AGILE OVERVIEW

The Agile Manifesto

12 Agile Principles

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

Adaptive Planning

- Schedule – locked
- Money – locked
- Scope – Flexible
- Backlog – Customer/Feedback
- Task list is flexible

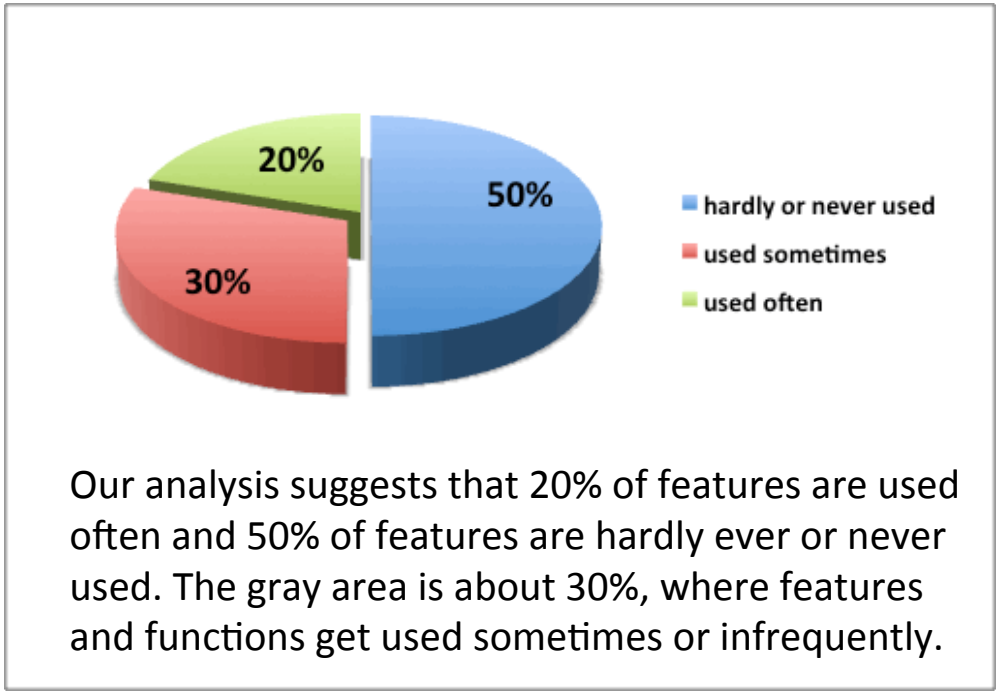
Versus

Feature Based Planning

- Schedule – flexible
- Money – partly flexible (but more fit to fixed price contract model)
- Scope – locked
- Backlog – Managers
- Task list is not flexible but locked

If a project manager delivers a project **on-time**, **within budget** and **all of the defined up-front scope**, is she/he successful?

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



Our analysis suggests that 20% of features are used often and 50% of features are hardly ever or never used. The gray area is about 30%, where features and functions get used sometimes or infrequently.

I'll know when I see it
If you can't know it upfront you will have to accept change.

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

Agile is getting to the wrong answer fast!

What sense does that make?

It's better than getting to the wrong answer slow!!

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

"I don't have time to work together with you daily."

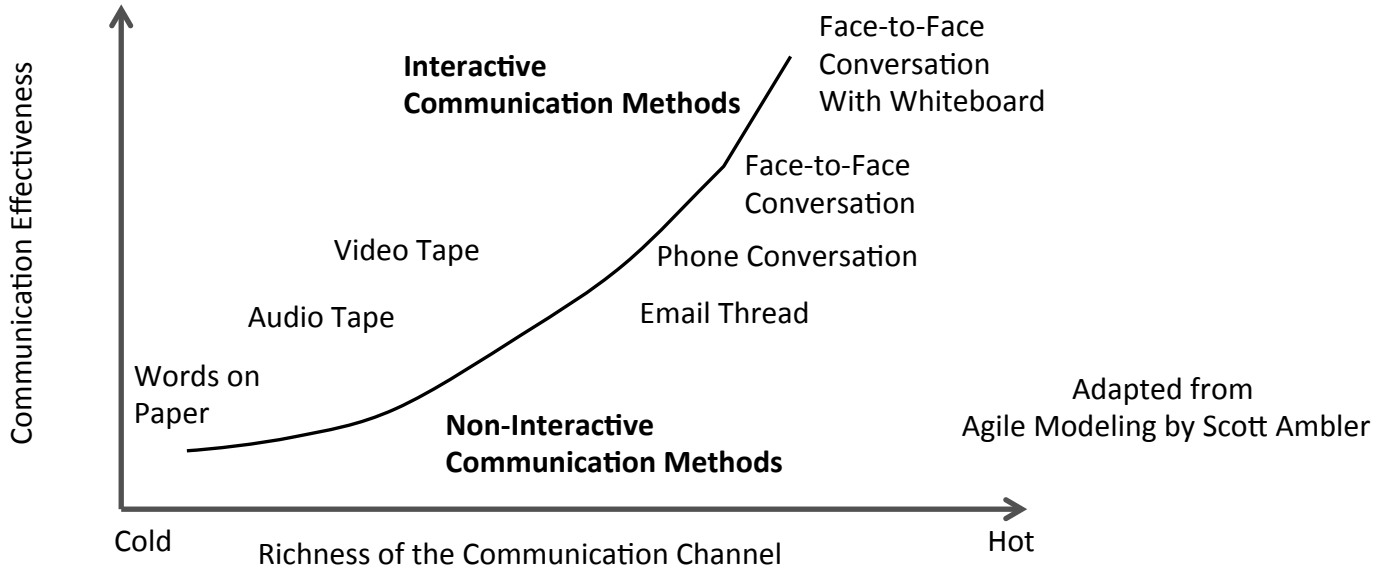
"Do you have time to fail?"

Getting feedback from stakeholders will only help developers

5. Build projects around motivated individuals.
Give them the environment and support they need, and trust them to get the job done.

Knowledge Workers are motivated by:
Autonomy, Mastery and Purpose

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.

Working Software represents Value in Software Development
Agile focuses on delivering Value

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

Would you run machinery in a factory night and day,
never give it a rest and never maintain it?

9. Continuous attention to technical excellence by reducing technical debt.

Huge impact of bad quality on value creation

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

Reduce complexity
Pay of technical debt
Induce adaptability

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

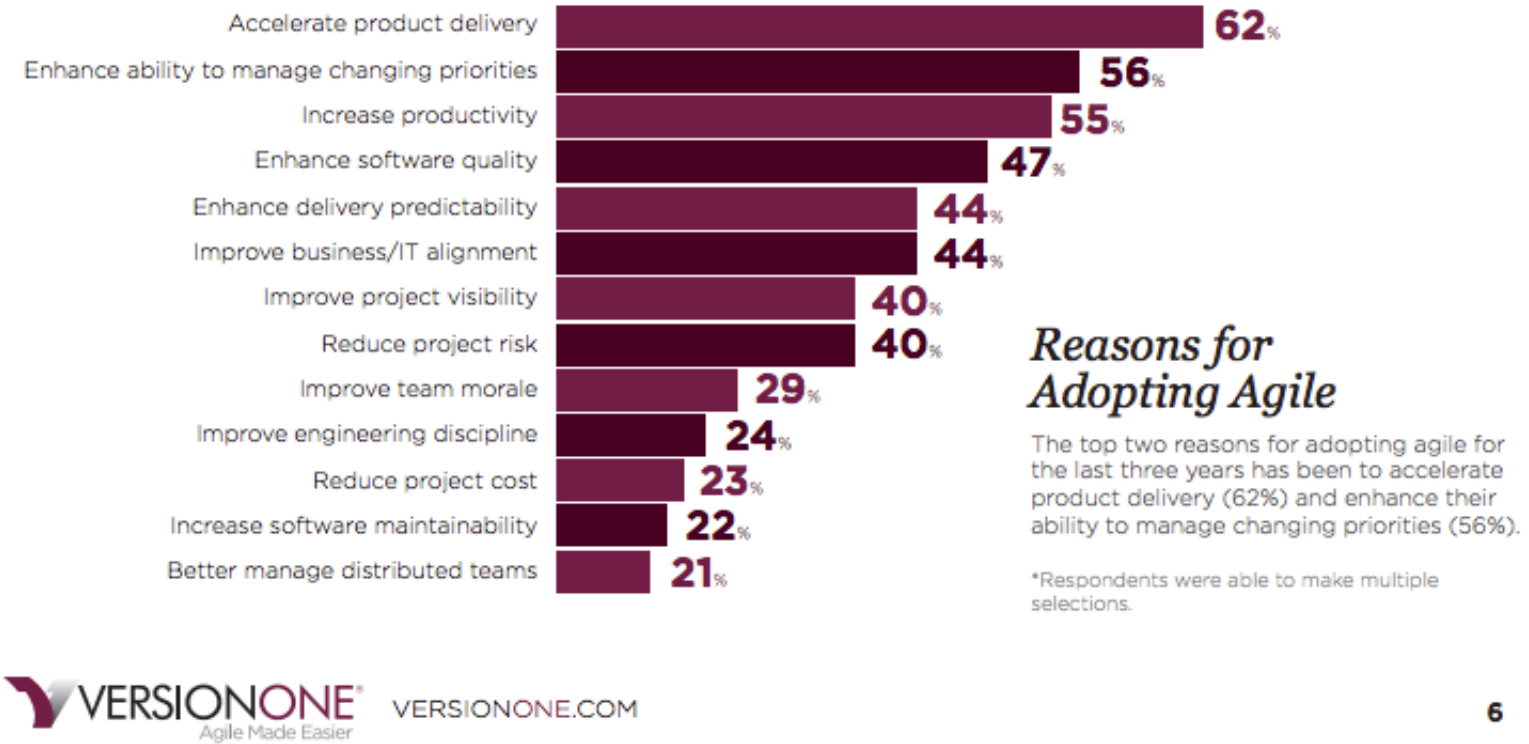
Group of motivated individuals
work together towards a goal,
have the ability and authority to take decisions
and readily adapt to changing demands

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

Continual Learning process
Experimentation, Visibility, Inspection, Adaption

Reasons for Adopting Agile

What to expect from Agile…



VersionOne – The 10th Annual State of Agile Development Survey - 2016

Reasons for Adopting Agile

Actual Improvements from Implementing Agile…

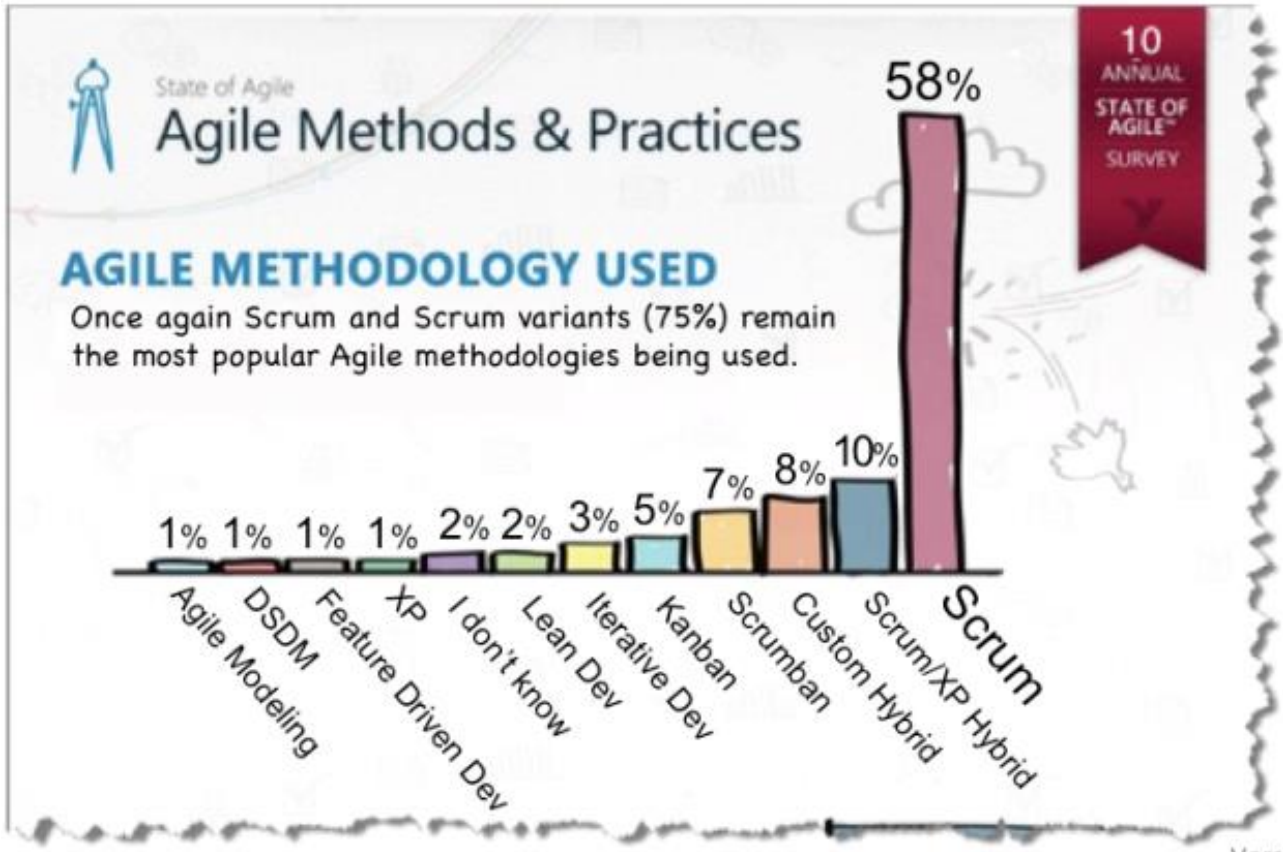
The top three benefits of adopting agile have remained steady for the past five years.

BENEFIT	GOT BETTER	NO CHANGE	GOT WORSE	DON'T KNOW
Ability to manage changing priorities	87%	3%	1%	9%
Increased team productivity	85%	3%	1%	11%
Improved project visibility	84%	3%	1%	12%
Increased team morale/motivation	81%	5%	3%	11%
Better delivery predictability	81%	6%	2%	11%
Faster time to market	80%	7%	1%	13%
Enhanced software quality	79%	6%	2%	14%
Reduced project risk	78%	6%	1%	15%
Improved business/IT alignment	77%	6%	1%	16%
Improved engineering discipline	73%	7%	2%	19%
Enhanced software maintainability	70%	8%	2%	21%
Better manage distributed teams	62%	11%	2%	25%

*Respondents were able to make multiple selections.

VersionOne – The 10th Annual State of Agile Development Survey - 2016

Agile Flavors



VersionOne, 2016

What is Lean?

- The core idea is to maximize **customer value** while minimizing waste.
- A lean organization understands customer value and focuses its key processes to continuously increase it.
- The ultimate goal: provide value to the customer through a value creation process that has zero waste.
- Optimize the Whole

Lean is the pathway to Agile

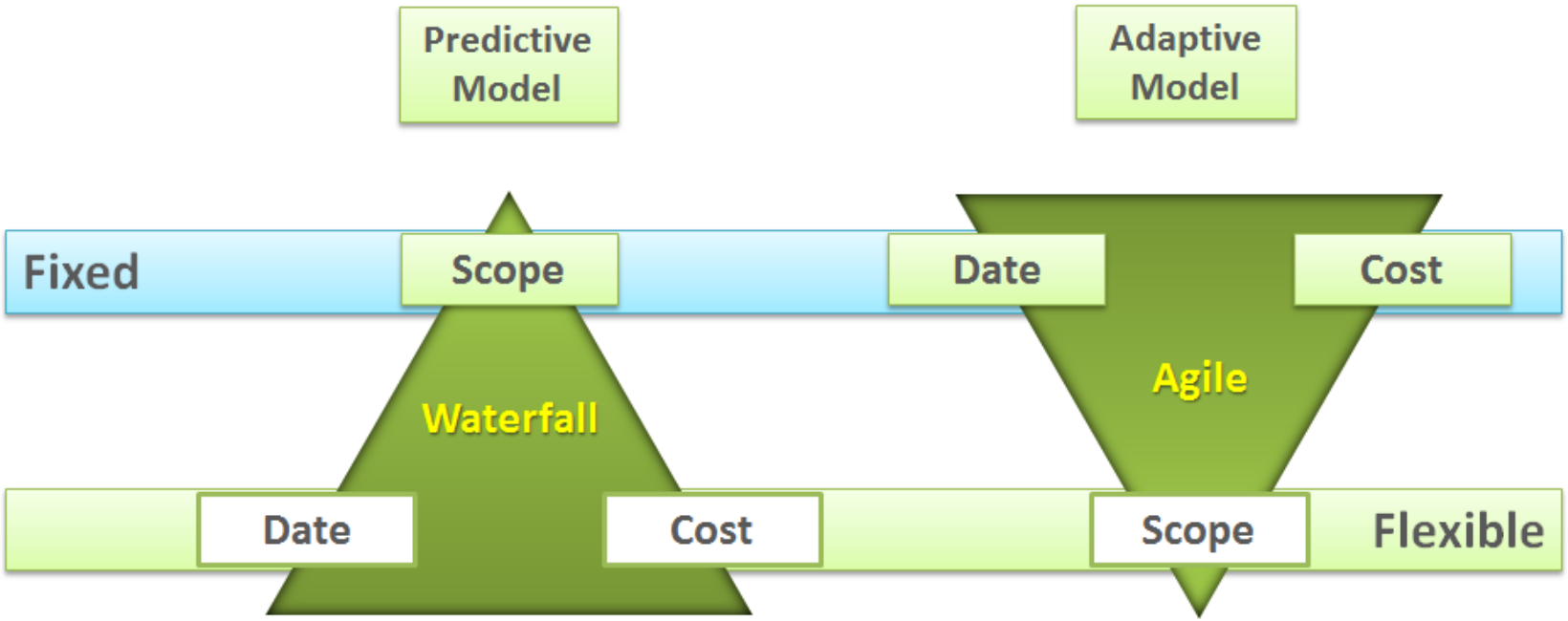
- Why does it matter?
  - Lean has over 50 years of proving and evolving.
  - Lean is not a one-size-fits-all solution.
  - Lean needs constant attention to remain effective.
  - If Agile is built on LEAN – Agile inherits Lean characteristics and Agile is not just a passing fad.

