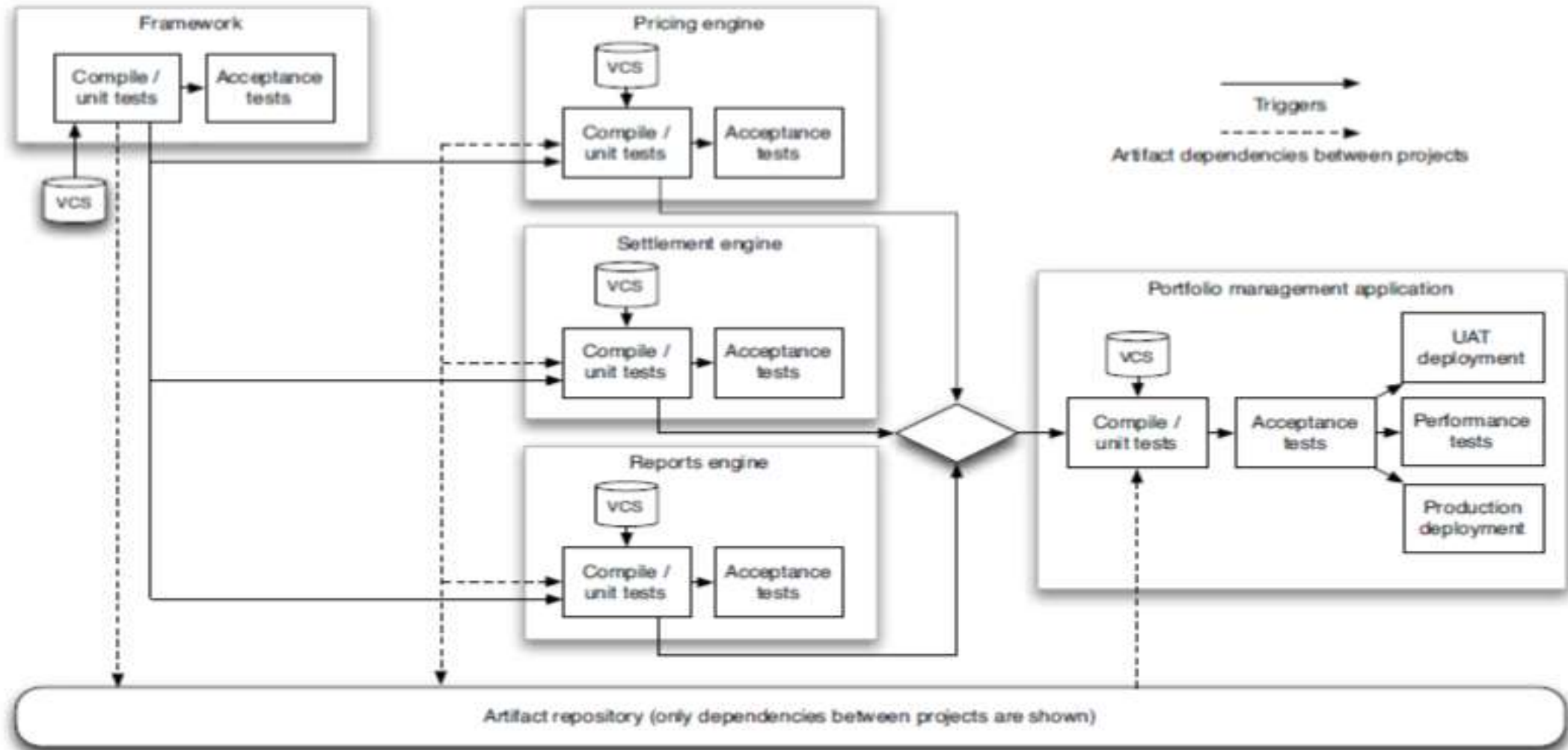
Managing Dependency Graphs



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

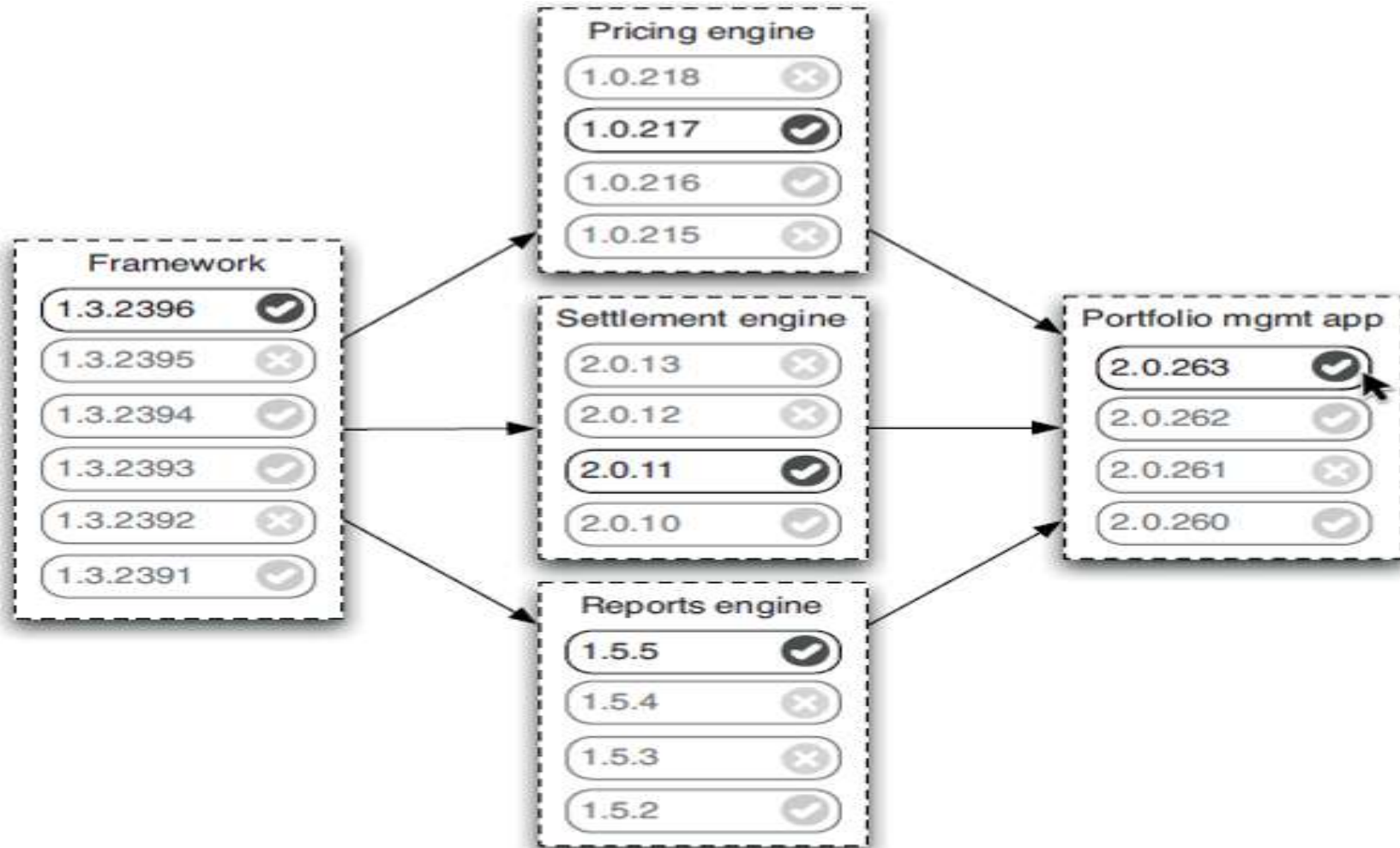
Managing Component

Pipelining Dependency Graphs



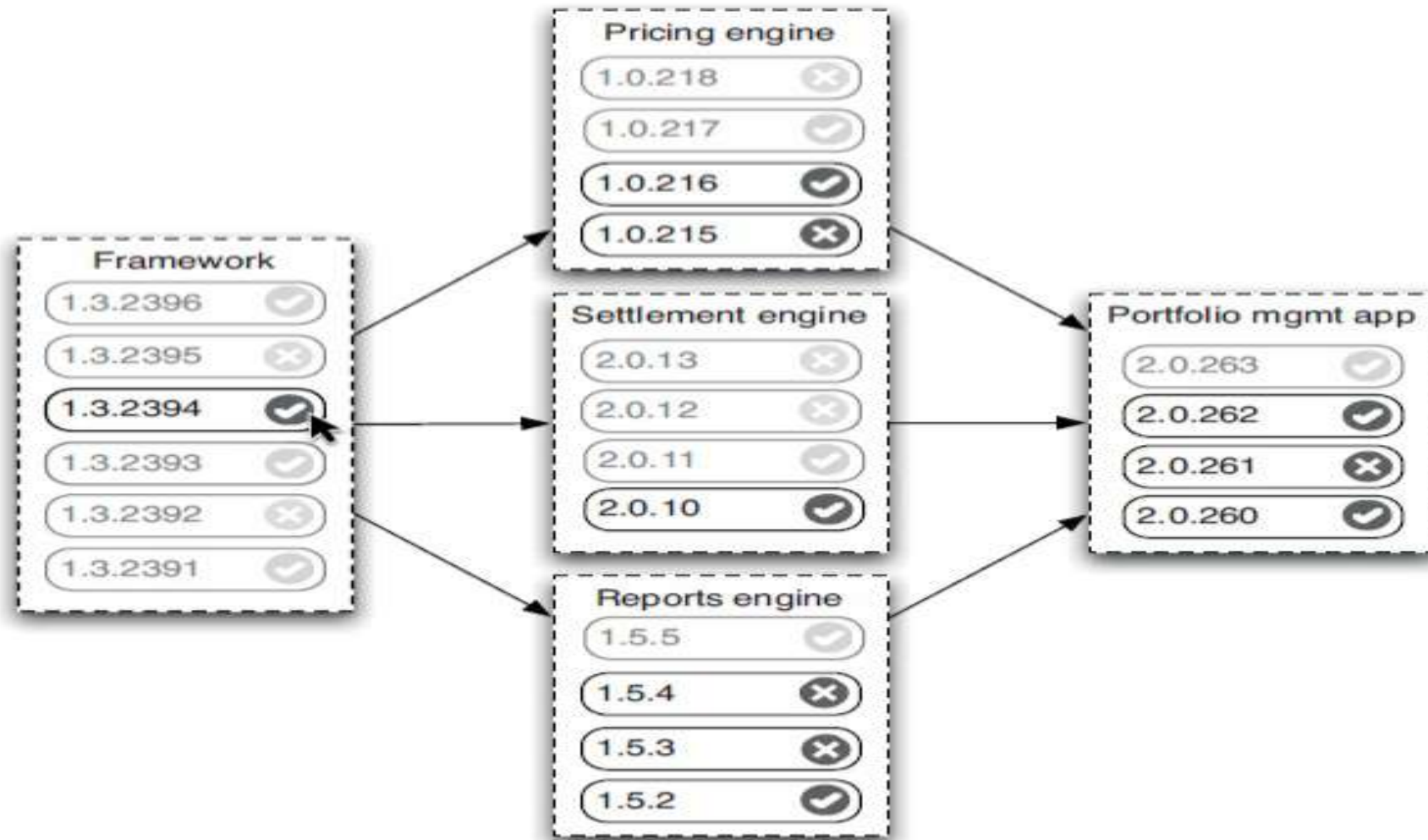
Managing Component

Visualizing upstream dependencies



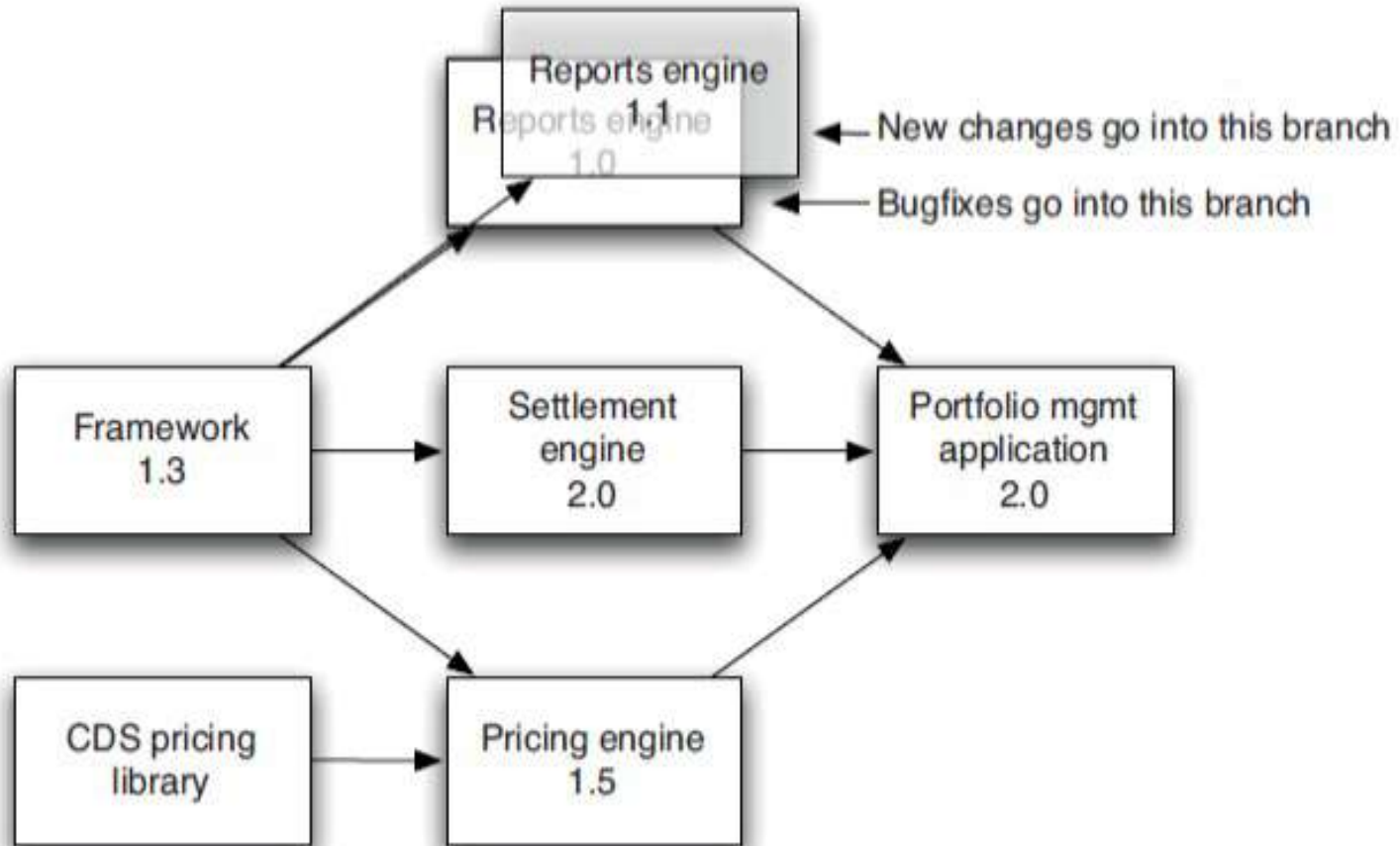
Managing Component

Visualizing downstream dependencies



Managing Component

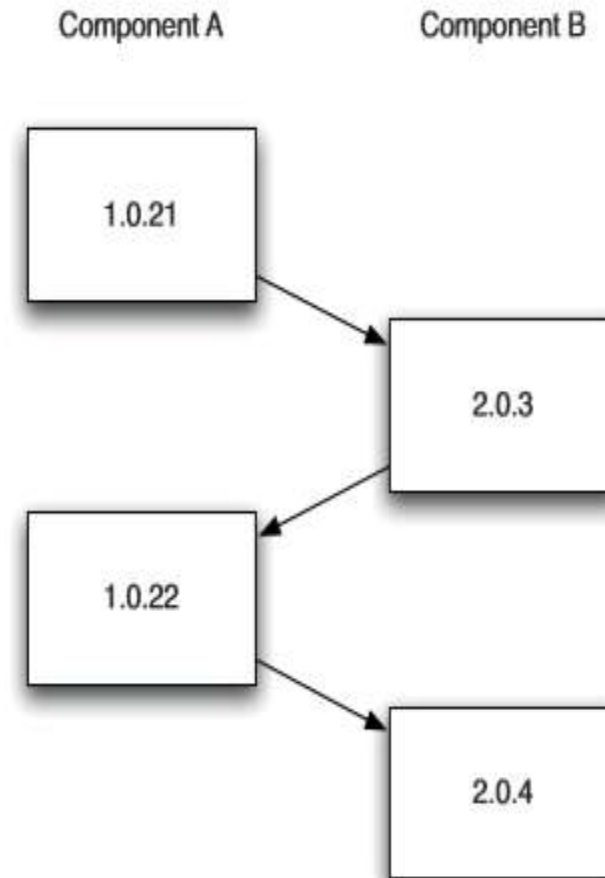
Branching Components



Managing Component

Circular Dependencies

- This occurs when the dependency graph contains cycles
- No build system supports such a configuration out of the box, so you have to hack your toolchain to support it



Circular dependency build ladder

Build automation

Build tools

- Steps of a build process

Compile the source code

Running & evaluating the unit test

Processing existing resource
files (configurations)

Generating artifacts (WAR, JAR,..)

Build automation

Build tools

- Additional steps that are often executed in the a build process:

Administering dependencies

Analyzing code quality (static
code analysis)

Running additional tests

Archiving generated artifacts and
packages in a central repository

Automating Build Process

Build Tools

- Maven
- Gradle

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

Maven

