While we're waiting to start...

What **unexpected issues** related to pipelines, continuous integration, and continuous delivery have you found?

Please write yours down, **one per sticky note**, and put on our Surprises wall chart.

We'll try to address them during the workshop.

Using Pipelines to Bring Product to Production

Abby Bangser and Lisa Crispin
European Testing Conference 2018

Why pipelines?

Some statements we have heard about pipelines:

- *Pipelines are “technical”, not business facing, they deal with automation and code*
- *“DevOps” teams / people are in charge of pipelines*
- *Pipelines are simple and execute the testing, they do not need testing themselves*

Instead, we prefer these statements about pipelines

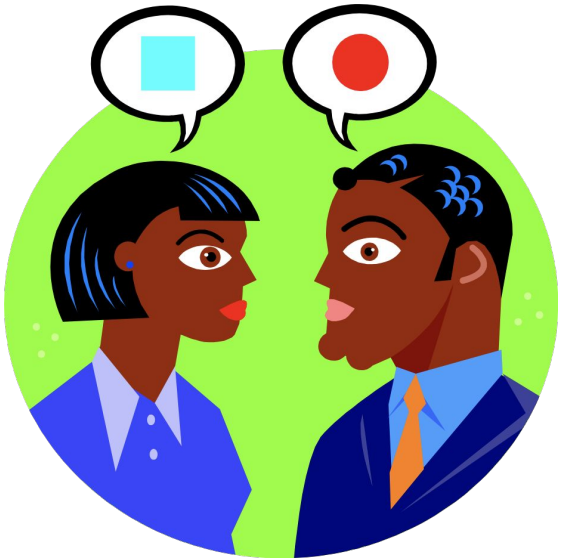
- Teams benefit from cross role pipelines engagement
- Value streams rely on ability to identify and fix bottlenecks
- As a key to risk mitigation, pipelines require analysis and testing to validate

Learning intentions

- Explore continuous integration and continuous delivery concepts
- Better define what we can expect from pipelines
- Learn vocabulary to engage with pipelines as a practice within your team
- Hands on experience layering in feedback loops

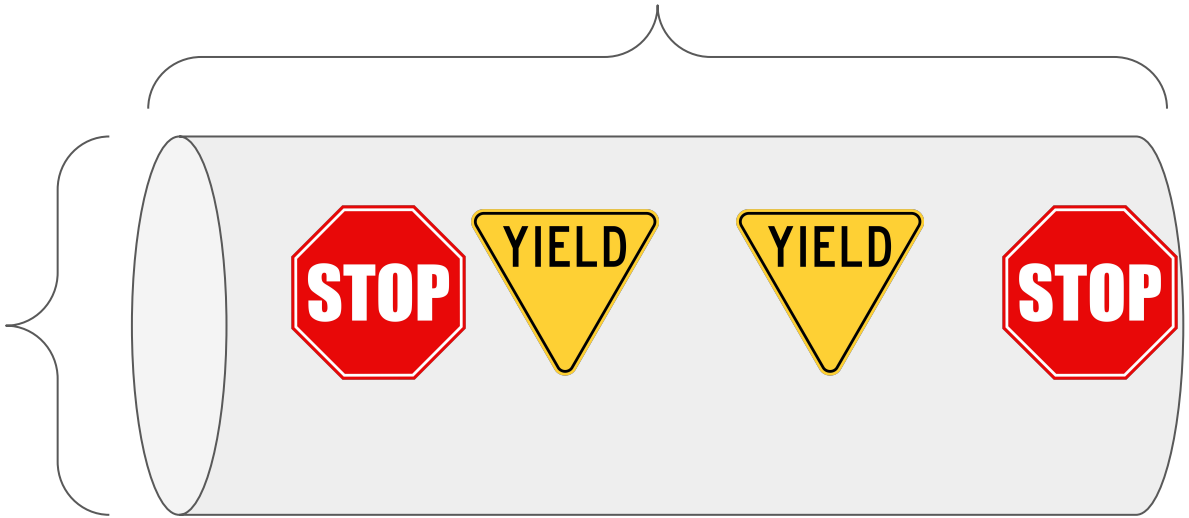
Clear up some terminology

- Make sure we share a common language across the team
- Avoid common misunderstandings
  - Throughput and latency
  - Continuous Integration
  - Deployment Pipeline
  - Continuous Delivery
  - Continuous Deployment



Latency is the amount of time to get through a pipeline line including any introduced delays

Throughput is the number and size of items that can be sent at any one time through a pipeline



How throughput and latency actually manifest

If a pipeline has **HIGH latency**, people will attempt to overcome these challenges by **increase BATCH SIZE of throughput**.