Waterfall Methodology

Waterfall is a plan-driven method of developing software

Agile Vs. Waterfall

Agile is a mindset… a different way to think about the work.



DSDM.org, 1994

Roles in Scrum

- Scrum Team Member – Dev, QA, Ops, Infosec
- Product Owner
  - Manage user stories and priorities
- Scrum Master
  - Facilitates ceremonies
- Engineering Manager
- Architect

User Stories

A user story is:

- A concise, written description of a piece of functionality that will be valuable to a user (or owner) of the software.
- A deliverable
- **As a [user role] I want to [goal] so I can [value**
- ***For example:*** **As a registered user I want to log in so I can access subscriber-only content**

User Stories

- **Who (user role)**
- **What (goal)**
- **Why (value)**
  - gives clarity as to why a feature is useful
  - can influence how a feature should function
  - can give you ideas for other useful features that support the user's goals

Invest in good user stories!

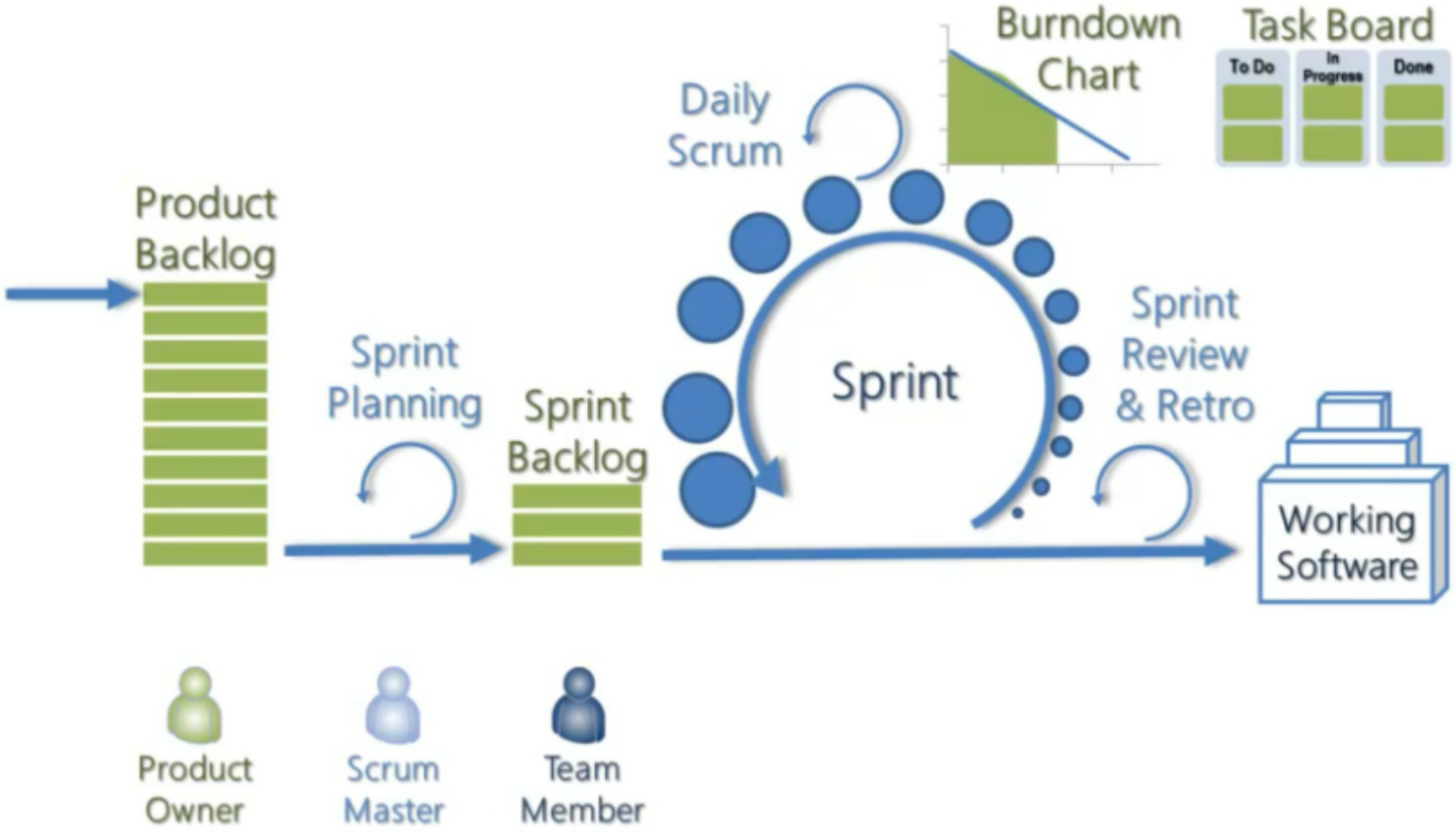
User stories should have these characteristics:

- **I**ndependent
- **N**egotiable
- **V**aluable
- **E**stimable
- **S**mall
- **T**estable

More terms in Scrum

- Agile Estimating and Planning (Preparing work for Process Consumption – a.k.a. Sprint/Iteration 0)
- Scrum = Value Stream Framework
- Daily standup
- Sprint = Iteration = time-box
- Product Backlog = Requirements list
- User Story = Requirement with Customer Value
- Non-User Story = Requirement that contributes to Customer value (Technical perspective)
- Epic = Large Story (can't fit in Iteration)
- Story Point = time-less unit of size (relative estimating)
- Velocity = # story points per Sprint for team
- Scrum Master = Agile Project Manager
- Product Owner = Voice of the Customer
- Team Member = Task takers

Scrum Framework



More on User stories

- Acceptance Criteria
- User Story points
- Definition of done

The Importance of Acceptance Criteria

- Represents the items that the PO will verify in order to confirm that a story is done.
- Gives the team the detail necessary to delimit the product and correctly size the story..
- The team will build the simplest solution to the acceptance criteria; if you care about something, communicate it to the team.

How to Estimate User Stories

- **Curse**
    - **C**omplexity
    - **U**ncertainty
    - **R**isk
    - **S**cope
    - **E**ffort

- X-Small, Small, Medium, Large, Extra-Large (known as “T-Shirt Sizing”)
- Fibonacci sequence: 1,2,3,5,8,13,21

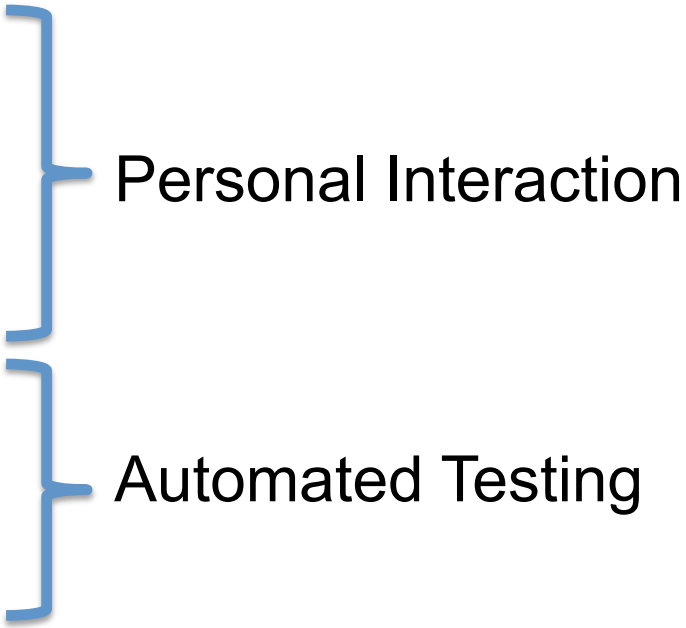
Definition of done

- Design
- Code
- Test – what to test?
  - Unit test sufficient
  - Or integration test is required
- Added: Review code for Security Vulnerabilities
- Review with the product manager

Feedback Loops make Scrum effective

Agile processes are “Like driving a car …
constantly observing and adjusting”
– Kent Beck

- Iterations (few weeks)
- Daily Scrum
- Continual team interaction
- Continuous Integration
- Test-Driven Development

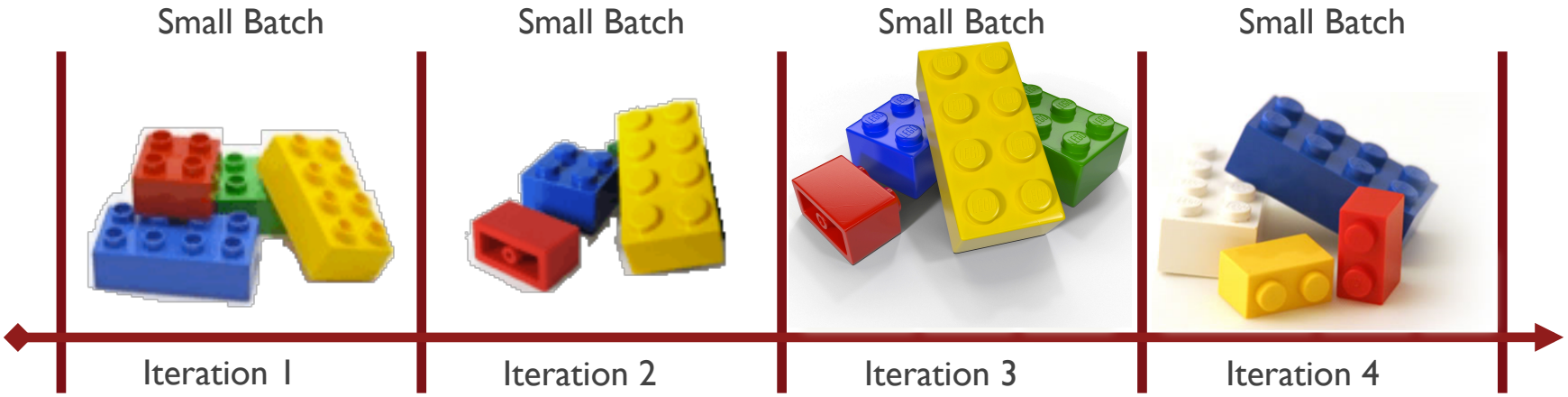


That’s **Scrum**

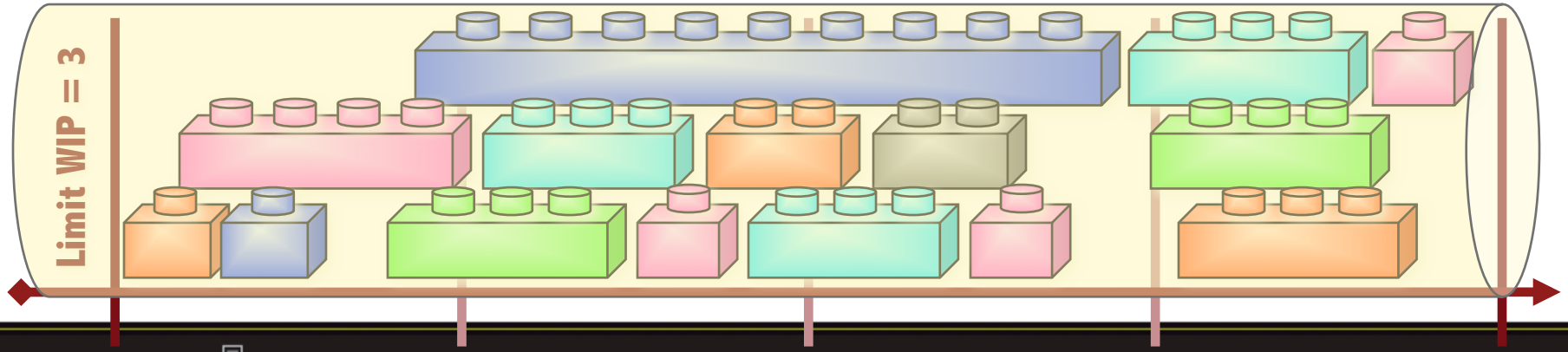
What about **Kanban** !!

Kanban is not Batching

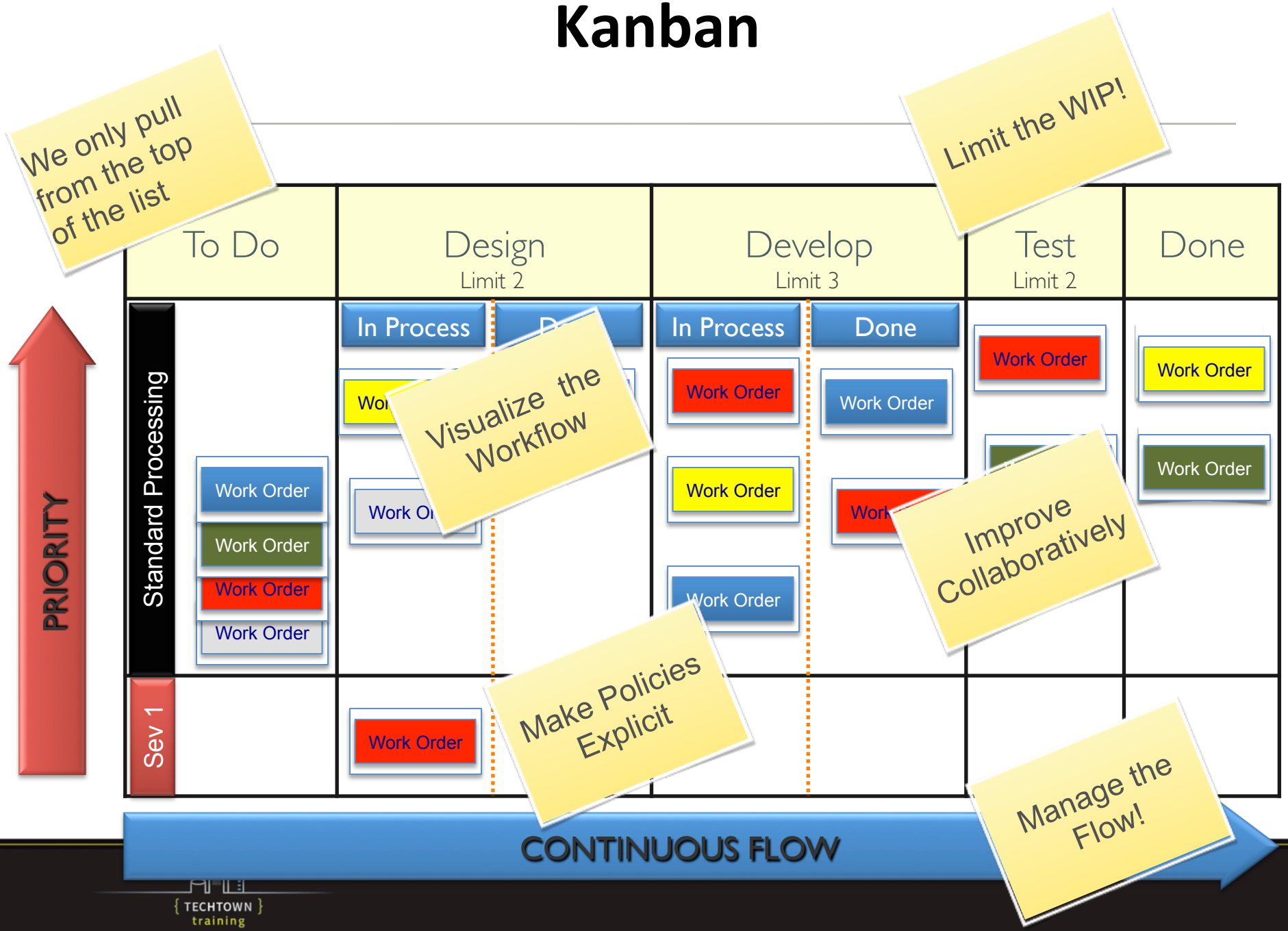
Scrum represents small batching



Kanban is continuous flow through a pipeline sized to the WIP Limit



Kanban


We only pull from the top of the list
Limit the WIP!
To Do
Design
Limit 2
Develop
Limit 3
Test
Limit 2
Done
PRIORITY
Standard Processing
Sev 1
In Process
In Process
Done
Work Order
Visualize the Workflow
Improve Collaboratively
Make Policies Explicit
Manage the Flow!
CONTINUOUS FLOW

Agile Defect Management

- Purposes
  - Ensure every defect is fixed or accepted
  - Provide data for Metrics and analysis
- Options
  - Handle defects face-to-face
  - Write & manage Defect Stories
  - Use a Defect or Issue Tracking System

Acceptance Test-Driven Dev (ATDD)

Product Owner defines (prior to coding):

- Acceptance Criteria for each User Story
- Acceptance Tests (User Story, Feature, Product)

Developers use Acceptance Tests:

- Ensure they develop what is needed
- Test for "Doneness"

Testers collaborate with Product Owner and Developers

Behavior-Driven Development (BDD)

Like ATDD, but focused on System Behavior in response to User Behavior

- User Story Acceptance Criteria:
  1 or more "Given/When/Then"
  - Given (a particular scenario)
  - When (the user does something)
  - Then (this should be the result)