What is Maven



What is Maven

Build Tool

- One artifact (Component, JAR, even a ZIP)
- Manage Dependencies

Management Tool

- Handles Versioning / Releases
- Describe Project
- Produce Javadocs / Site information



Who owns Maven

Apache Software Foundation

Maven official site is built with Maven
Open Source

Maven

Maven Structure

- `src/main/java` : by default maven looks for a `src/main/java` directory underneath of project)
- `target` folder : compiles all our source code to target directory
- `pom.xml` : maven compile source code in way provided by `pom.xml`
- Different language : `src/main/groovy` or `src/main/resources`
- Unit testing: `src/test/java`
- Target directory : Everything get compile
Even your test gets run and validated
Packaged Contents (like JAR, WAR or ZIP depending on what we have provided in `pom.xml`)

Maven

Pom.xml

- Maven uniquely identifies a project using
- groupId :
 - The groupId is often the same as our package
 - Ex: com.bits or com.maven.training
 - groupId is like, business name or application name as you would reference it as a web address
- artifactId :
 - Same as name of your application
 - Ex. HelloWorld, OnDemandService etc.
- version :
 - Version of project
 - Format {Major}.{Minor}.{Maintenance} if it is RELEASE
 - '-SNAPSHOT' to identify in development
 - Ex: 1.0-SNAPSHOT

