Build Your Own DevOps Roadmap

Lisa Crispin, Mike Hrycyk, Rob Bowyer
With material from Christin Wiedemann, Abby Bangser, Ashley Hunsberger & Lisi Hocke

It takes a village…

@lisacrispin, @robbowyer, @qaisdoes, @c_wiedemann

Copyright 2018 Lisa Crispin

A little about us...

yvrTesting
Vancouver, BC · 809 members

@lisacrispin, @robbowyer, @qaisdoes, @c_wiedemann

How about you?

How do you primarily identify yourself? Please go stand by the sign that best describes you - or make your own!

- Tester/QA
- Developer/coder
- Manager
- Business analyst
- Coach
- Other roles?

Learning intentions

- DevOps and continuous delivery concepts
- Common terminology
- Questions to help you & your team build a DevOps culture
- Test suite canvas to help design your deploy pipeline
- Use pipelines for feedback, mitigating risks, answering questions
- Create your own DevOps roadmap

Expectations

- Please ask questions and contribute your stories!
- There are no “best practices”, there are “leading practices”.
- Read more, and experiment with these ideas to really learn them.
- DevOps is a big area. We’ll focus on pipelines.

Schedule

9:00 Start

10:30 Break

12:00 Lunch

1:00 Start afternoon

3:00 Break

5:00 End

Your stories & questions always welcome!

Building the roadmap

- You have a blank map to fill out as we go through the day
- As you learn today, fill in terms as we cover them, and check off items relevant to you and plan your path
- This will be your action plan after you get back to your daily work
- Think about where YOU want to go!

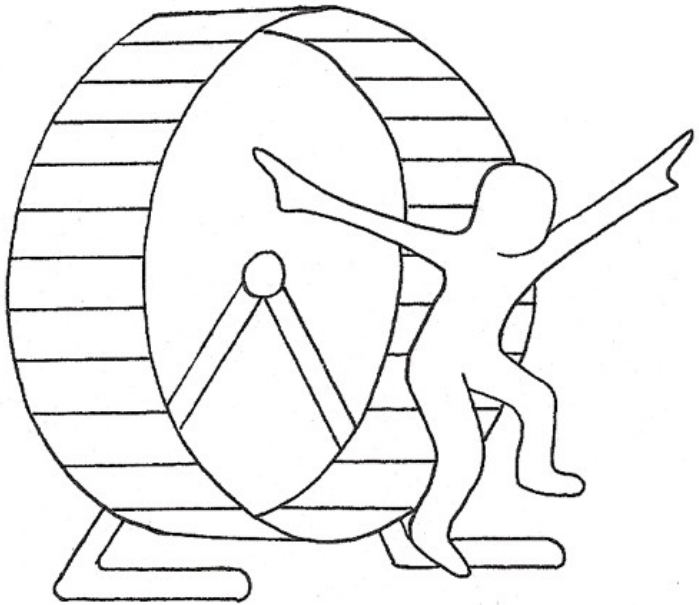
What’s blocking your road to DevOps?

- Write down challenges you want help with today, 1 per sticky note
- Go around your table, each person briefly introduce their challenges
- Put the sticky notes on your wall chart. Group similar topics.
- Dot vote to choose your 3 top topic areas - each person gets 3 votes
- Be ready to share your chosen topics with the large group

What do you think when you hear “DevOps”?

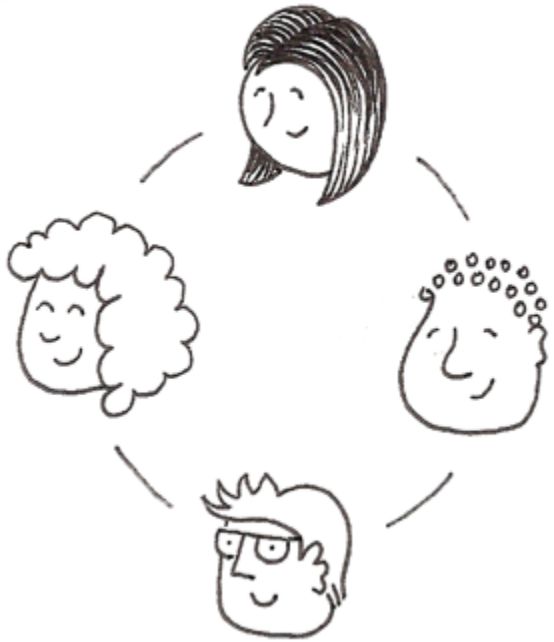
How about “Continuous Delivery”?

What do you think when you hear “Continuous testing”?



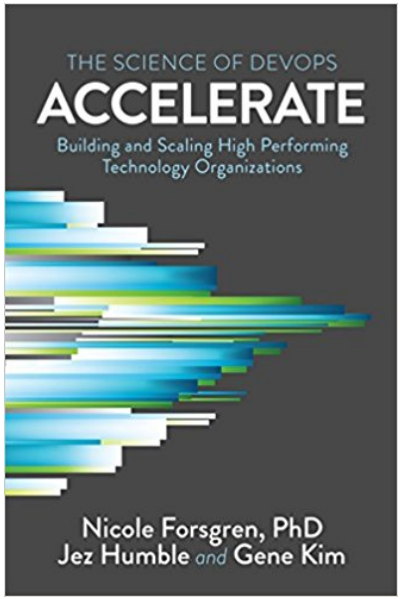
Whole team responsibility for quality

- Commitment to a level of confidence
  - Bug prevention over bug detection
  - Learning from production use, errors
  - …and responding fast
  - Focus on what's valuable to customers
- Diverse perspectives, skill sets, biases



“DevOps”

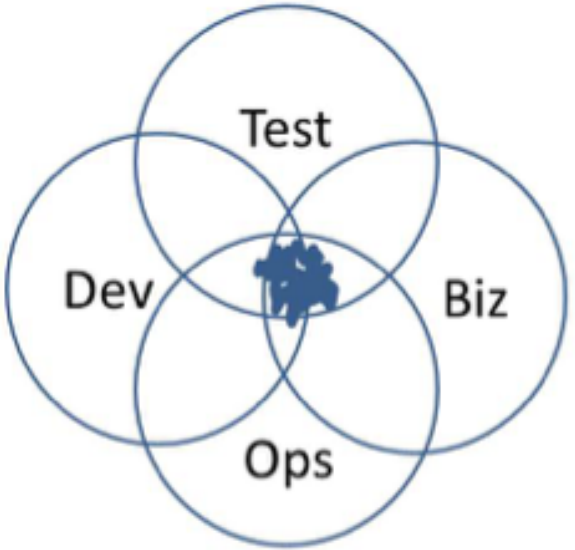
- Term coined in 2009, grew from agile values, principles & practices
- Devs, testers, ops, others collaborate
  - Create & maintain infrastructure for CI, deployments, test & prod environments
  - Support continuous delivery & testing
  - Improve our customers’ lives



It’s all about:

- Collaboration
- Continuous improvement
- Continuous learning

Testing is the heart of DevOps



Building a DevOps Culture

DevOps isn’t a role or a team

It’s collaboration between the software delivery team (including testers) and the system administration and operations team

Building relationships

This whole team approach sounds nice, but…

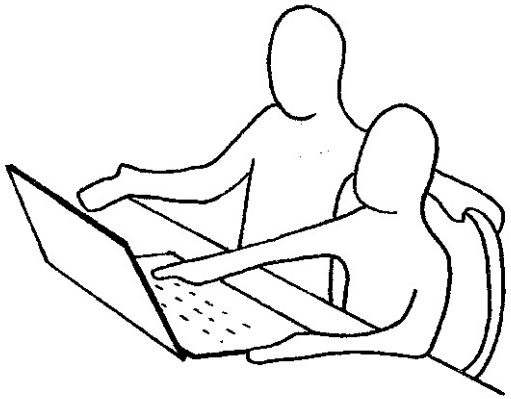
How can we engage others to collaborate?

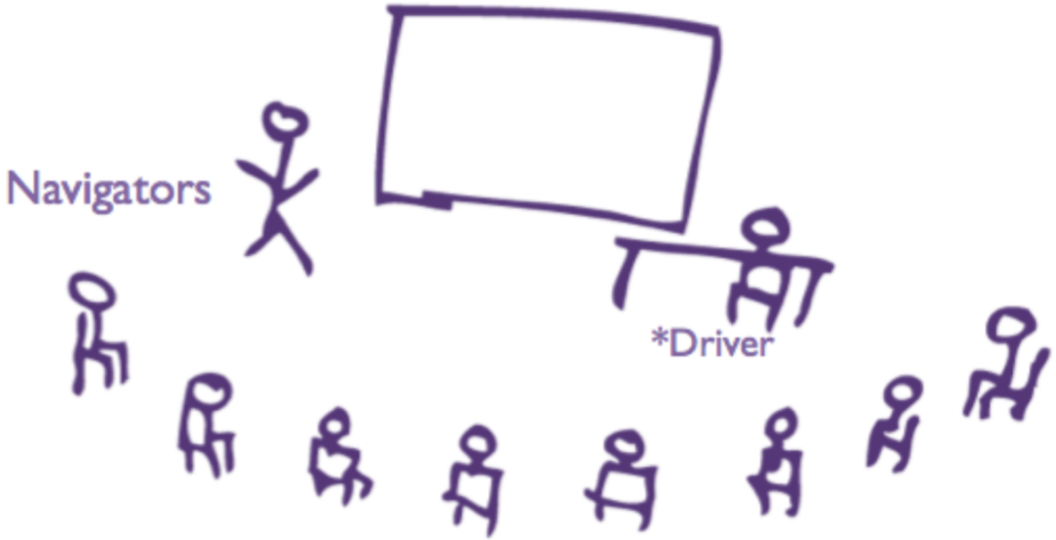
We're humans! (or possibly dragons, donkeys, unicorns…)

- Start with casual, friendly conversations
- Do food
- Share something useful
- Ask people in other roles/teams to participate, share their knowledge

Katrina Clokie has excellent tips in *A Practical Guide to Testing in DevOps*

Cross-discipline pairing, mobbing





Picture from Mob Programming Guidebook, Maaret Pyhäjärvi and Llewellyn Falco

Pillars of Lean work for DevOps too

- Respect for people
- Continuous improvement

(note: NOT “waste reduction” or tools)

Challenge everything, embrace change

(Bas Vodde, Craig Larman)

“Stop the line” mentality - from Toyota

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

What relationships do you want to build?