DEVOPS CULTURE

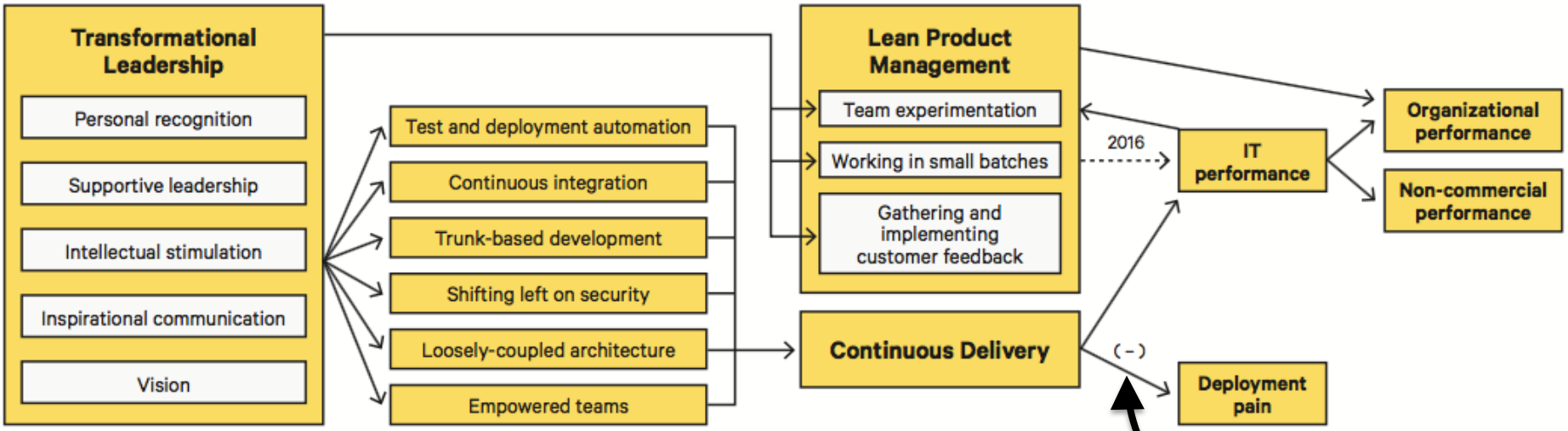
DevOps Defined

Originally

Tools, methods and processes
that enable collaboration
~~between~~ → among all
~~Application Development~~ → professionals
~~and IT Operations~~ → in an IT shop
to increase
the velocity and reliability
of ~~software deployment~~ → everything we do

NOW

High-Performance is Predicted by Transformational Leadership



⟶ = "predicts, affects, drives, or impacts"

High-Performing Teams Better Throughput and Stability

Survey Question	High IT Performers	Medium IT Performers	Low IT Performers
Deployment Frequency	On demand (multiple deploys per day)	Between once per week and once per month	Between once per week and once per month*
Lead time for changes	Less than one hour	Between one week and one month	Between one week and one month*
Mean time to recover (MTTR)	Less than one hour	Less than one day	Between one day and one week
Change failure rate	0-15%	0-15%	31-45%

2017 State of DevOps Report | presented by Puppet + DORA

High-Performing Teams Automate – Do Less Manual Work

	High performers	Medium performers	Low performers
Configuration management	28%	47%[a]	46%[a]
Testing	35%	51%[b]	49%[b]
Deployments	26%	47%	43%
Change approval processes	48%	67%	59%

[a, b] *Not significantly different*

Percentage of work that is done manually

High Performance Requires Loosely-Coupled Architecture & Teams

The biggest contributor to continuous delivery

A Team can do these:	Without these:
Make large-scale changes to the design of its system	Permission from someone outside the team
	Other teams making changes in their systems
	Creating significant work for other teams
Complete its work	Fine-grained communication and coordination with people outside the team
Deploy and release on demand	Coordinating with the release and deployment of services they depend upon
Do most of its testing on demand	An integrated test environment
Perform deployments during normal business hours	Appreciable downtime

2017 State of DevOps Report | presented by Puppet + DORA

The Core Chronic Conflict

Two Simultaneous Goals:

Respond to
rapidly changing needs

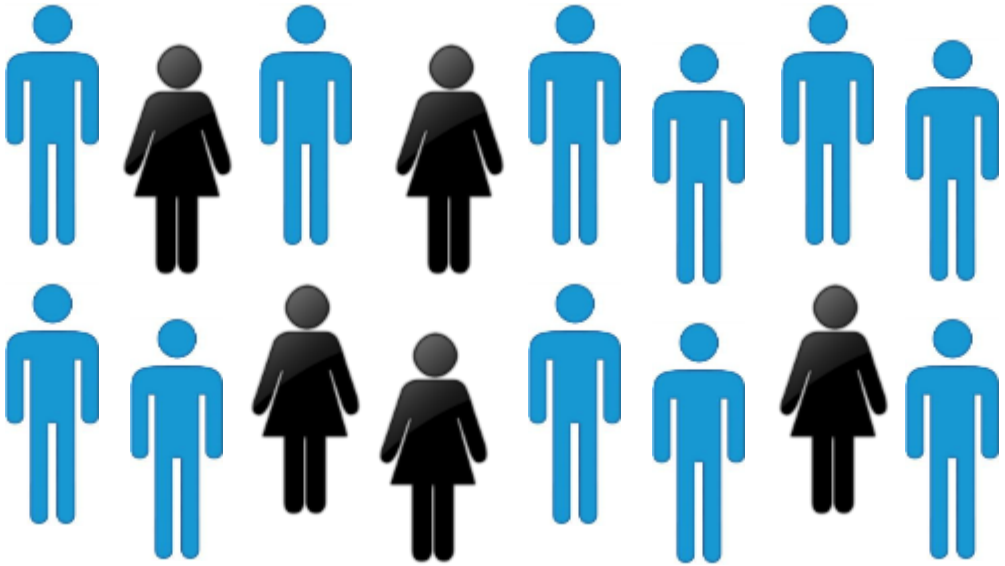
Provide stable,
reliable, secure service

Dev

Ops

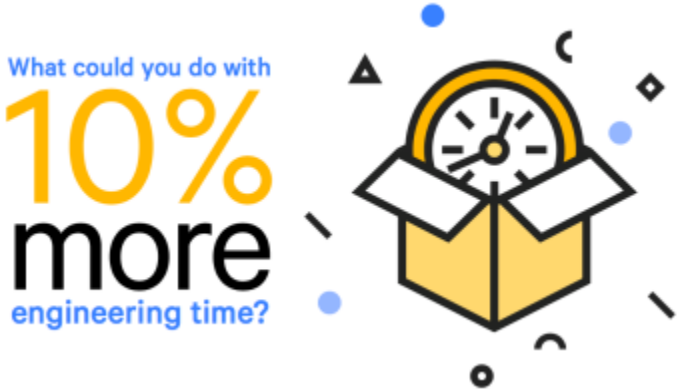
How DevOps Helps

One IT organization, Cross-functional teams, shared goals



Business Value of DevOps

- Costs Avoided
  - Excess Rework
  - Downtime reduced
- Reinvested
  - Accelerate time to market
  - Gain business intelligence
  - Improve customer engagement & retention



Case Study: Knight Capital
Business Value of DevOps Illustrated

Background

- 1 August 2012, Knight Capital Group needed to update its infrastructure to keep up with growing demand
  - Largest trader of US equities with a 17% Market share of NYSE and a 16.9% Market share of Nasdaq
  - 19,000 US securities traded daily
  - Daily trading volume of $21B
  - Worth approx $1.5B
  - Employed 1450 people

Case Study: Knight Capital
Business Value of DevOps Illustrated

The New Application

- New software router, SMARS, to enable its customers to take advantage of a new Retail Liquidity Program (RLP) that started August 1, 2012.
- Code from Power Peg (old router) was used in SMARS.
- Power Peg had been unused since 2003
  but was still installed on the production servers

Case Study: Knight Capital
Business Value of DevOps Illustrated

Manual Deployment

- July 27 SMARS was deployed to servers in stages on successive days
  - 7 of the 8 servers successfully updated, but one was missed
  - No one remembered that the old Power Peg application was still running on the 8th server

Manual Recovery

- August 1st, Knight began participating in the RLP
  - 7 of the 8 servers functioned as expected
  - The eighth server began executing Power Peg code instead of the new SMARS
  - 97 alert emails were sent, beginning at 8:01
  - 45 minutes later, the eighth server was identified and stopped

Case Study: Knight Capital
Business Value of DevOps Illustrated

Business Impact

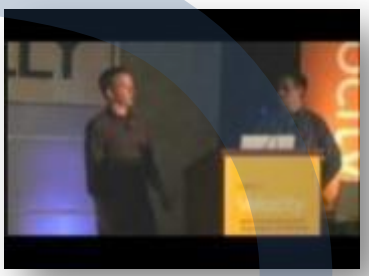
Knight Capital Group lost $460M in 45 minutes

- The rogue server executed 212 erroneous orders, resulting in 4 million executions in 143 stocks
  for more than 397 million shares.
- That day, the company stock plunged 33% and by the next day 75% of the companies value had been erased
- Six months later, Knight Capital Group was forced to merge with KCG Holdings

The History of DevOps

2007

Agile2008 Conference



puppet labs

ANSIBLE

CIO

DEVOPS DAYS

flickr

SALTSTACK

CHEF

redhat

View Damon Edwards' 12-minute video on the history of DevOps at www.itrevolution.com/the-history-of-devops

Highlights From Video

Early Key Points from "10+ Deploys Per Day…"

1. Automated infrastructure
2. Shared version control
3. One step build and deploy
4. Feature flags
5. Shared metrics
6. IRC and IM robots

1. Respect
2. Trust
3. Healthy attitude about failure
4. Avoiding Blame

flickr

John Allspaw (Director of Development) and John Hammond (Director of IT Operations)

Highlights From Video

Where Did DevOps Come From?

1. W Edwards Deming
2. Total Quality Management
3. The Lean Movement
4. Toyota Production System
5. Agile Development
   & Agile Infrastructure Movements
6. Continuous Delivery Movement

1. W Edwards Deming

- Greatest quality guru of the 20th century
  (He lived 1900-1993)
- Articulated the 14 Principles (in notes below)
- Built on the work of Walter A Shewhart
- Laid the foundation for
  - Total Quality Management
  - The Lean Movement
  - The Toyota Production System

2. Total Quality Management

- Developed for the US Navy (1970-80s)
- Response to Japan's quality supremacy
- Based on the work of W Edwards Deming
- TQM Key Concepts:
  - Quality is defined by customers' requirements.
  - Top management has direct responsibility for quality improvement.
  - Increased quality comes from systematic analysis and improvement of work processes.
  - Quality improvement is a continuous effort and conducted throughout the organization.

3. The Lean Movement

Codified in the Toyota Production System (1980s)

- Value Stream Mapping
- Kanbans
- Major Tenets:
  - **Short Lead Time** is the best predictor of quality, customer satisfaction and employee happiness
  - **Small Batch Size** is the best predictor of short lead times

4. Toyota Production System

8 Wasters

Defects
Effects of rework, scrap and incorrect information

Overproduction
Producing more than is needed, or before it is needed

Waiting
Time spent waiting for the next step of the process

Non-Utilized Talent
Underutilizing people's talents, skills, and knowledge

Transportation
Unnecessary movement of products or materials

Inventory
Excess products and materials being produced

Motion
Unnecessary movement by people (e.g. walking)

Extra Processing
More work of higher quality than is required by the customer

Exercise 1.1
Identify Waste

- Identify 3 or more areas of waste in your team or organization.
- How can you communicate these to the affected teams or executives?
- What is one thing you can attempt to do to better understand or even eliminate one of these items.

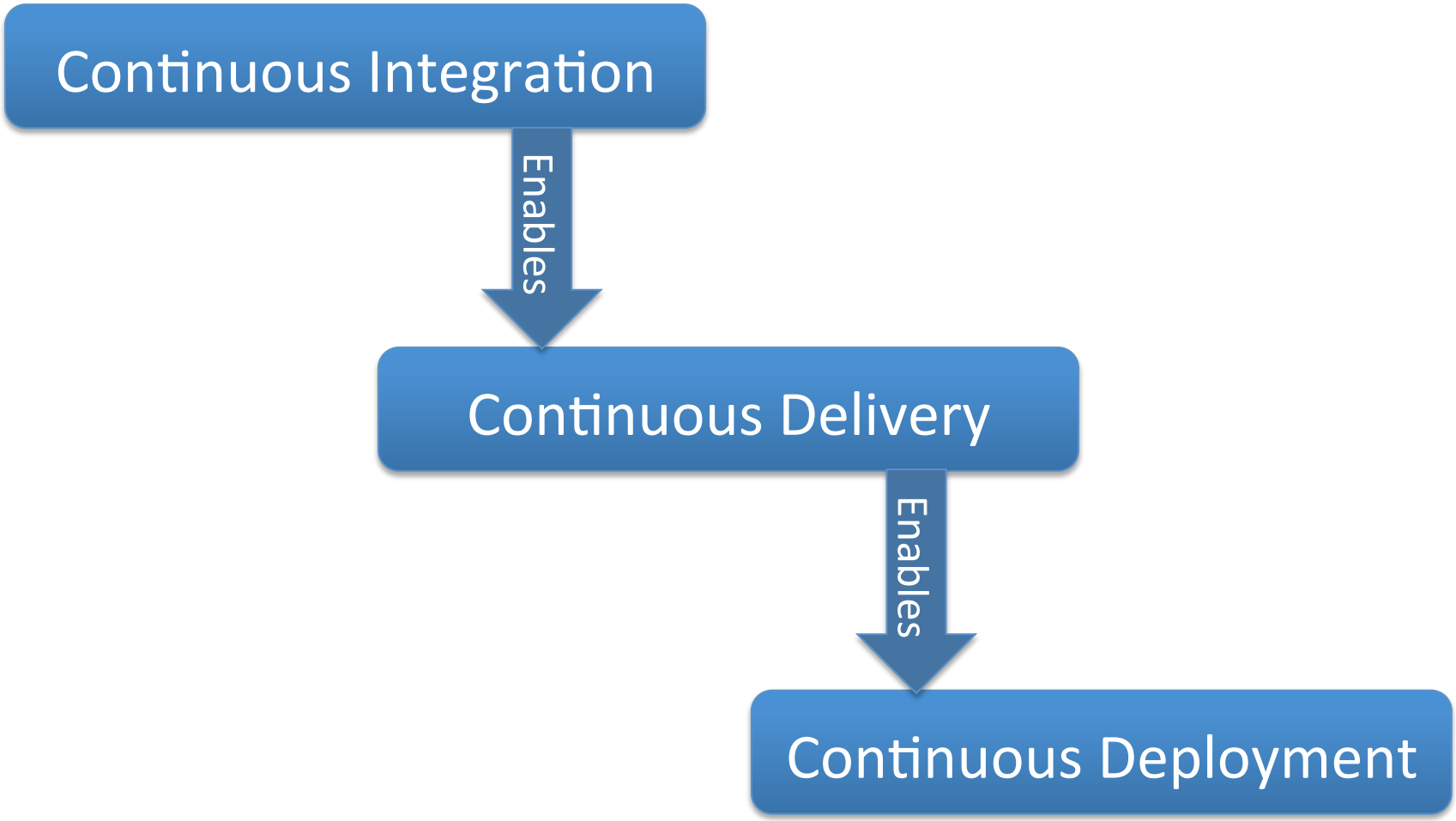
5. Agile Development & Agile Infrastructure Movements

The Agile Manifesto *(2001)*: We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions	**OVER**	**Processes and tools**
Working software	**OVER**	**Comprehensive documentation**
Customer collaboration	**OVER**	**Contract negotiation**
Responding to change	**OVER**	**Following a plan**

That is, while there is value in the items on the right, we value the items on the left more.

6. Continuous Delivery Movement



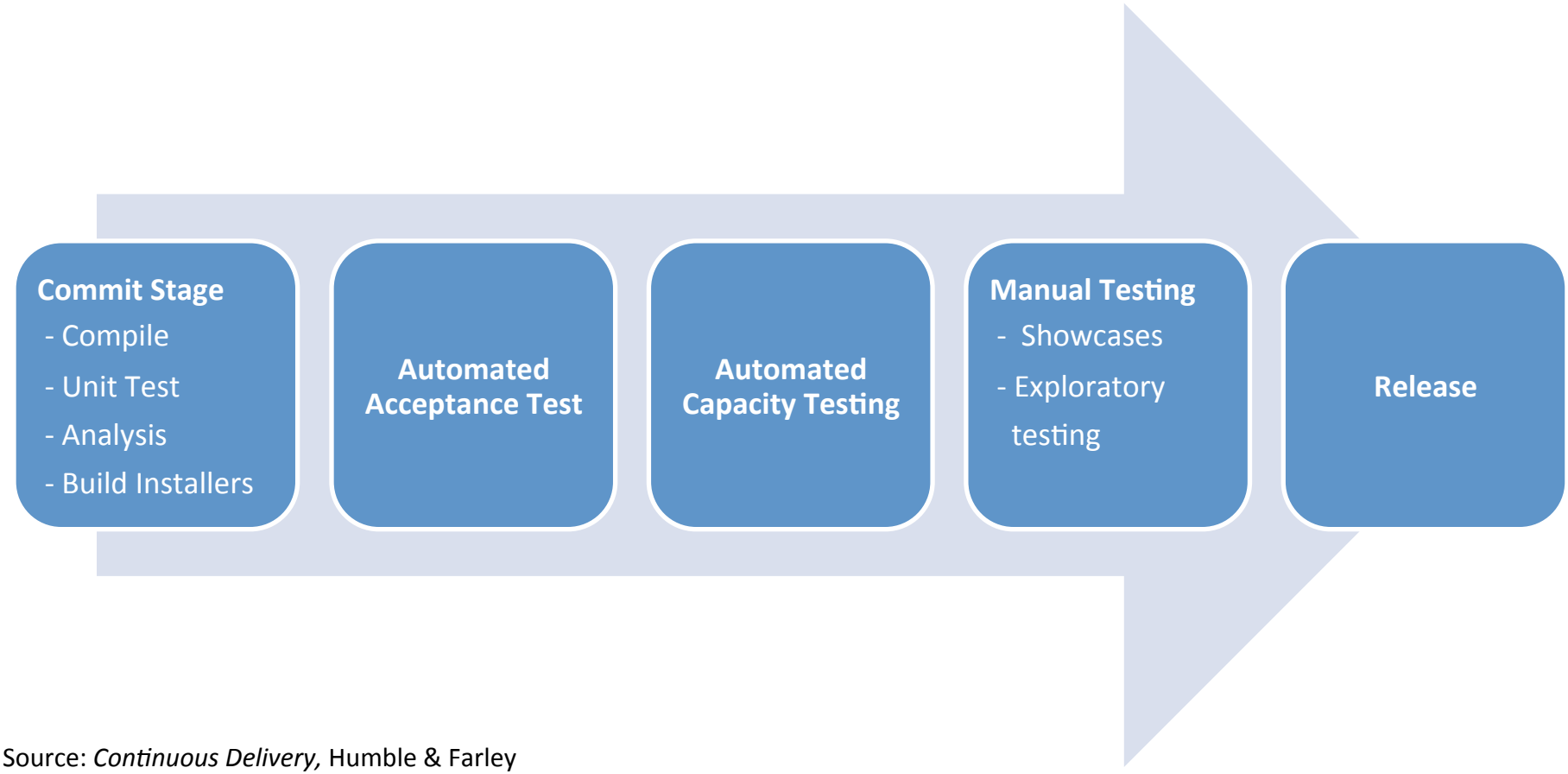
Continuous Delivery

We are doing Continuous Delivery if:

- We are doing Continuous Integration
  - Develop in small batches
  - Develop on trunk or merge to trunk daily
  - Fix broken builds quickly
- We keep trunk in a releasable state
- We are able to release:
  - On demand
  - With the push of a button
  - During business hours

Jez Humble
The DevOps Handbook
Chapter 12

Continuous Delivery
The Deployment Pipeline



Source: *Continuous Delivery,* Humble & Farley

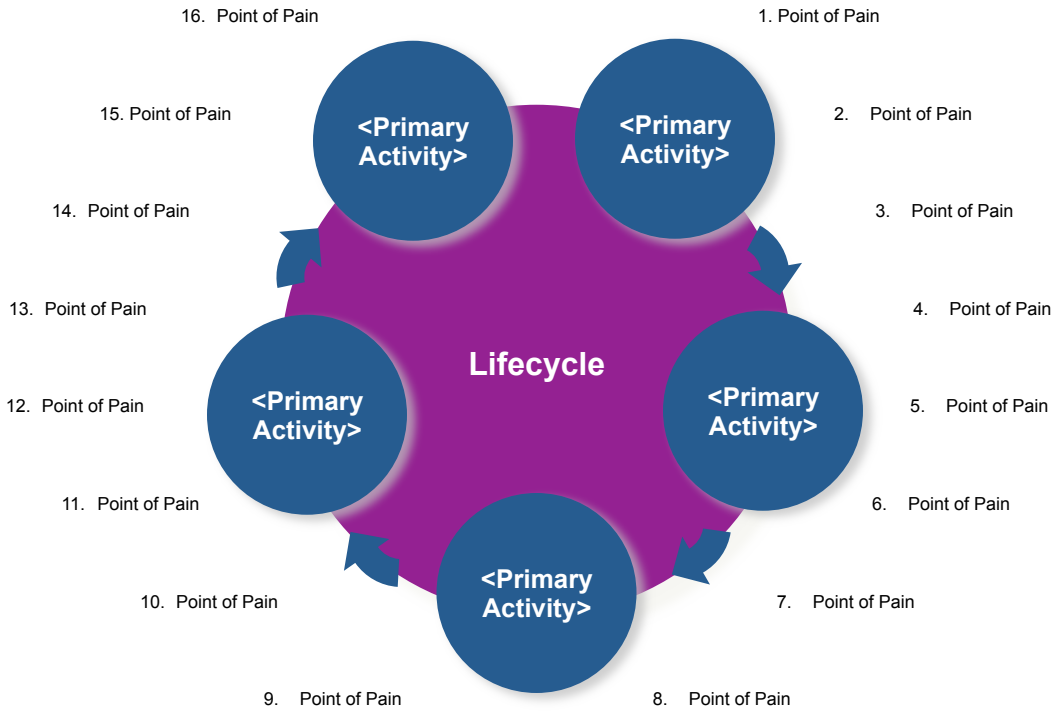
Continuous Delivery Principles

Make releasing software low-risk, repeatable & reliable

- Reduce risk and increase velocity
  - Automate everything you can
  - Attack problematic activities
    - Do them more often (make them more routine)
    - Perform them earlier in the Deployment Pipeline
  - Continually improve the Deployment Pipeline
    - It is everyone's responsibility

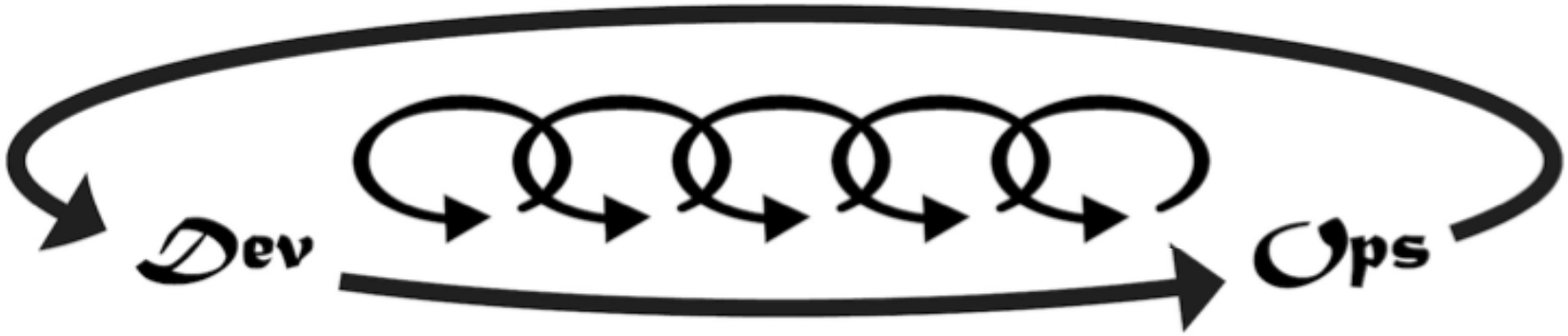
Exercise 1.2
Lifecycle – Points of Pain

1. List your work activities
2. Brainstorm Points of Pain (bottlenecks, problems, etc.)
3. Willing to share with the class?



DevOps is great, but how to enable DevOps ?

Gene Kim's Three Ways



The First Way: Optimize Flow

The Second Way: Amplify Feedback

The Third Way: Continual Learning & Experimentation

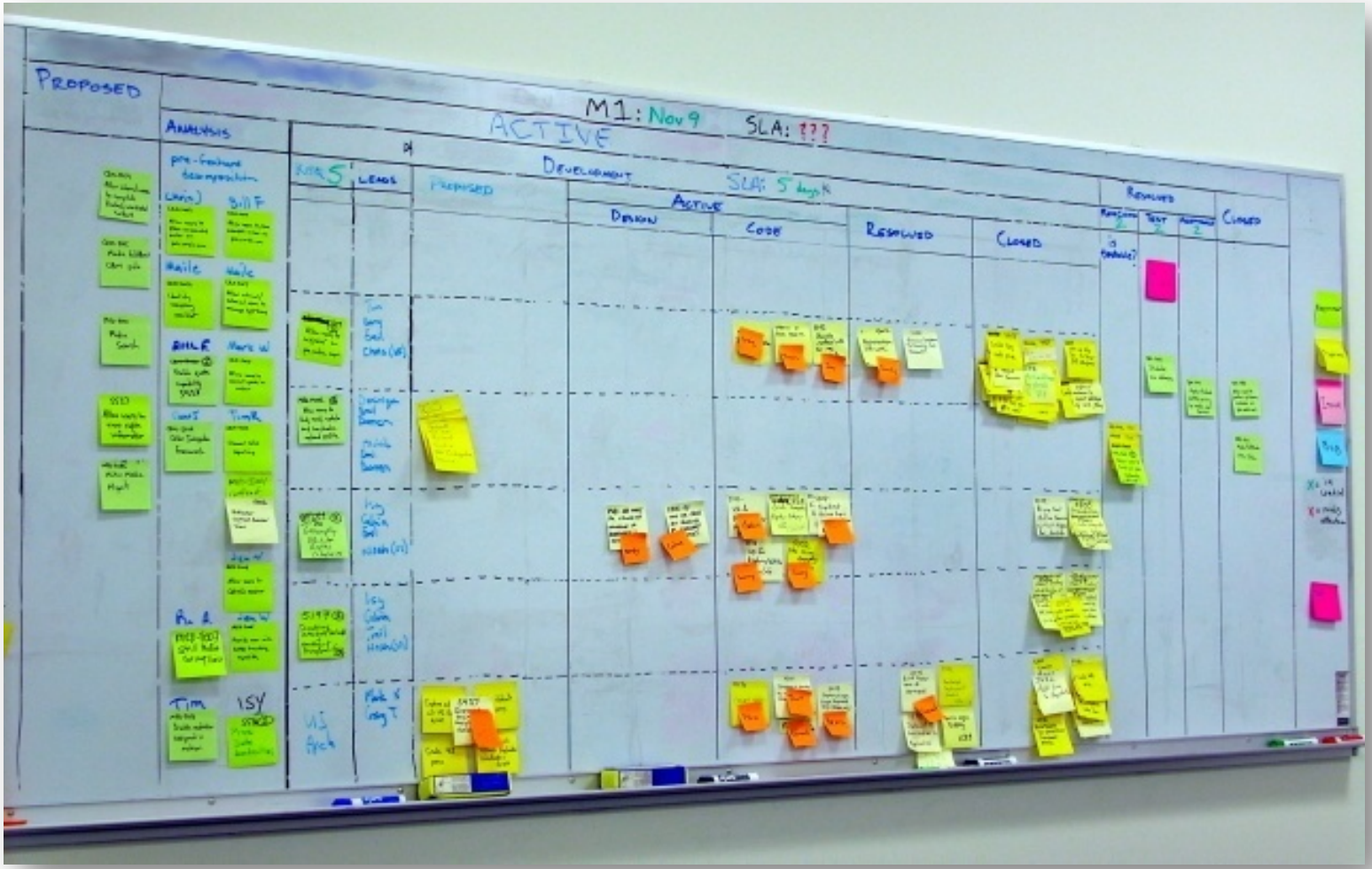
Optimize Flow

1. Principles of Flow
2. Infrastructure As Code
3. Deployment Pipeline
4. Shared Version Control
5. Automated Testing
6. Continuous Integration
7. Containerization
8. Architecture for Reduced-Risk Deployment
9. Change Review and Coordination

1. Principles of Flow

- Make Work Visible
- Limit Work in Process (WIP)
- Reduce Batch Sizes
- Optimize the Process to
  Reduce the Number of Handoffs
- Continually Identify and Elevate Constraints
- Eliminate Waste in the Value Stream

Make Work Visible



Reduce Batch Sizes

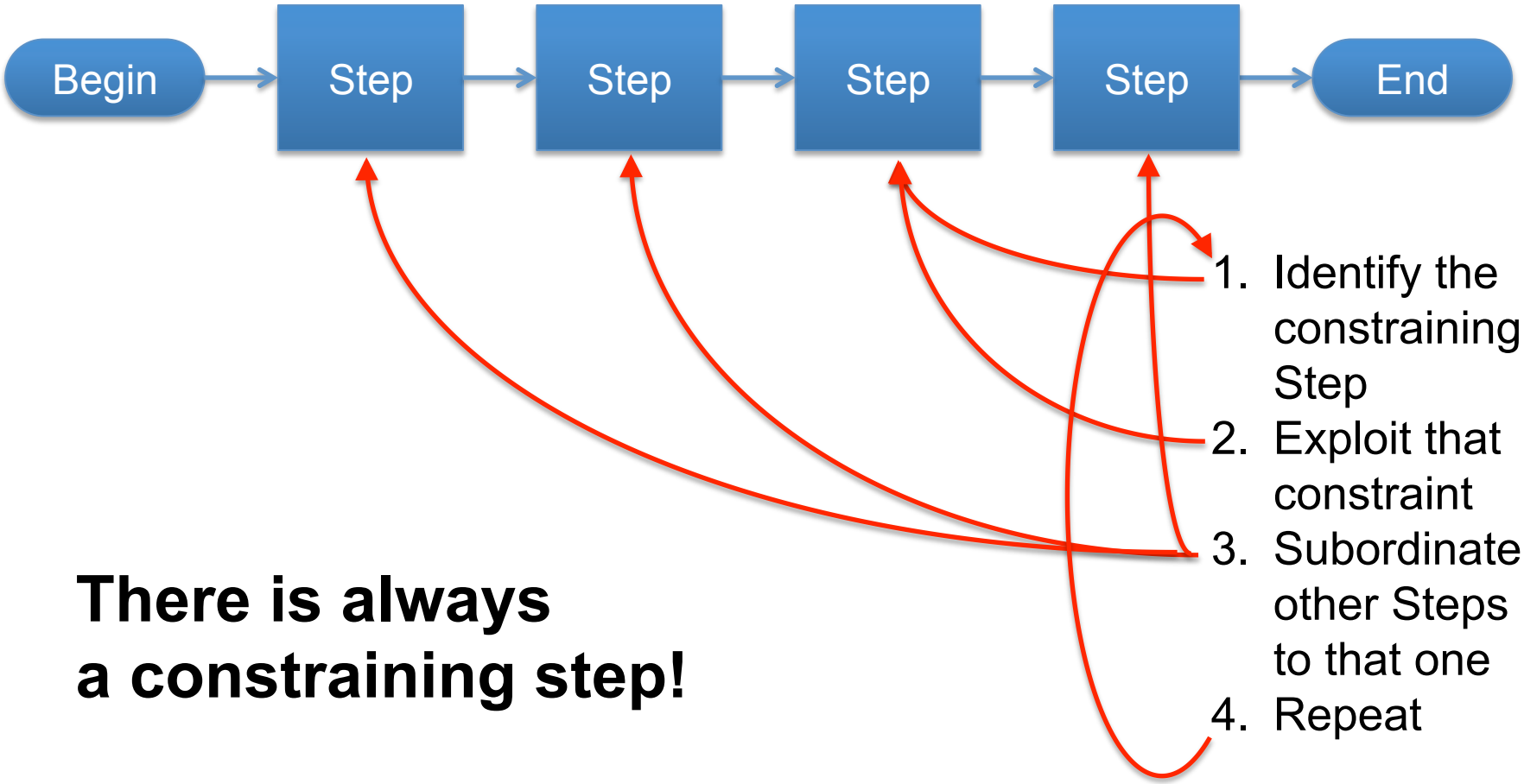
Benefits:

- Lower WIP
- Less variability in Flow
- Reduced Lead Time (Reduced Waiting)
- Better Quality (Reduced Rework)

Optimize the Process to Reduce the Number of Hand-Offs

- Handoffs
  - Increase Lead Time (waiting)
  - Reduce %CA (mistakes)
- Reduce Handoffs:
  - Cross-functional Teams
  - Automation

Continually Identify and Elevate Our Constraints



There is always a constraining step!

2. Infrastructure As Code

Topics and Tools

- Infrastructure Configuration Management
- The Deployment Pipeline
- Shared Version Control

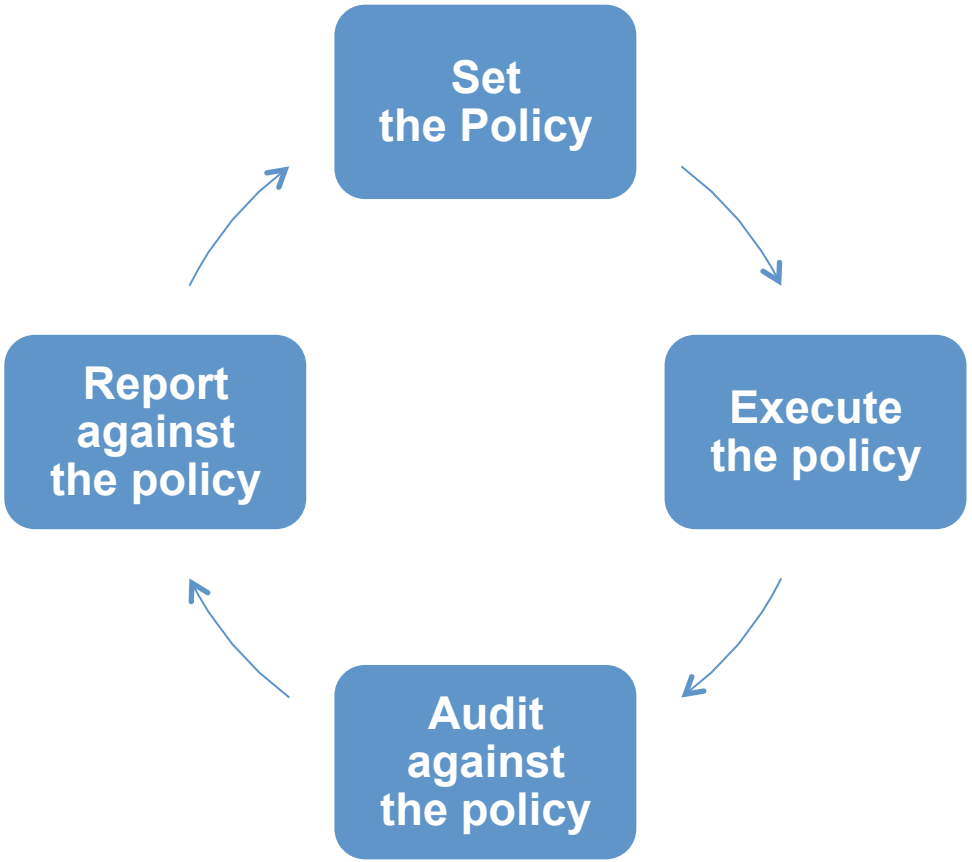
Infrastructure Configuration Management

- Automate configuration of systems & devices
  - Remove human error as a cause of incidents
  - Increase deployment velocity
  - Enable self-service for Dev teams
- Automate configuration verification
  - Establish & enforce standards
  - Audit and provide visibility
  - Automatically correct deviations