Maven

Pom.xml

- packaging :
 - Packaging is how we want to distribute our application
 - Ex. JAR file, a WAR file, RAR file or an EAR file
 - The default packing is JAR
- dependencies :
 - Just add it to our dependency section of POM file
 - Need to know our three things for dependency i.e. groupId, artifactId, and version
- plugins
 - Just add it to plugins section of POM file

Maven

Pom.xml example

```
pom.xml > project > build > plugins > plugin > version
1  <project>
2    <groupId>com.bits</groupId>
3    <artifactId>HelloWorld</artifactId>
4    <version>1.0-SNAPSHOT</version>
5    <modelVersion>4.0.0</modelVersion>
6    <packaging>jar</packaging>
7
8    <dependencies>
9      <dependency>
10        <groupId>org.apache.commons</groupId>
11        <artifactId>commons-lang3</artifactId>
12        <version>3.8.1</version>
13      </dependency>
14    </dependencies>
15
16    <build>
17      <plugins>
18        <plugin>
19          <groupId>org.apache.maven.plugins</groupId>
20          <artifactId>maven-compiler-plugin</artifactId>
21          <version>3.7.0</version>
22          <configuration>
23            <target>10</target>
24            <source>10</source>
25            <release>10</release>
26          </configuration>
27        </plugin>
28      </plugins>
29    </build>
30  </project>
```

Maven

Maven Goals



clean



compile



package



install




deploy

- First run compile goal
 - Then runs unit test
 - Generate artifact/package as per provided in pom.xml
 - Ex. JAR, WAR
- First run package goal
 - Then install the package in local repository
 - Default it is .m2 folder
- Runs install goal first
 - then deploy it to a corporate or remote repository
 - Like file sharing

Maven

Project Inheritance

- Pom files can inherit configuration
 - groupId, version
 - Project Config
 - Dependencies
 - Plugin configuration
 - Etc.



```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <parent>
    <artifactId>maven-training-parent</artifactId>
    <groupId>org.lds.training</groupId>
    <version>1.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>maven-training</artifactId>
  <packaging>jar</packaging>
</project>
```

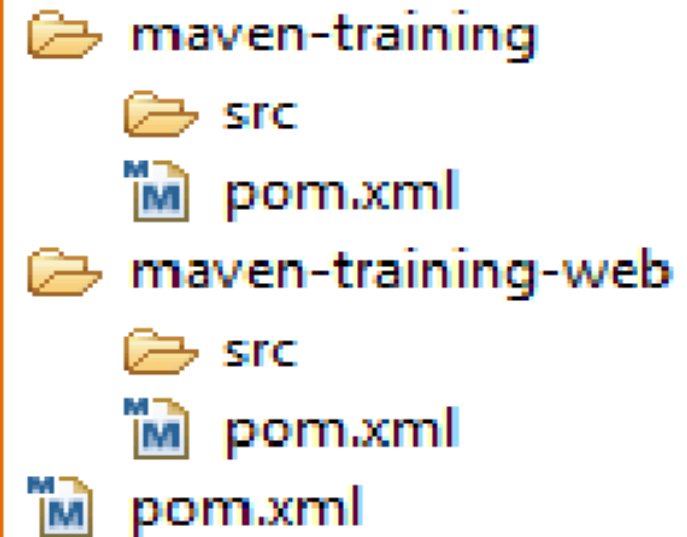
Maven

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



```
<project>
  ...
  <packaging>pom</packaging>
  <modules>
    <module>maven-training</module>
    <module>maven-training-web</module>
  </modules>
</project>
```



Gradle

What is Gradle



What is Gradle

Open source build automation tool
Gradle build scripts are written in Domain Specific Language [DSL]



Why Gradle

High performance

- runs only required task; which have been changed
- Build cache helps to reuse tasks outputs from previous run
- ability to have shared build cache within different machine

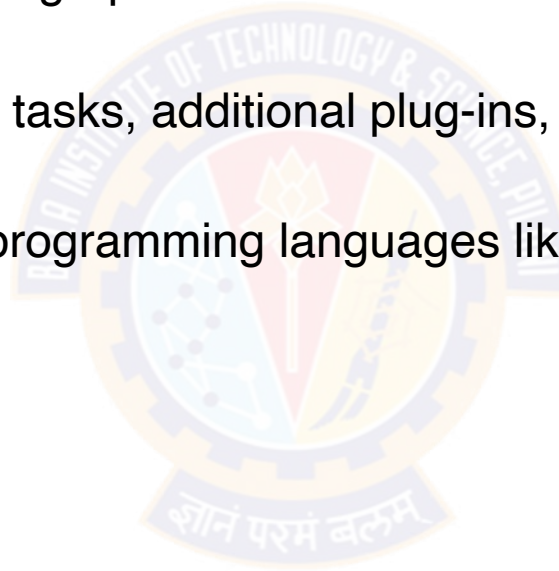
JVM foundation

- Java Development Kit is prerequisite
- It is not limited to Java

Gradle

Core concepts

- Tasks and the dependencies between them
- Gradle calculates a directed, acyclic graph to determine which tasks have to be executed in which order
- Graph can change through custom tasks, additional plug-ins, or the modification of existing dependencies
- Plug-ins allows to work with other programming languages like Groovy, kotlin C++ etc.,

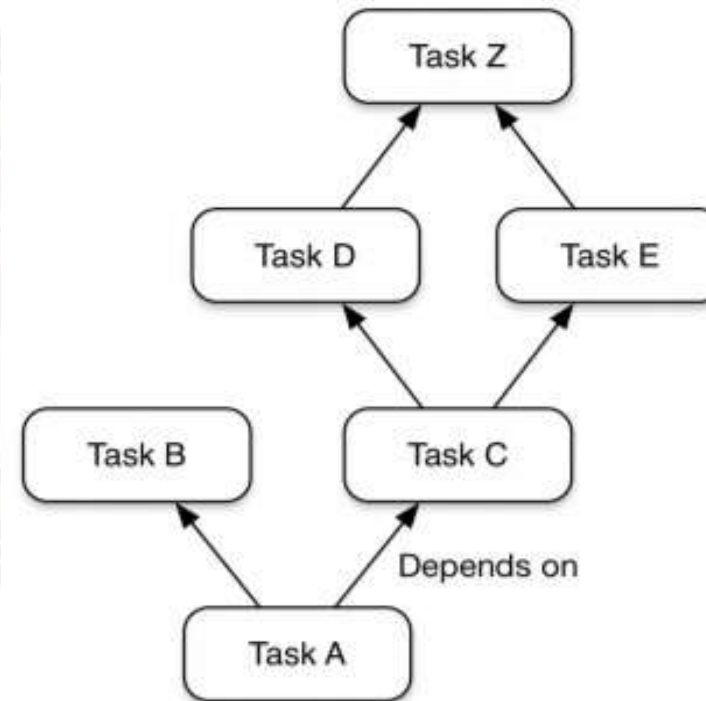


Gradle

Core model

- The core model is based on tasks
 - Directed Acyclic Graphs (DAGs) of tasks
- Example of a Graph
- Tasks Consists of:
 - Action : To perform something
 - Input : values to action
 - Output : generated by

Generic task graph



Build Tool

Summary



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

References

CS 4,5,6 & 7

- Chapter 5 from Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis and Katherine Daniels
- Transformation to Enterprise DevOps culture: <https://devops.com/six-step-approach-enterprise-devops-transformation/>
- **Cloud as a Catalyst:**
- TextBook 2: Continuous Delivery : Chapter 11 Managing Infrastructure and Environments
- TextBook 1: DevOps A S/W Architect Pres: Chapter 2 the Cloud as a Platform
- Chapter 3, from DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu ,
- Chapter 5, Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis
- <https://git-scm.com/>



Q&A



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Automating Build Process

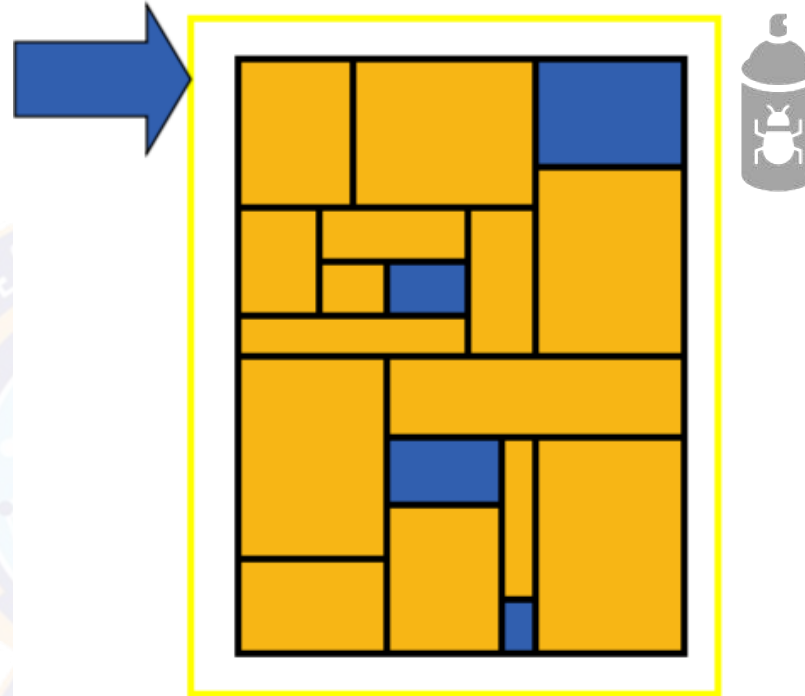
- Unit testing
- Automates Test Suite - Selenium
- Continuous Code Inspection
- Code Inspection Tools
 - Sonarqube