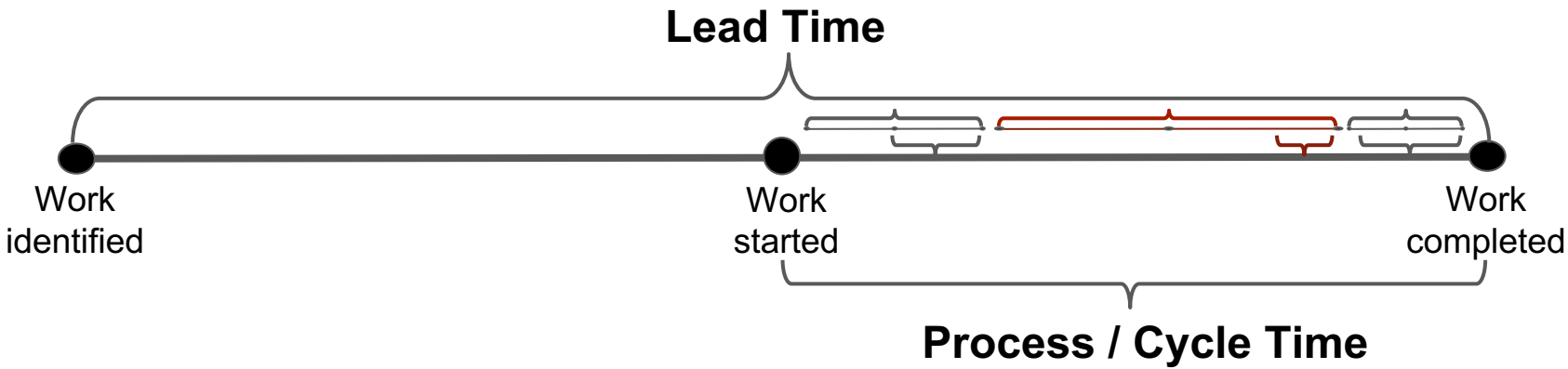
Individually:

- Make a list of people or groups outside your immediate team that could potentially help you/your team
- Write down one idea how to start building a bridge to the first three
- Pair or trio up and
  - Each share your list

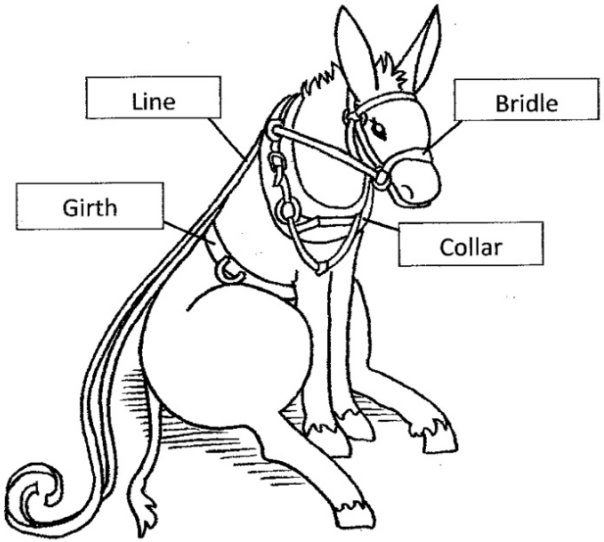
Measuring our flow of work

Cycle time: how long from start to delivery?

- Re-work slows us down
- Shared understanding speeds us up



Some common terminology around CI, CD, pipelines



And some guiding principles...

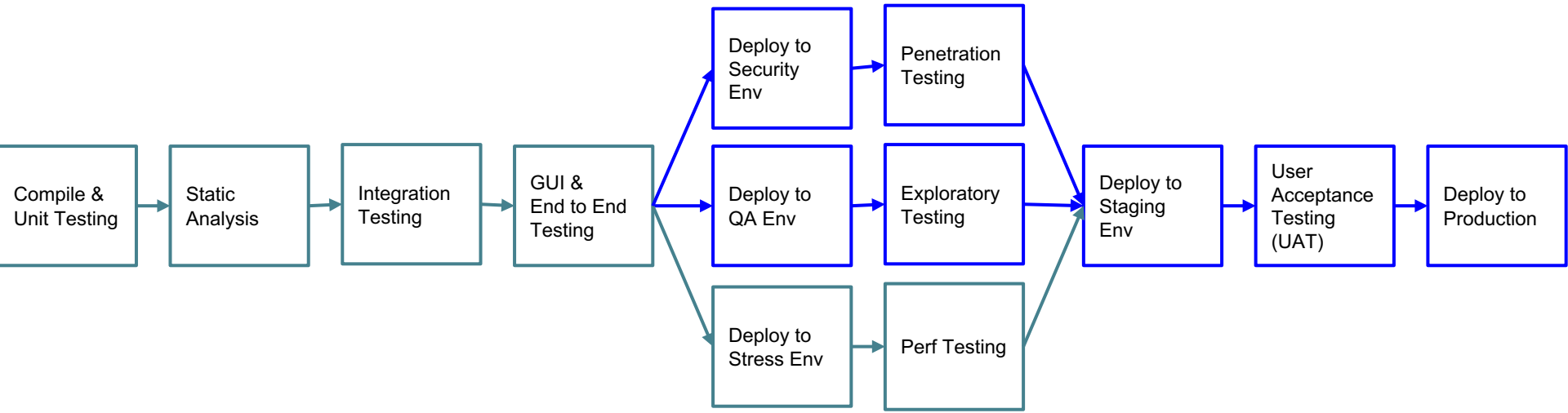
Continuous Integration

- 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

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

Continuous Delivery Example



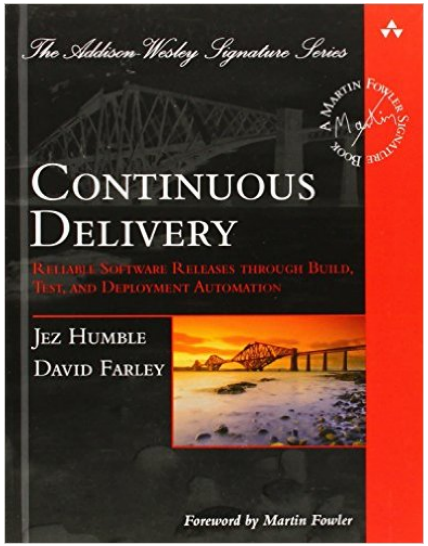
Remember...

Some, or many, steps in the pipeline may be manual!
And that is ok!

- Deploys
- Exploratory testing
- Visual checking
- … what else can you think of?

Principles of continuous delivery

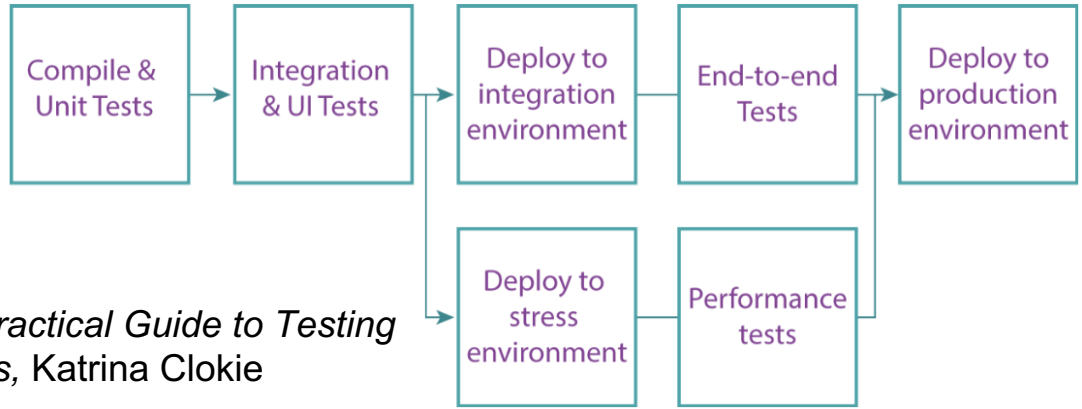
From Jez Humble and David Farley, continuousdelivery.com:



- Build quality in
- Work in small batches
- Computers perform repetitive tasks, people solve problems
- Relentlessly pursue continuous improvement
- Everyone is responsible