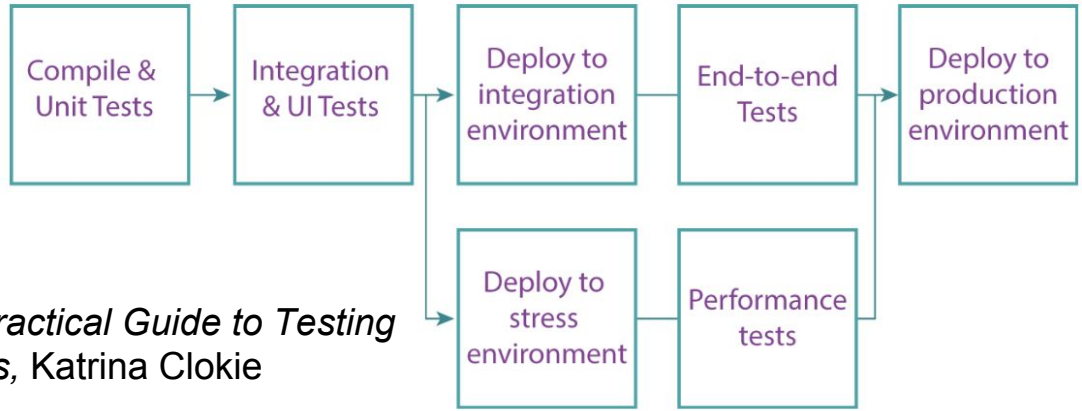
A pipeline with **LOW latency**, will encourage people to **decrease BATCH SIZE** and **increase FREQUENCY of throughput**.

Continuous Integration (CI)

- Integrate code into a shared repository multiple times per day
- All code is integrated onto the same branch multiple times per day
- Typically the start of a pipeline
- Each check-in can be verified by an automated build with automated regression tests

Deployment pipeline

- Break the build into stages to speed up feedback
- Each stage takes extra time & provides more confidence
- Early stages can find most problems -> faster feedback
- Later stages probe more thoroughly
- Automated deployment pipelines are central to continuous delivery

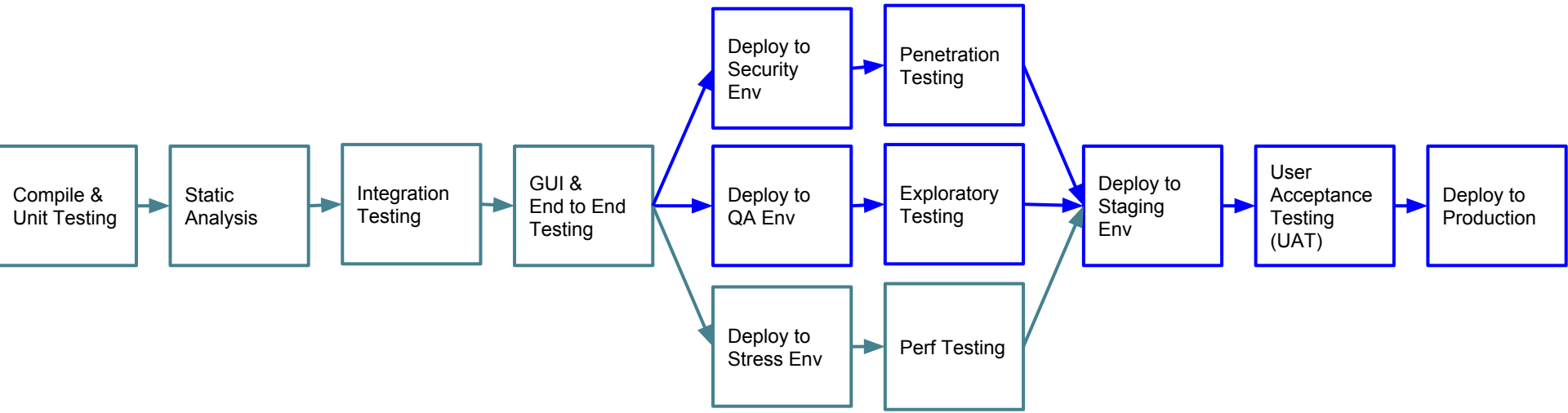


From *A Practical Guide to Testing in DevOps,* Katrina Clokie

Continuous Delivery (CD)