Unit Testing

Traditional Testing

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



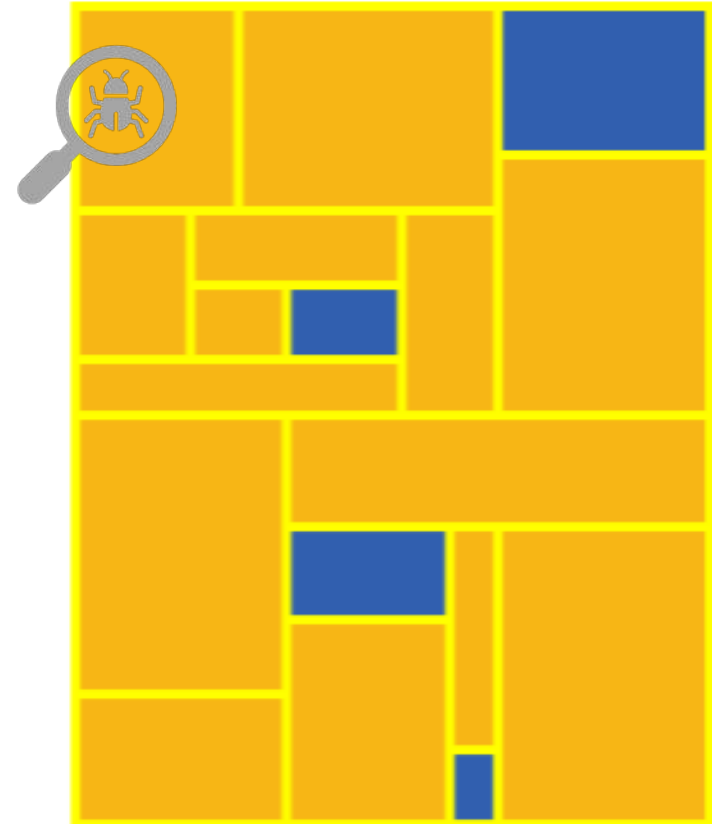
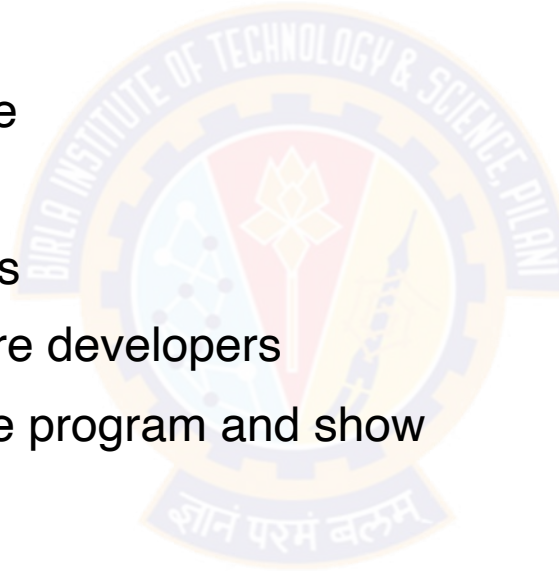
Traditional Testing Strategies

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

Unit Testing

What is Unit Testing

- Is a level of the software testing process where individual units/components of a software/system are tested
- Each part tested individually
- All components tested at least once
- Errors picked up earlier
- Scope is smaller, easier to fix errors
- Typically written and run by software developers
- Its goal is to isolate each part of the program and show that the individual parts are correct



Unit Testing

Why Unit Testing

Concerned with

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

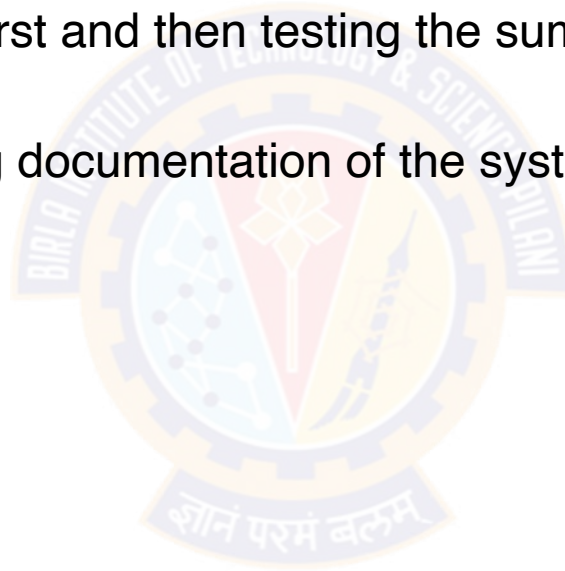


- Faster Debugging
- Faster Development
- Better Design
- Excellent Regression Tool
- Reduce Future Cost

Unit Testing

Benefits

- Unit testing allows the programmer to refactor code earlier and make sure the module works correctly
- By testing the parts of a program first and then testing the sum of its parts, i.e. 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
- Unit tests should be fully automated and non-interactive
- Make unit tests simple to run
- Measure the tests
- Fix failing tests immediately
- Keep testing at unit level
- Keep tests independent
- Name tests properly
- Prioritize testing



Test Automation

Selenium

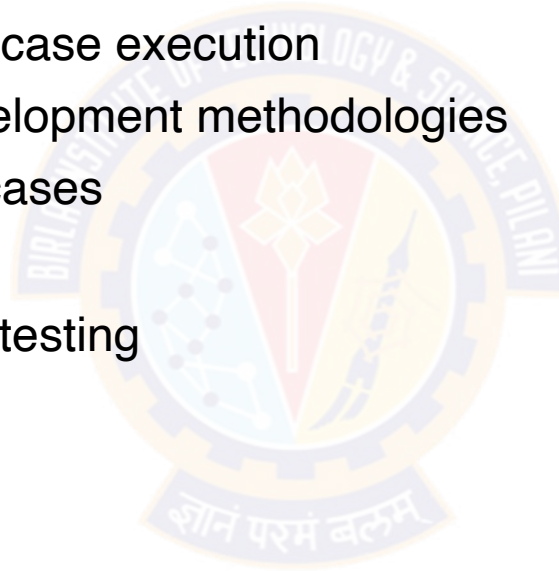
- Perform an sort of interaction
- Selenium helps to automate web browser interaction
- Scripts perform the interactions



Selenium

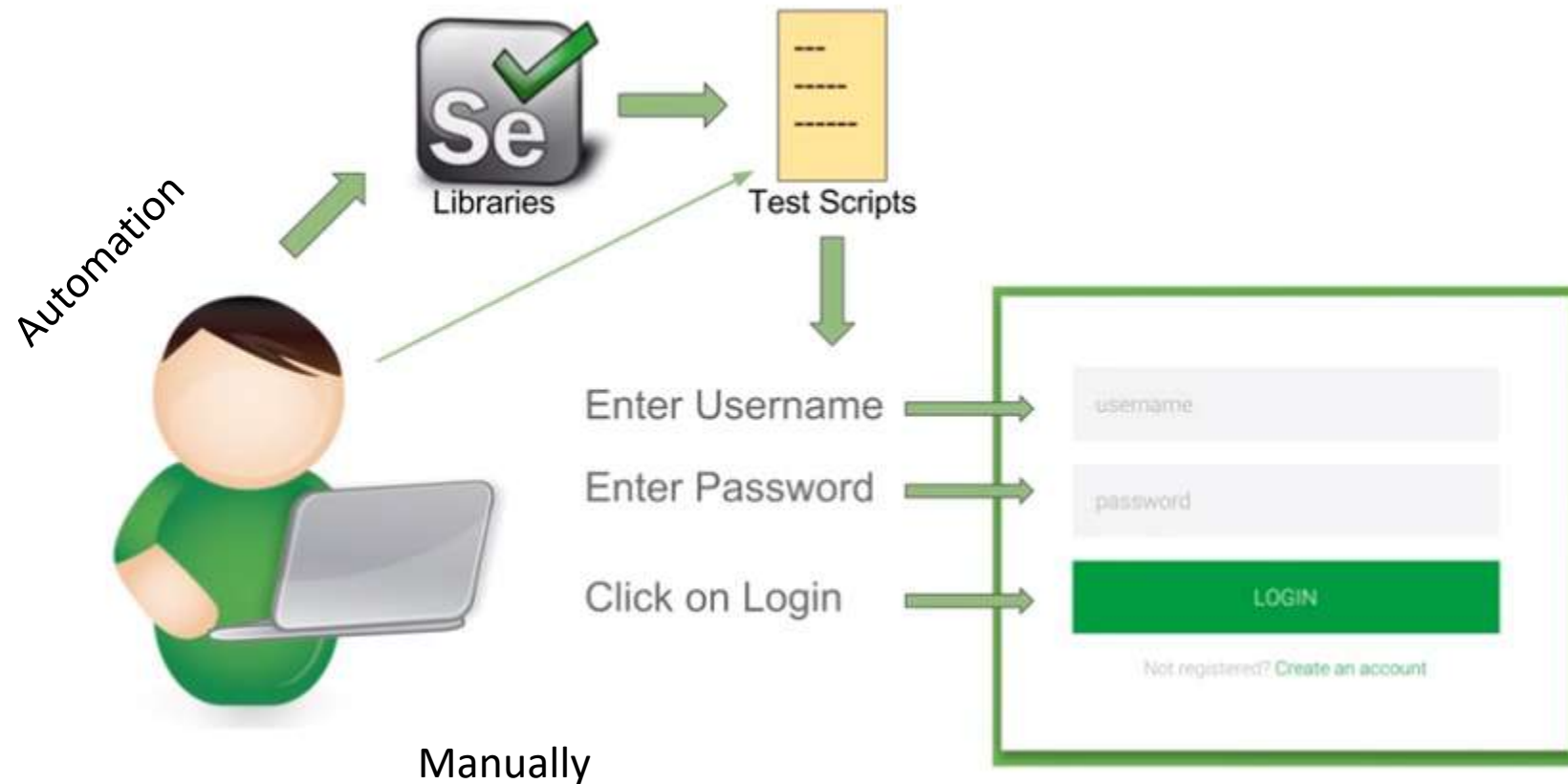
Benefits

- 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
- Reduced Business Expenses
- Reusability of Automated Tests
- Faster Time-to-Market