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



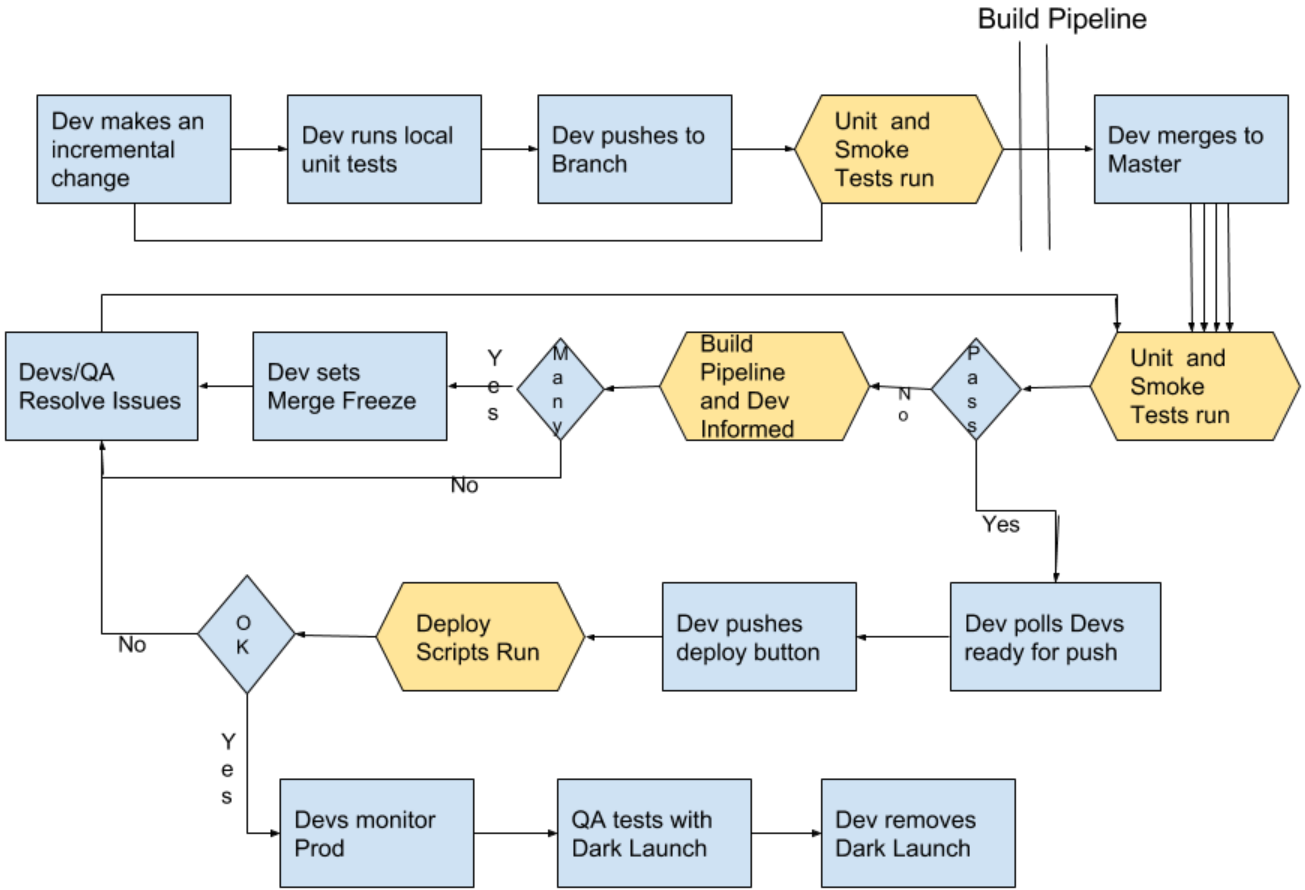
From *A Practical Guide to Testing in DevOps,* Katrina Clokie

Continuous Deployment (also CD :-/)

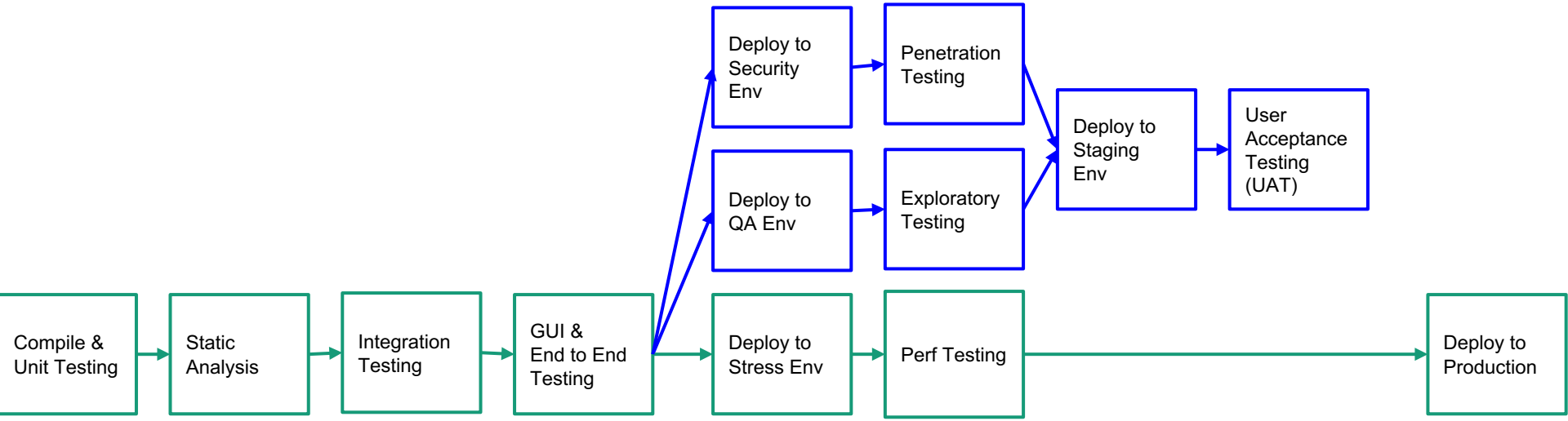
- Deployments occur on every successfully verified commit. Often many a day.
- Heavily from automated testing and Continuous Delivery environment, but does not actually require either

Image: www.squirrelpicnic.com

Extremely High Level Dev → Deploy Flow - Idealized Version



Continuous Deployment Example



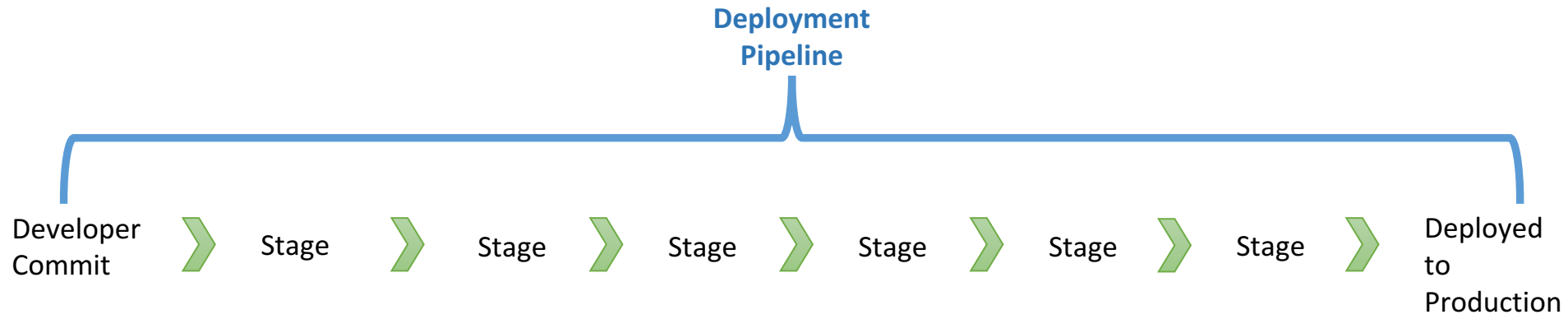
CD (either one) without automation…

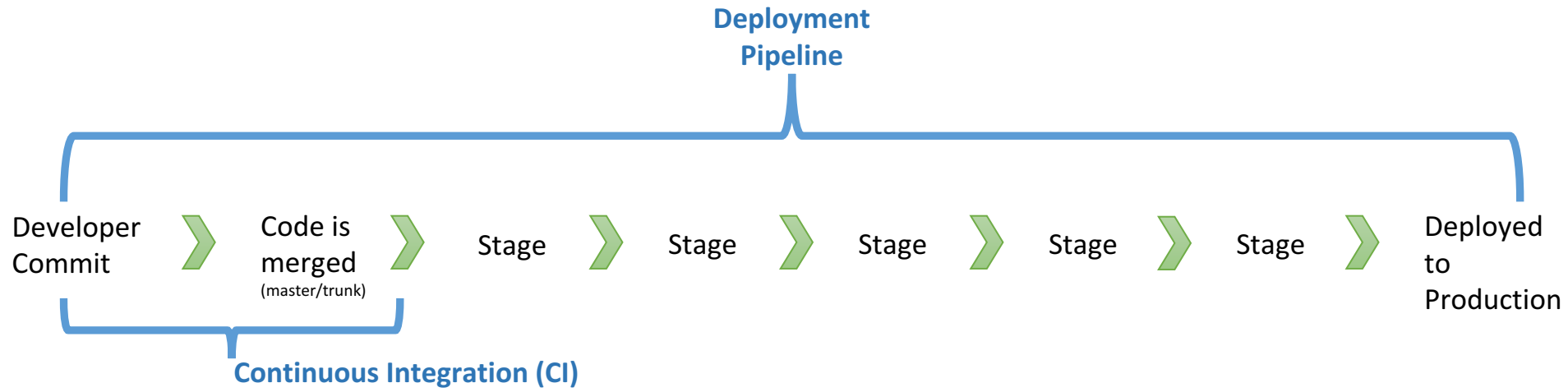
Like driving at night without your headlights. It's possible… but headlights reduce the risk.