Configuration Management Tools

The big names

- Puppet
- Ansible
- Chef
- Salt

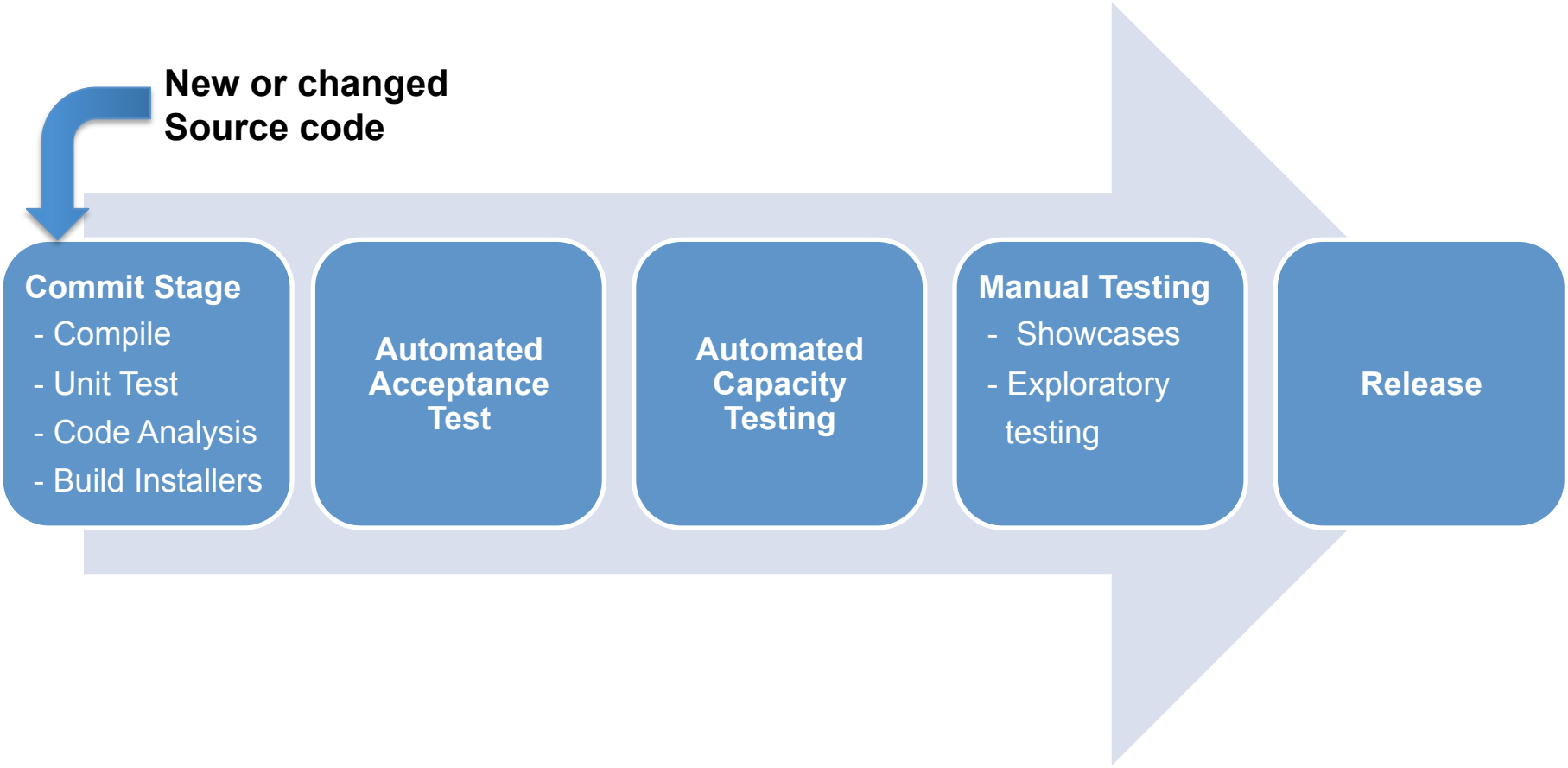


3. The Deployment Pipeline

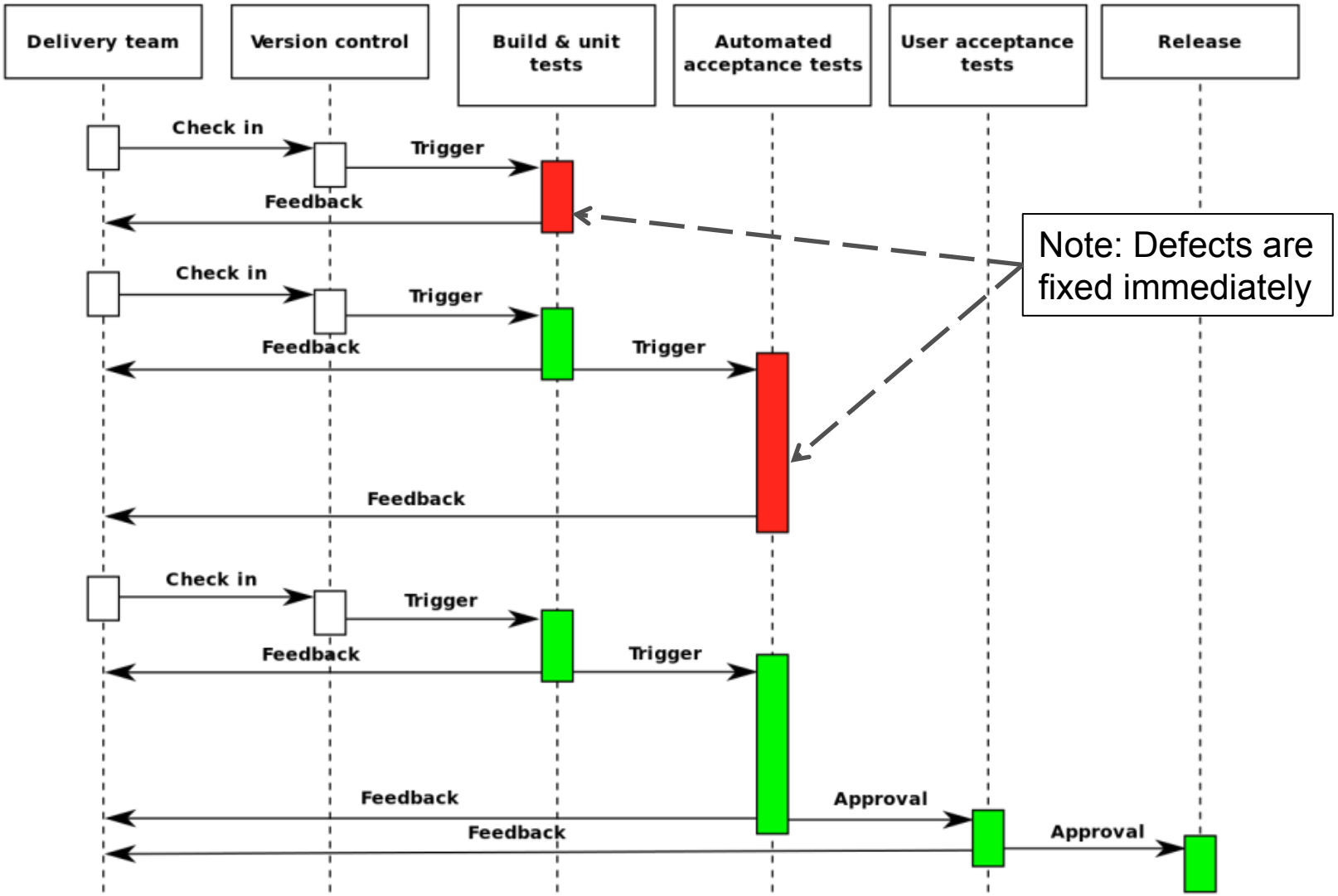
- The steps to build, test and deploy:
  - Application software
  - Systems or servers
  - Configuration updates

    e.g. firewalls
- Automating the Deployment Pipeline:
  - Improves Flow & velocity
  - Reduces human error
  - Enables needed testing

Deployment Pipeline Example Application Software



Deployment Pipeline Flow

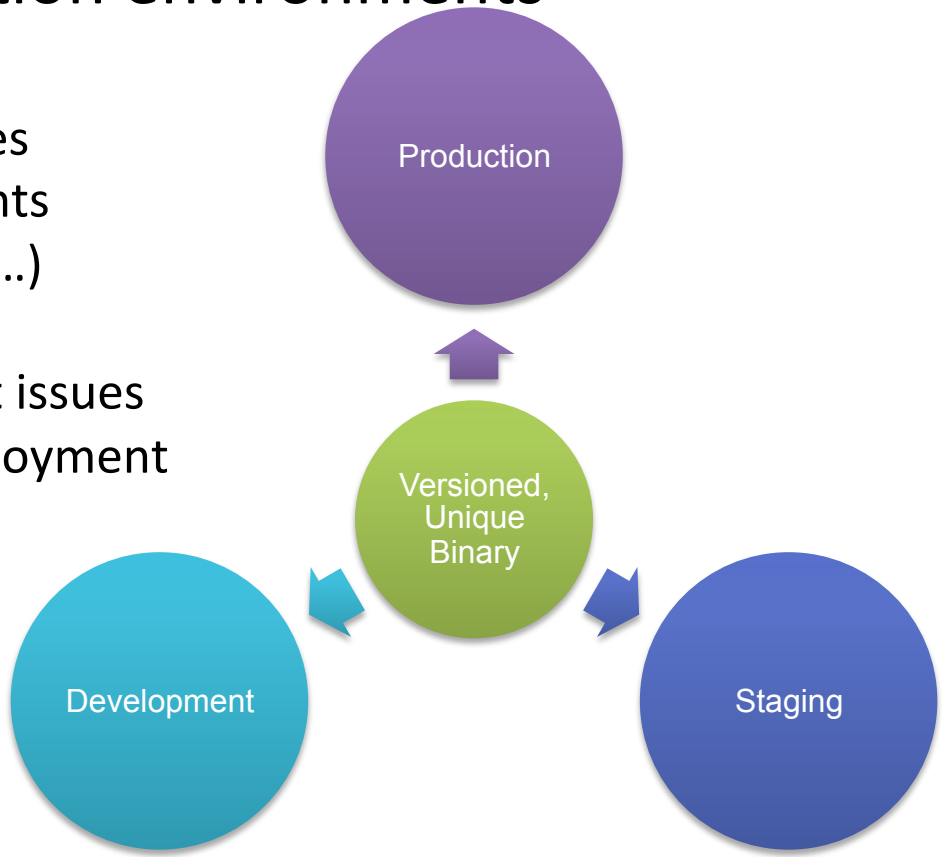


Build “Binaries” Once

“Binaries” = All files that are necessary to deploy into all test and production environments

Includes executables, scripts, config files

- Use the same files in all environments (development, staging, production …)
- More efficient than rebuilding
- Easier to troubleshoot environment issues
- Files tested many times before deployment



Antipattern:
Rebuilding for each environment (especially for production release)

- Less efficient
- Invalidates testing

Deploying Database Changes

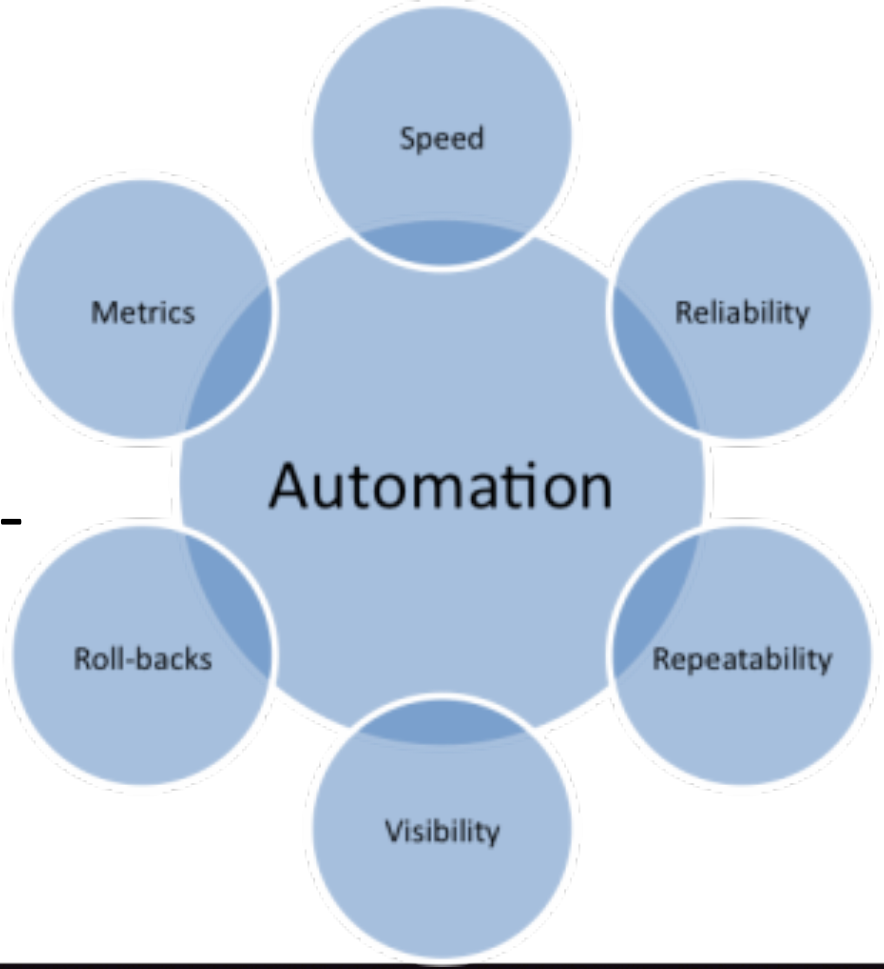
"Automate Everything You Can" includes Database Administration

- Scripts to update Database Schema
- Scripts to transform data
- Files & scripts to load new data
- Scripts to revert to the prior version

  or restore the prior version from backup

Automated Deployment Pipeline

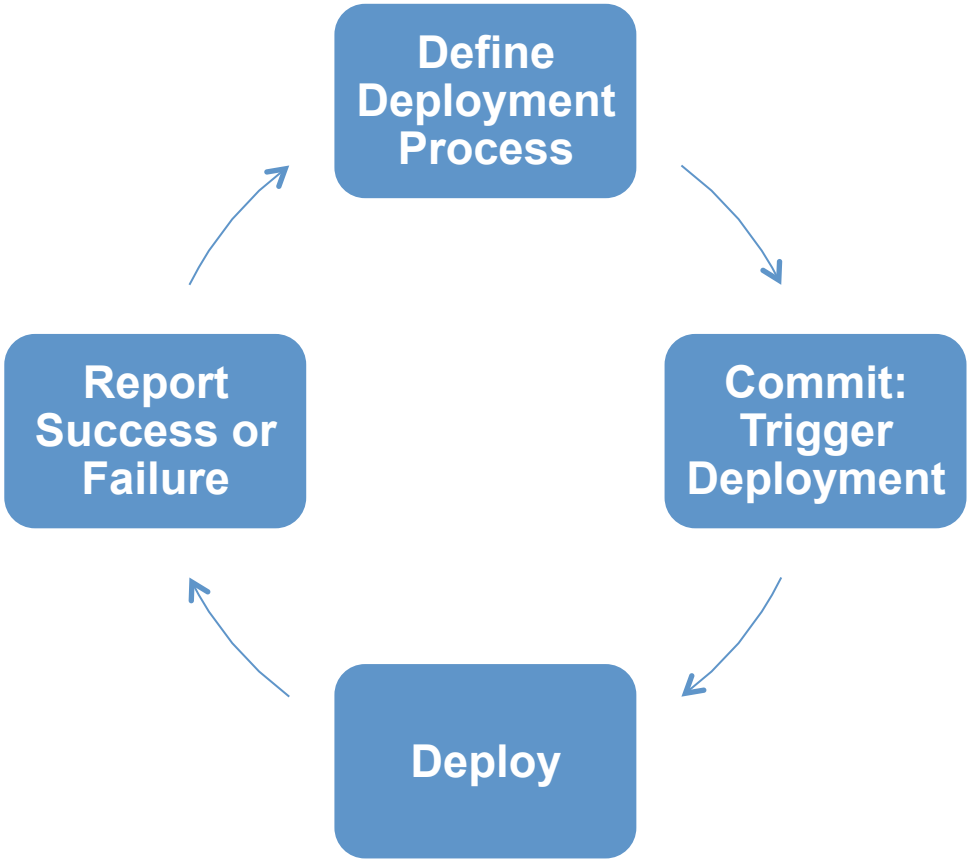
- Anyone can deploy
- New environments easily supported
- Less overhead
- More frequent deployments are possible
- More scalable
- Fewer errors



Deployment Tools

The big names

- Jenkins
- Capistrano
- TFS



4. Shared Version Control

- Single source of truth for all teams (Dev, QA, InfoSec, Ops)
- ***All*** artifacts in one repository

Application Software
- Source Code
- Test Scripts

Infrastructure Management
- Configuration Files
- Deployment Scripts

Database Administration
- Databases
- Update Scripts

Third Party Artifacts
- External Libraries
- Open Source Components

Do You Have Everything in Version Control?

- Can a new developer setup his/her workstation from one source?
- Can operations duplicate production systems?
- Can the testing team set up a production-like test environment?
- Can a business user create a demonstration for a potential customer?