Selenium

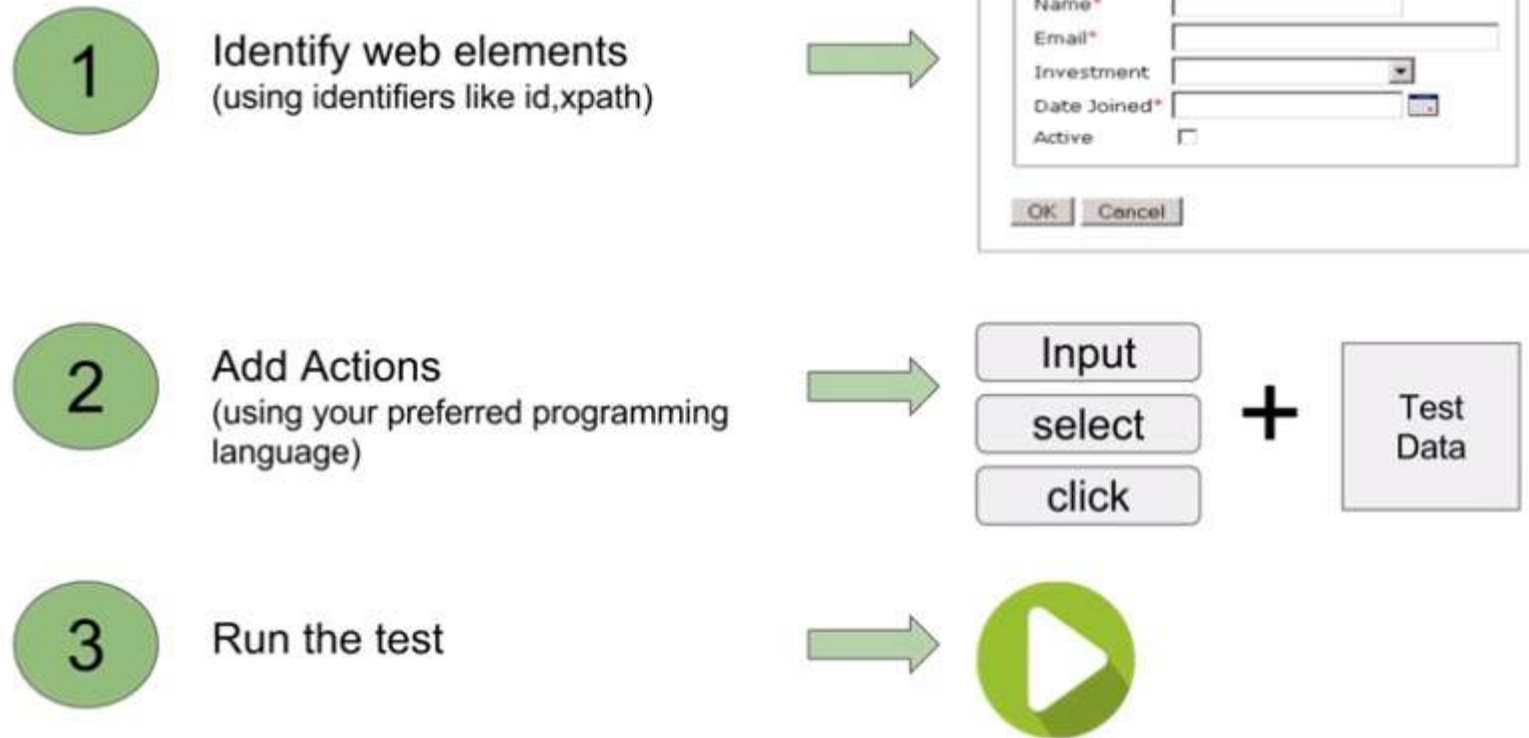
Lets say you want to test one login page



Selenium

Example

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



Selenium

Components

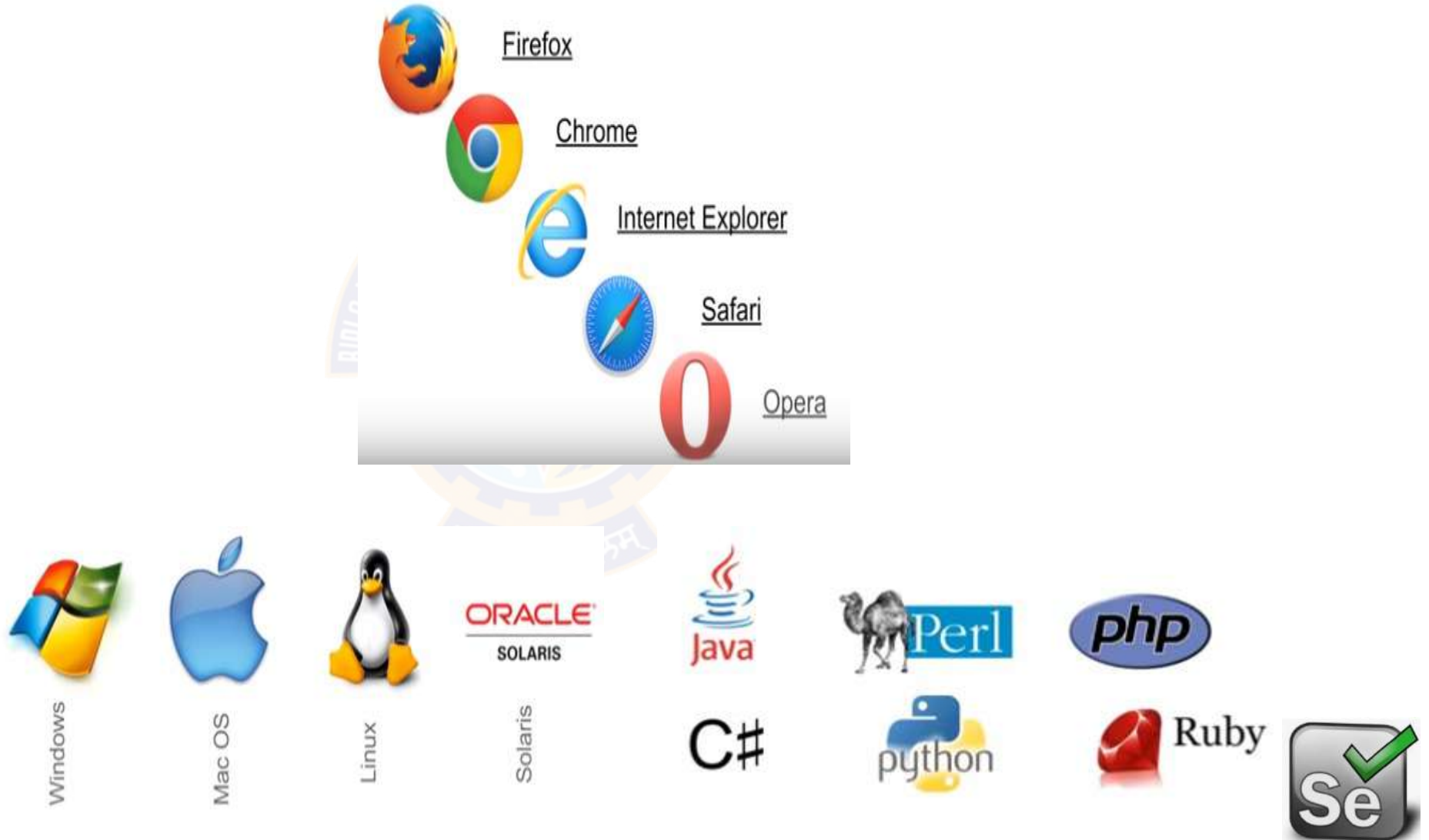
- Selenium IDE
 - A Record and playback plugin for Firefox add-on
 - Prototype testing
- Selenium RC (Remote Control)
 - Also known as selenium 1
 - Used to execute scripts (written in any language) using Javascript
 - Now Selenium 1 is deprecated and is not actively supported
- WebDriver
 - Most actively 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