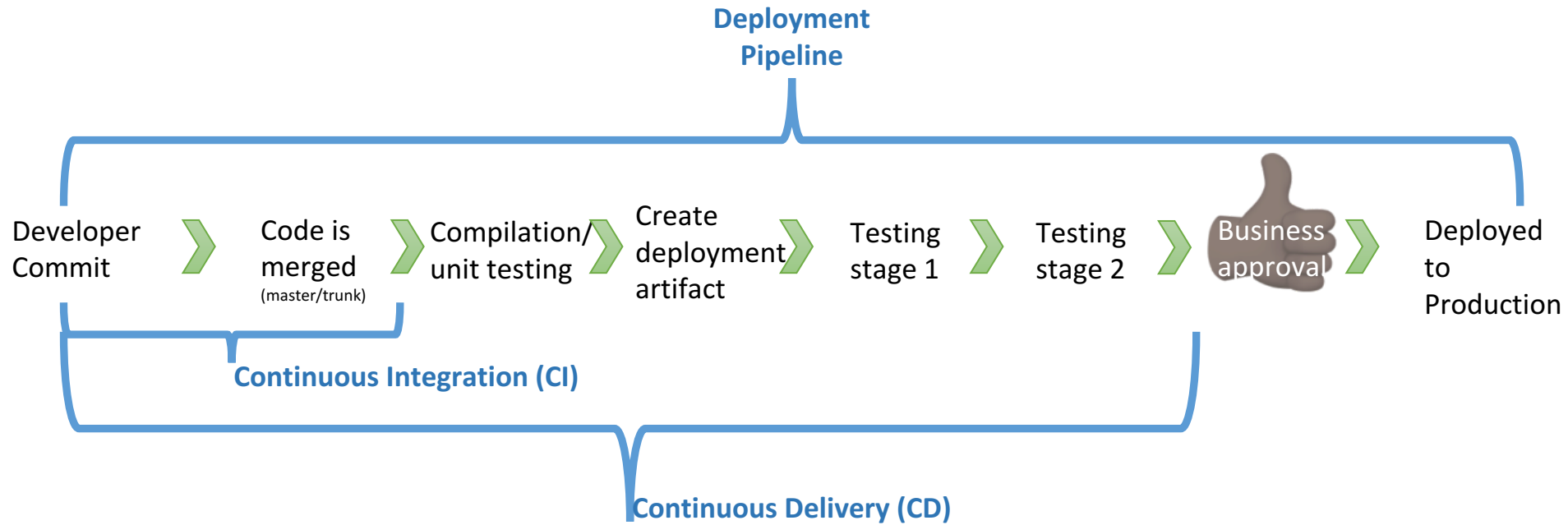
Automation is one headlight
Monitoring is the other

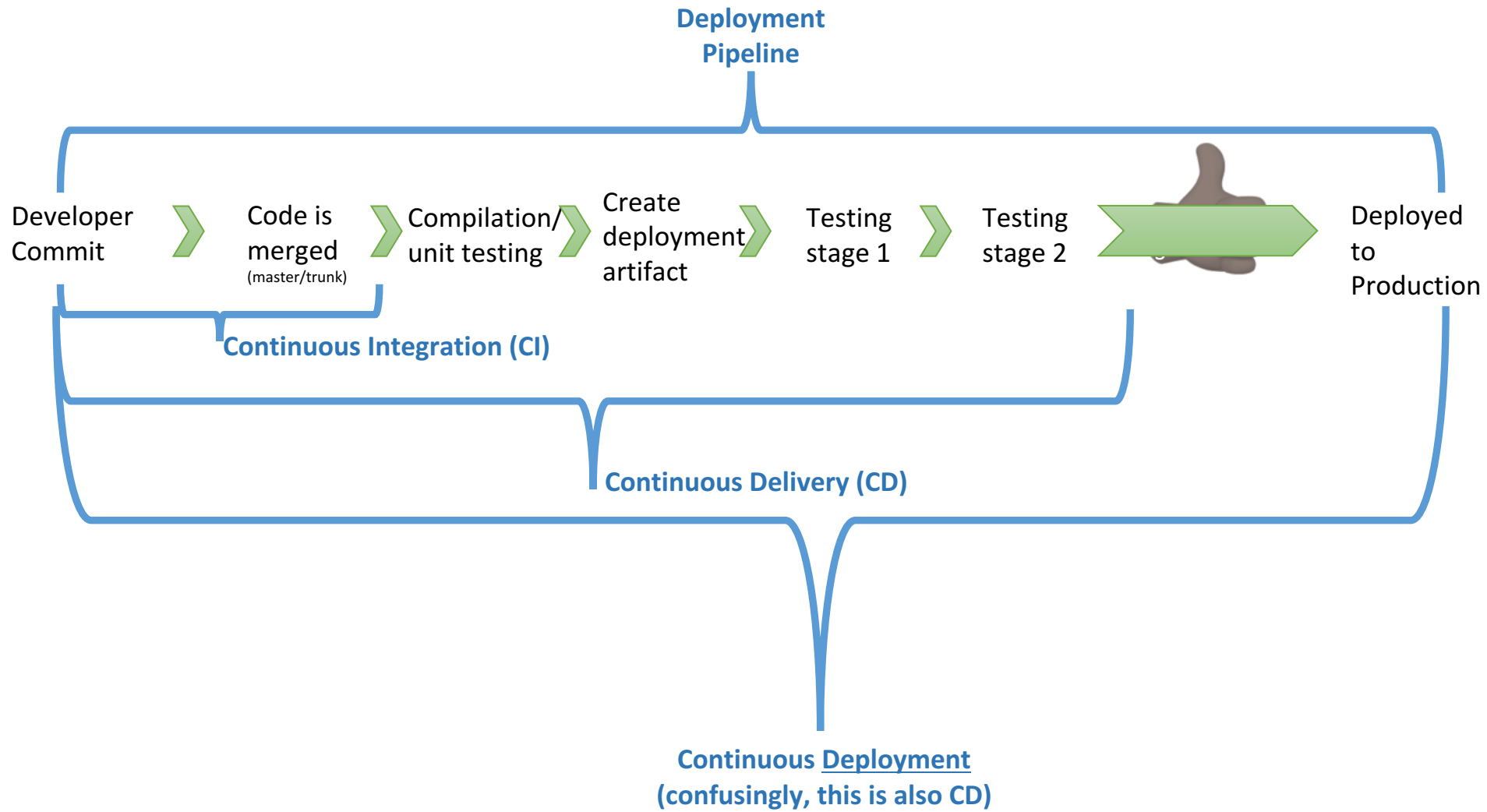
Props to Ashley Hunsberger for the analogy