- Ability to get many types of changes into production safely, quickly and sustainably
  - eg. new features, configuration changes, bug fixes experiments
- Heavily benefits from automated testing but is not an explicit dependency
- Each commit is independently verified as a deployable release candidate
- A deployable release candidate is always available

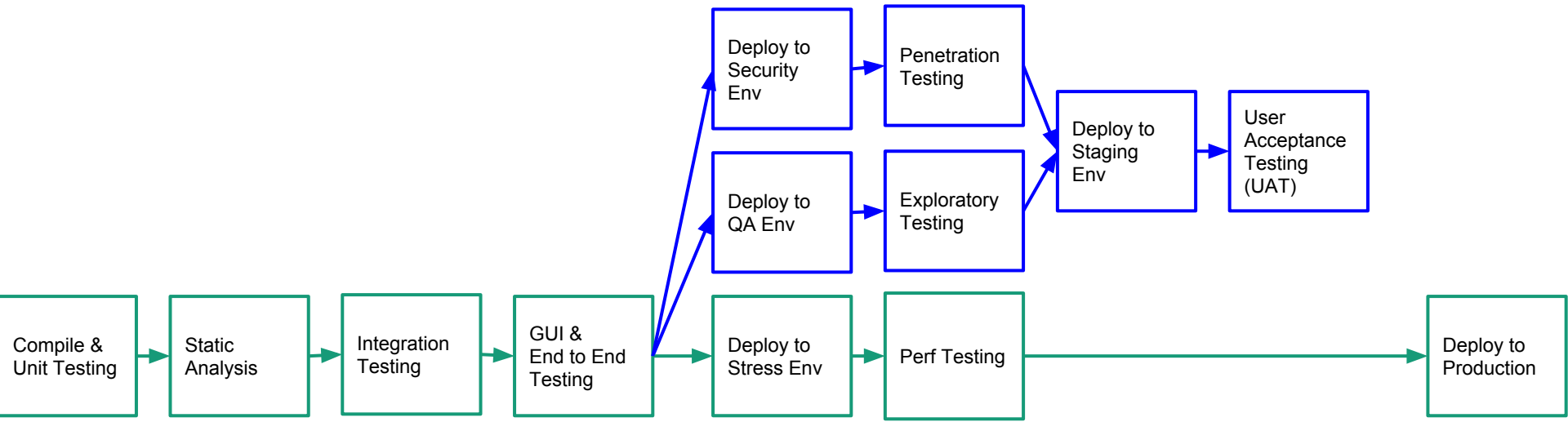
Example Continuous Delivery



Continuous Deployment (CD *…also*)

- Each commit is independently verified as a deployable release candidate and given it passes all verifications is automatically deployed to production
- Again, heavily benefits from automated testing and Continuous Delivery environment, but does not actually require either
- Deployments occur on every successfully verified commit. Often many a day.

Example Continuous Deployment



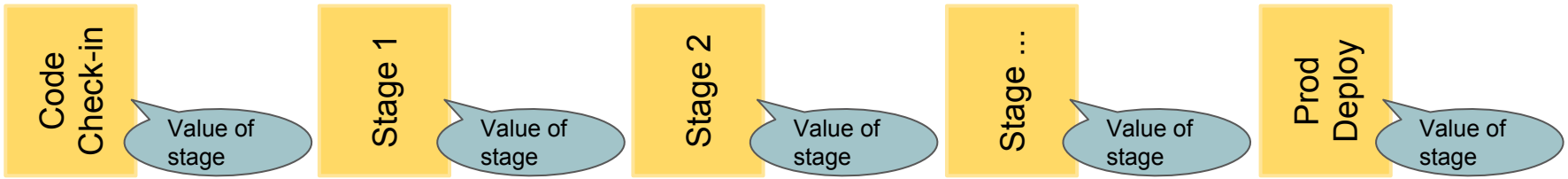
Activity time

Building the **first** layer of a pipeline

code change is committed → **a production release**

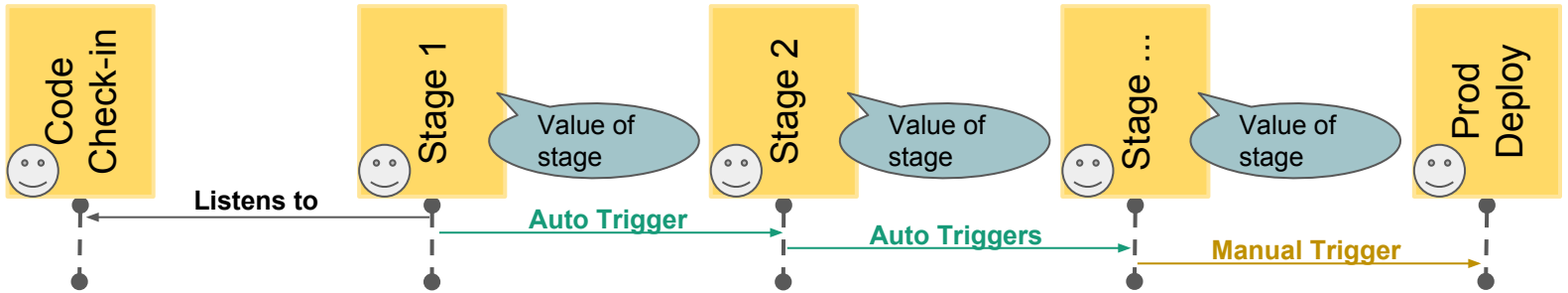
In your table groups, each person select a predetermined step and in turn (using stickies for notes)

- Place your step in order on the group's chart
- Identify what value and information the step can provide in its outputs



In your table groups, each person select a predetermined step and in turn:

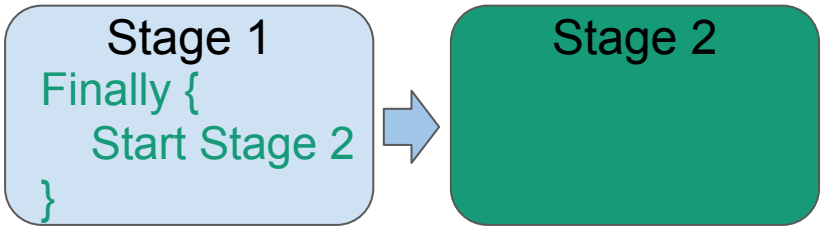
- Place your step in order on the group's chart
- Identify what value and information the step can provide in its outputs
- **Add in who on the team benefits from that information**
- **Add to or change the the drawing to reflect your answers to:**
  - **Could it be speeded up with parallelization?**
  - **Will the step be started automatically? Manually? Based on the state of another step?**



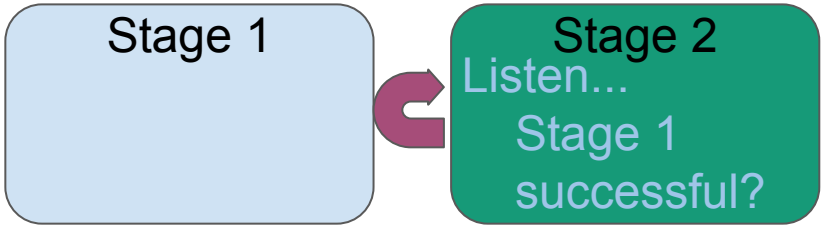
Some vocabulary to go with these ideas

Push vs Pull -

Asking a successful stage to kick off the next is pushing



Giving a stage prerequisites to poll (check periodically) for or to listen to a stage is pulling

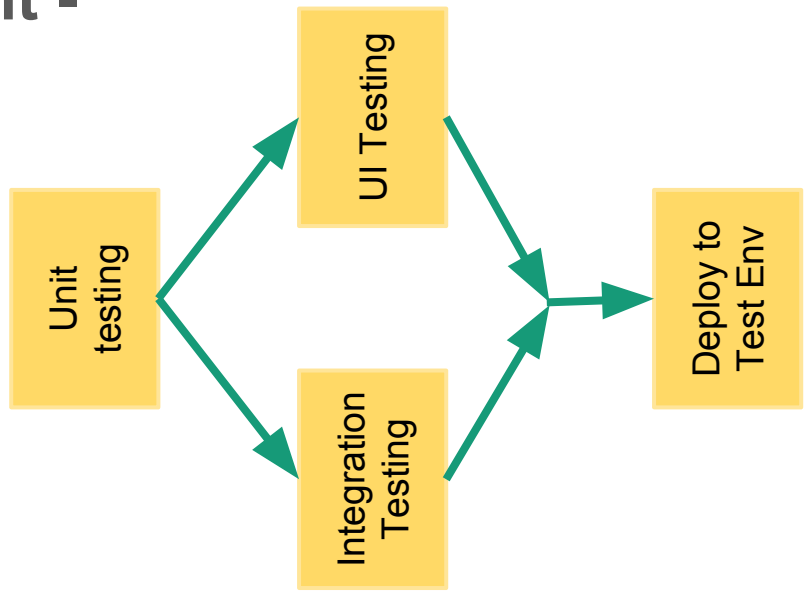


Something to think about: Push can fail a stage for reasons associated with the next stage. Pull can silently not work correctly if checking the wrong thing.

Some vocabulary to go with these ideas

“Fan in” Dependency Management -

Something to think about: if only one of the Prerequisite stages passes, do you run the shared step?



Building a **second** layer of value for a pipeline

Let's dig into why we chose those stages for our pipeline.

Let's explore the  *that we have created*

In your table groups, each person select a pipeline step and in turn share:

- How long is it until feedback (pass or fail etc) is provided from each stage
- Add details about who might be alerted and how an alert may be communicated or displayed

Some vocabulary to go with these ideas

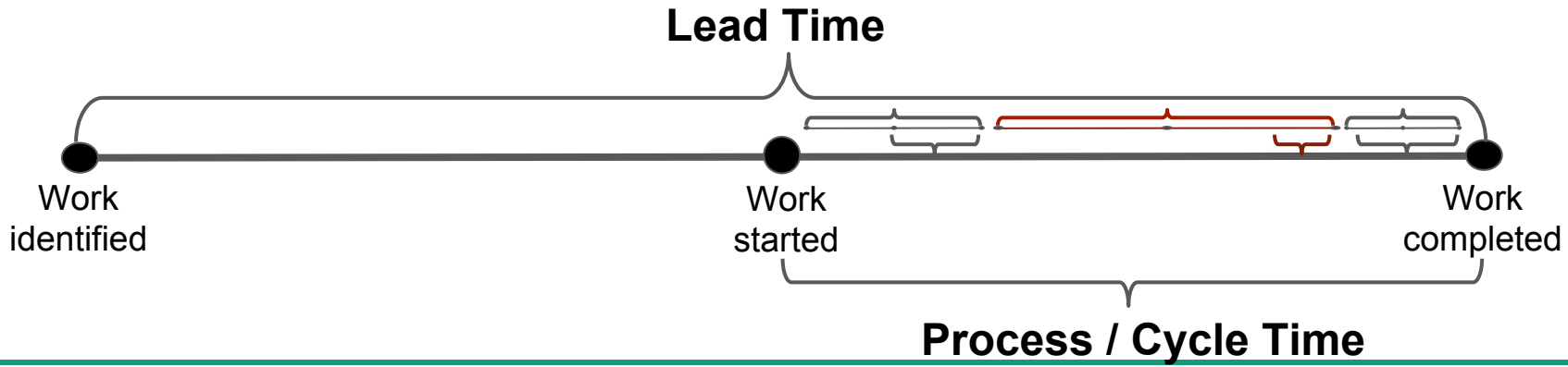
“Stop the line” mentality - from Toyota

- Every employee on the assembly line has a responsibility to “stop the line” when they see a defect
- Pushing the “big red button” is an investment that leads to improvements:
  - Knowledge sharing
  - Cost, speed
  - Reliability

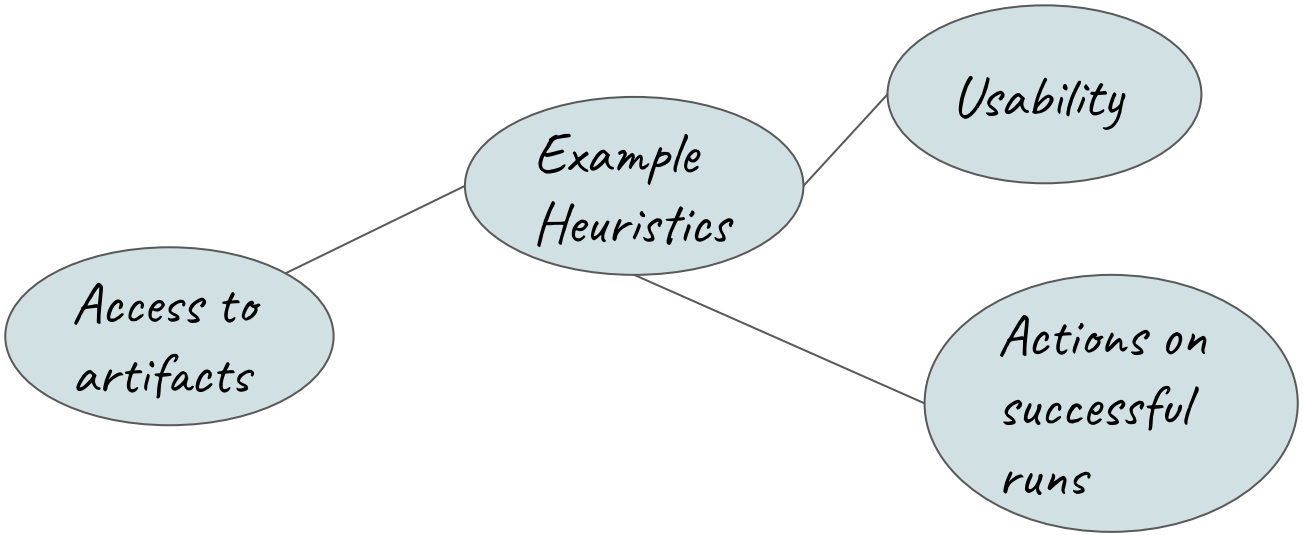
Some vocabulary to go with these ideas

“The First Way” - Improving the fast left-to-right flow of work. Encourages focus on breaking constraints which includes finding the *right* constraints to break first.

Note: This is a part of the Three Ways as introduced in The DevOps Handbook.



How we can test a pipeline and its infrastructure



Design your experiment!

How will you use or change your team's pipeline?

- Set a SMART goal - how will you measure whether your experiment is working
- It's ok to fail! Focus on learning
- Pair up to encourage each other, report on your outcomes

Feedback for a chocolate :)

Your Feedback Matters

We would like to know whether you found this workshop valuable and how you plan to use what you learned. Please take a few minutes to fill in this feedback form.

1. My overall learning experience was (circle one): Great / Good / Unsatisfactory

2. I learned things that will help me in my job (circle one): Many / Some / None

3. I got the most value from the following part:

4. I got the least value from the following p

Agile Testing and More Agile Testing

Save 35%* off the books or ebooks

- eBook formats include EPUB, MOBI, & PDF

Agile Testing Essentials video

Save 50%* on *Agile Testing Essentials LiveLessons* Video Training

Use code **AGILETESTING** at informit.com/agile

*Discount taken off list price. Offer only good at informit.com and is subject to change.

informIT the trusted technology learning source

Pearson

Thank you!

Let’s keep the conversation going...

@a_bangser

@LisaCrispin

Glossary: **A - C**

Blue/green deployment. Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any given time, one of the environments serves all production traffic, while the other is idle or accessible only by internal users for testing purposes. At deployment time, the new version can be deployed to the “idle” environment, tested, and then the router is switched to send all production traffic to the new production environment.

Canary release. A technique to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users before rolling it out to the entire infrastructure and making it available to everybody. (Danilo Sato)

Continuous delivery (CD). The ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, *safely* and *quickly* in a *sustainable* way. (Jez Humble)

Continuous deployment. Every change goes through the pipeline and automatically gets put into production, resulting in many production deployments every day. (Martin Fowler)

Continuous integration (CI). A development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. (Thoughtworks) Production deployments are frequent, but not every change results in automatic production deployment.

Glossary: **D - M**

Dark launch. The practice of deploying the very first version of a service into its production environment, well before release, so that you can soak test it and find any bugs before you make its functionality available to users. (Jez Humble)

DevOps. The term was coined in 2009 from “Development” and “Operations”, but the concept goes much further back. Developers, testers, operations staff and others collaborate to build and maintain build, test and production infrastructure that enables them to improve their customers’ lives.

Feature flag (or feature toggle). A configuration option that defines whether or not a feature within your software should be executed. You might also hear this concept called feature flags, flippers, switches, feature bits, or latent code. (Katrina Clokie)

Latency is the amount of time to get through a pipeline line including any introduced delays.

Logging. Log files record transactional and status information, along with errors and warnings that are generated by unexpected activity. (Katrina Clokie) They provide detailed, low-level information to diagnose the problems.

Monitoring. The process of maintaining surveillance over the existence and magnitude of state change and data flow in a system. Monitoring aims to identify faults and assist in their subsequent elimination. (Katrina Clokie)

Glossary: **O - Z**

Observability. The instrumentation you need to understand what's happening in your software is available. Observability focuses on the development of the application, and the rich instrumentation you need, not to poll and monitor it for thresholds or defined health checks, but to ask any arbitrary question about how the software works. (Charity Majors)

Pipeline. A repeatable, recorded communication of automated feedback. It includes test and deployment scripts, from development and operations respectively, along with a pipeline to illustrate the process by which the scripts run. Also known as automated deployment pipeline. (Katrina Clokie)

Staged rollout. A canary release with a different focus. Instead of creating a canary by limiting changes to infrastructure, the rollout intentionally limits the number of users with access to the new code. (Katrina Clokie)

Throughput is the number and size of items that can be sent at any one time through a pipeline.

Reading list: **Internet resources**

“Architecting for Continuous Delivery”, Vishal Naik, https://www.thoughtworks.com/insights/blog/architecting-continuous-delivery “...the goal of CD is to be able to release software frequently, and reliably, in a frictionless manner.”

“Continuous Testing”, Jez Humble, https://continuousdelivery.com/foundations/test-automation/ “Get started by building a skeleton deployment pipeline—create a single unit test, a single acceptance test, an automated deployment script that stands up an exploratory testing environment, and thread them together. Then increase test coverage and extend your deployment pipeline as your product or service evolves.” *Also from Jez: https://continuousdelivery.com/*

“Incremental Steps to Continuous Releases”, Maaret Pyhäjärvi , http://visible-quality.blogspot.com/2017/05/incremental-steps-to-continuous-releases.html “...this fundamentally changes the testing I do. It enables me to test each change, isolate it and see its impacts all the way through production.”

Reading list: **Internet resources (continued)**

"Continuous Testing in DevOps", Dan Ashby, https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/
"For me, testing fits at each and every single point in this (DevOps) model."

"What Happens to Testing in Continuous Delivery", Fanny Pittack, https://www.slideshare.net/traumstoff/what-happens-to-testing-in-continuous-delivery

"Testing in Production: Rethinking the Conventional Deployment Pipeline", Jacob Winch, https://www.theguardian.com/info/developer-blog/2016/dec/20/testing-in-production-rethinking-the-conventional-deployment-pipeline

https://dojo.ministryoftesting.com/lessons/the-tester-s-survival-guide-to-joining-a-continuous-delivery-project-amy-phillips (behind the paywall for the Dojo – but worth paying for!)

https://dojo.ministryoftesting.com/lessons/human-pipeline-optimize-your-feedback?s_id=4472 (behind the paywall for the Dojo – but worth paying for!)

Reading list: **Books**

Humble, Jez, and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test and Deployment Automation*, Addison-Wesley, 2010.

Clokie, Katrina, *A Practical Guide to Testing in DevOps*, LeanPub, 2017.

Kim, Humble, Debois, Willis, *The DevOps Handbook*, IT Revolution Press, 2016.

Reading list: **Continuous Integration Tools**

Circle CI: https://circleci.com/docs/2.0/workflows/ (docs)

Concourse: concourse.ci/introduction.html (docs) & ci.concourse.ci/teams/main/pipelines/main?groups=develop (example)

Jenkins: https://ci.jenkins.io/job/Core/ (example) & https://jenkins.io/doc/tutorials/ (docs / tutorial)

Jenkins with BlueOcean pipelines: https://jenkins.io/projects/blueocean/ (docs)

Identifying **additional** layers of value for a pipeline

An often underutilised capabilities of pipelines is how to build auditability into the delivery process.

For example:

- Traceability of who makes changes (commit messages are made in a certain format and must match credentials committing)
- Segregation of duties on who can push to prod (done through authorisation)
- Proof of code quality (through static analysis and/or automated test packs)
- Example - API doc auto generated
- Commit messages - who did the commit, who signed off on it
- Living doc - can be shown to auditors
- Who needs the outputs
- How to create outputs
- “Build once deploy everywhere”? (for trust of outputs)