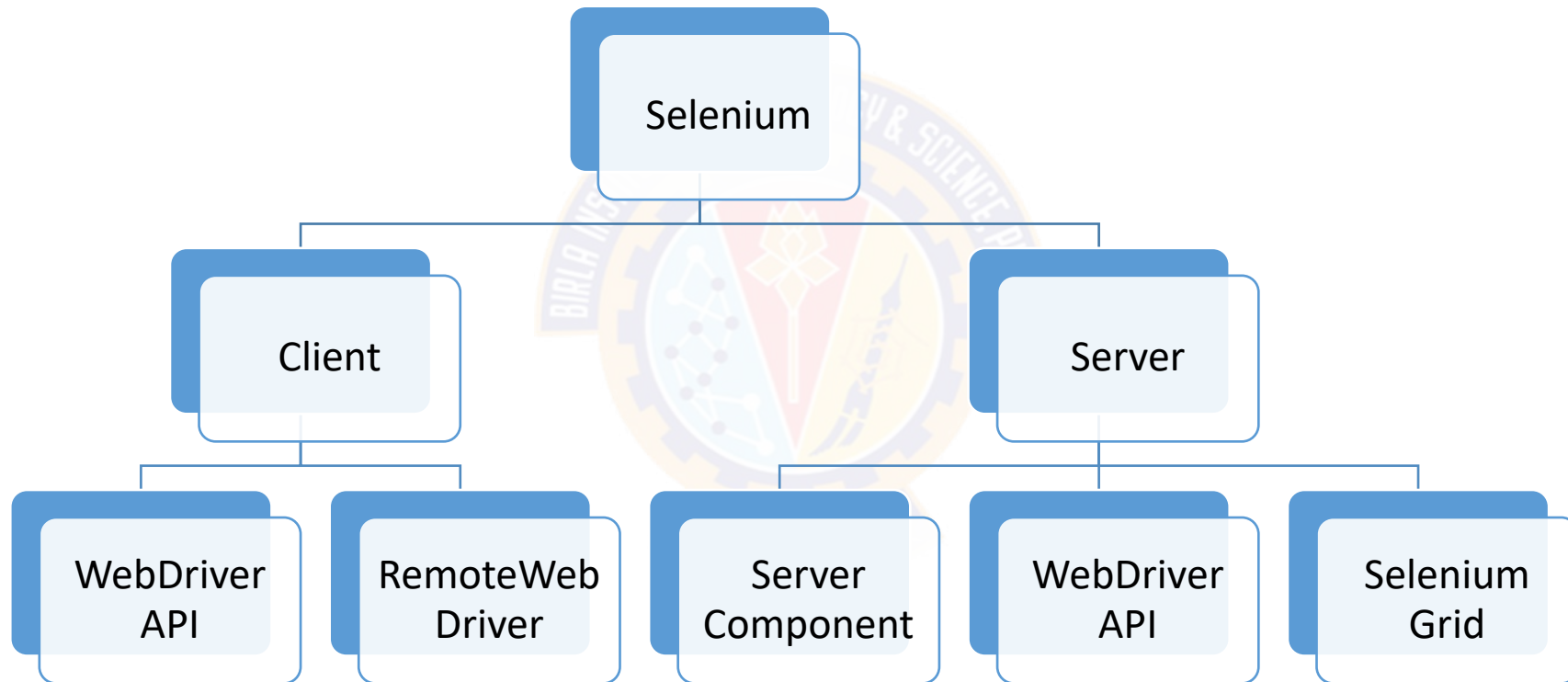
Supports

- Browsers
- OS
- Language



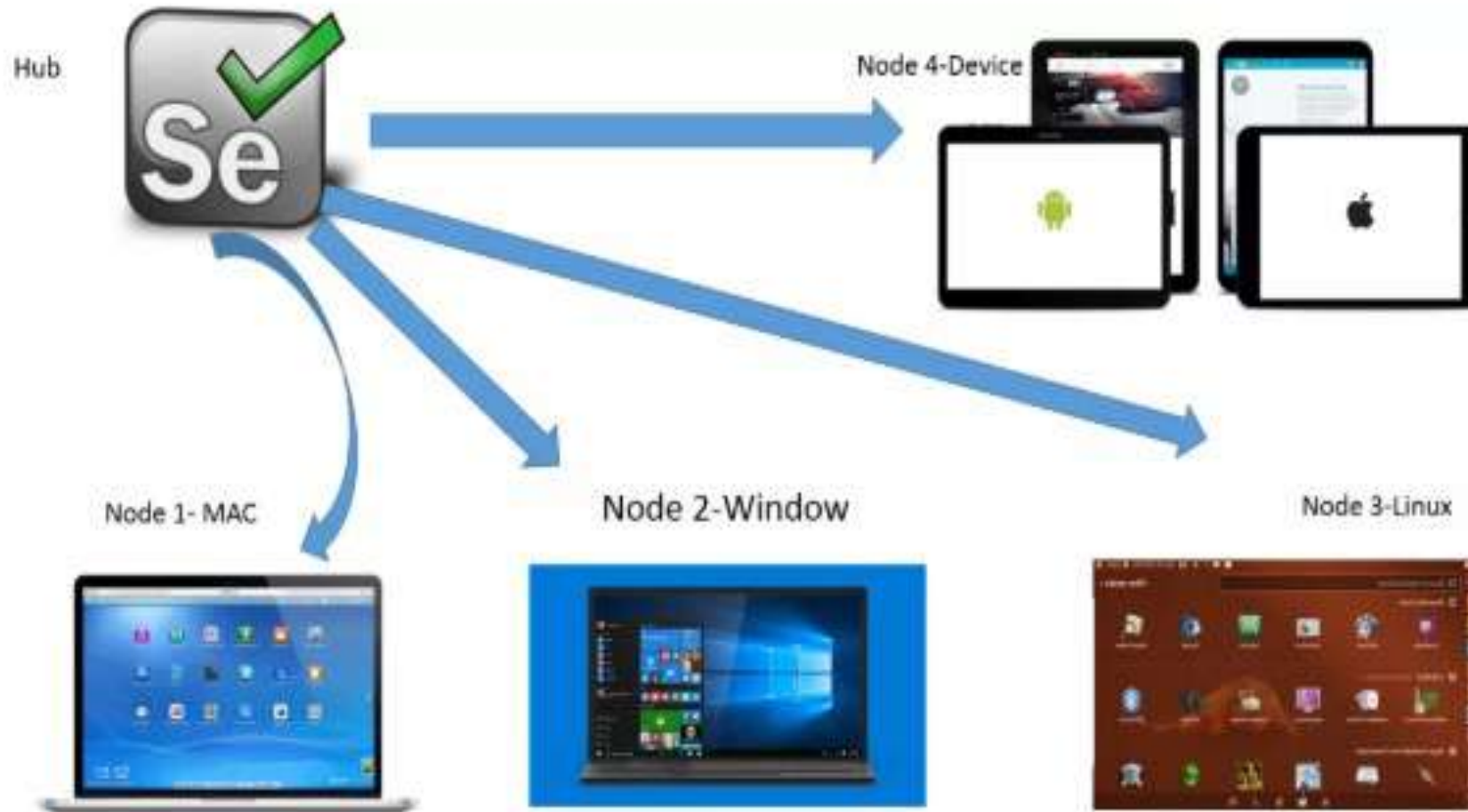
Selenium

Architecture



Selenium

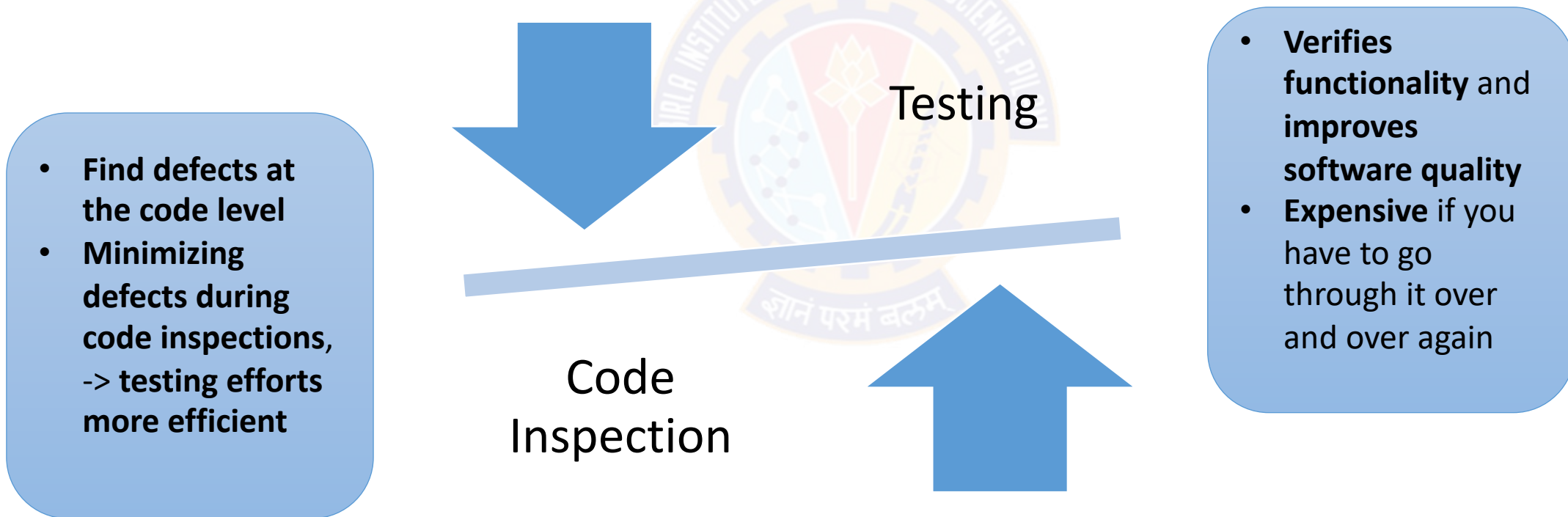
Selenium Grid



Continuous Code Inspection

Continuous code inspection = Constantly scanning code

- Identify if any defects
- It is a process of code review
- Its been proved 90% of defects can be addressed using code inspections tools