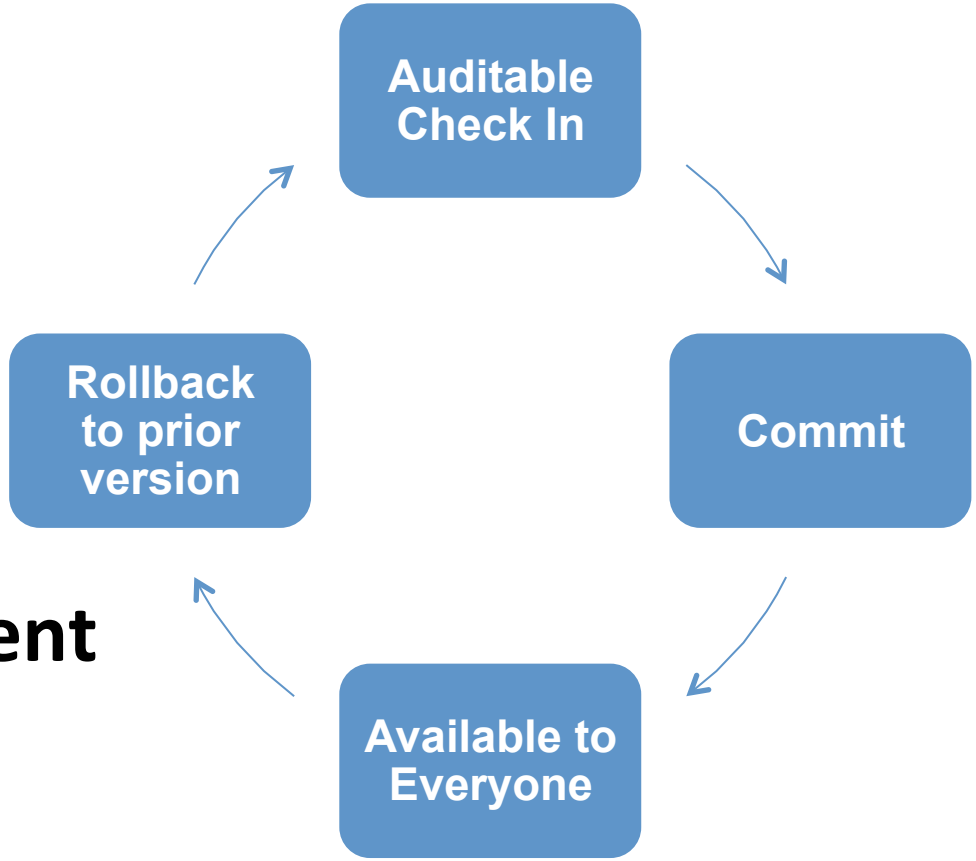
Version Control Tools

The big names

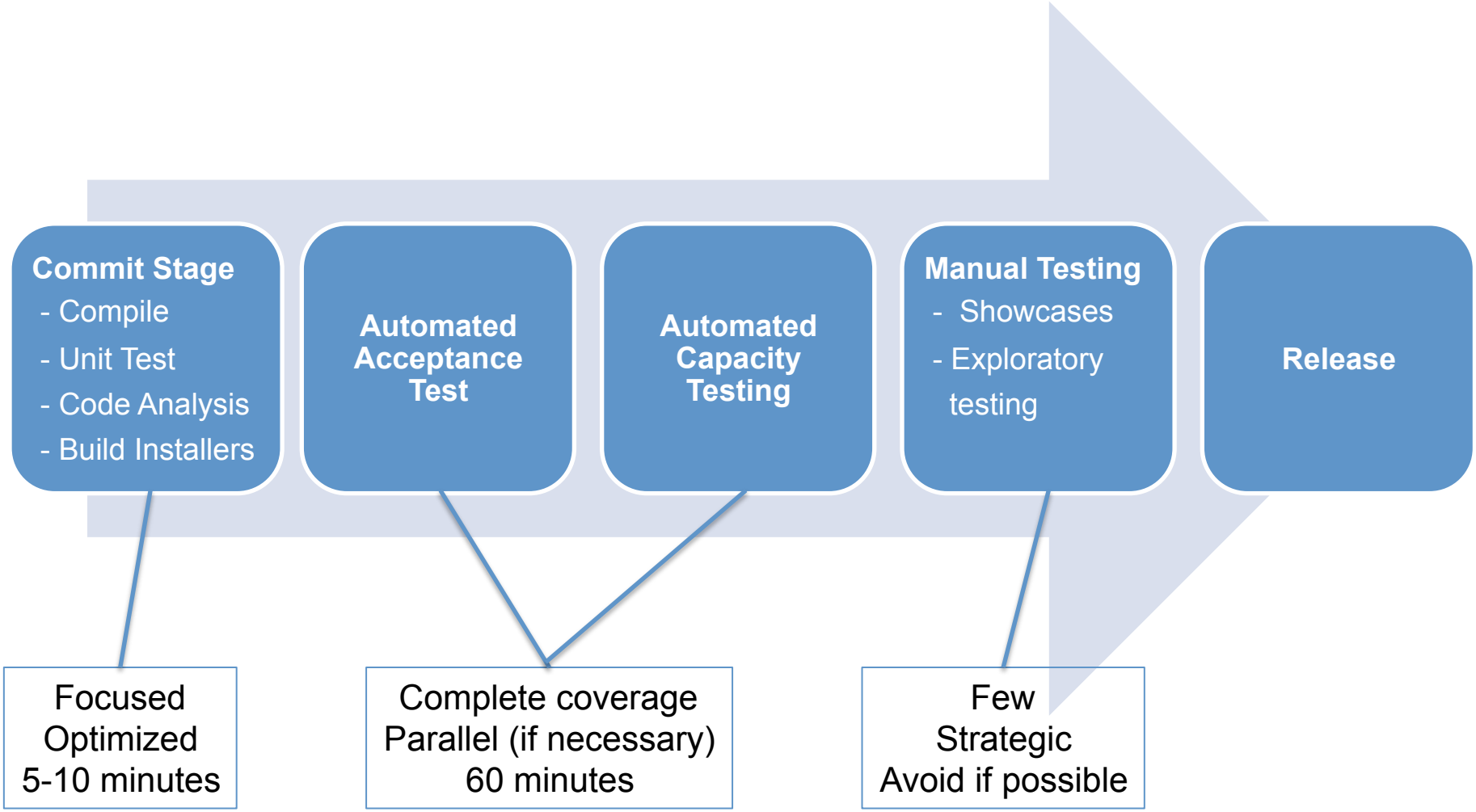
- Github
- Git
- TFS

Artifact Management

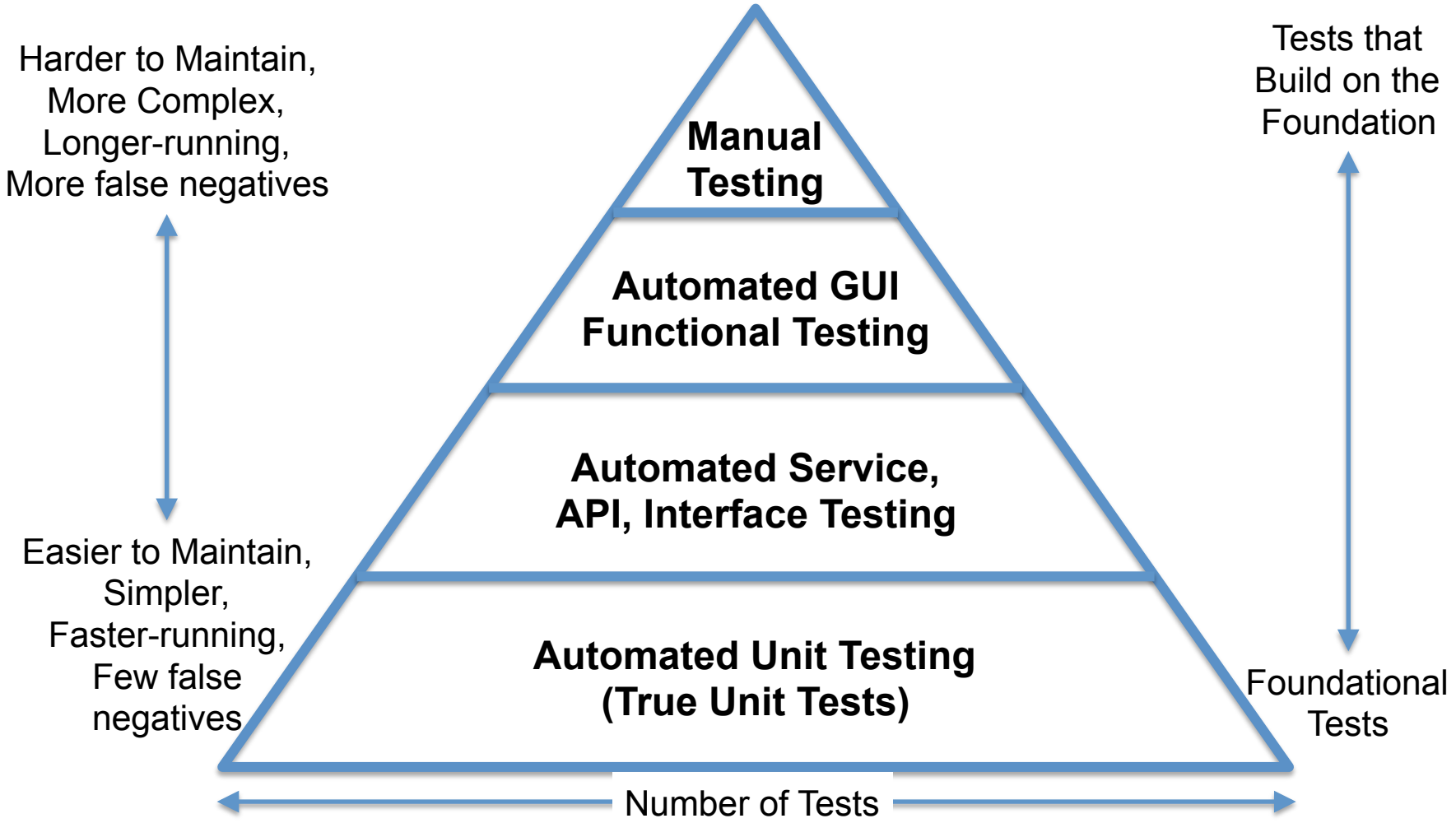
- Artifactory
- Sonatype



5. Automated Testing



Test Automation Pyramid



Strategies for Managing Test Data

Test Doubles

- Stubs: Returns fake values when a method is called
- Mocks: Set expected values and verify them

In-Memory Databases

- In-memory version of database per test
- Typically per developer and restricted by database type

Test Data

- Create database from Schema or backup
- Reinitialize Database to custom application settings

Self-Contained Tests

- Each test puts the database into a known state before running
- Helps avoid dependencies (tests that must be run in a specific order)

Automated Testing Tools

- Unit Testing
  - JUnit, NUnit, etc
- Functional, Performance Testing
  - HP Application Lifecycle Manager (ALM)
  - Cucumber
  - Selenium
  - Ranorex
- Security Testing
  - Penetration Test tools
  - Active analysis tools

Static Analysis Tools

- Language-specific
  - Python (Pylint & Pyflakes)
  - Java (PMD Java, CheckStyle)
  - C++ (Cppcheck, clangtidy)
  - Php (Code Sniffer, PHPMD)
  - Ruby (Rubocop, Reek)
  - SonarQube: 20+ languages
- Security Scanners
- Complexity Analyzers

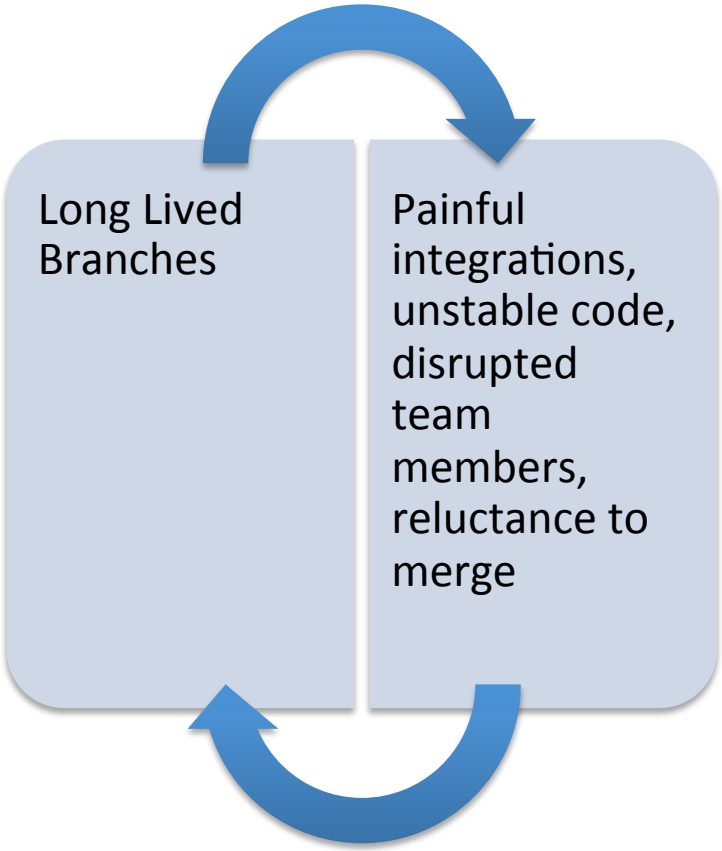
6. Continuous Integration

Is your team doing Continuous Integration?

- Do developers check into trunk
  at least once per day?
- Does every check-in trigger testing
  (unit and regression) and an automated build?
- If the build is broken or tests fail,
  is the problem resolved in a few minutes?

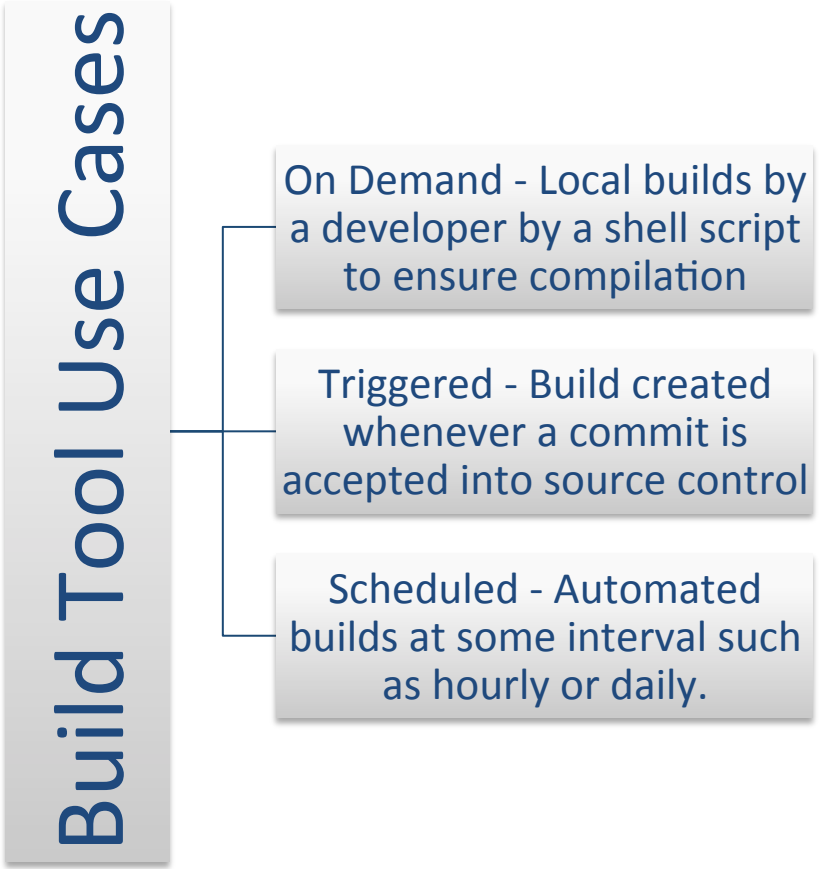
Check Into Trunk Often

- Develop on trunk (main line) means integrate every day
  - Must break down work into small commits
  - Use architectural techniques we will discuss soon
- Run tests after each change and quickly revert or fix changes that break build
  - Ensure software is always in a good state



Build Automation

- A developer can easily reproduce a build on his or her local machine.
- Dev team can schedule regular builds
- Anyone can trigger a build without developer help (e.g. Product Owner can build a demo version)



Build Automation Tool

maven.apache.org

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

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

7. Containerization

In Shipping	In IT
A system by which merchandise of any kind	A mechanism by which an application and its dependencies
is contained in a standard shipping container	are encapsulated in a software “container”
that can be attached to and transported on	that can be installed and run on
any transport modality (truck, train, ship, plane)	any computer (with any operating system)
that has the necessary connectors.	that has the necessary interface hooks.

Containerization Tools

- Docker
  www.docker.com

- Team Foundation Server (TFS)
  visualstudio.com

Container Orchestration Tools

- Docker Swarm

  www.docker.com/products/docker-swarm

- Kubernetes

  kubernetes.io

- Amazon EC2 Container Service

  aws.amazon.com/ecs

- Azure Container Service

  azure.microsoft.com/en-us/blog/azure-container-service-preview

- Red Hat OpenShift

  www.openshift.com

8. Architecture for Reduced-Risk Deployment

- Feature Flags (Feature Toggles)
- Microservices
- Blue-Green Deployments

What Are Microservices?

- Object-Oriented Design – OOD (1980's)
  - Design the application as a collection of objects
  - (Encapsulation, Information hiding, Tight cohesion, Loose coupling, SOLID principles – in notes)
- Service Oriented Architecture – SOA (2000's)
  - Take OOD the next step
  - Implement the application as a collection of Services
  - Each service is a separate executable object
- Microservices (2010's)
  - Take SOA to the extreme
  - Each service is small and simple

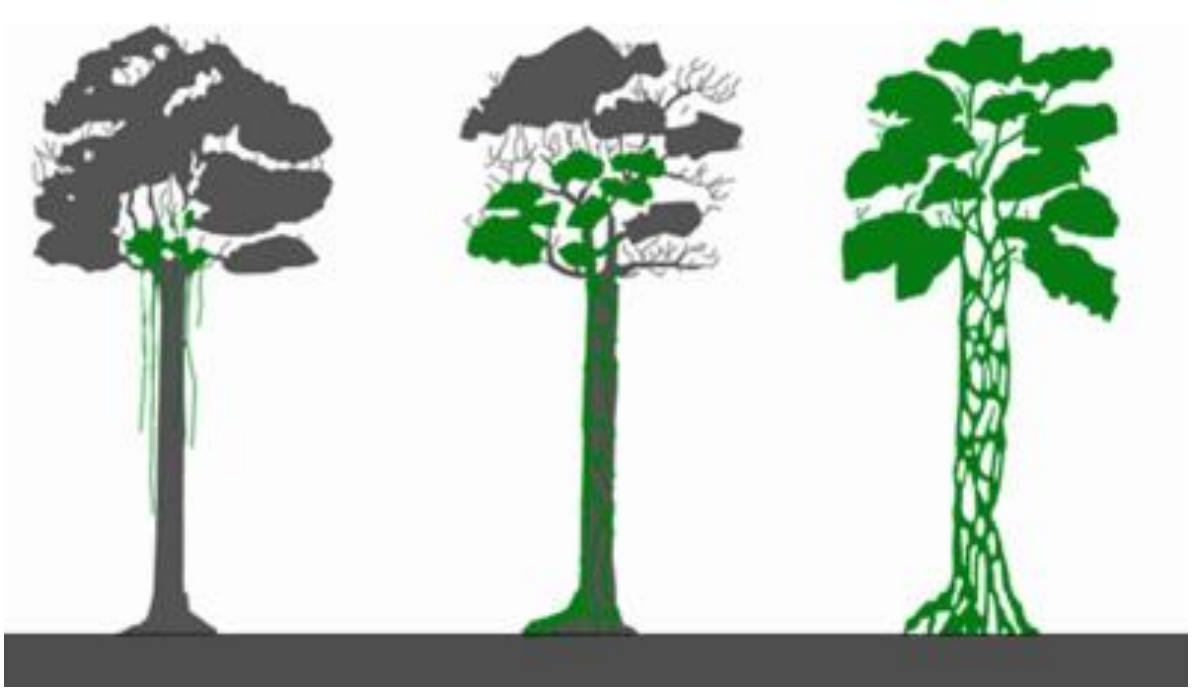
Why Use Microservices?

- Overall system resilience
  - Service failure does not kill the application
  - Service failure may be invisible to users
- Scalability
  - Replicate a Microservice to accommodate demand
- Observe Conway's Law
  - Small Agile Teams & Microservices

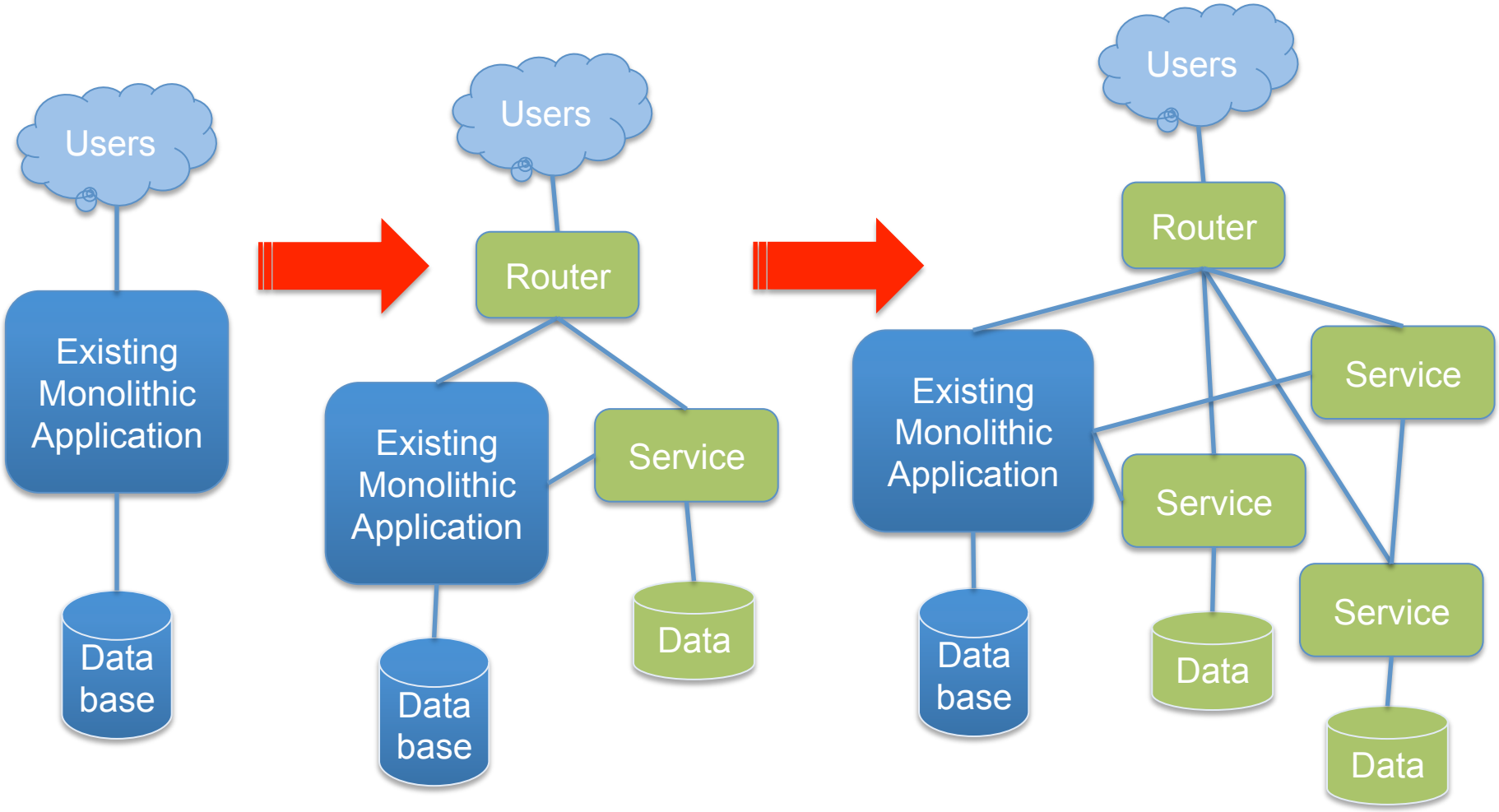
Re-architecting Applications Gradually
The Strangler Pattern

Why The Strangler Pattern?

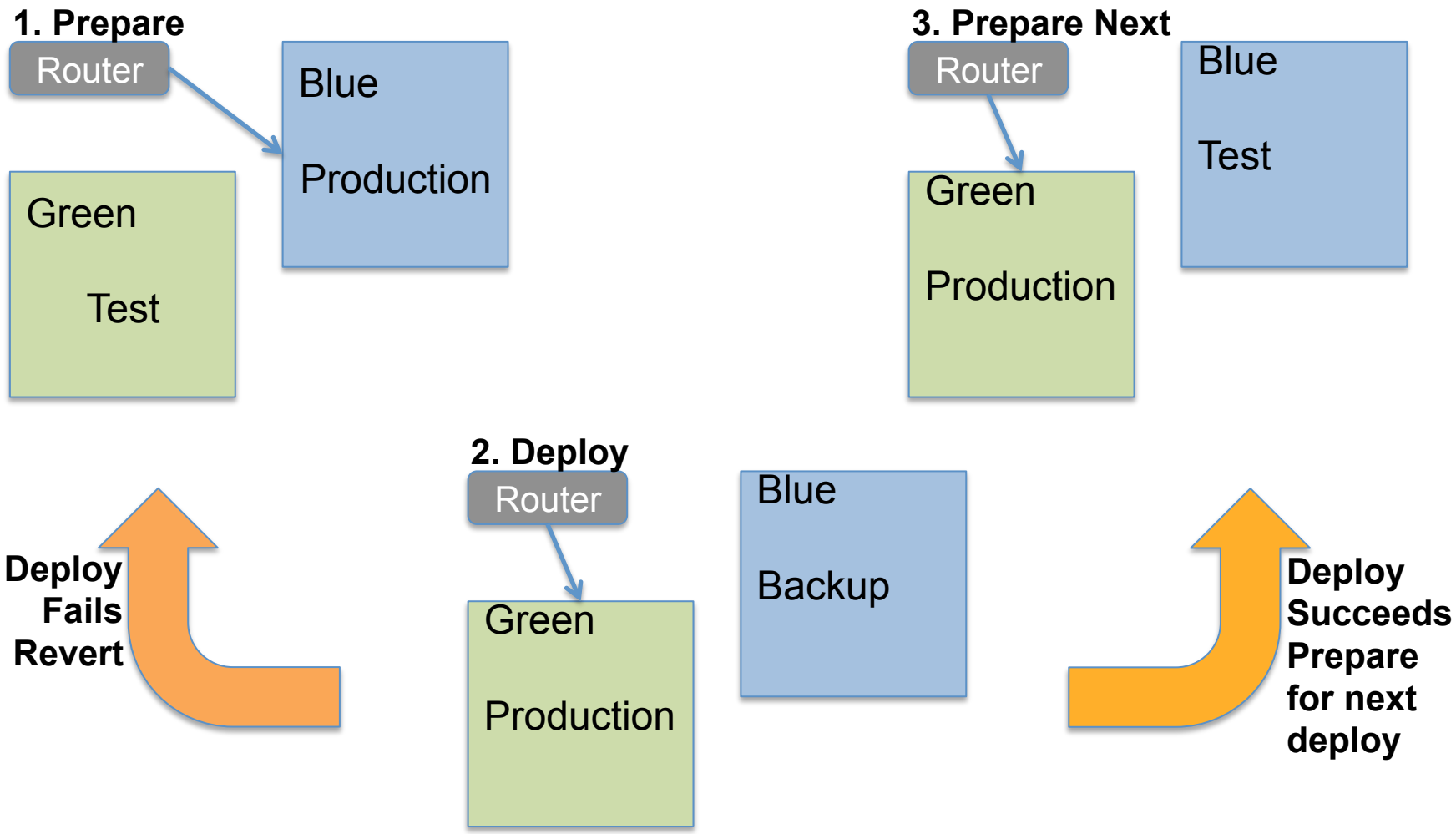
- Avoid risky application rewrite projects
- Start seeing the benefits of SOA quickly
- Begin moving toward smaller market-oriented teams



How the Strangler Pattern Works



Blue-Green Deployment Pattern

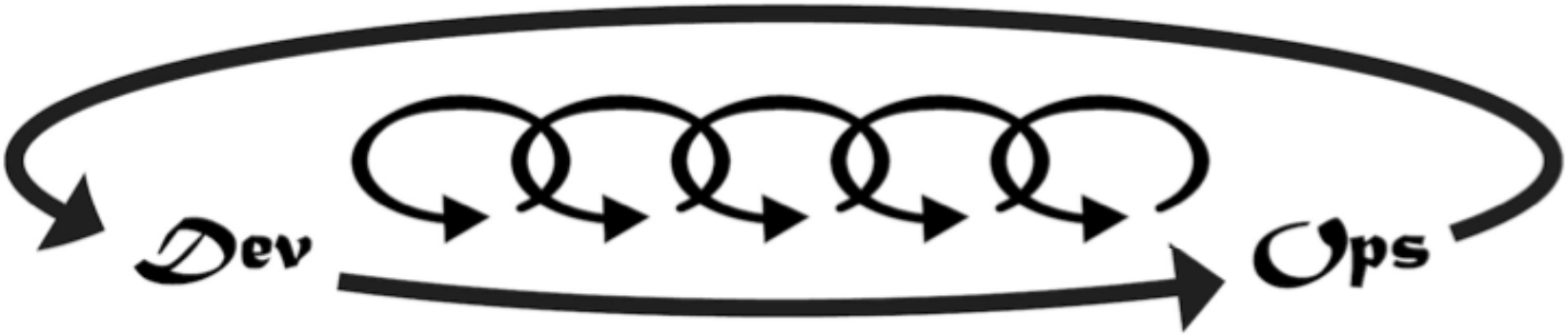


9. Change Review and Coordination

Principle: Those who are closest to the work understanding it best

- Management review = ineffective
- Peer Review = best
  - Technical review within team
  - Cross-team collaboration

Gene Kim's Three Ways



The First Way: Optimize Flow

The Second Way: Amplify Feedback

The Third Way: Continual Learning & Experimentation

Amplify Feedback

1. Principles of Feedback
2. Telemetry: Metrics, Monitoring and Alerting
3. Use Telemetry to Anticipate Problems
4. Feedback for Safe Deployment of Code
5. Hypothesis-Driven Development

1. Principles of Feedback

- See Problems As They Occur
- Swarm and Resolve Problems to Build New Knowledge
- Keep Pushing Quality Closer to the Source
- Optimize for Downstream Work Centers

See Problems As They Occur

- Fast feedback and feed-forward mechanisms
- Pervasive telemetry for constant visibility

Swarm and Solve Problems to Build New Knowledge

Toyota Andon Cord – Example here as well!

- Benefits of Swarming:
  - Fix the problem sooner when it is easier and less expensive to repair
  - Prevent the introduction of more errors
  - Organizational learning from problems

Keep Pushing Quality Closer to the Source

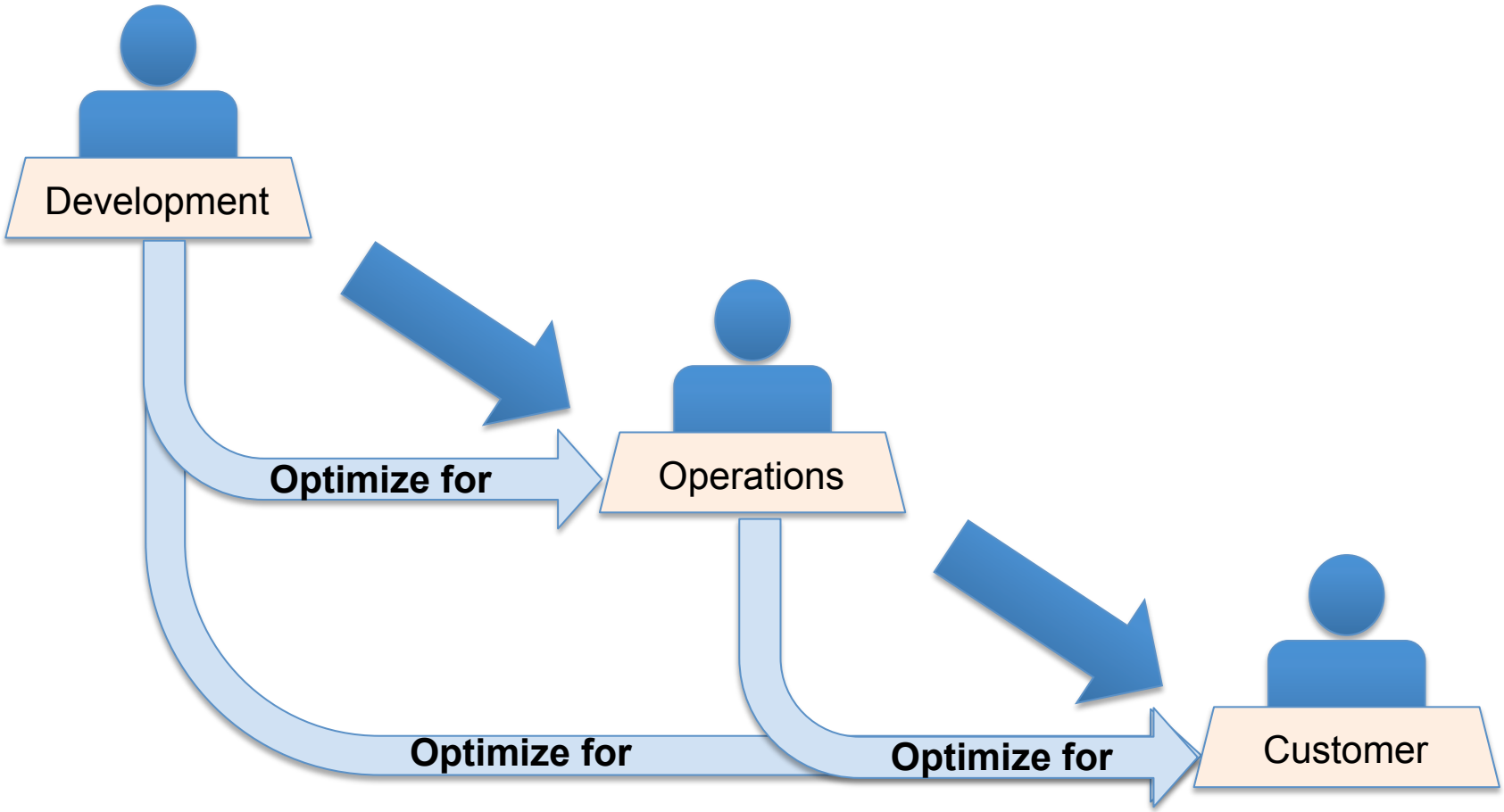
Inefficient/Ineffective

- Manual checks by a separate team that could be automated for the source team
- Approvals by distant authorities who lack detailed knowledge
- Detailed documentation that will quickly become obsolete
- Sending large batches to special committees & waiting for processing or approval

Efficient & Effective

- Enabling teams to find problems in their own work
- Use Peer Reviews
- Automate quality checks where possible
- Make Quality each person's job

Optimize for Downstream Work Centers



2. Telemetry: Metrics, Monitoring, Alerting

Telemetry = "an automated communications process by which measurements and other data are collected at remote points and are subsequently transmitted to receiving equipment for monitoring."

Event = a change in the state of something that is relevant

Metric = data that are captured because they are useful for detecting Events and understanding current state and history

Monitoring = watching metrics to identify Events that are of interest

Alerting = Notifying people when an Event requires action

Telemetry Principles

- Automate Telemetry
- Build Monitoring and Measurement in:
  - Production Environment
  - Development & Test environments
  - Deployment Pipeline
- Use alerts to boost efficiency
- Make information visible to all
  - Within IT and to Customers

Create Alerts

For every new or changed
system, service, application, server, device

- Identify Events that require intervention

  To prevent or respond to Incidents

- Automate Intervention where possible
- Create alerts to the appropriate people

  (where automated intervention is not possible)
  Dev, Ops, InfoSec

Review and Tune Alerts Periodically

- Identify “noise” alerts
  - Reduce, eliminate
    or route them to more appropriate people
- Identify failures
  that could have been prevented
  - Add alerts
- Automate Event response
  - Eliminate the need to Alert

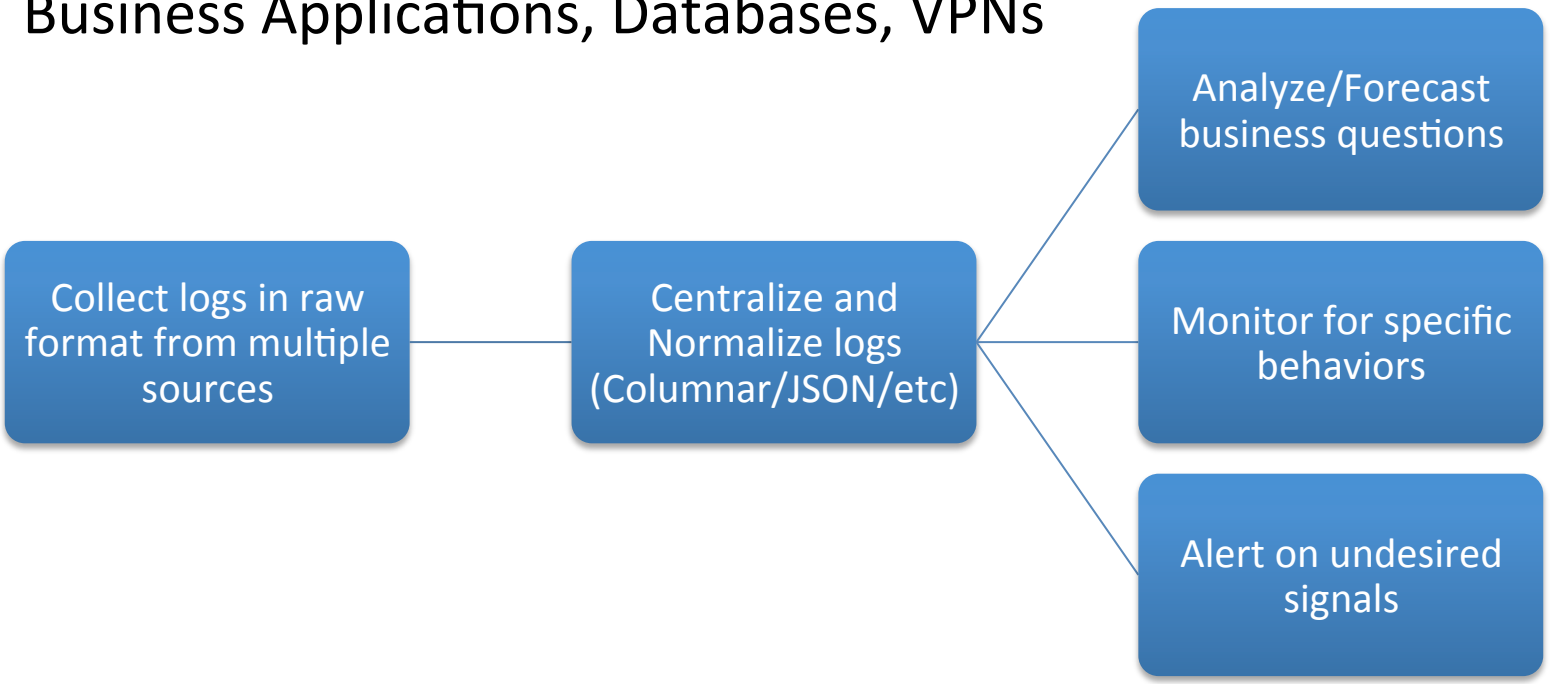
System Monitoring Tools

- Librato
- Monit
- Nagios
- PagerDuty

Log Aggregation

All logs – Audit records, Transaction logs, Intrusion alerts, Connection logs, User activity, Various alerts/Messages

From all sources – Firewalls, Routers/Switches, Web Servers, Business Applications, Databases, VPNs



Log Aggregation Tools

- Splunk
- ELK

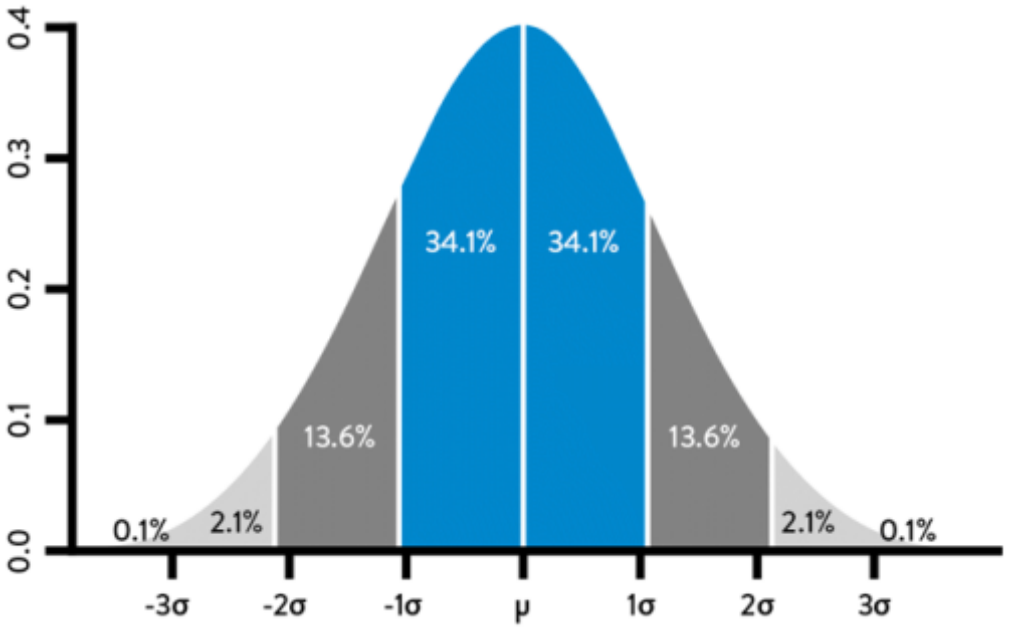
Common DevOps Metrics

- Number and frequency of software releases
- Volume of defects
- Time/cost per release
- MTTR (Mean Time to Recover)
- Number and frequency of outages / performance issues
- Revenue/profit impact of outages / performance issues
- Number and cost of resources

3. Use Telemetry To Anticipate Problems

Use statistical techniques to understand “normal”, then address the “abnormal”

- e.g. Mean and Standard Deviation



- Example: NetFlix (below)

Use Telemetry To Anticipate Problems

Monitor for (and alert on) trends

e.g.:

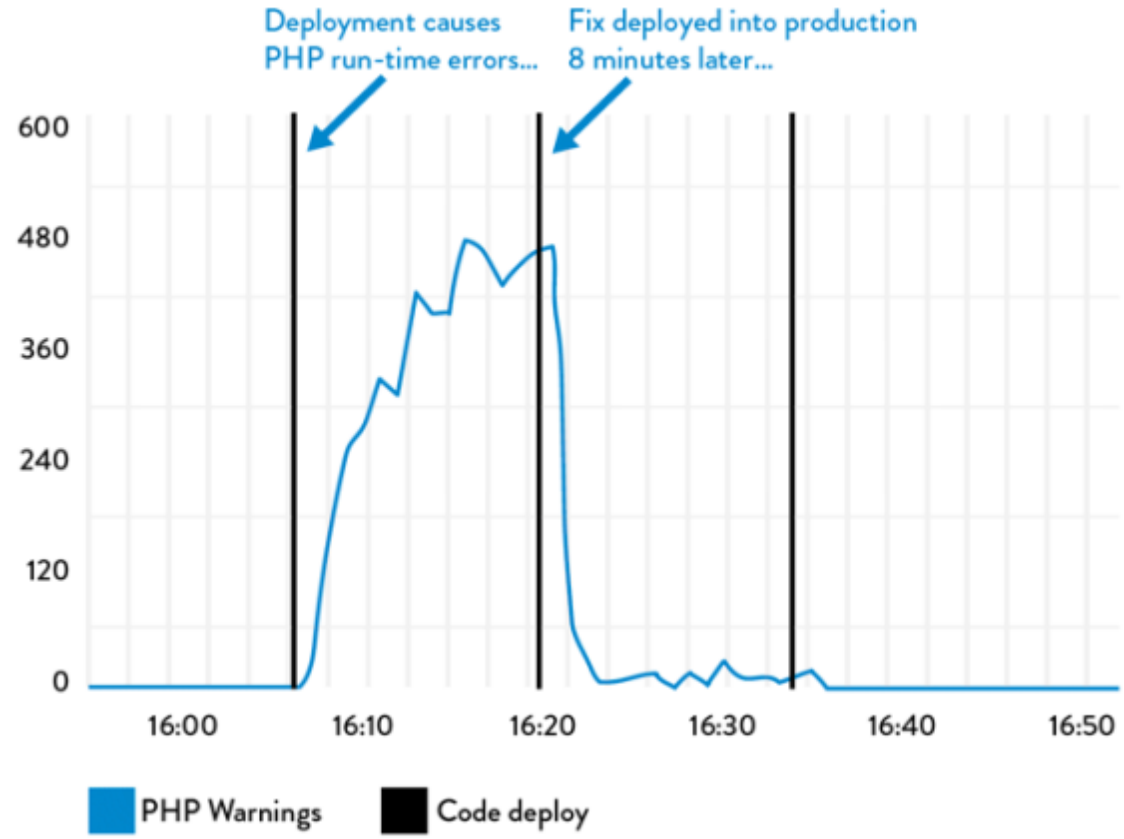
- Increasing web page load times
- Decreasing server free memory
- Decreasing available disk space
- Growing database transaction times
- Declining number of functioning servers behind the load balancer

4. Feedback For Safe Deployment of Code

Actively Monitor key metrics immediately before and after each deploy

- Take action on unexpected changes

Etsy.com Example:



5. Hypothesis-Driven Development

Backlog = Hypotheses not Requirements

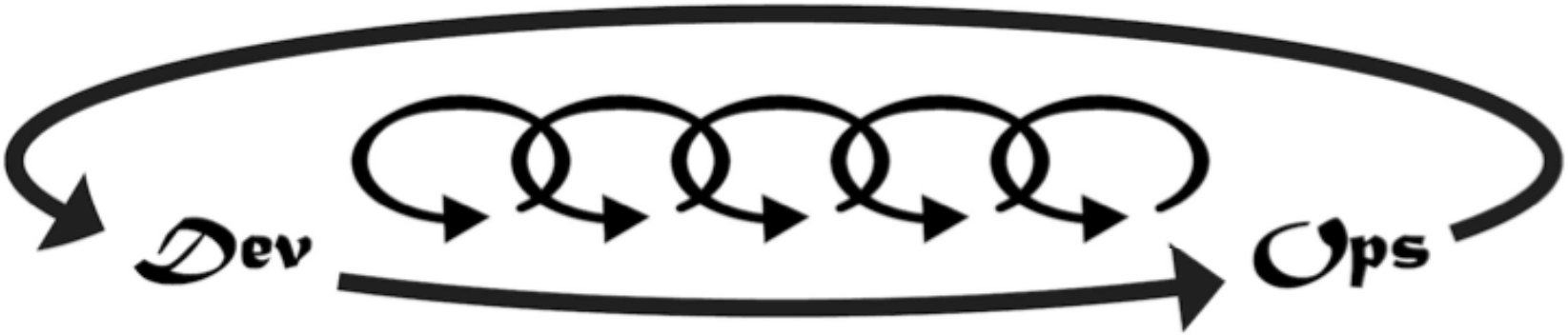
- Implement a quick experiment
- Monitor results in production
- Adapt the backlog based on results

A/B Testing

A type of Hypothesis-Driven Development

- Deploy both options (A & B)
- Randomly assign users to A or B
- Monitor results
- In future development,
  build on one,
  eliminate the other

Gene Kim's Three Ways



The First Way: Optimize Flow

The Second Way: Amplify Feedback

The Third Way: Continual Learning & Experimentation

Continual Learning and Experimentation

- **Learning Culture**
  - Blameless Postmortems
  - Enable Organizational learning and a safety culture
  - Reserve Time for Organizational Learning
  - Sessions, coaches, trainers, mentors
  - Institutionalize the Improvement of Daily Work
  - Transform Local Discoveries into Global Improvements
- **Innovation Culture**
  - Inject Resilience Patterns into Daily Work
  - Rehearse Failures
  - Special Days

Responses to Failure

Blaming Culture	Learning Culture
"Who caused the failure?"	"What caused the failure?"
Active self-protection	Active research
Information hidden	Information revealed
Find the culprit	Find the root cause
Correct the person	Correct the root cause
Learn to avoid responsibility	Learn to avoid failure

Blameless Postmortems

Hold the Blameless Postmortem
ASAP after incident resolution

Participants: Anyone who:

- May have contributed to the problem
- Identified the problem
- Responded to the problem
- Diagnosed the problem
- Was affected by the problem
- Wants to participate

Enable Everyone to Teach and Learn

- Lunch n Learn Sessions
  - New topics
  - Target topics of high business impact
  - Target biggest pain points
- Example from Nationwide Insurance: Teaching Thursday - Every employee spends 2 hours teaching or learning any topic of interest
- Attend/speak at public conferences - Bring back knowledge and schedule sessions to share it
- Hold Internal Conferences

  e.g. Nationwide Insurance, Capital One, Target

Establish Internal Consulting & Coaches

- Identify internal subject matter experts

  Make it part of their job to:

  - Coach
  - Mentor
  - Consult
- Formally establish an internal consulting or coaching group
  - Staff it with DevOps or subject matter experts
- Establish online communities of practice
  - People can post questions & get answers

Transform Local Discoveries into Global Improvements

Knowledge Sharing

Searchable repositories

- Postmortem reports

  (Blameless)

  - Problem Root causes
  - Resolution techniques
- Effective practices
- Problem avoidance techniques

Shared Artifacts

Common repositories

- Source code
- Test cases
- Scripts
- Configuration files

Knowledge Sharing

- Use Chat Rooms and Chat Bots
  - Chat Bot accepts commands in the Chat and posts responses & status
  - Chat log provides real time visibility and contains complete history
- Standardize via automation
  - Not written procedures in word documents!
- Everyone Shares one repository
- Document details in Automated tests
- Capture Ops requirements for use in all Dev projects
  - User Stories for operation and support (Telemetry, Troubleshooting)
  - Non-functional (Degradation, compatibility, performance)
- Identify technologies that are problematic for Ops
  - So Dev can migrate away from them

Inject Resilience Patterns into Our Daily Work

Normal daily work

(not in response to failure)

- Reduce Lead Times
- Increase test coverage
- Improve productivity
- Refactor for resilience
  - Systems & Applications
  - Processes & Methods

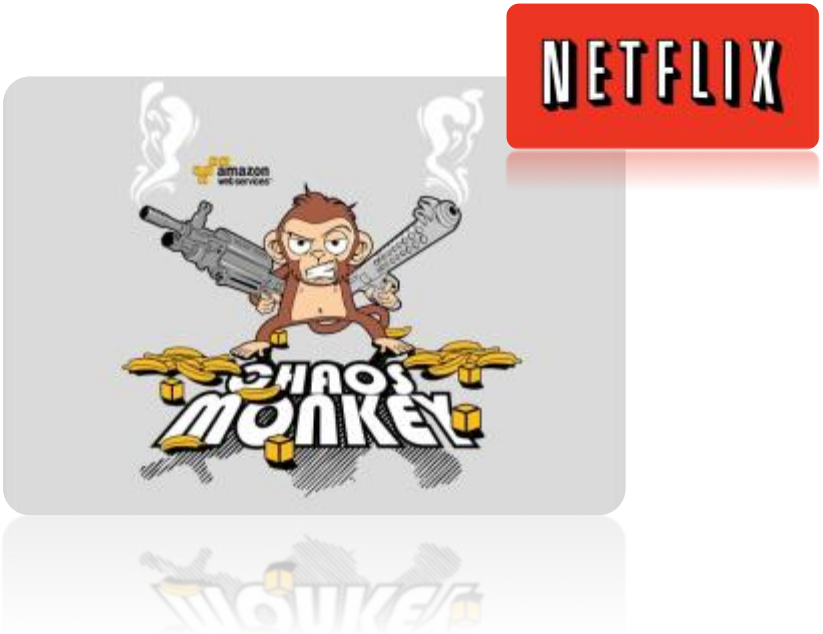
Special Exercises

"Game Day"

- Rehearse large scale failures
- Inject faults
  (e.g. Chaos Monkey)

Inject Production Failures to Continue Learning

- As your production resilience improves
- Keep pushing
- Consider your own Chaos Monkey! (or something like it)



Special days

On a Regular basis,
Schedule a few days to a week, e.g.

- Improvement (Kaizen) Blitz

  The Kaizen Blitz is a focused, short-term project to improve a process. Engage all people involved in a value stream to solve a known problem.

- Hackathons

  Try new things or
  experiment with new ways to do things

- Lab Weeks

Exercise 1.3 (Optional)
Optimizing Flow

- Identify specific steps your organization can take to Optimize Flow. Consider:
  - Infrastructure As Code
  - Infrastructure Configuration Management
  - Deployment Pipeline
  - Shared Version Control
  - Automated Testing
  - Continuous Integration
  - Architecting for Low-Risk Deployments
  - Change Management

Exercise 1.4 (Optional)
Amplifying Feedback

Identify specific steps you can take to amplify Feedback in your team

Consider:

- Telemetry: Metrics, Monitoring, Alerting
- Using Telemetry to anticipate problems
- Deployment Monitoring
- Developers following Applications downstream
- Hypothesis-Driven Development and A/B Testing

Exercise 1.5 (Optional)
Innovation Culture and Learning Culture

Identify way in which your IT shop could be made more innovative and have more learning culture

Consider:

- Blameless Postmortem
- Risk-taking
- Failure injection (e.g. Game Days)
- Ways to share knowledge
- Scheduled teaching and learning opportunities
- Internal consultants or coaches