To sum up the terms again…

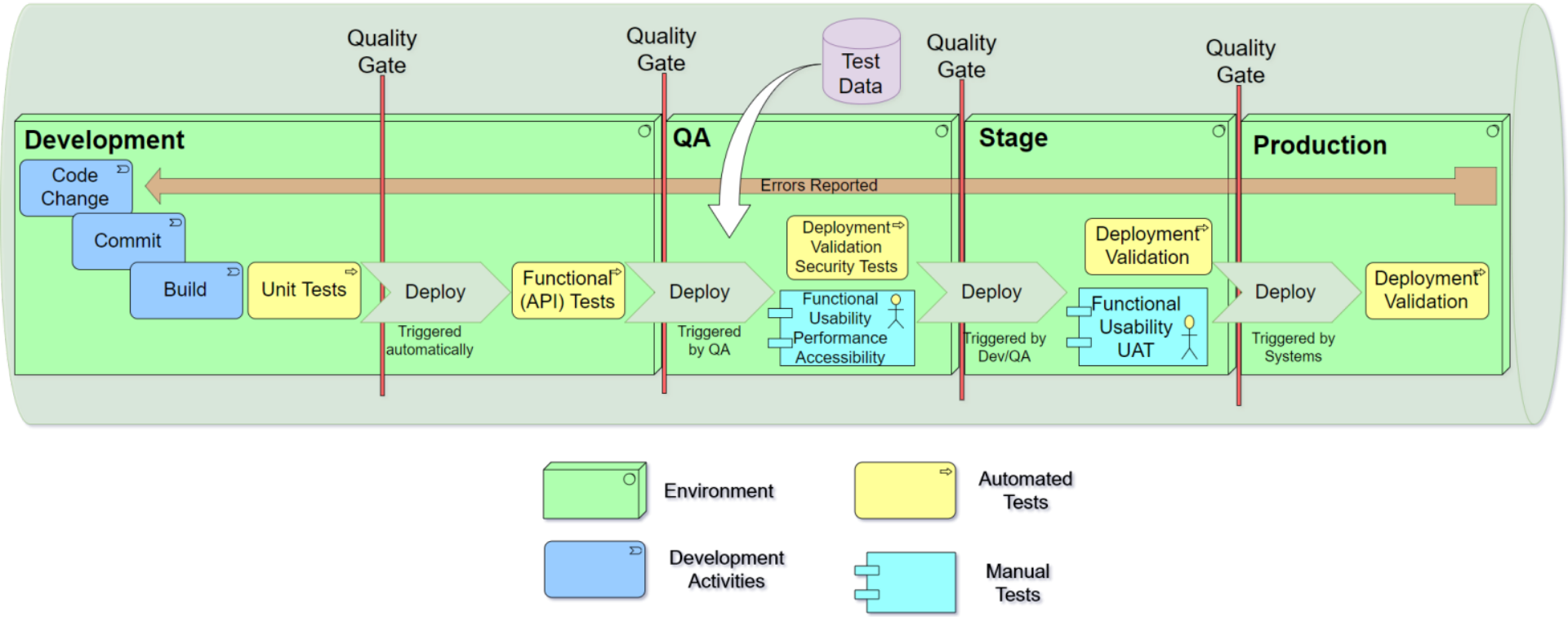

Deployment Pipeline
Developer Commit
Stage
Stage
Stage
Stage
Stage
Stage
Deployed to Production


Deployment Pipeline
Developer Commit
Code is merged
(master/trunk)
Stage
Stage
Stage
Stage
Stage
Deployed to Production
Continuous Integration (CI)

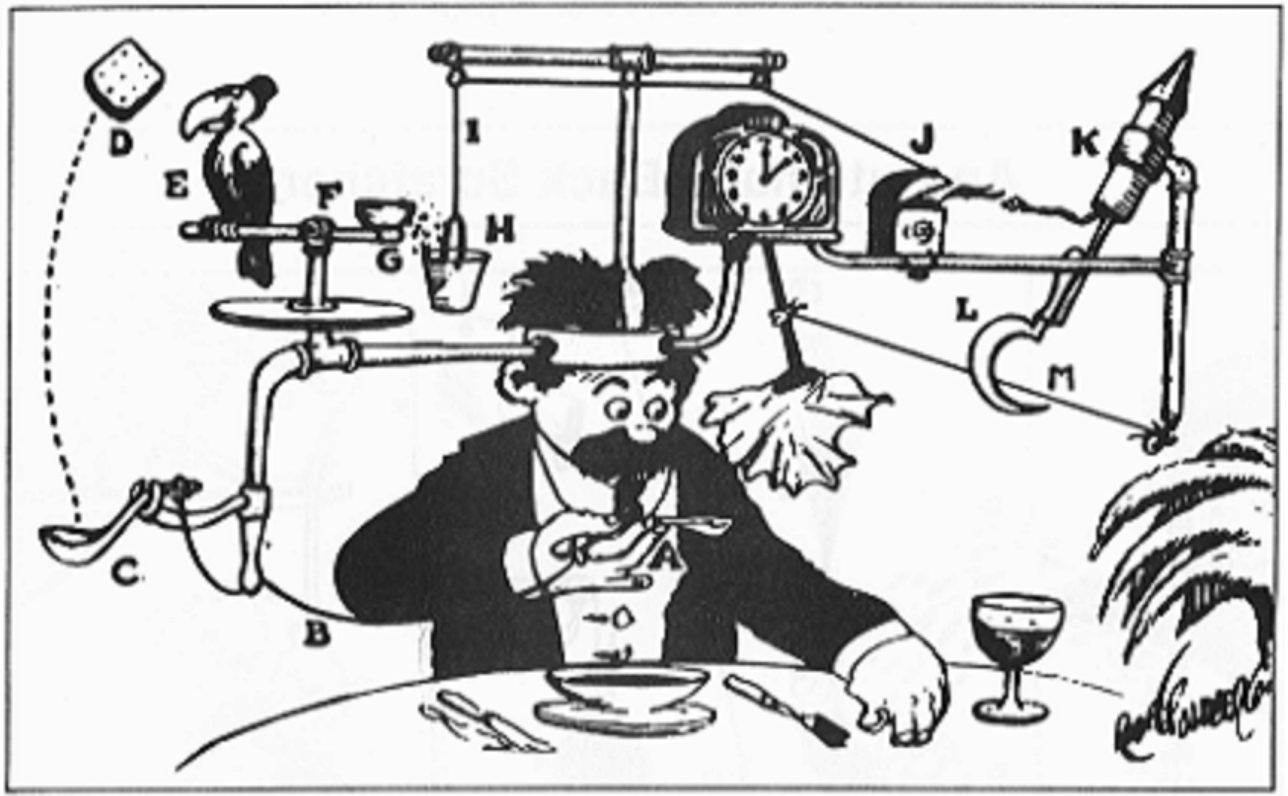

Deployment Pipeline
Developer Commit
Code is merged
(master/trunk)
Compilation/ unit testing
Create deployment artifact
Testing stage 1
Testing stage 2
Business approval
Deployed to Production
Continuous Integration (CI)
Continuous Delivery (CD)


Deployment Pipeline
Developer Commit
Code is merged
(master/trunk)
Compilation/ unit testing
Create deployment artifact
Testing stage 1
Testing stage 2
Deployed to Production
Continuous Integration (CI)
Continuous Delivery (CD)
Continuous Deployment
(confusingly, this is also CD)

Build Pipeline



Pipelines can get complicated!



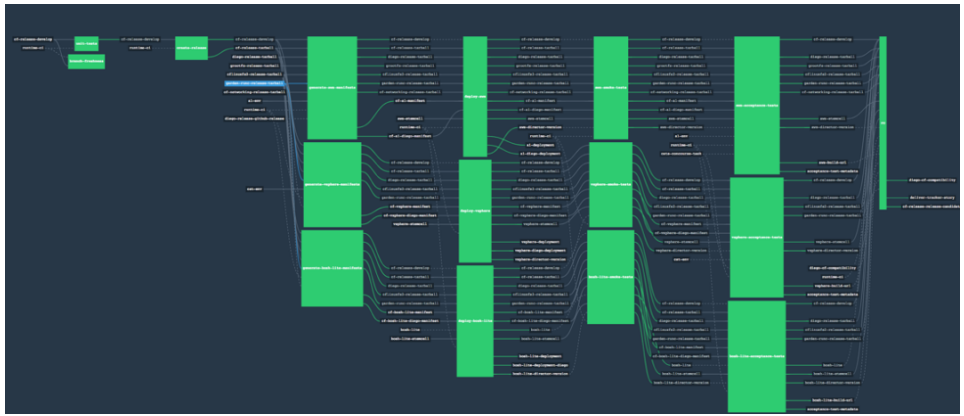
Let’s explore pipelines

Your table group is going to be a team on one of these products for today:

- Waze/Google Maps (Navigation app)
- Etsy (E-commerce something homemade)
- Kayak (book a trip)
- Paypal (banking)
- Make up your own!

How does your code flow to prod? In your table groups:

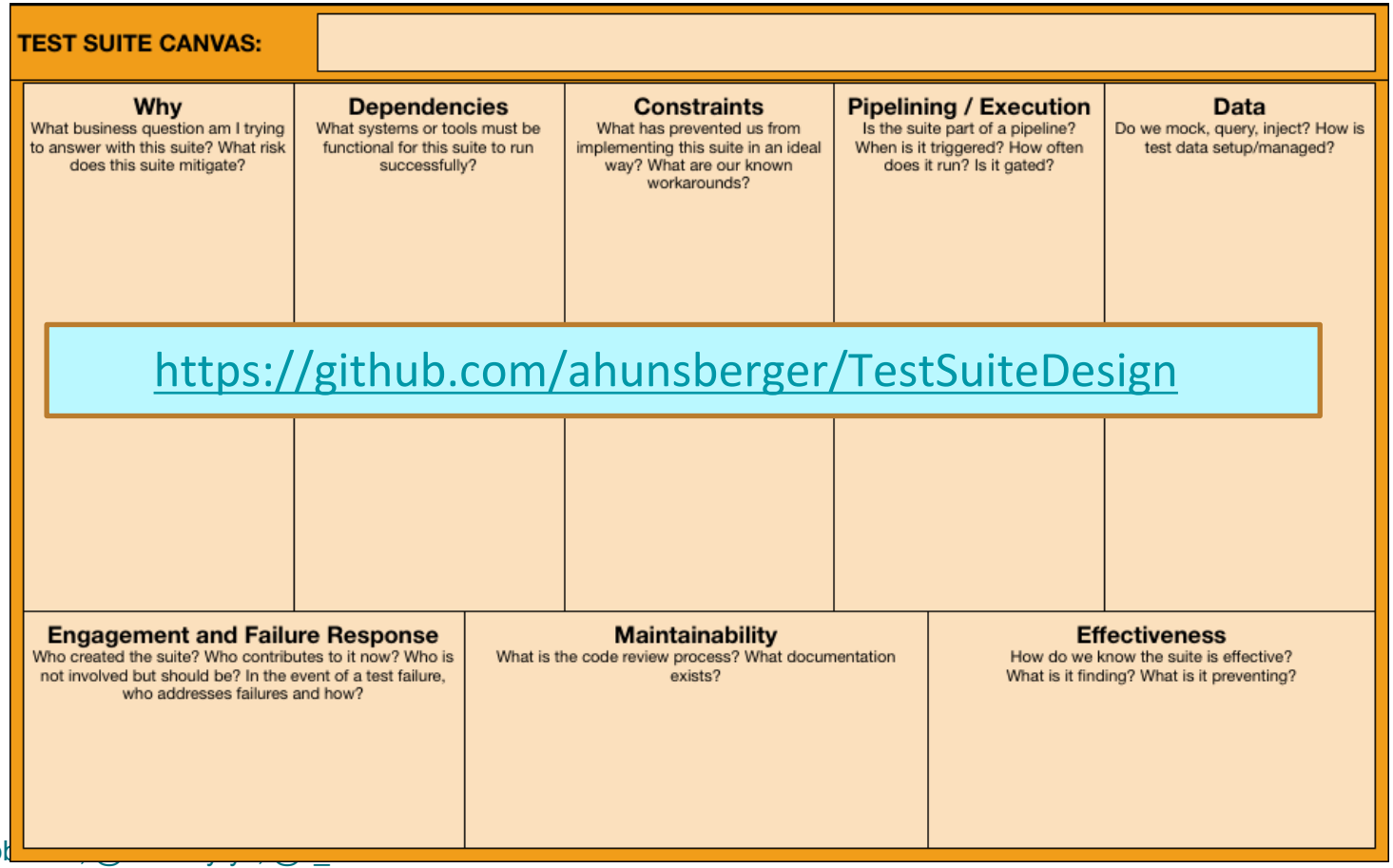
- Write up a bug for your chosen product
- Now the bug is fixed and the change has been committed.
- How does that fix get to the customer?
    - Create step cards provided to visualize the flow.
    - The handout on your table has some example steps



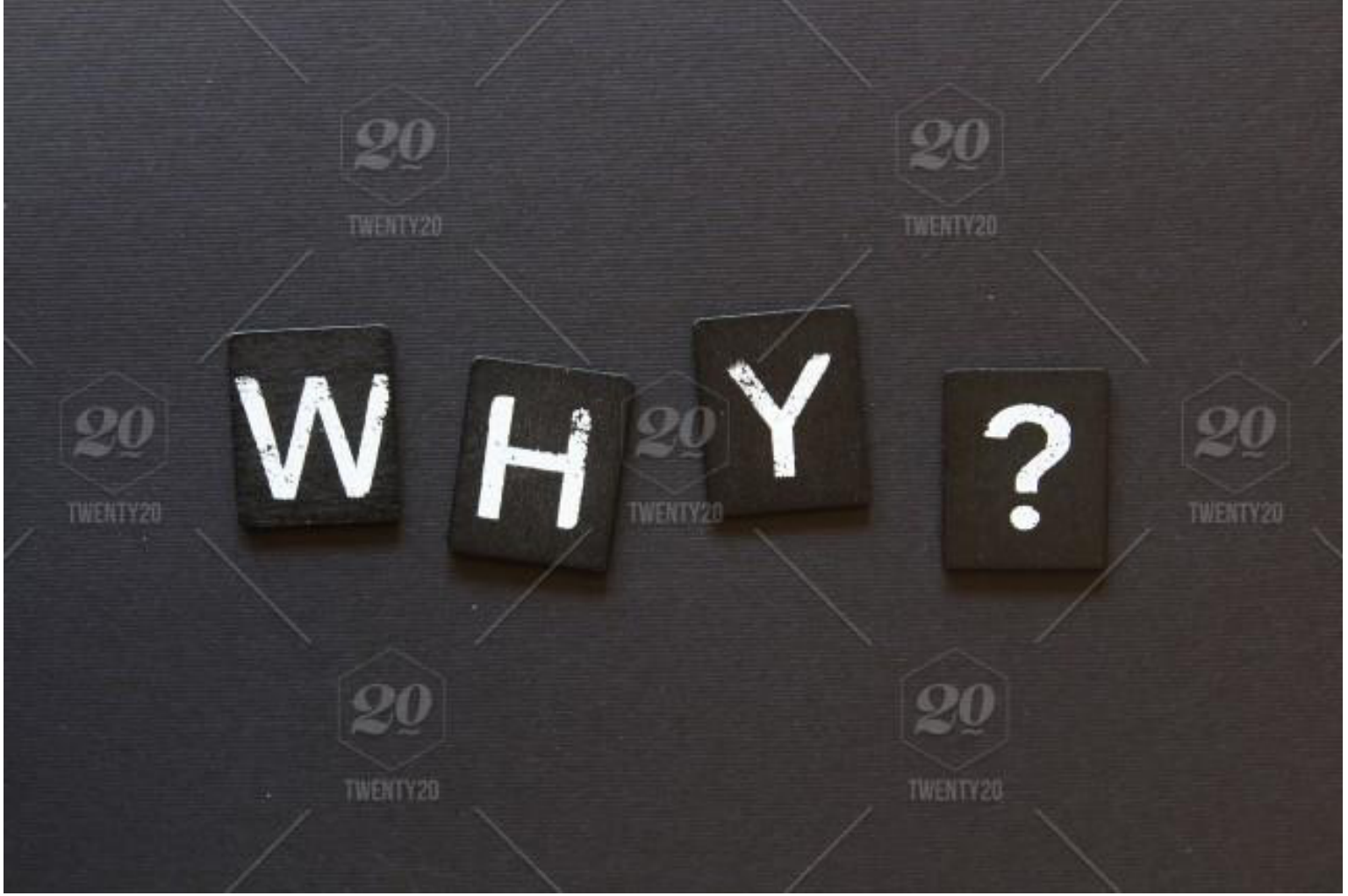
Outcomes of the code flow exercise

- Did anything surprise you?
- Is anything missing from your visual?
- Were the customers happy?

The Test Suite Canvas (from Ashley Hunsberger, inspired by Katrina Clokie)



Let’s explore a few canvas discussion points



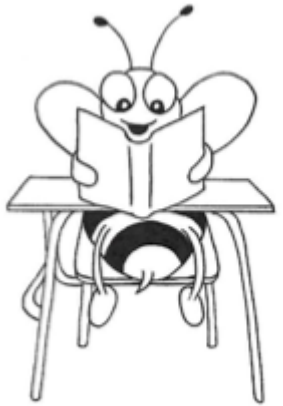
@lisacrispin, @robbowyer, @qaisdoes, @c_wiedemann

What do we want to learn from each step?

What business questions can each step in our pipeline answer?

- Integration and build
- Static code analysis
- Automated test suites
- Manual testing

What risks can we mitigate with each step?



A few real world examples…

API Test Suite

- Am I getting proper responses that warrant UI testing?

Static Code Analysis

- Are we meeting accessibility standards?

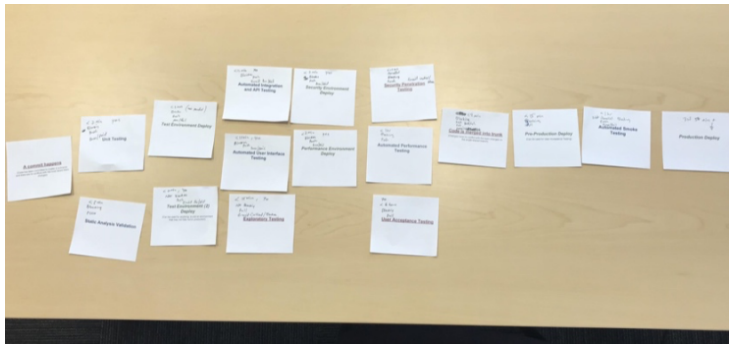
Build Installer Testing

- Does the build install without error so that it is worth further testing?

In your table groups, ask “why”

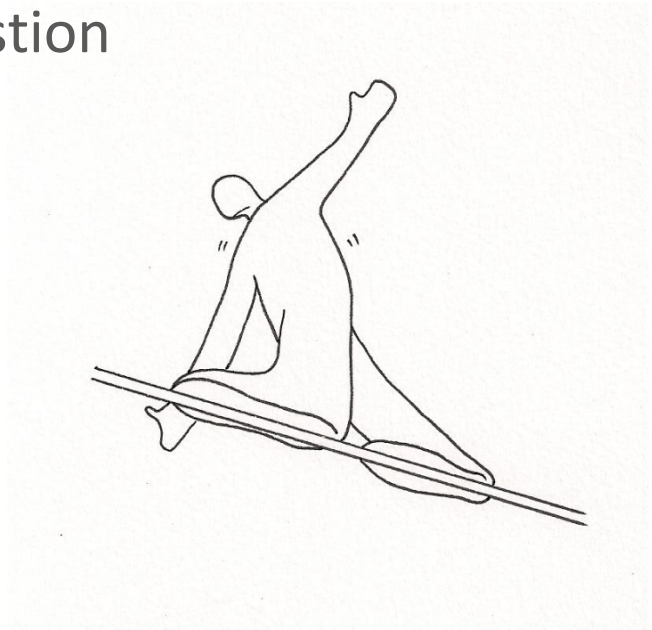
For each step:

- Identify business questions that can be answered, risks that can be mitigated
- Who on the team could benefit from that information?
- Write on sticky notes and put with the step



Why is each step needed?

- What questions and risks did you talk about?
- Give an example of a step in the pipeline that mitigated that risk or answered the question



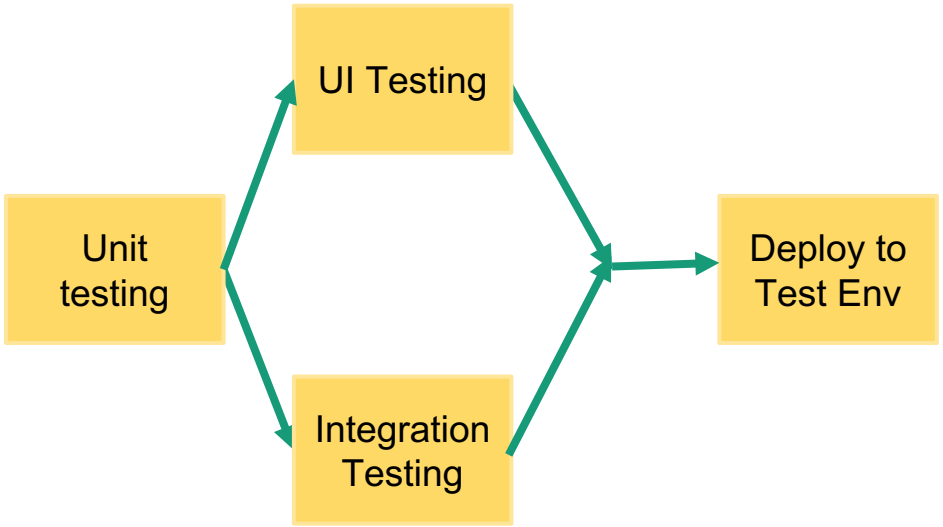
Dependencies

What needs to be in place for this step to run successfully?

- Other systems
- Tools
- Data, environments…
- What else?

“Fan in” dependency management

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



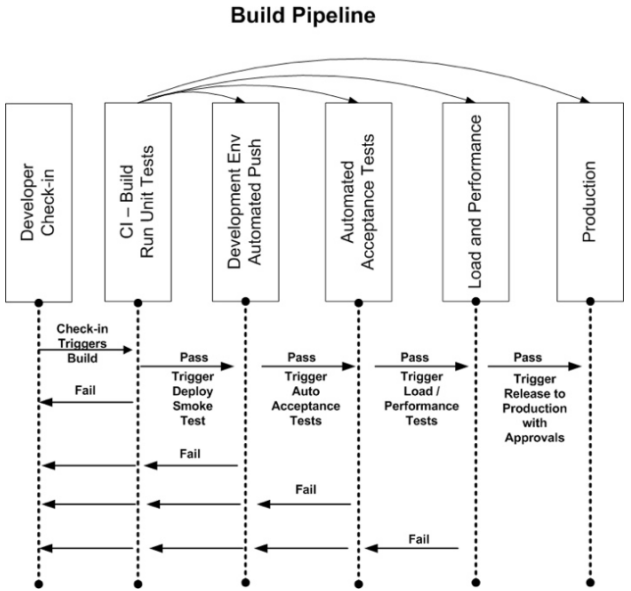
Constraints

- What's preventing us from optimizing this step?
- For example: is it a manual step we could automate?
- What are our known workarounds?

Triggering each step in your pipeline

What kicks off each step in your pipeline?

Is everything sequential?



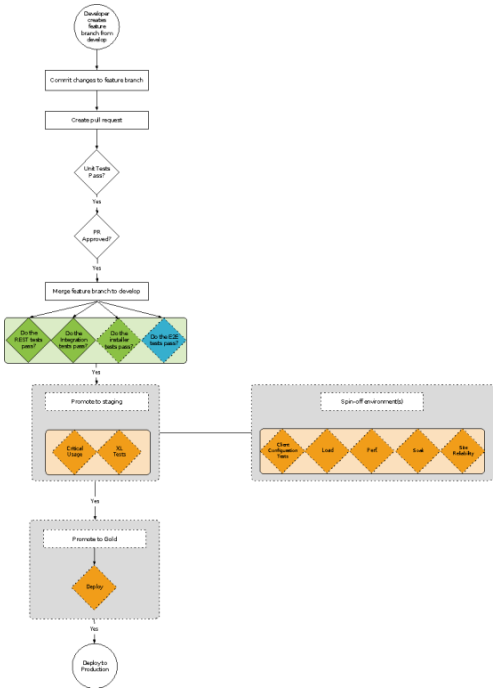
Gates!

- Automatically stop defects from making it any further down stream
- TRUST your tests (reliability)
- Whole team culture

In your table groups, revisit your pipeline:

- Which steps are dependent on others? Which are potentially constrained by lack of skills, time, other factors?
- What triggers each step to execute? What would stop the build and alert the team?
- How long do you think your feedback loops would be?
- What actions might you take to reduce dependencies and constraints? To speed up feedback?

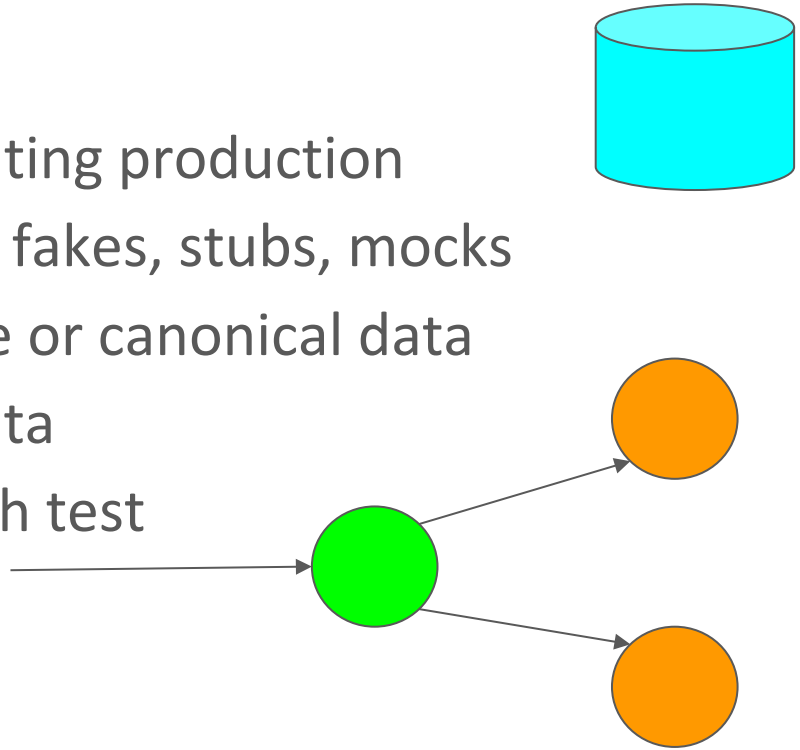
Could you find ways to shorten your feedback loops?



Test data

How do we manage test data?

- Tradeoff of speed vs. simulating production
- Unit tests use test doubles - fakes, stubs, mocks
- Higher level tests use fixture or canonical data
  - which simulates prod data
- Setup and teardown for each test



What test data do your current teams use?

- Gather with your table group around a sheet of flip chart paper on the wall.
- Take turns briefly explaining the test data that your team currently uses for different types of automated tests.
- If your team doesn’t have automated tests yet, explain the data you use for manual testing.

Test Data Discussion

- Did you discover new ways to create test data?
- What were some advantages or disadvantages of your approach over another?

Learning from production use

Monitoring

- Keep the product / system under review - known baseline
- Testing in production is a necessity
- Big data and the tools to monitor it are here
- Requires instrumenting the code
- AI (ANI), ML allow us to process the data
- Need ability to respond quickly to pain points
- Team discipline to respond to alerts
- Can miss unanticipated problems

Observability

- Software is exponentially more complex
- Many combinations of things going wrong
- Monitoring for known problems misses “unknown unknowns”
- “The ability to ask arbitrary questions about your environment without knowing ahead of time what you wanted to ask”

https://www.honeycomb.io/observability/

Learning from production

- Use “learning releases” aka “minimum viable product” (MVP)
- Usage trends can inspire new features
- A/B, beta testing
- What else?

Exposure control

- Your team needs to master feature toggles!
- Dogfooding, canary release, staged rollout, dark launch

What infrastructure do you need to support DevOps?

Discuss in your table groups

- Where would you start to build infrastructure for DevOps?
- What type of tools might help?
  - Build and deploy pipelines
  - Monitoring, instrumenting
- Servers, serverless, virtual, cloud?
- Who would build each item? Do you need to test it?
- What needs to be done manually? What can be done manually for now but later automated?

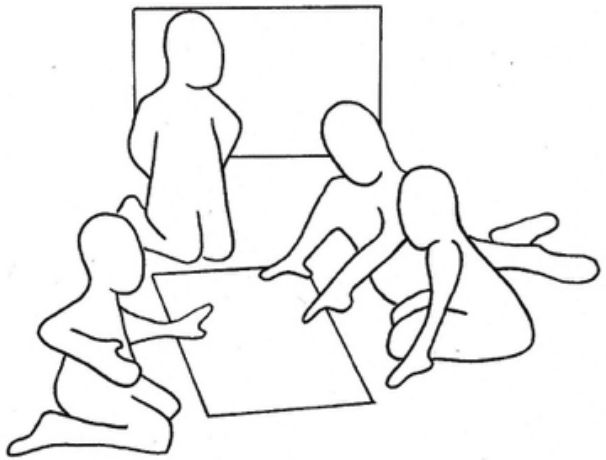
Where would you start?

- What infrastructure would you put in place first, for a team that has none?
- What experiences have you already had on your teams at work - what's in place, what do you need next?
- What needs testing?

Fitting testing into continuous delivery

What ideas do you have for how testing can “keep up”?

How can you feel confident about deploying to production more often than once a week?



Some keys to confidence

- Those feature toggles
- Small changes, frequently
- Developers exploratory test & more at story level
- Testers coordinate testing at epic/feature level
  - Pair & mob with devs, designers, product people…

Keep testing visible

- Stories for all types of testing at feature/epic level go into the backlog along with feature stories
  - Exploratory test charters
  - A11y, i18n, security, reliability, performance …
  - Regression automation not already done
- Anyone can pick up a testing task
- The feature isn’t ready until all testing activities are done
- Other ideas?

So how do we get started with DevOps?

Let’s design an experiment!

Design an experiment - part 1

In the context of your group's "product":

- Take your top 3 challenges from our opening exercise
  - If your group desires, you can substitute new ones
- Dot vote to choose the top challenge - each person gets 3 votes
- Become a cross-functional team: role-play if needed so you have a variety: product owner, tester, coder, designer, operations...

Design an experiment - part 2

For your chosen challenge:

- Create a measurable hypothesis with a prediction
- Design an experiment to test the hypothesis in the chosen work context (be specific)
- Make a poster showing challenge, hypothesis & experiment

Challenge: _____

Hypothesis:

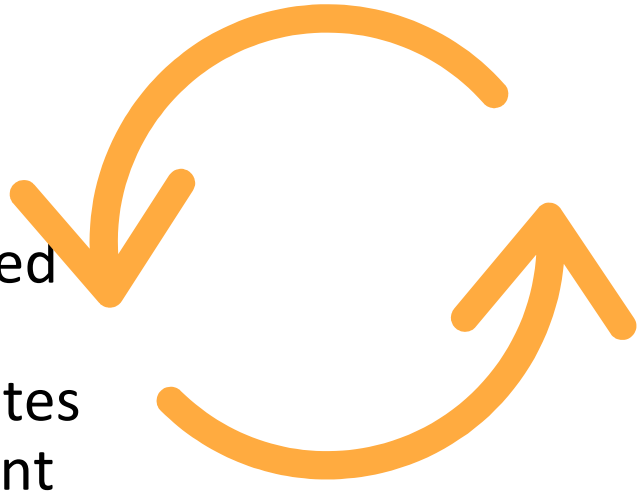
- We believe that _____
- will result in _____
- We'll know we have succeeded when _____

Experiment:

- _____
- _____

Design an experiment - part 3: Get some feedback!

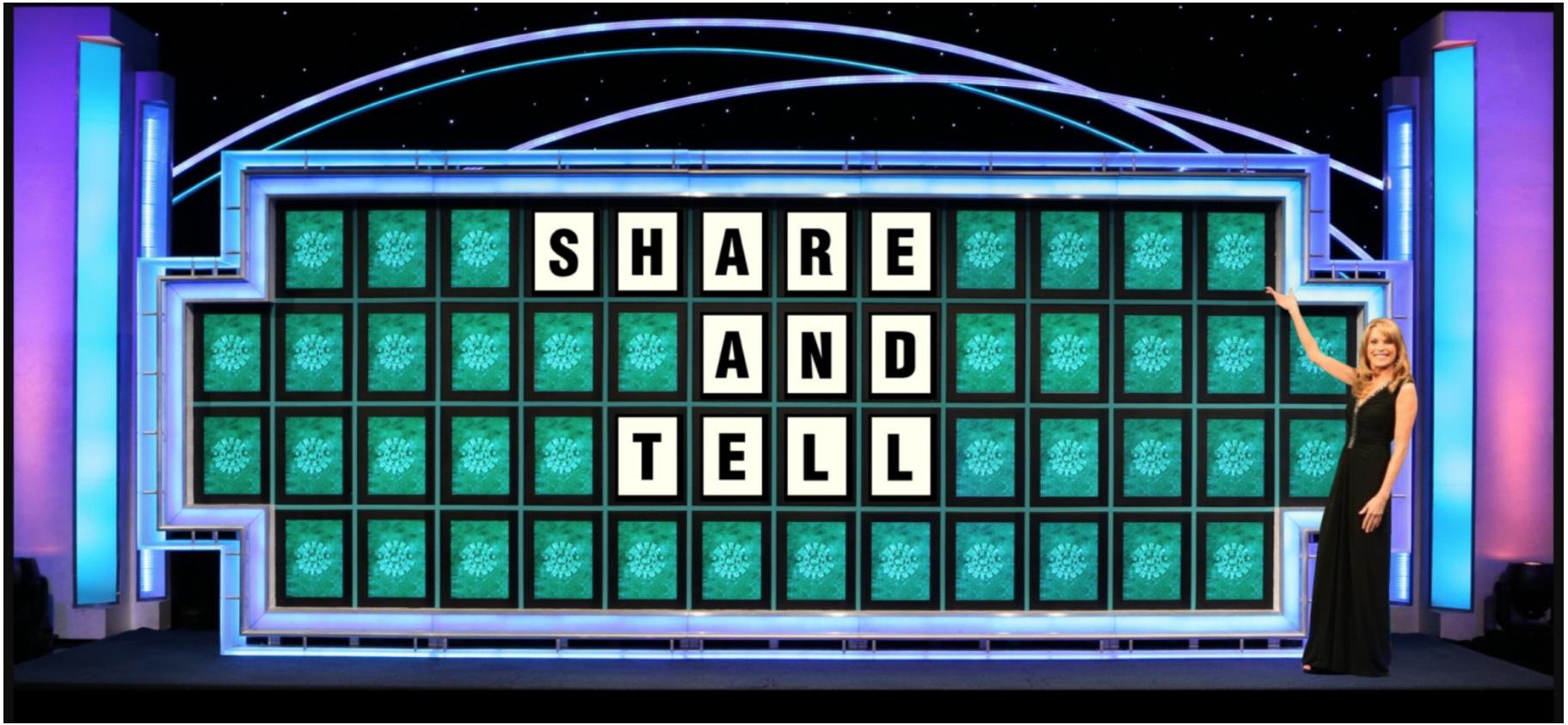
- Choose a spokesperson to take your poster and travel to the opposite table
- Everyone else stays at their table
- Spokespeople explain the challenge, hypothesis and experiment to the visited group
- The group gives feedback and contributes their ideas for the presented experiment



Design an experiment - part 4: Incorporate feedback

- Tweak your hypothesis and experiment based on the feedback you got from others
- Make a new poster if needed
- Get ready to present your challenge, hypothesis and experiment to everyone ☺

Group presentations



@lisacrispin, @robbowyer, @qaisdoes, @c_wiedemann

Each group please present:

- the challenge/problem,
- the measurable hypothesis,
- the designed experiment,
- the most valuable idea you got from the other groups that you decided to incorporate

Reflecting on our experiments

- Would you be willing to try this with your real world team?
- What was helpful about getting feedback?

Important DevOps topics we're not covering today

- Infrastructure as code
- Configuration management
- Containers
- Cloud
- Environment management
- Infrastructure testing

… See *Continuous Delivery* and *A Practical Guide to Testing in DevOps*

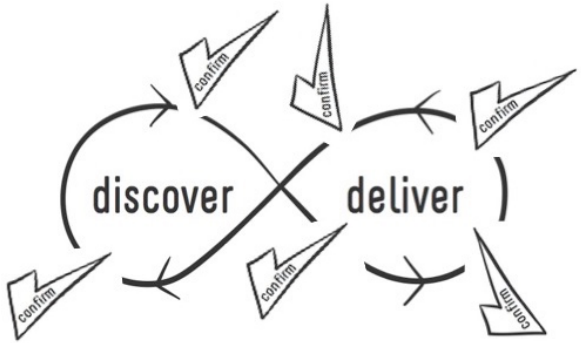
What’s on your roadmap?

Individually: Reflect on and finish up your roadmap to DevOps

Who’s willing to share?

Succeeding with the whole team approach

- Collaborating to continuously improve pipelines, feedback
- Whole team commitment, engagement
- Visualize together, experiment
- Baby steps - it's a process
- Not "shifting left or right" – it's infinite!



Ellen Gottesdiener and Mary Gorman, *Discover to Deliver*

Questions?

Resources list - see our handout too

- https://github.com/ahunsberger/CAST2018
- Agile Testing (Lisa and Janet)
- More Agile Testing (Lisa and Janet)
- Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation (Jez Humble and David Farley)
- A Practical Guide to Testing in DevOps (Katrina Clokie)
- Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations (Nicole Forsgren, Jez Humble, Gene Kim)
- https://github.com/ahunsberger/TestSuiteDesign

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

@lisacrispin, @robbowyer, @qaisdoes, @c_wiedemann

Thank you!

Let’s keep the conversation going...

@lisacrispin
@robbowyer
@mikehrykryk