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

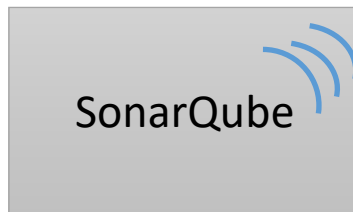
Continuous Code Inspection

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

Continuous Code Inspection Tool

SonarQube



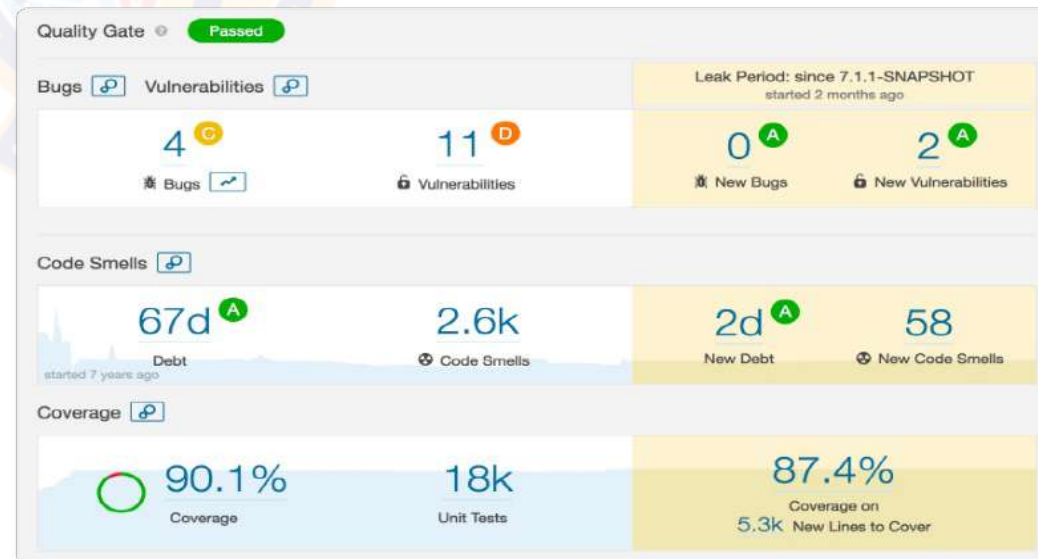
capability to show health of an application



Highlight issues newly introduced



Quality Gate, you can fix the leak and therefore improve code quality systematically



SonarQube

Overall health

- Bug:
 - An issue that represents something wrong in the code
 - If this has not broken yet, it will, and probably at the worst possible moment
- Code Smell:
 - A maintainability-related issue in the code
 - Examples: Dead Code, Duplicate code, Comments, Long method, Long parameter list, Long class etc.,
- Vulnerability:
 - A security-related issue which represents a backdoor for attackers



SonarQube

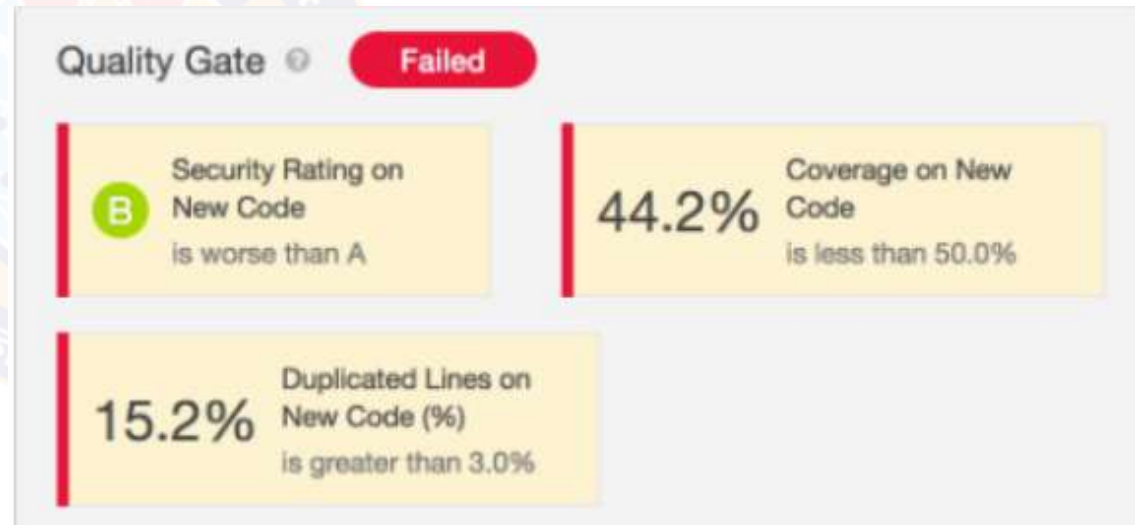
Enforce Quality Gate

- To fully enforce a code quality practice across all teams, need a Quality Gate
- A set of requirements that tells whether or not a new version of a project can go into production
- SonarQube's default Quality Gate checks what happened on the Leak period and fails if your new code got worse in this period

A quality gate is the best way to enforce a quality policy in your organization

Define a set of Boolean conditions based on measure thresholds against which projects are measured

It supports multiple quality gate definitions

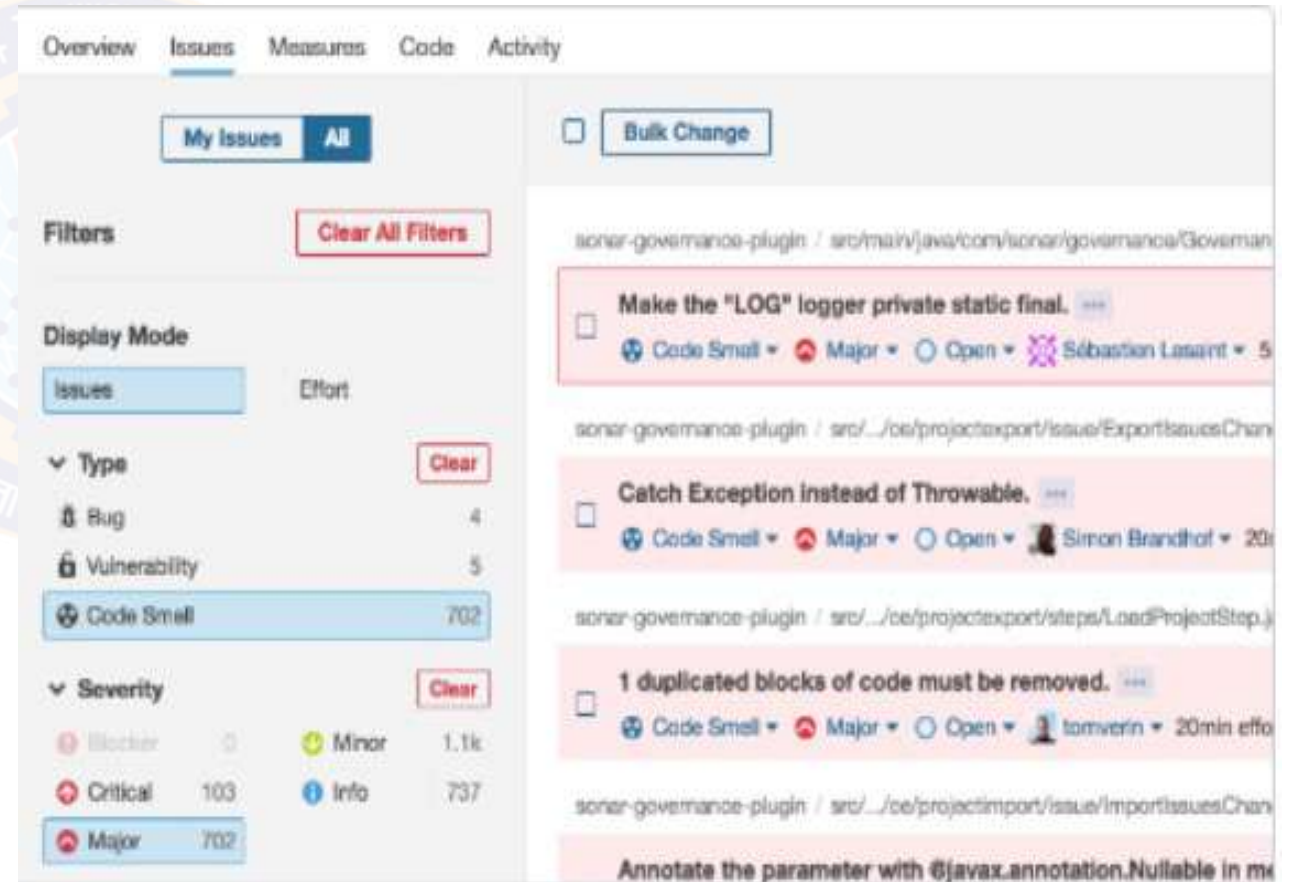


Example: Failed Project

SonarQube

Dig into issues

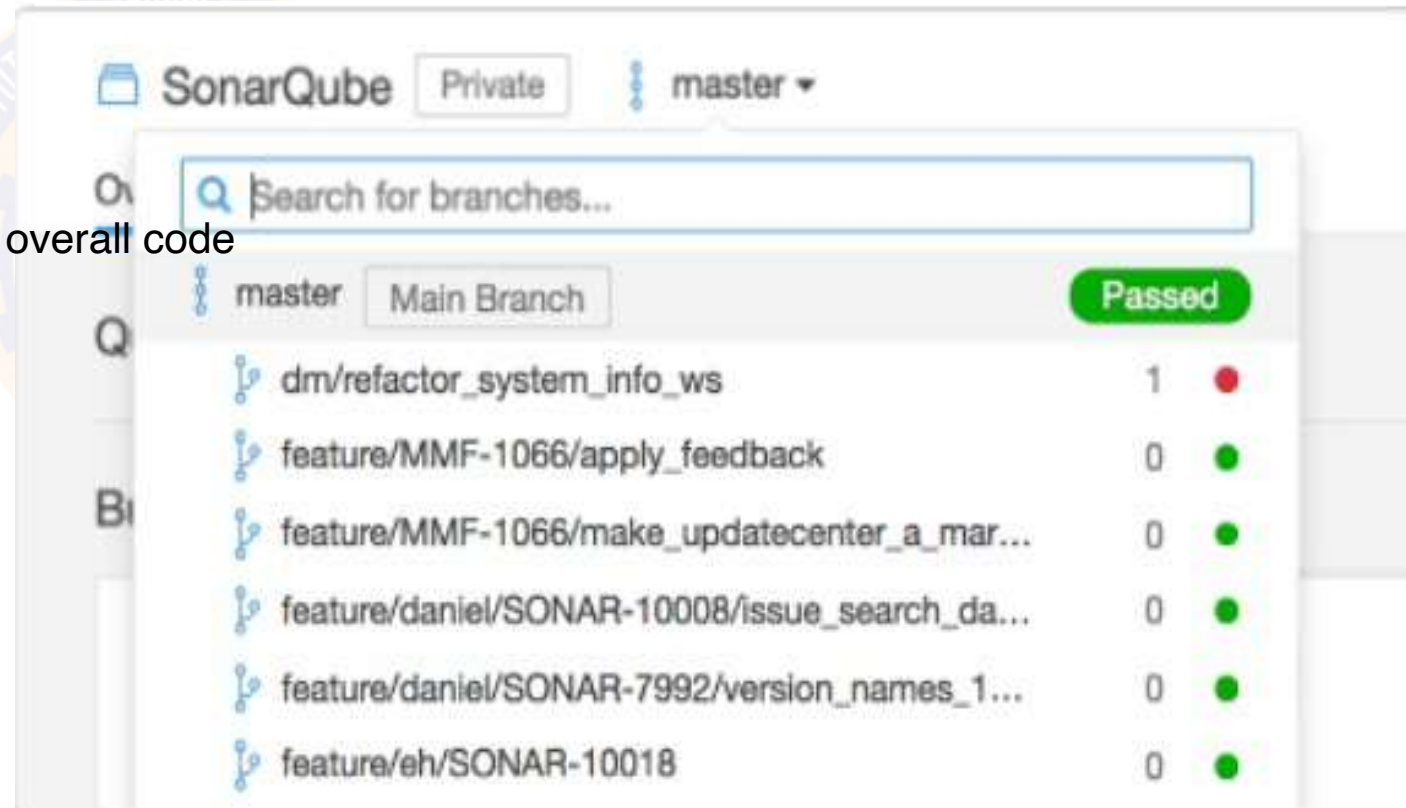
- The “Issues” page of your project gives you full power to analyze in detail
- What the main issues are?
- Where they are located?
- When they were added to your code base?
- And who originally introduced them?



SonarQube

Analyzing Source Code

- Analyze pull requests
 - Focuses on new code – The Pull Request quality gate only uses your project's quality gate conditions that apply to "on New Code" metrics.
- Branch Analysis
 - Each branch has a quality gate that:
 - Applies on conditions on New Code and overall code
 - Assigns a status (Passed or Failed)



SonarQube

Integration for DevOps

maven



Gradle



Makefile

MSBuild

 **Bamboo**

 **Travis CI**

 **Jenkins**

 **AppVeyor**

 **Azure DevOps**

 **TeamCity**



Q&A



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Continuous Integration

- Continuous Integration
- Prerequisites for Continuous Integration Version Control
- Continuous Integration Practices



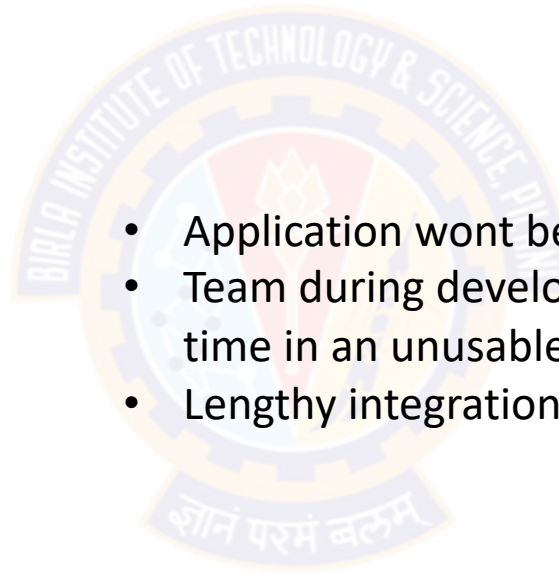
Continuous Integration

Continuous integration (CI)

- Process of integrating new code written by developers with a mainline or “master” branch frequently throughout the day

“Nobody is interested in trying to run the whole application until it is finished”

- Application won't be in a working state
- Team during development spends significant proportion of time in an unusable state
- Lengthy integration phases at the end of development

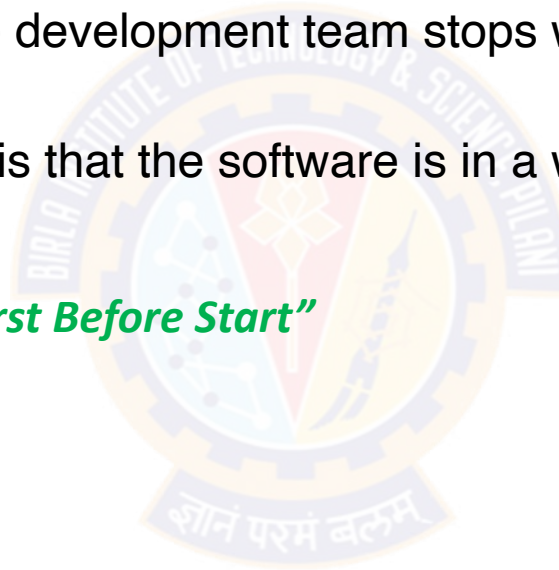


Continuous Integration

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”



Implementing Continuous Integration

Pre-requisites

1. Version Control



2. An Automated Build



3. Agreement of the Team



Continuous Integration Pre-requisite

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 your 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

CI Tools

- Open Source :
 - Jenkins
 - Cruise Control
 - GitLab CI
 - GitHub Actions
- Commercial:
 - ThoughtWorks Studios
 - TeamCity by JetBrains
 - Bamboo by Atlassian
 - BuildForge by IBM



How it was before Continuous Integration

Just a glance !!!

- Nightly Build 😞



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

Continuous Integration

Pre-requisite & Best Practices

Prerequisite

Check In Regularly -> frequent check-ins



Create a Comprehensive Automated Test Suite

Unit Test

Component Test

Acceptance Test



Will provide extremely high level of confidence that any introduced change has not broken existing functionality

Keep the Build and Test Process Short

Standard Recommendation:

10 min is Good

5 min is Better

90 Sec is IDEAL

Managing Your Development Workspace

local Development Workspace must be replica of Production

Continuous Integration

Pre-requisite & Best Practices

Best Practices

Don't Check In on a Broken Build

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

Commit locally than direct to Production

Wait for Commit Tests to Pass before Moving On

At time of Check-in, you should monitor the build progress
Never go home with broken build

Always Be Prepared to Revert to the Previous Revision

The previous revision was good because; you don't check in on a broken build

Continuous Integration

Scenario 1

Friday at 5.30 PM; your build is fail.

- You will be leaving late, and try to fix it
- You can revert changes
- You can leave the build broken

What Option you will opt here?



✗ Stay late to fix the build after working hours

Check in early enough so you have helping hands around
If it is late to Check In then Save your Check in for Next Day

Make rule of not checking in less than an hour before the end of work

Best Solution if you are alone then:

Revert it from your Source Control

Leaving Broken Build:

- On Monday your memory will no longer be fresh
- It will take you significantly longer to understand the problem and fix it
- Everyone at work will yell at you
- If you are late on Monday; be ready to answer a number of calls
- Not the least your name will be mud

Continuous Integration

Scenario 2

If you try to revert every time then; how can you make progress?

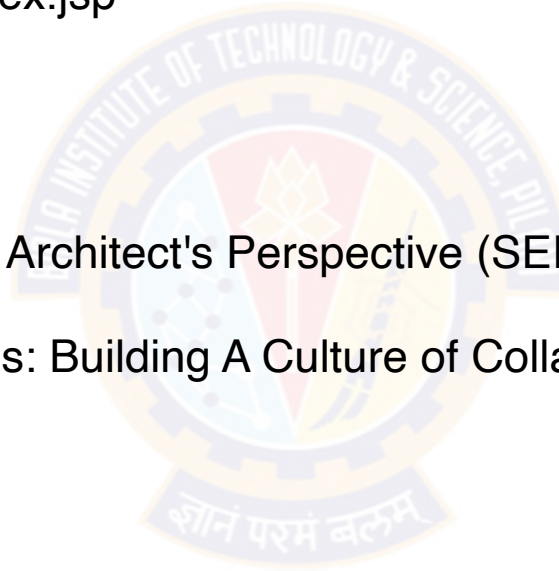


Time Boxing

- Establish a team rule
- When the build breaks on check-in, try to fix it for ten minutes or any approximate time your 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

CS 8 and 9

- Chapter 4, from DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu ,
- <https://www.seleniumhq.org/docs/index.jsp>
- <https://www.sonarsource.com>
- <https://docs.sonarqube.org>
- Chapter 5, from DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu ,
- Chapter 11,12, from Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis





Q&A



Thank You!

In our next session: