CS 4530 Software Engineering

Module 14: Continuous Development Processes

Jonathan Bell, Adeel Bhutta, Mitch Wand
Khoury College of Computer Sciences

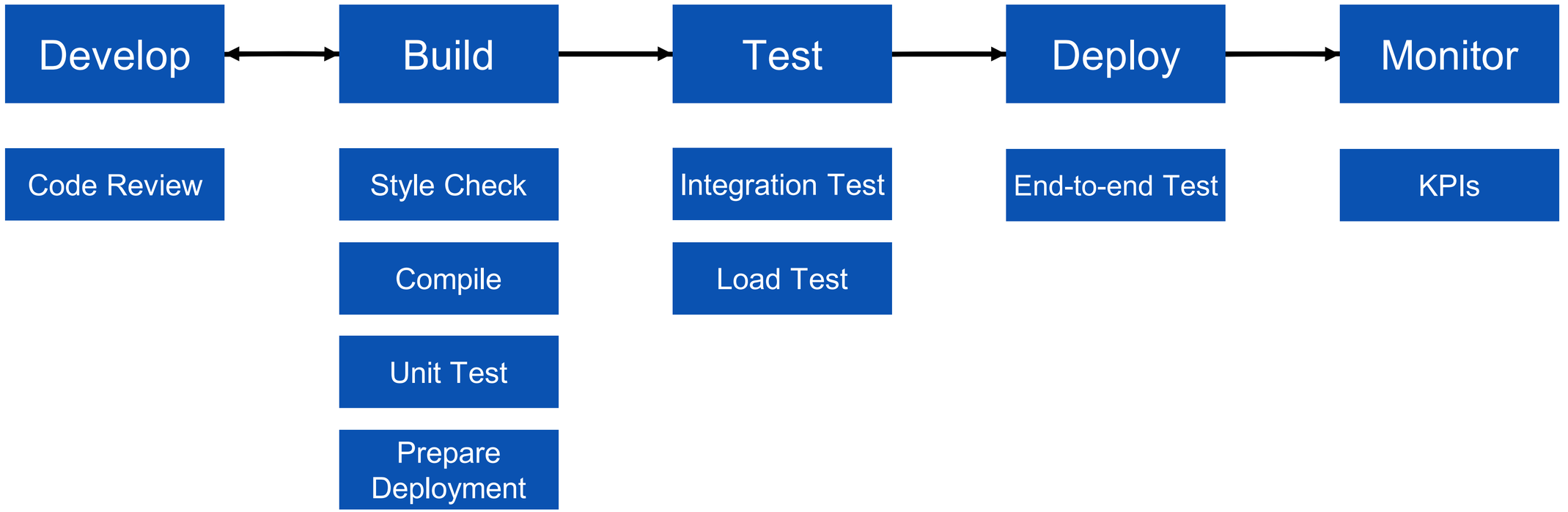
Learning Objectives for this Lesson

By the end of this lesson, you should be able to…

- Describe how continuous integration helps to catch errors sooner in the software lifecycle
- Describe the benefits of a culture of code review
- Describe strategies for performing quality-assurance on software as and after it is delivered

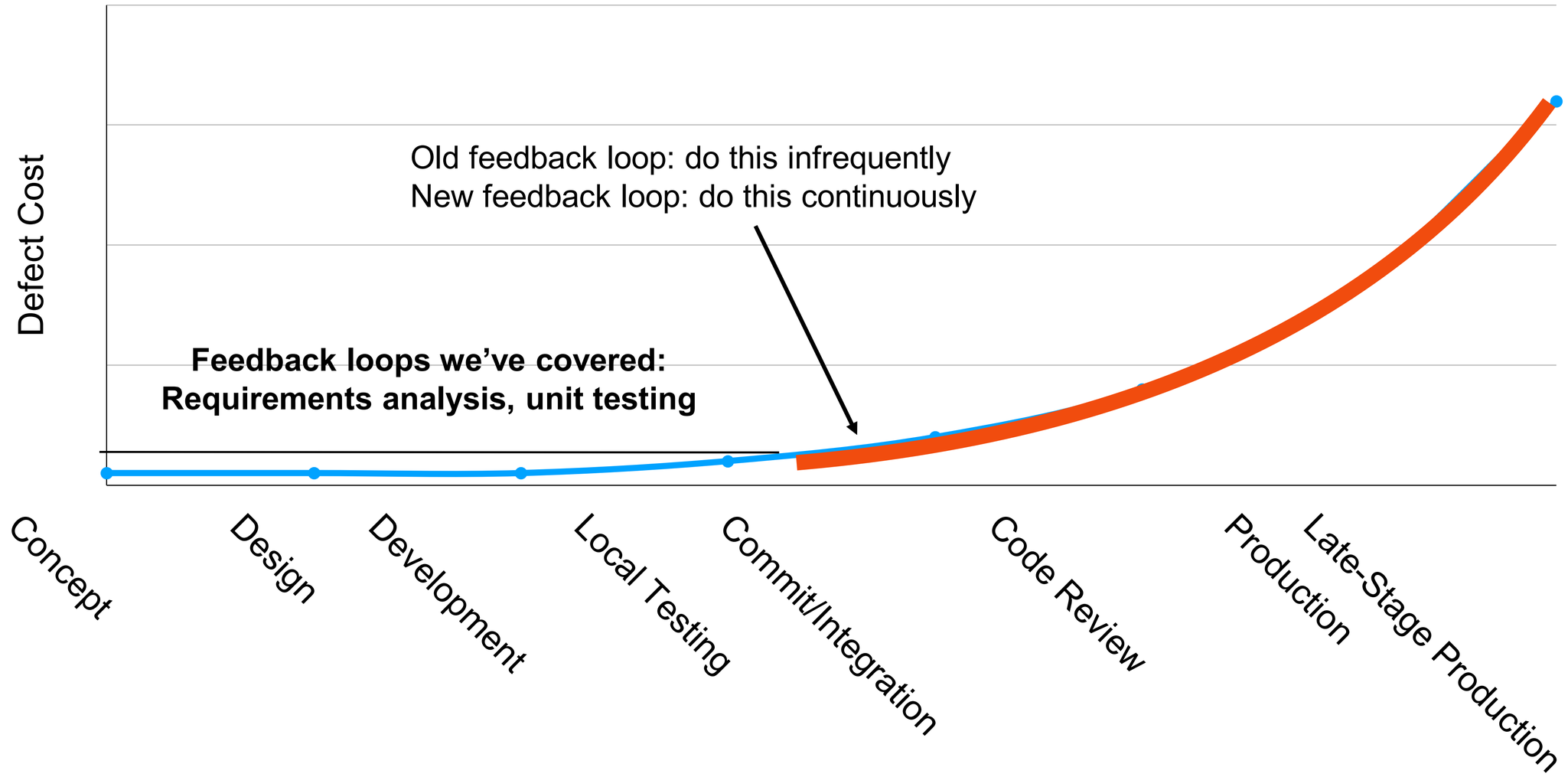
Continuous Development

Improving quality & velocity with frequent, fast feedback loops



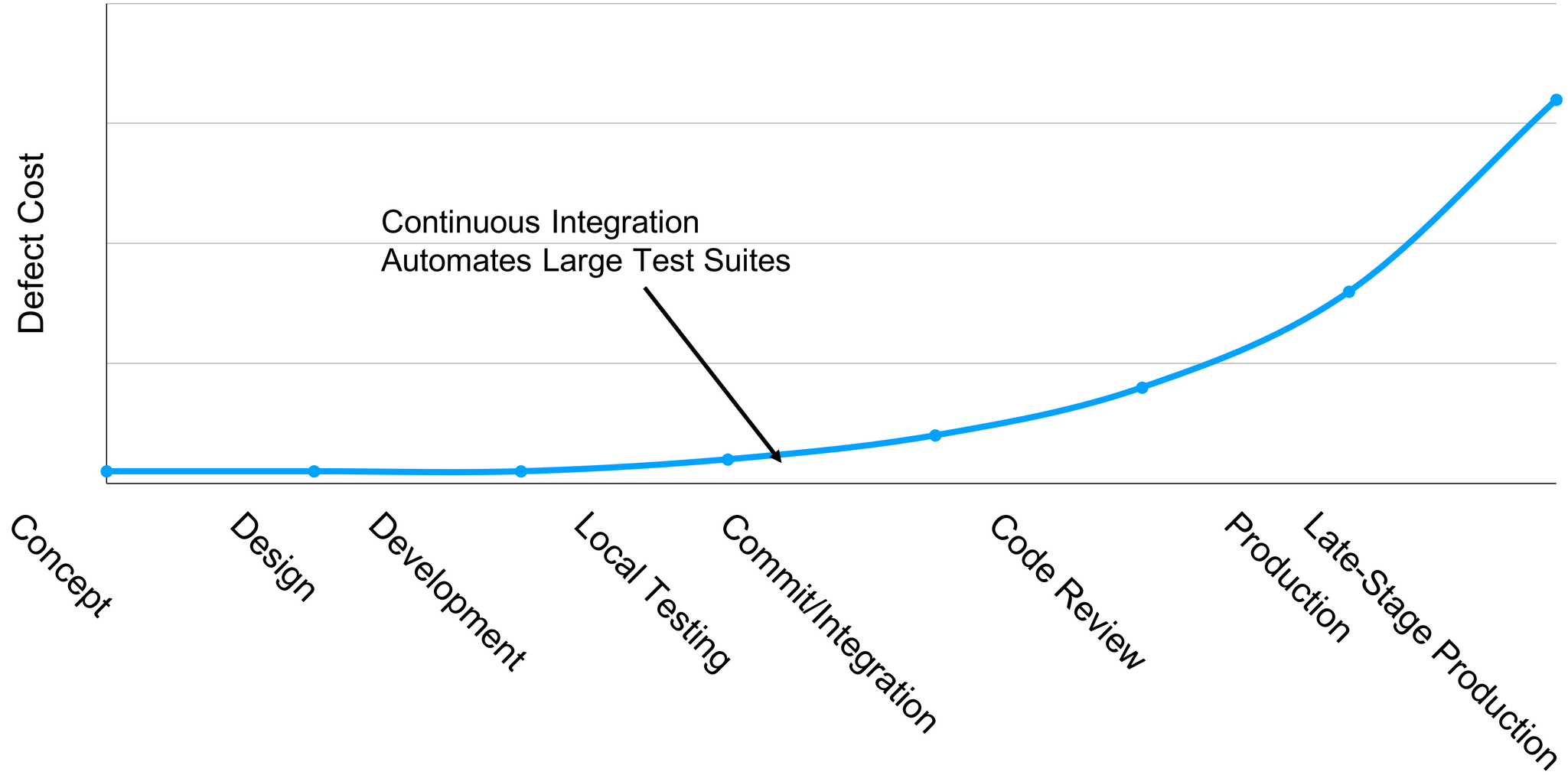
Agile Values Fast Quality Feedback Loops

Faster feedback = lower cost to fix bugs



Continuous Integration

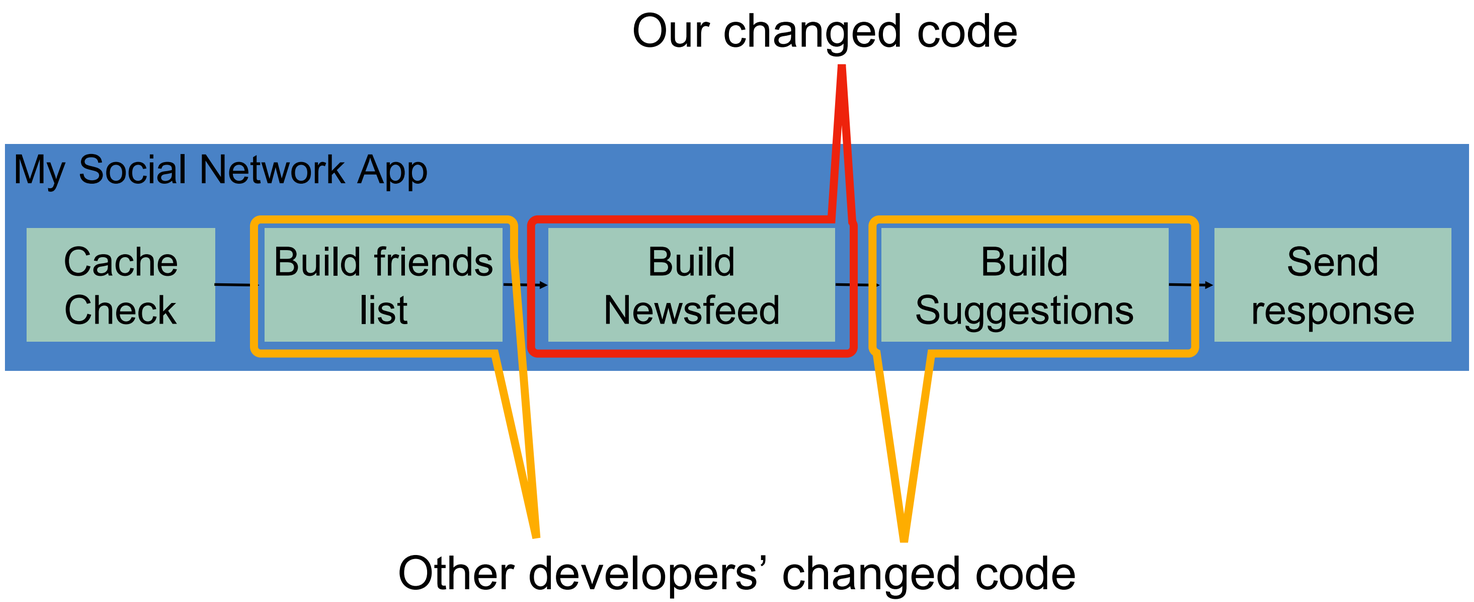
Fast feedback on integration errors



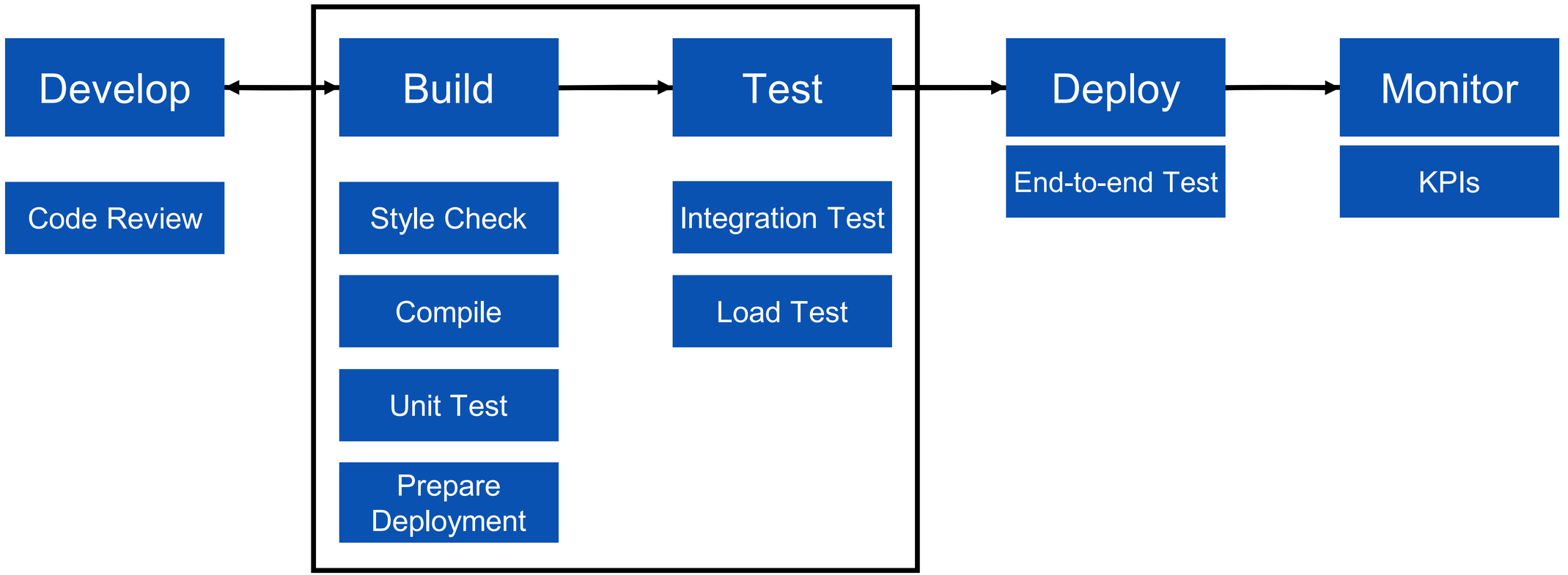
Continuous Integration

Motivation

- Our systems involve many components, some of which might even be in different version control repositories
- How does a developer get feedback on their (local) change?



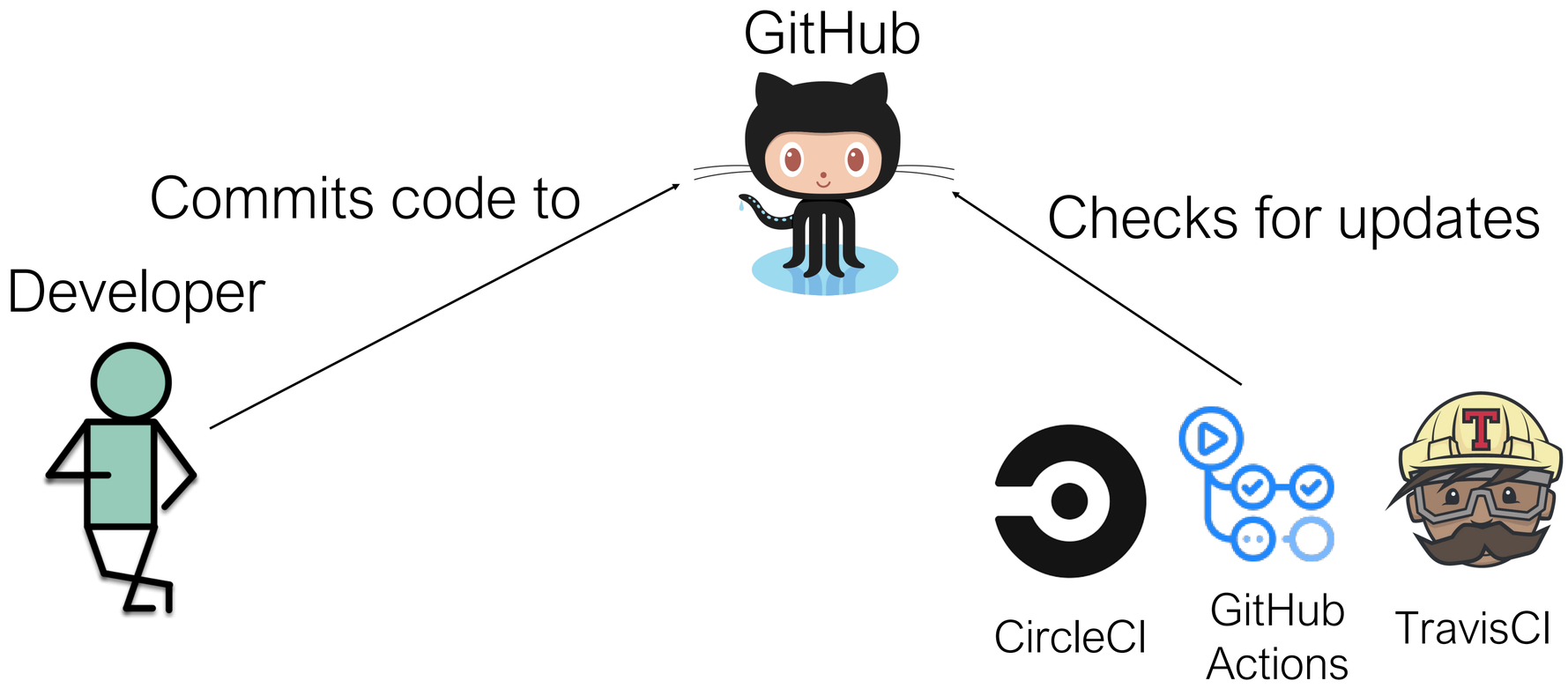
Continuous Integration is a Software Pipeline



Automate this centrally, provide a central record of results

Continuous Integration in Practice

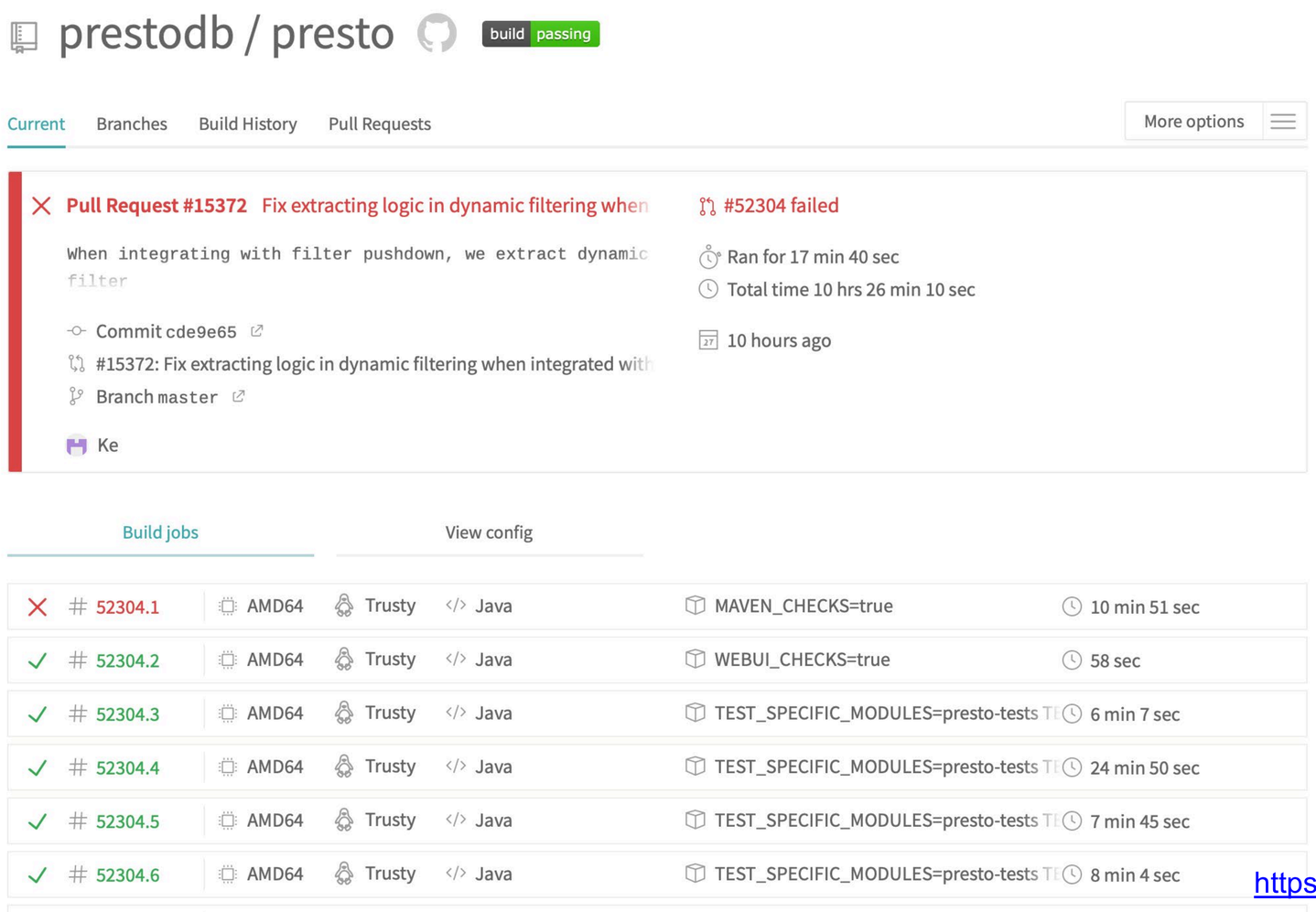
Small scale, with a service like CircleCI, GitHub Actions or TravisCI



Runs build for each commit

Example CI Pipeline

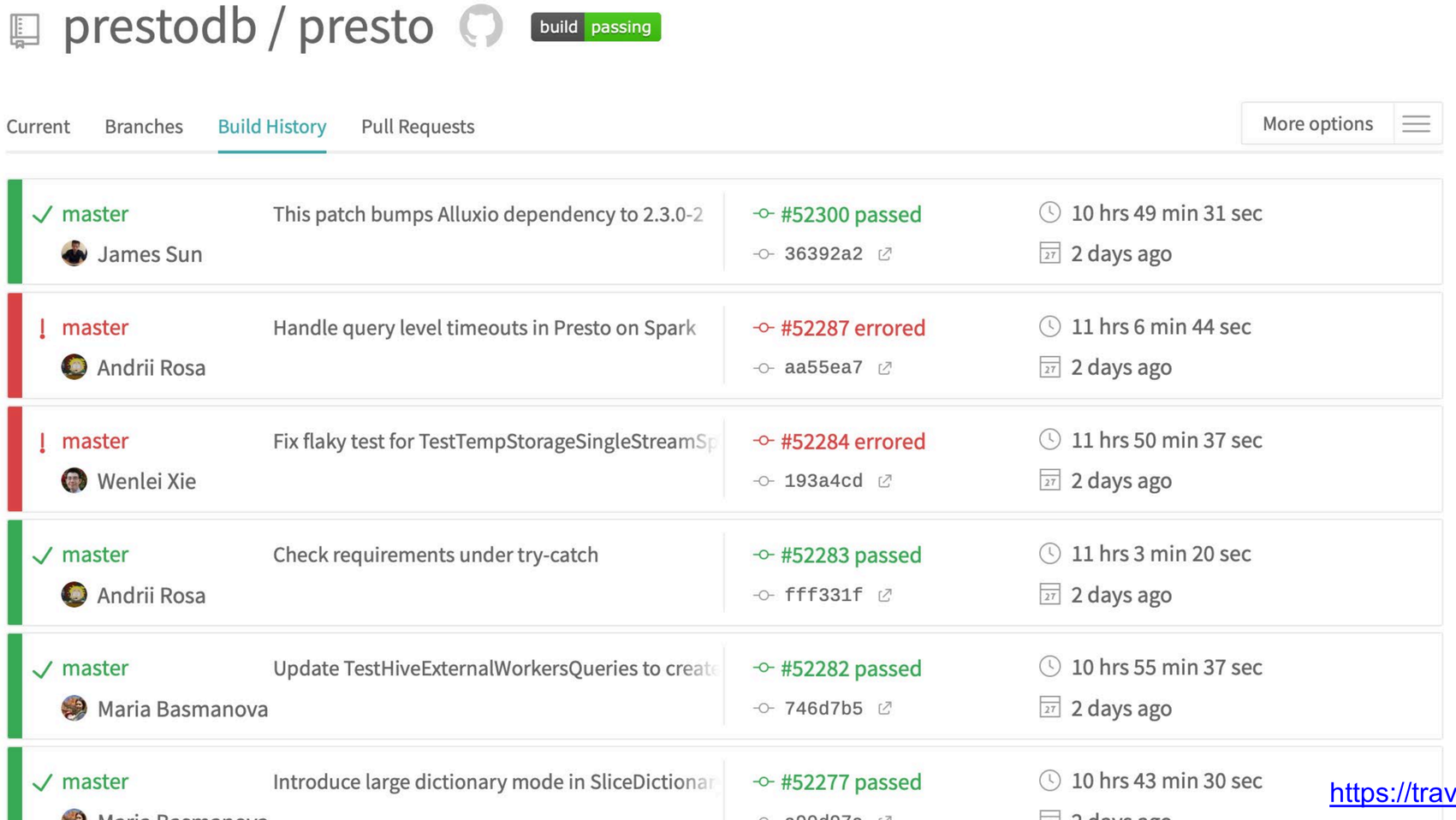
Open source project: PrestoDB



https://travis-ci.com/github/prestodb/presto

Example CI Pipeline - TravisCI

At a glance, see history of build



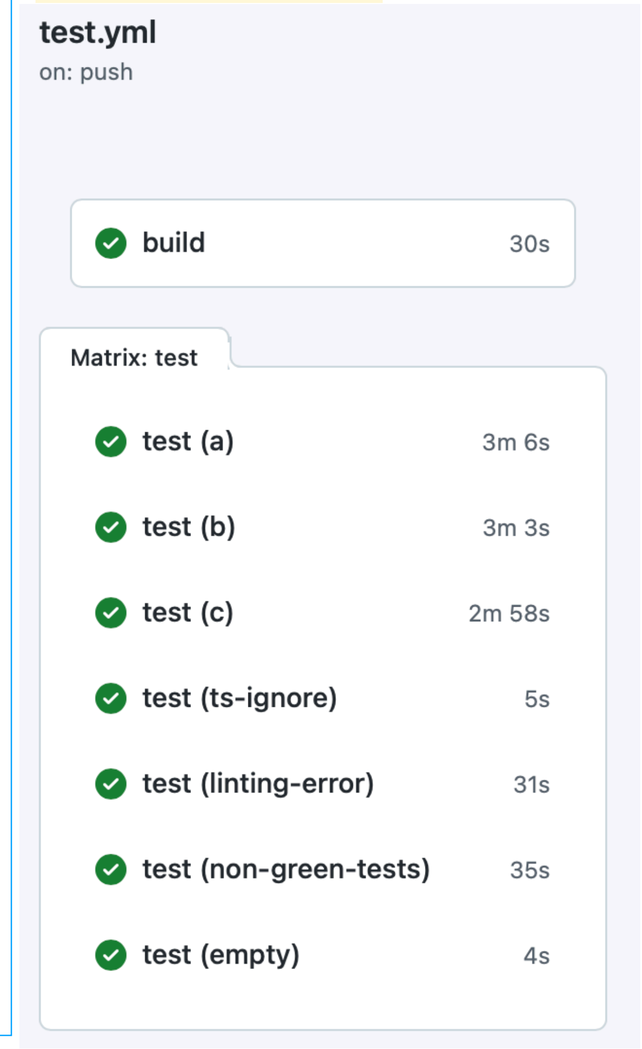
https://travis-ci.com/github/prestodb/presto

CI In Practice: Individual Project Autograder

test.yml (CI workflow file)

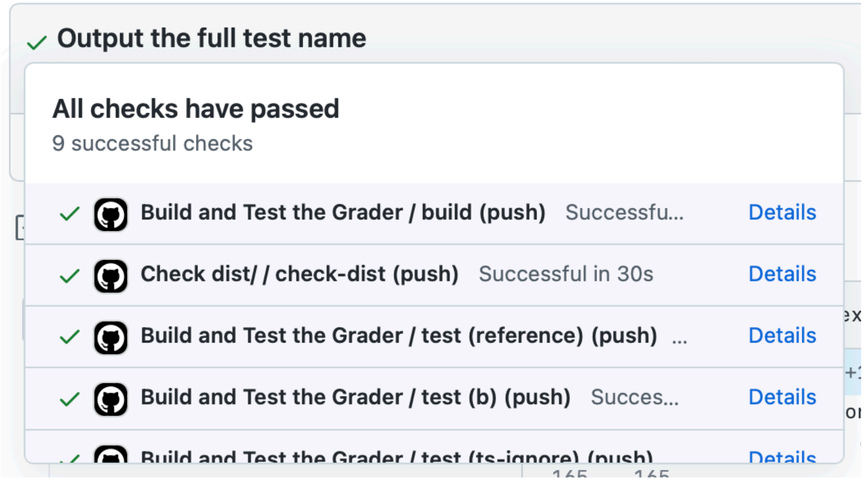
```yaml
name: 'Build and Test the Grader'
on: # rebuild any PRs and main branch changes
  pull_request:
  push:
    branches:
      - main
      - 'releases/*'
jobs:
  build:
    runs-on: self-hosted
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '16'
      - run: |
          npm install
  test:
    runs-on: self-hosted
    strategy:
      matrix:
        submission: [a, b, c, ts-ignore, linting-error, non-green-tests, empty]
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '16'
      - uses: ./
        with:
          submission-directory: solutions/${{ matrix.submission }}
```

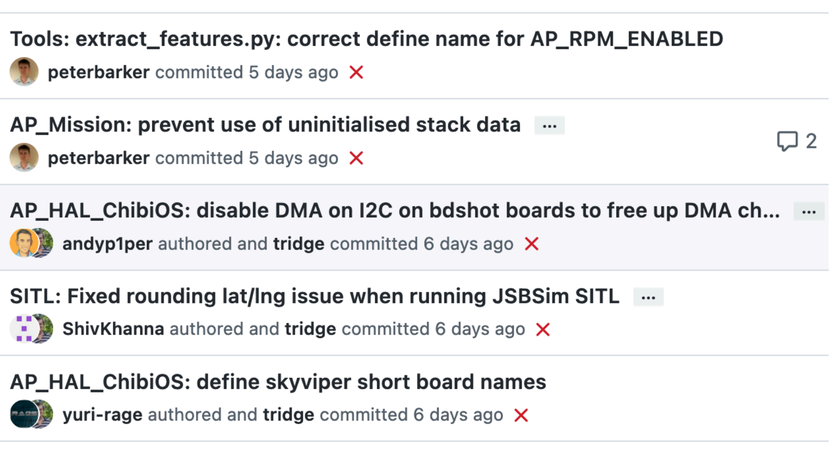
GitHub Actions Results



Attributes of Effective CI Processes

- Do not allow builds to remain broken for a long time
- CI should run for every change
- CI should be *fast*, providing feedback within minutes or hours
- CI should not completely replace pre-commit testing

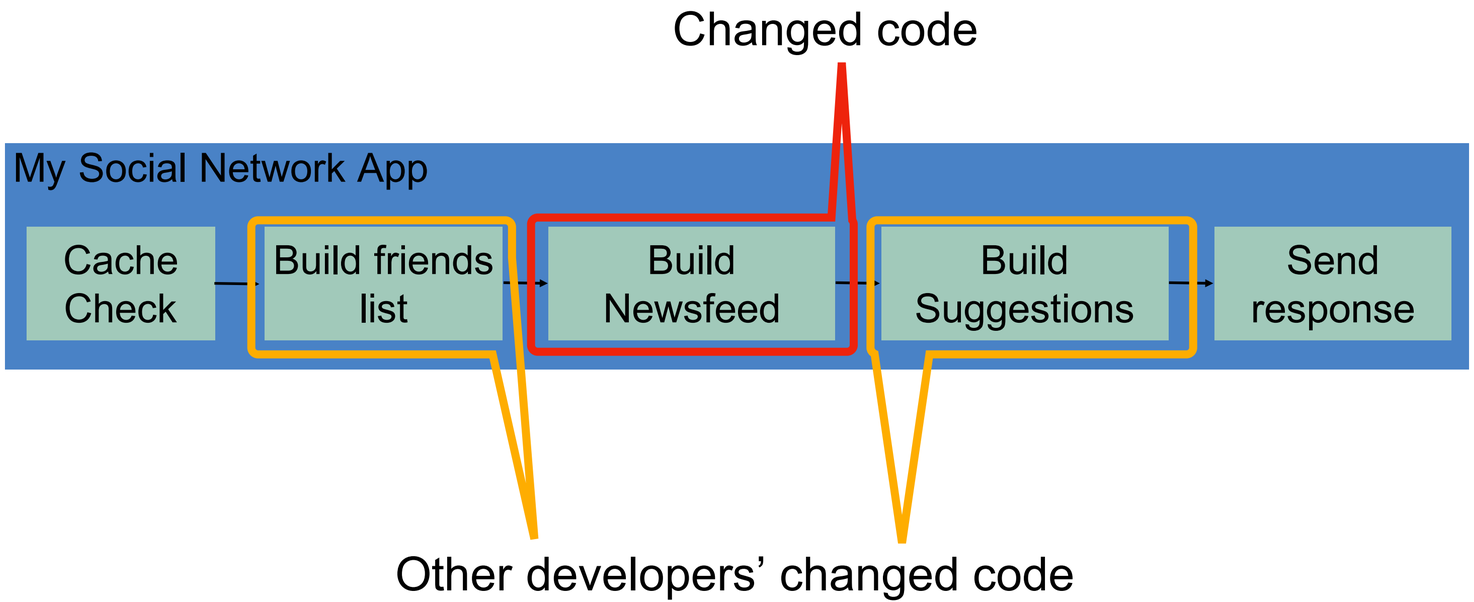




How do we apply continuous integration?

Testing the right things at the right time

- Do we integrate changes immediately, or do a pre-commit test?
- Which tests do we run when we integrate?
- How do we compose the system under test at each point?

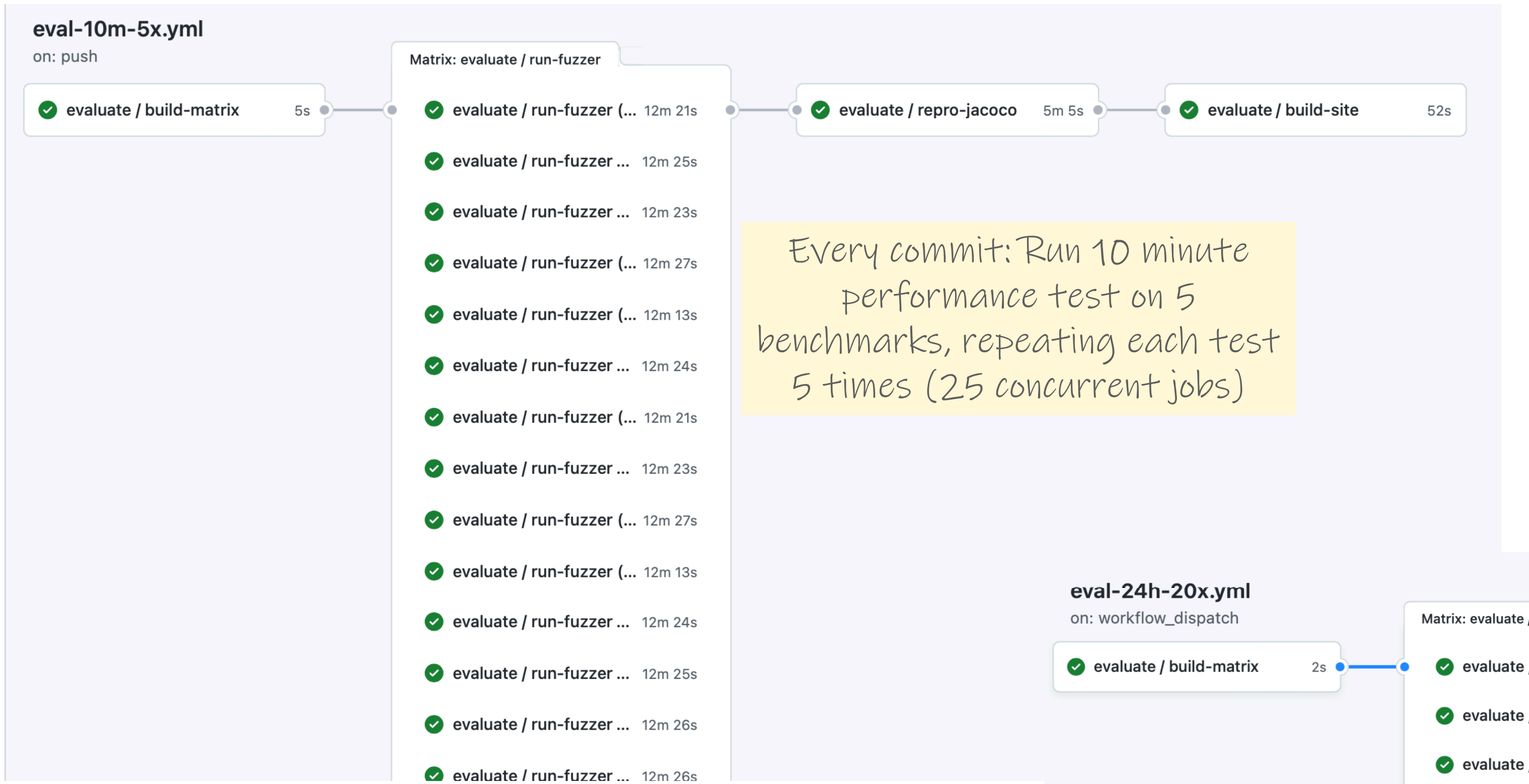


Use Scalable Cloud Resources for CI

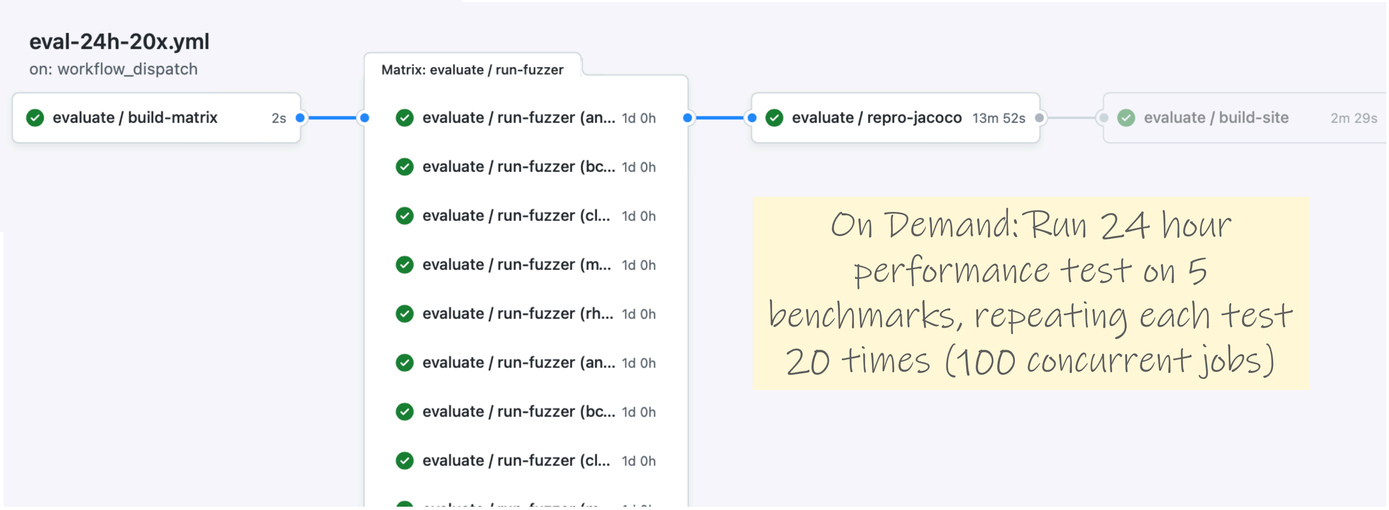
Example: Developing a Fuzzer

- “Fuzzers” are automated testing systems that aim to automatically generate inputs to programs that cover code and reveal bugs
- Fuzzers are non-deterministic: to evaluate with confidence, need repeated, long-running trials
- Evaluating fuzzers is time consuming, determining which changes impact performance is confusing
- How to run experiments in the cloud?

CI Pipelines Automate Performance Testing



Every commit: Run 10 minute performance test on 5 benchmarks, repeating each test 5 times (25 concurrent jobs)



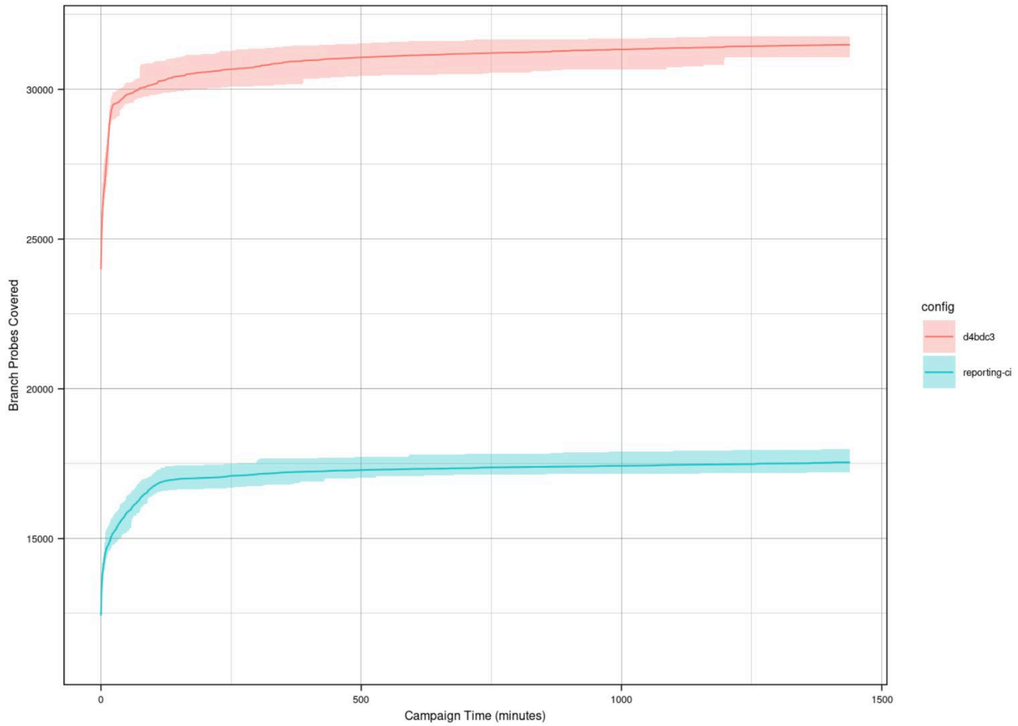
On Demand: Run 24 hour performance test on 5 benchmarks, repeating each test 20 times (100 concurrent jobs)

https://github.com/neu-se/CONFETTI/actions

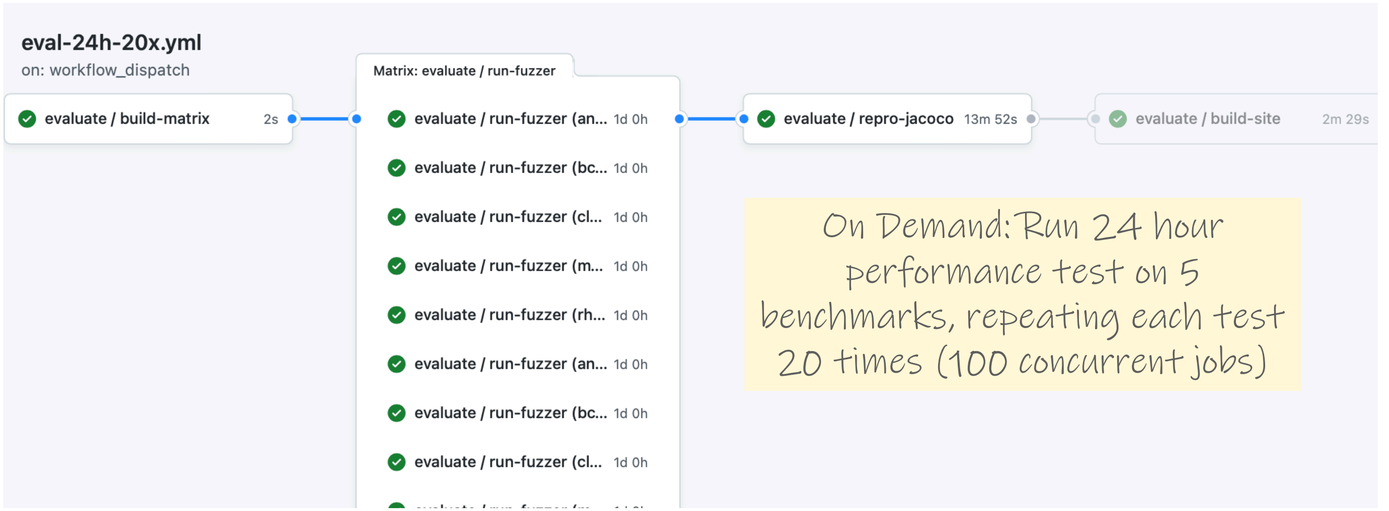
CI Pipelines Automate Performance Testing

closure

Branch Probes Over Time



Download this graph as PDF



https://github.com/neu-se/CONFETTI/actions

Continuous Integration in Practice

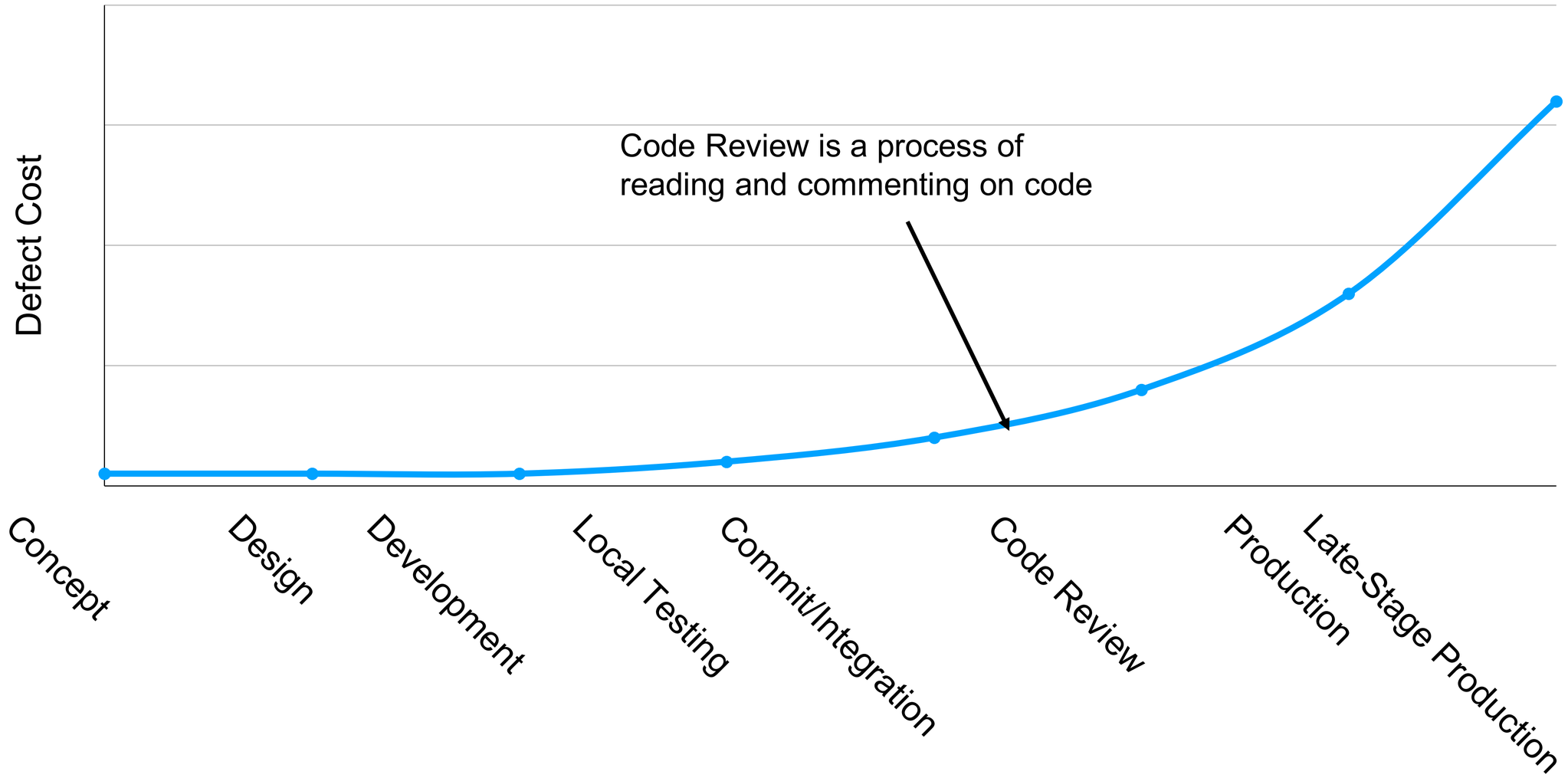
Large scale example: Google TAP

- >50,000 unique changes per-day, > 4 billion test cases per-day
- Pre-submit optimization: run fast tests for each individual change (before code review). If fast tests pass, allow the merge to continue
- Then: run all affected tests; “build cop” monitors and acts immediately to roll-back or fix
- Build cop monitors integration test runs
- Average wait time to submit a change: 11 minutes

“Software Engineering at Google: Lessons Learned from Programming Over Time,” Wright, Winters and Manshreck, 2020 (O’Reilly)

Cost to Fix a Defect Over Time

Rough estimate



Code Review should be a Formal Process

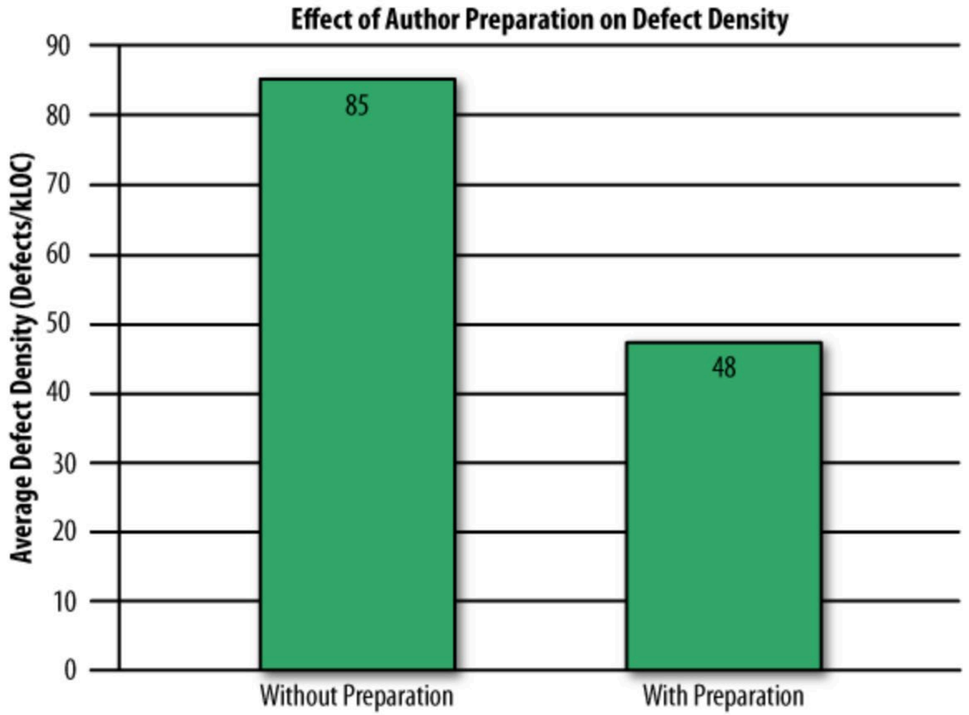
- A code review is the process in which the author of some code is asked to explain it to their peers:
  - What purpose the code has;
  - How the code accomplishes this purpose;
  - How the author is confident of this information,
    - E.g., show results of running tests (CI results)
- A code review often concerns a code *change* (“diff”)

Why should we perform code review?

- Code review increases breadth of knowledge of code:
  - Other people ”know” the code;
  - Easier to handle someone cycling off project.
- Verbalizing decisions improves their quality:
  - The process of writing an explanation encourages critical thinking.
- Code reviews improve quality of code base:
  - Knowing code will be reviewed pushes developers to make code more presentable and understandable.

Self-Review is no substitute to Peer Review

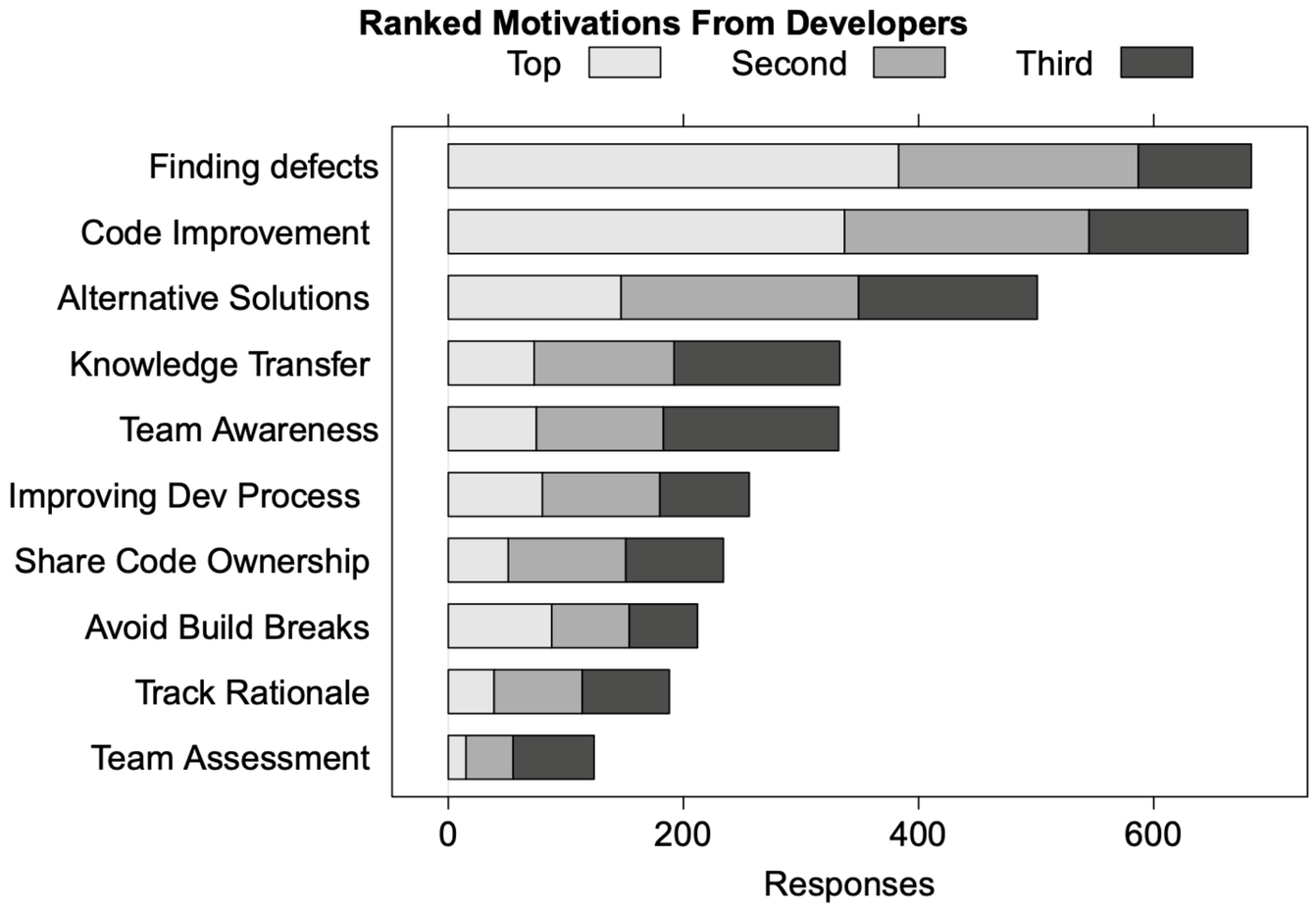
Study of 300 reviews at Cisco in 2006



Even if developers pre-review their code, many defects still found in peer review

"Best Kept Secrets of Peer Code Review", Jason Cohen, SmartBear Software, 2006

Code Reviews Have Many Benefits (Microsoft)



"Expectations, Outcomes, and Challenges of Modern Code Review", Bacchelli & Bird, ICSE 2013

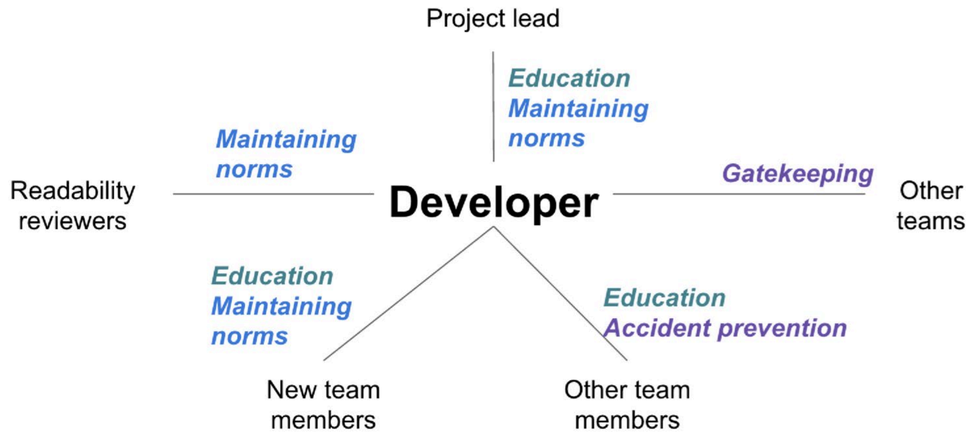
Code Reviews Descend from Traditional Code Inspection

- Formal process of reading through code as a group;
- Applied to all project documents;
- A 3-5 person team reads the code aloud and explains what is being done;
- Each person has a specific role (moderator, reviewer, reader, scribe, observer, author)
- Usually a 60 minute meeting;
- Less efficient (defects/cost) than modern review processes.
- Very waterfall.
- Traceable, measurable

Many Stakeholders can Benefit from Code Review

Reviewers might be…

- An owner of the code being changed or added to
- Someone to verify that the code meets standards.
- Someone to ensure documentation is consistent.
- Other people:
  - Interested in this code base;
  - Experts in the code base.

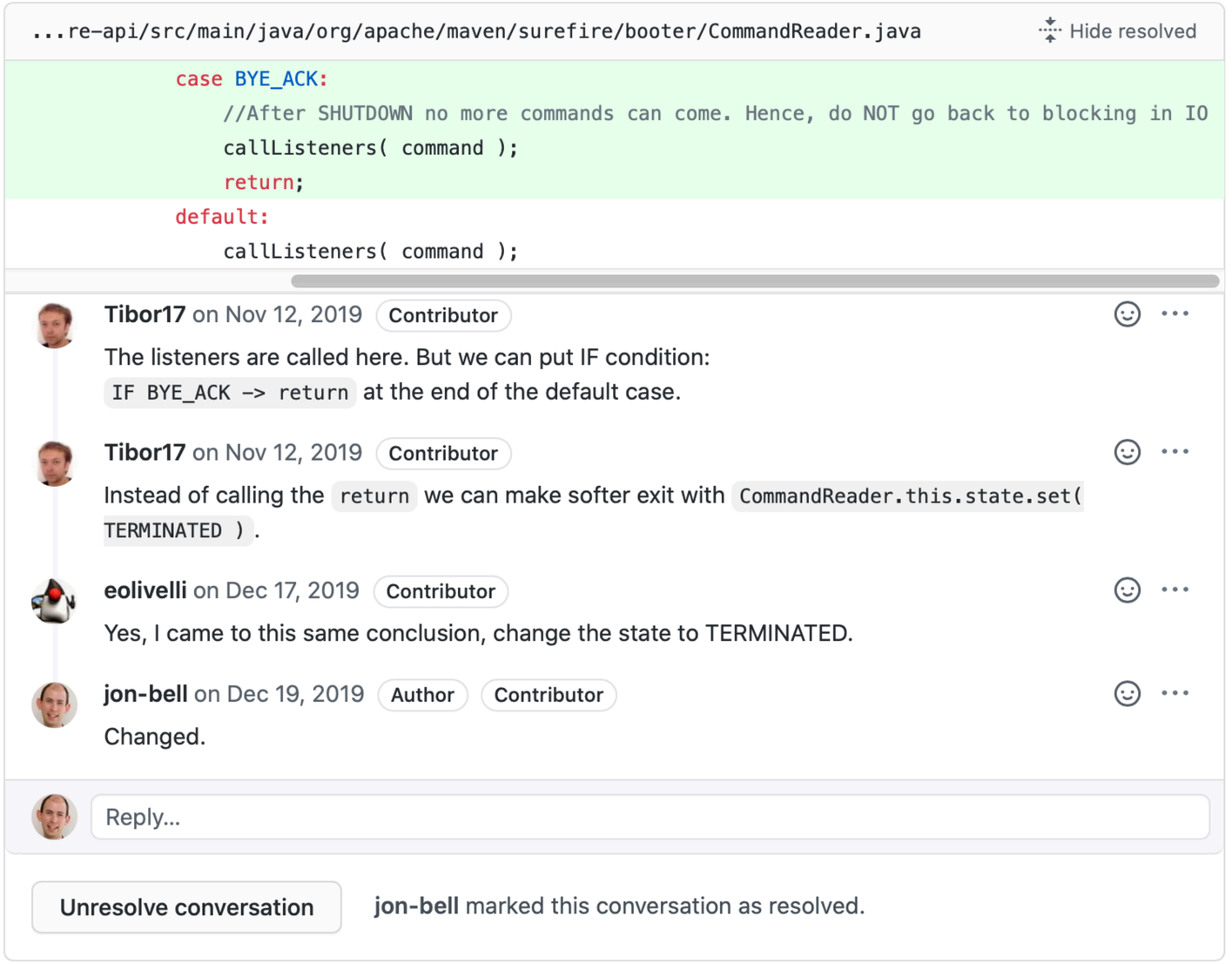


"Modern Code Review: A Case Study at Google", Sadowski et al, ICSE 2018

Code Review: How

- At Google, reviewers get access to changes, explanation and all relevant test results: review is asynchronous.
- Elsewhere reviews can be in person:
  - More heavyweight, cannot be as common.
- Review must be professional and impersonal:
  - No one is being “attacked” (or, no one *should* be).
- Don’t rehash design arguments (defer to author).
- All suggestions and criticisms must be addressed:
  - At least in the negative.

Code Review: Example on Pull Request



Code Review: Sample Check-List

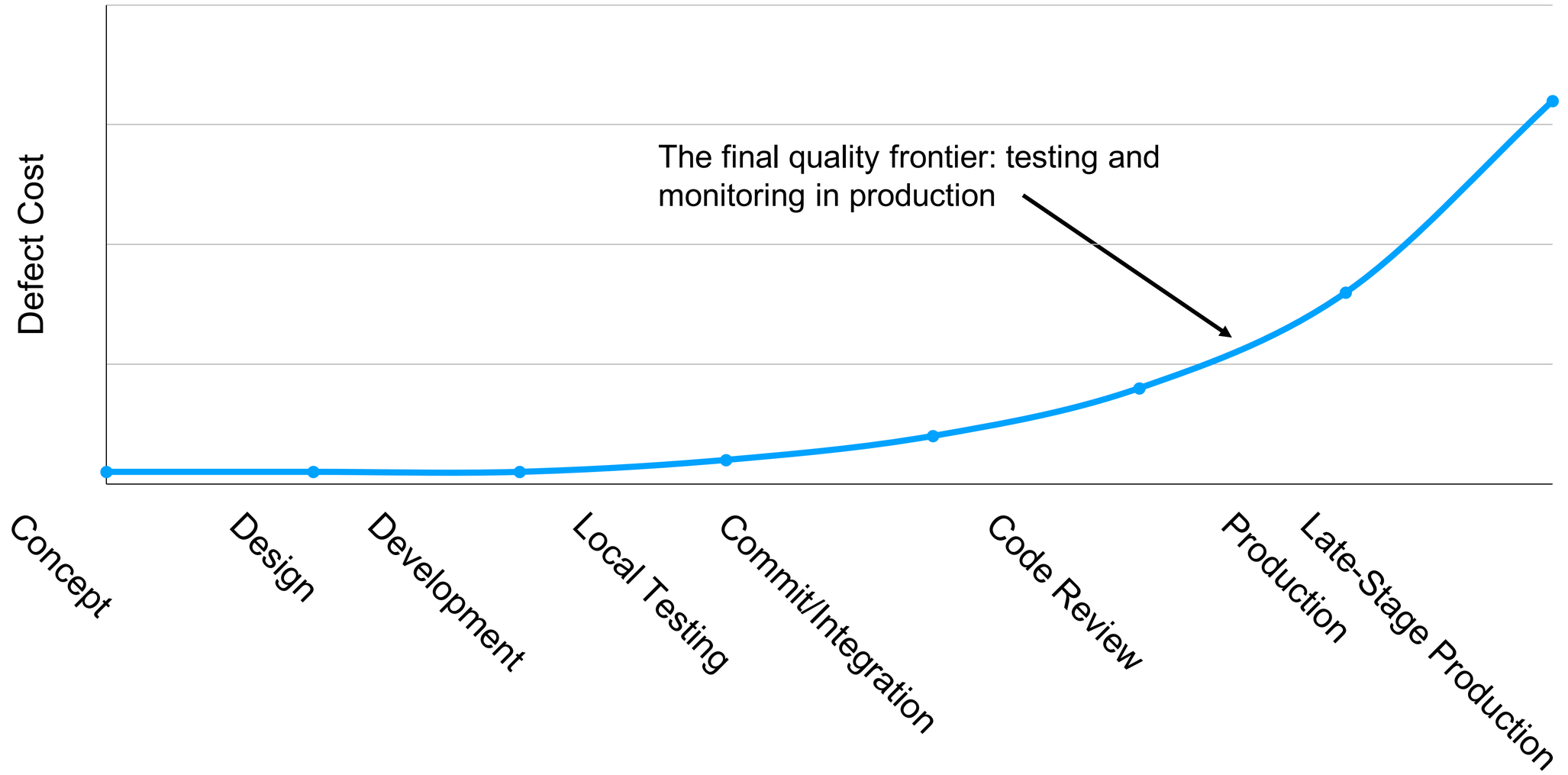
- Am I able to understand the code easily?
- Does the code follow our style guidelines?
- Is the same code duplicated more than once?
- Is this file (or change) too big?
- Does this code meet our non-functional requirements?
- Is this code maintainable?
- Does this code have unintended side-effects?

Code Reviews and Programmer's Ego

- Code review means someone's looking over your work
- You might have some attachment to it
- Criticisms: sometimes hard not to take personally
- Acknowledge a criticism and move on
- Acknowledgment doesn't imply that the author agrees with the content of the criticism
- Remember: The review is not about *you*, the goal is to improve code

Cost to Fix a Defect Over Time

Rough estimate



Case Study of a Failed Deployment: Knight Capital

Knightmare: A DevOps Cautionary Tale

D7 DevOps April 17, 2014 6 Minutes

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story to demonstrate the importance making deployments fully automated and repeatable as part of a DevOps/Continuous Delivery initiative. Since that conference I have been asked by several people to share the story through my blog. This story is true – this really happened. This is my telling of the story based on what I have read (I was not involved in this).

This is the story of how a company with nearly $400 million in assets went ban
minutes because of a failed deployment.

"In the week before go-live, a Knight engineer manually deployed the new RLP code in SMARS to its eight servers. However, the engineer made a mistake and did not copy the new code to one of the servers. Knight did not have a second engineer review the deployment, and neither was there an automated system to alert anyone to the discrepancy. "

https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html

What Could Knight Capital Have Done Better?

- Use capture/replay testing instead of driving market conditions in a test
- Avoid including “test” code in production deployments
- Automate deployments
- Define and monitor risk-based KPIs
- Create checklists for responding to incidents

Deployment Philosophy: Instagram

“Faster is safer”

“If stuff blows up it affects a very small percentage of people”

Instagram cofounder and CTO Mike Krieger

https://www.fastcompany.com/3047642/do-the-simple-thing-first-the-engineering-behind-instagram

Continuous Delivery

“Faster is safer”: Key values of continuous delivery

- Release frequently, in small batches
- Maintain key performance indicators to evaluate the impact of updates
- Phase roll-outs
- Evaluate business impact of new features

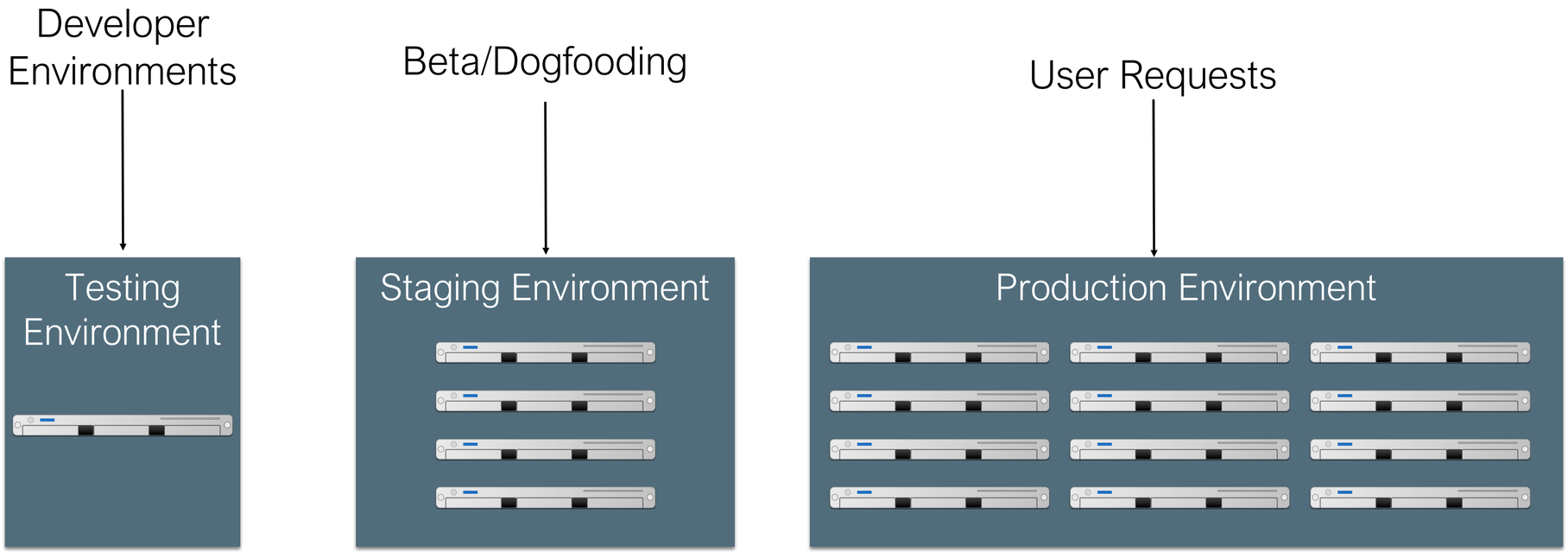
Staging Environments

Enabling Continuous Delivery

- As software gets more complex with more dependencies, it's impossible to simulate the whole thing when testing
- Idea: Deploy to a complete production-like environment, but don't have everyone use it
  - Examples:
    - “Eat your own dogfood”
    - Beta/Alpha testers
- Lower risk if a problem occurs in staging than in production

Test-Stage-Production

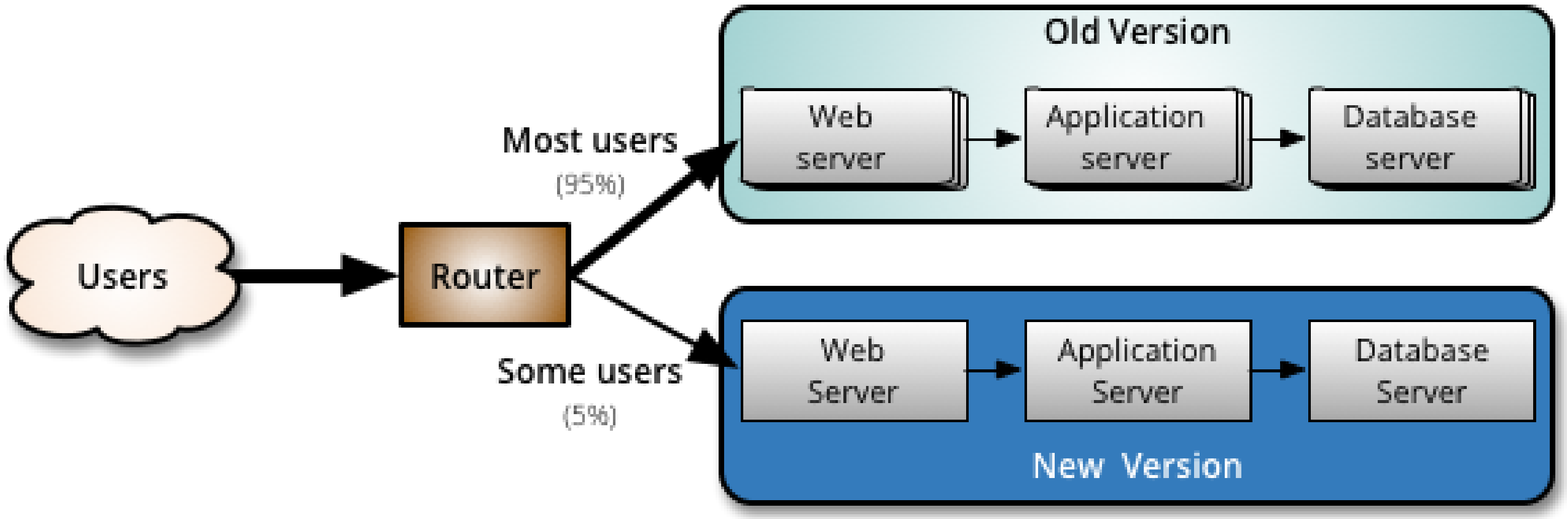
Continuous Delivery in Action



Q/A takes place in each stage (including production!)

A/B Deployments with Canaries

Mitigating risk in continuous delivery

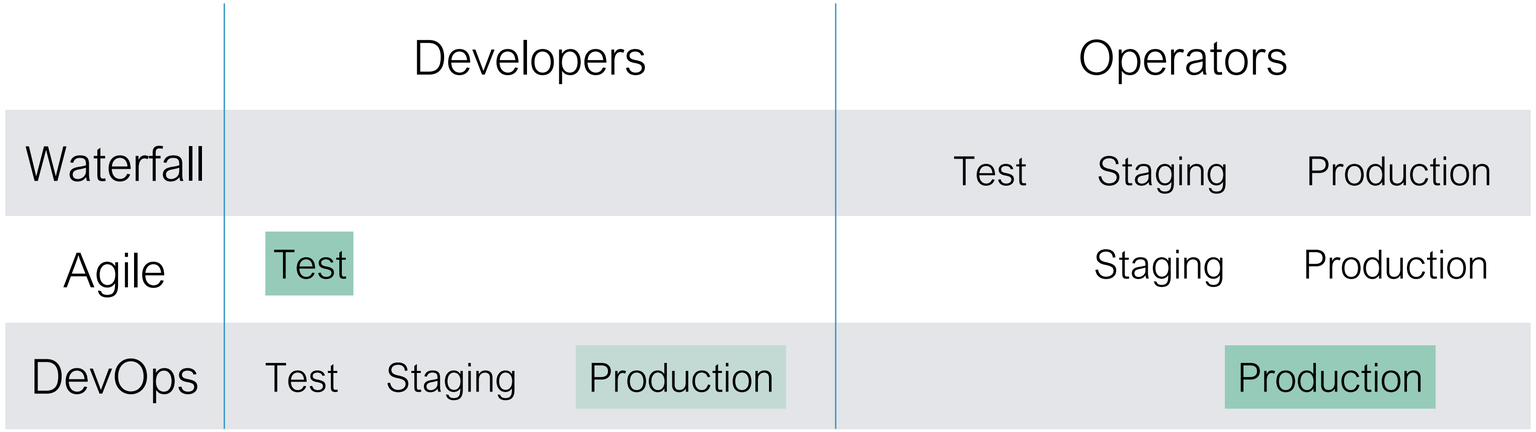


Monitor both:
But minimize impact of problems in new version

Operations Responsibility

DevOps in a slide

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
  - Test, Staging, Production
- Whose job is it?



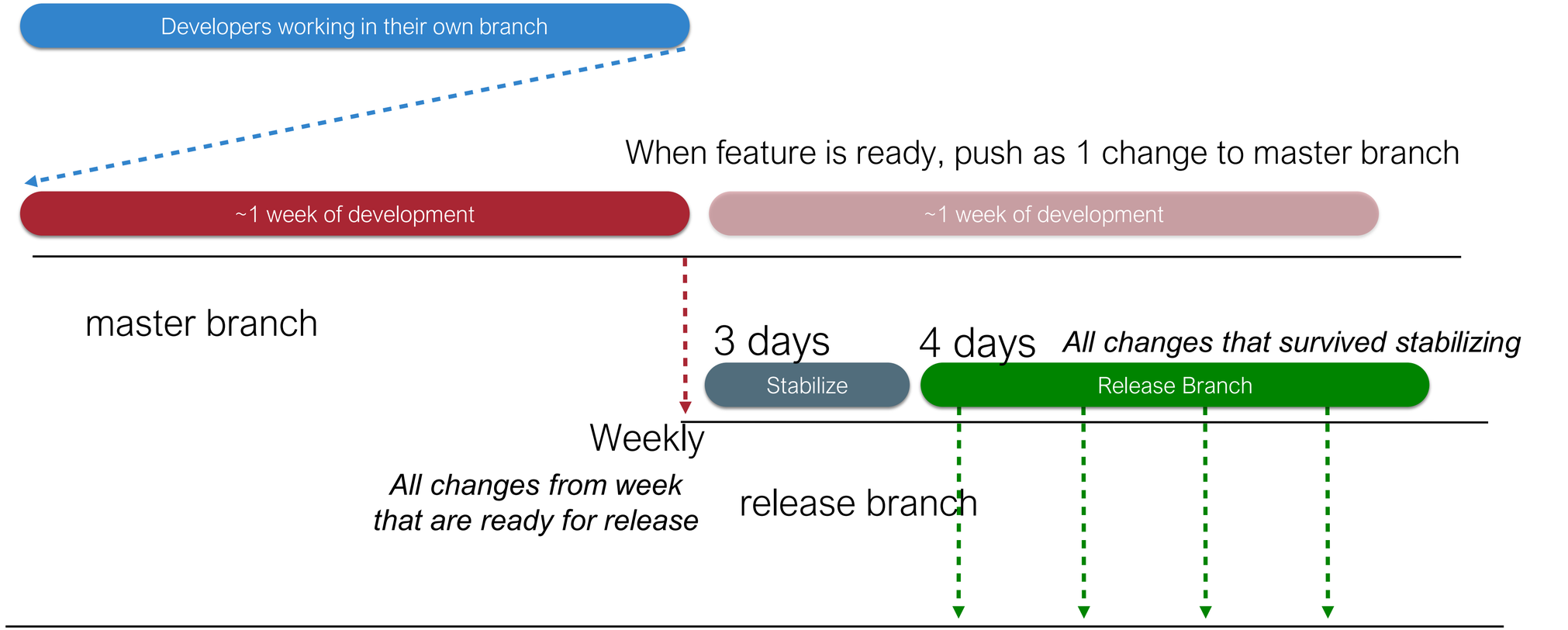
Release Pipelines

How quickly is my change deployed?

- Even if you are deploying every day, you still have some latency
- A new feature I develop today won't be released today
- But, a new feature I develop today can begin the **release pipeline** today (minimizes risk)
- **Release Engineer**: gatekeeper who decides when something is ready to go out, oversees the actual deployment process

Deployment Example: Facebook.com

Pre-2016



Deployment Example: Facebook.com

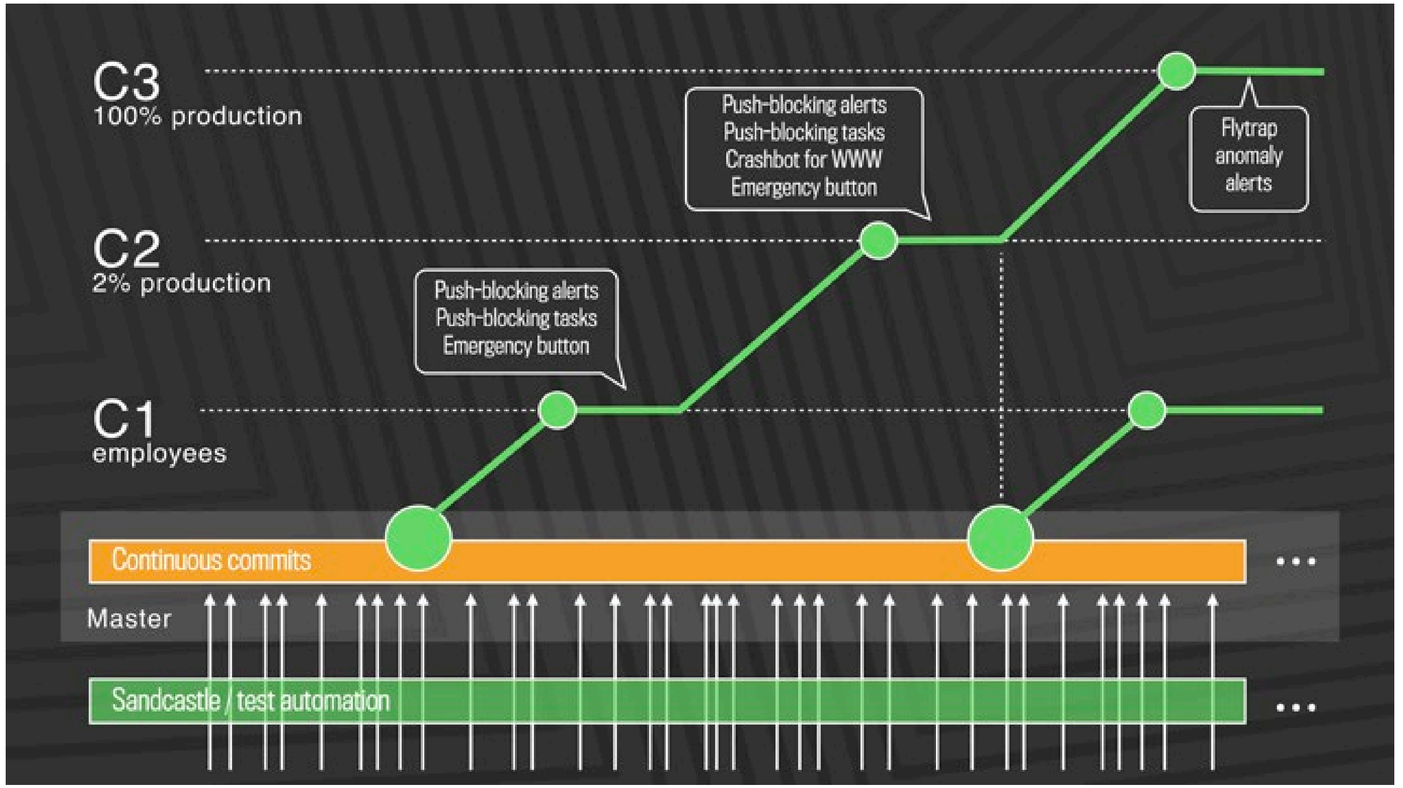
Chuck Rossi, Director Software Infrastructure & Release Engineering @ Facebook

"Our main goal was to make sure that the new system made people's experience better — or at the very least, didn't make it worse. After almost exactly a year of planning and development, over the course of three days in April 2017 **we enabled 100 percent of our production web servers to run code deployed directly from master**."

"Rapid release at massive scale" https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

Deployment Example: Facebook.com

Post-2016: Truly continuous releases from master branch



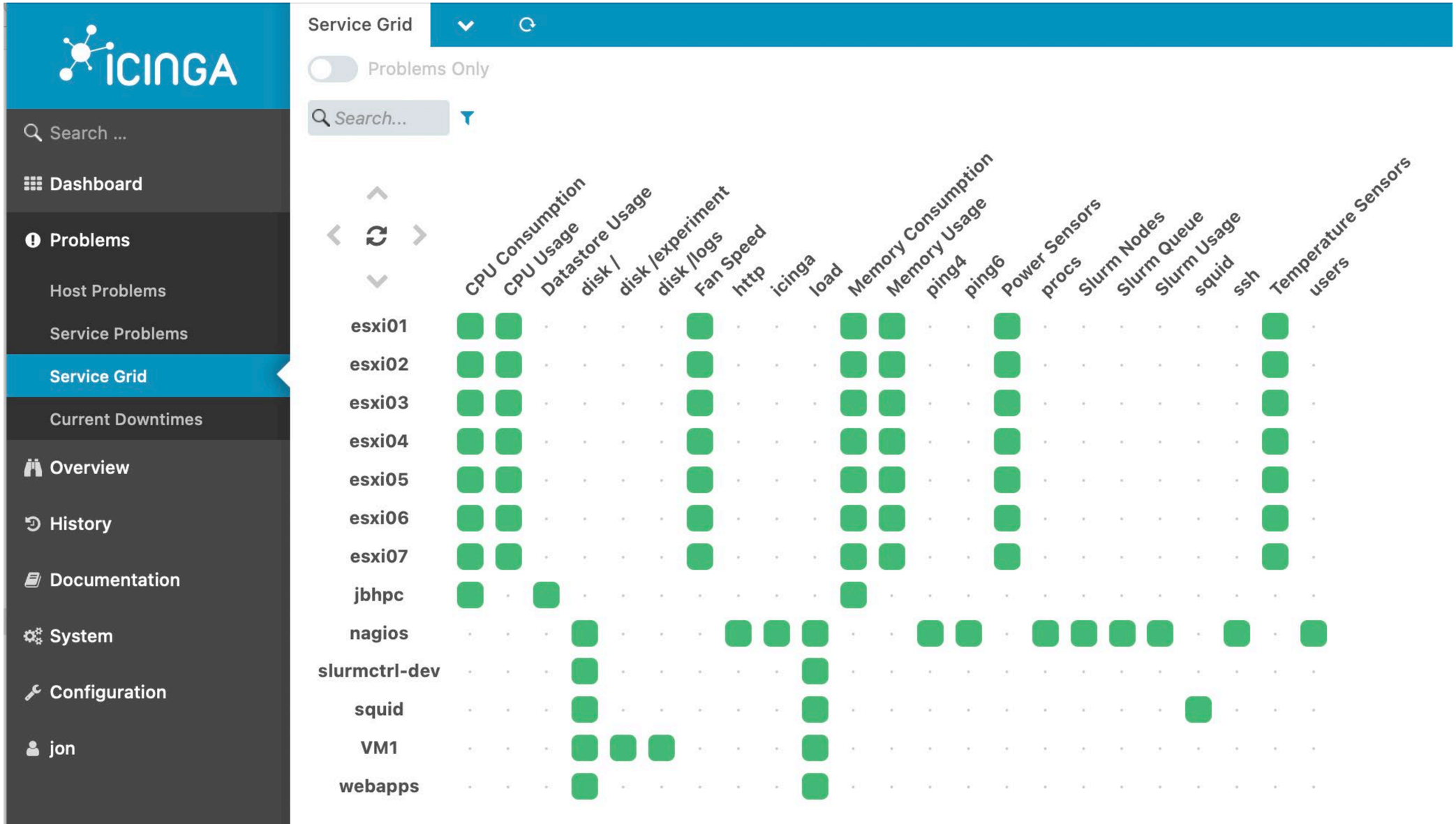
https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

Monitoring

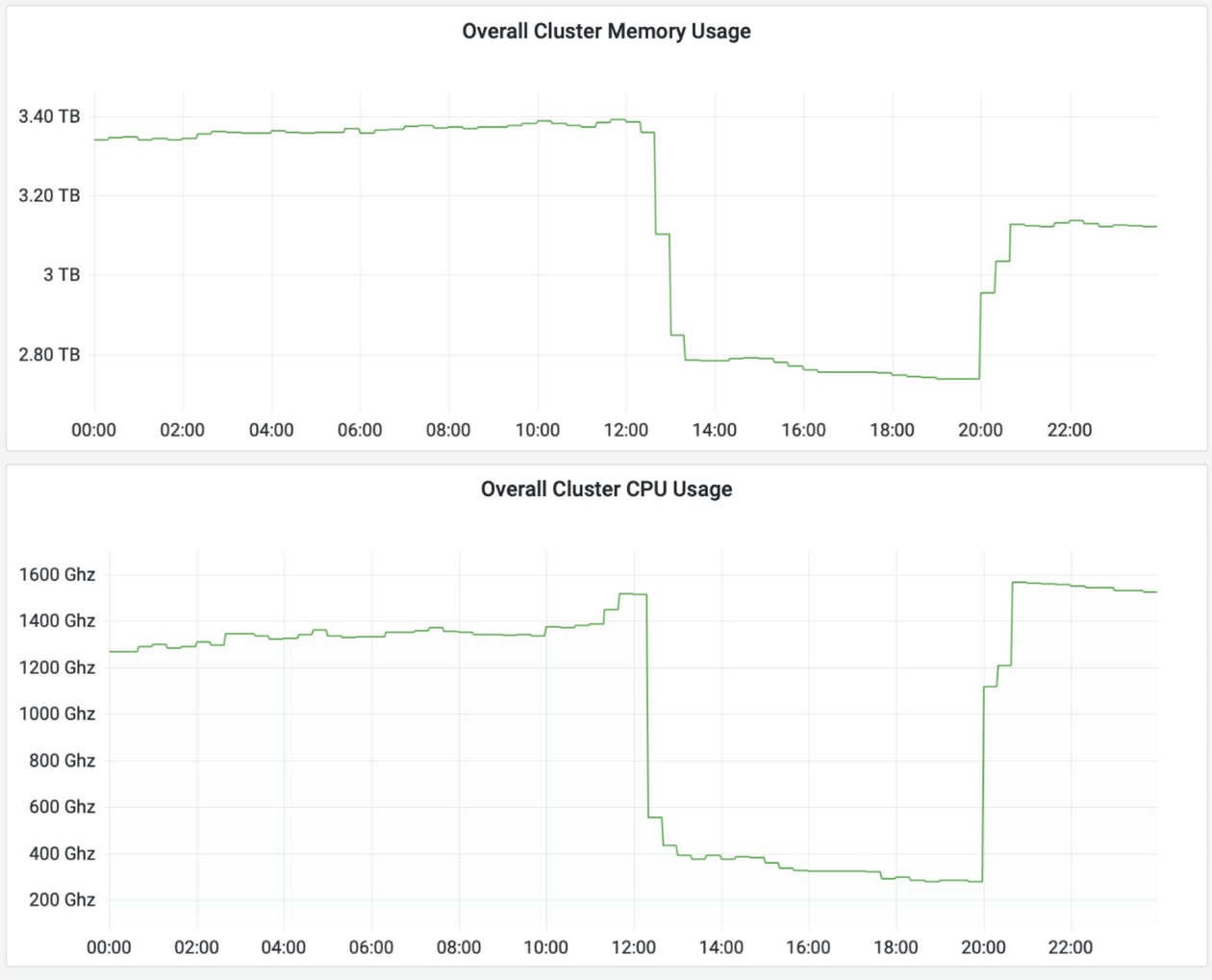
The last step in continuous deployment: track metrics

- Hardware
  - Voltages, temperatures, fan speeds, component health
- OS
  - Memory usage, swap usage, disk space, CPU load
- Middleware
  - Memory, thread/db connection pools, connections, response time
- Applications
  - Business transactions, conversion rate, status of 3rd party components

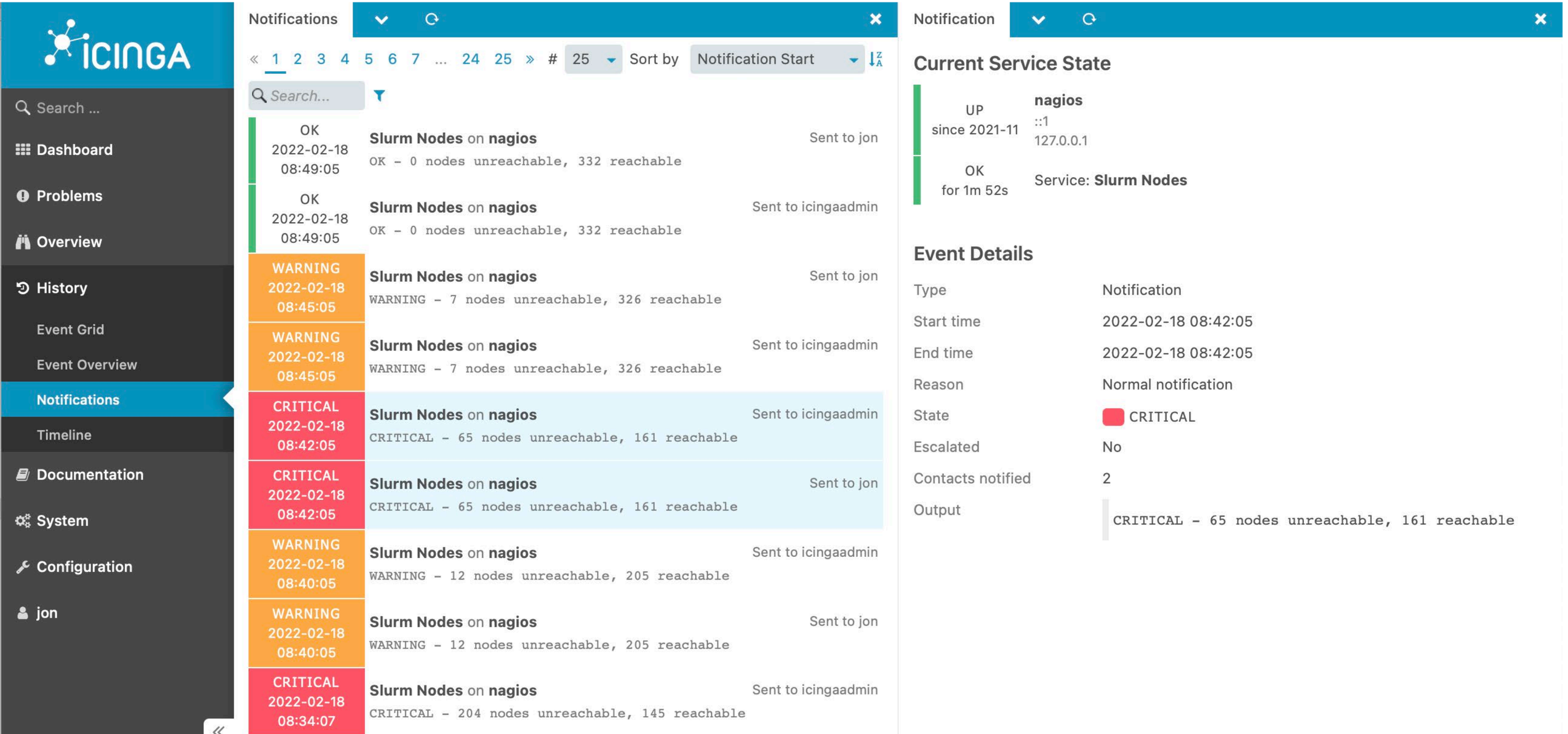
Monitoring Services Aggregate System Status



Monitoring Dashboards Help Gather Insights



Monitoring Services Take Automated Actions



Monitoring Services Take Automated Actions

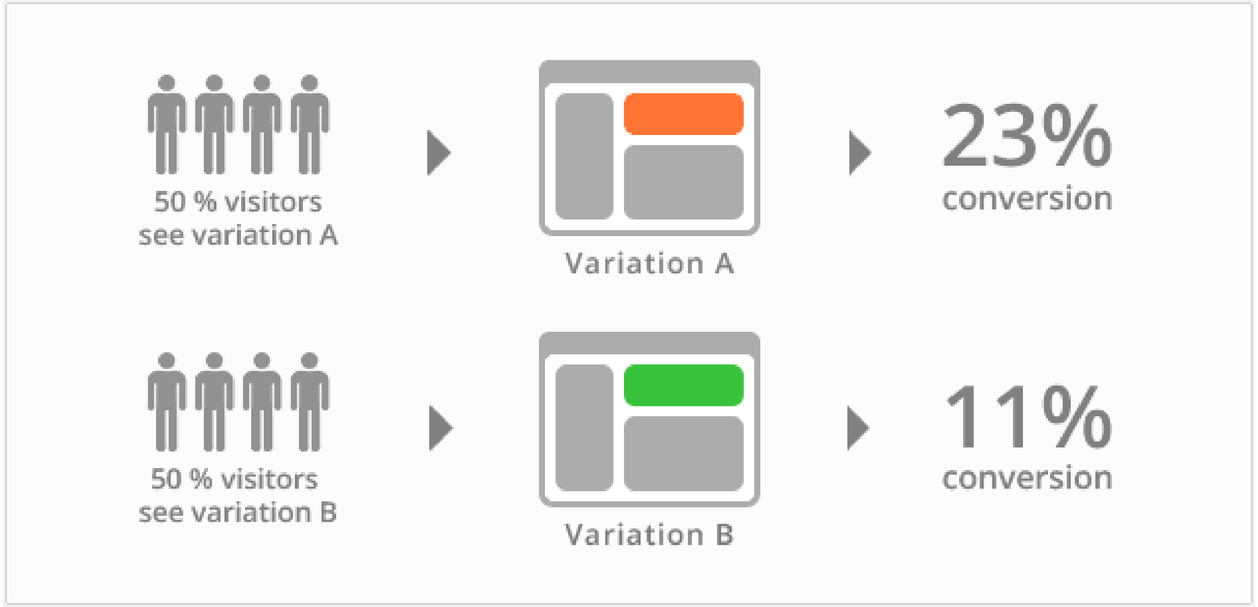
Automatically detecting irregular behavior at Netflix

https://www.youtube.com/watch?v=qyzymLIj9ag

Usability Testing in Continuous Development

A/B Testing

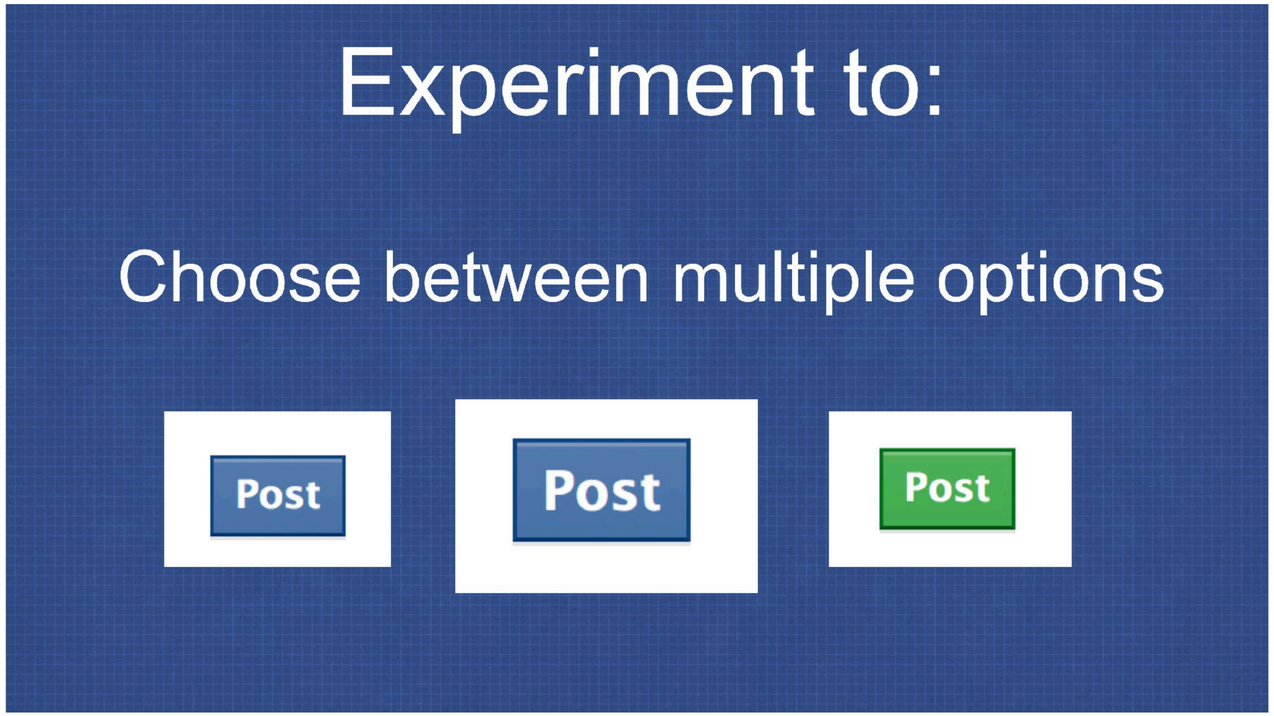
- Ways to test new features for usability, popularity, performance without a focus group
- Show 50% of your site visitors version A, 50% version B, collect metrics on each, decide which is better



Usability Testing in Continuous Development

A/B Testing: PlanOut from Facebook ("N=10^9 user study")

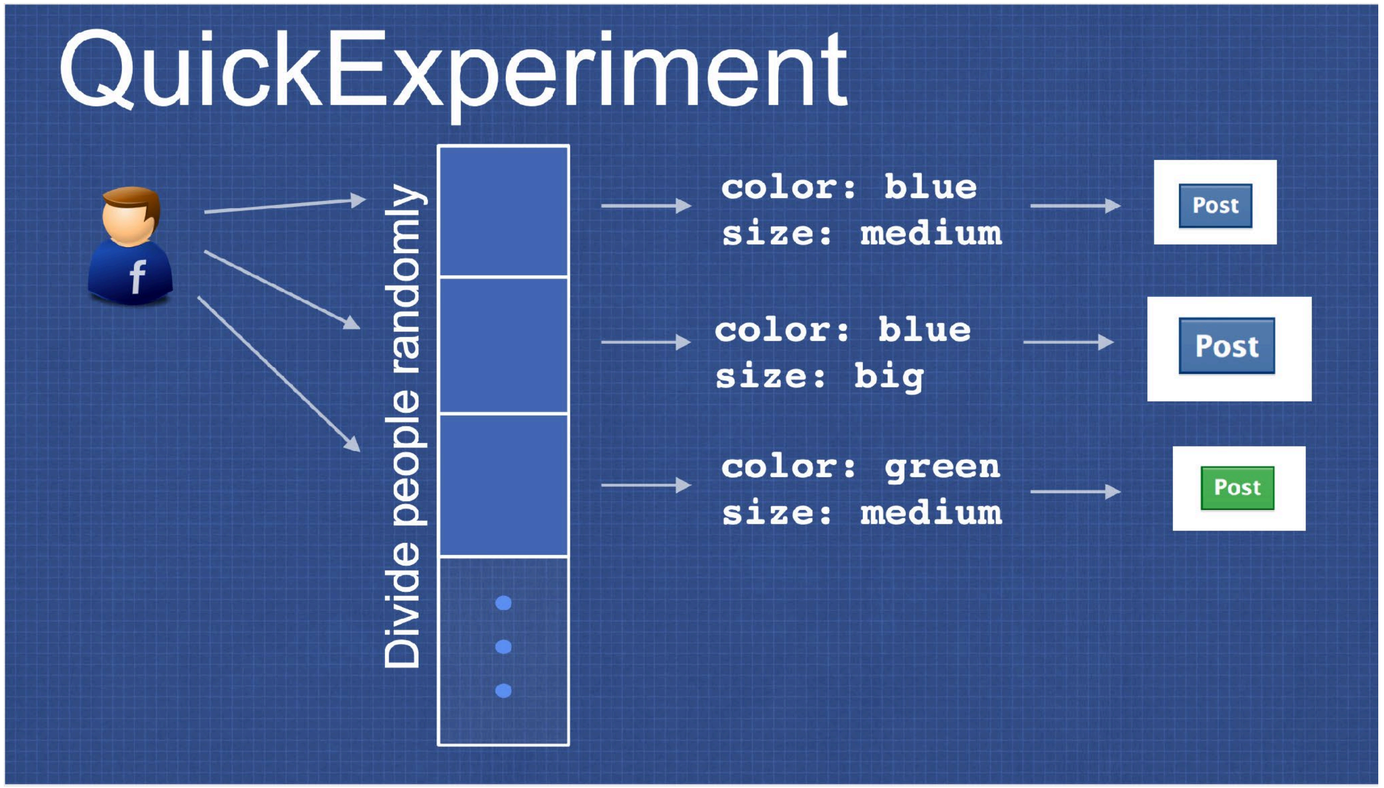
- Used to test advertising strategies (and Facebook functionality)
- Segment audience and define KPIs, collect results



https://github.com/facebook/planout https://www.slideshare.net/optimizely/opti-con-2014-automated-experimentation-at-scale

Usability Testing in Continuous Development

A/B Testing: PlanOut from Facebook ("N=10^9 user study")

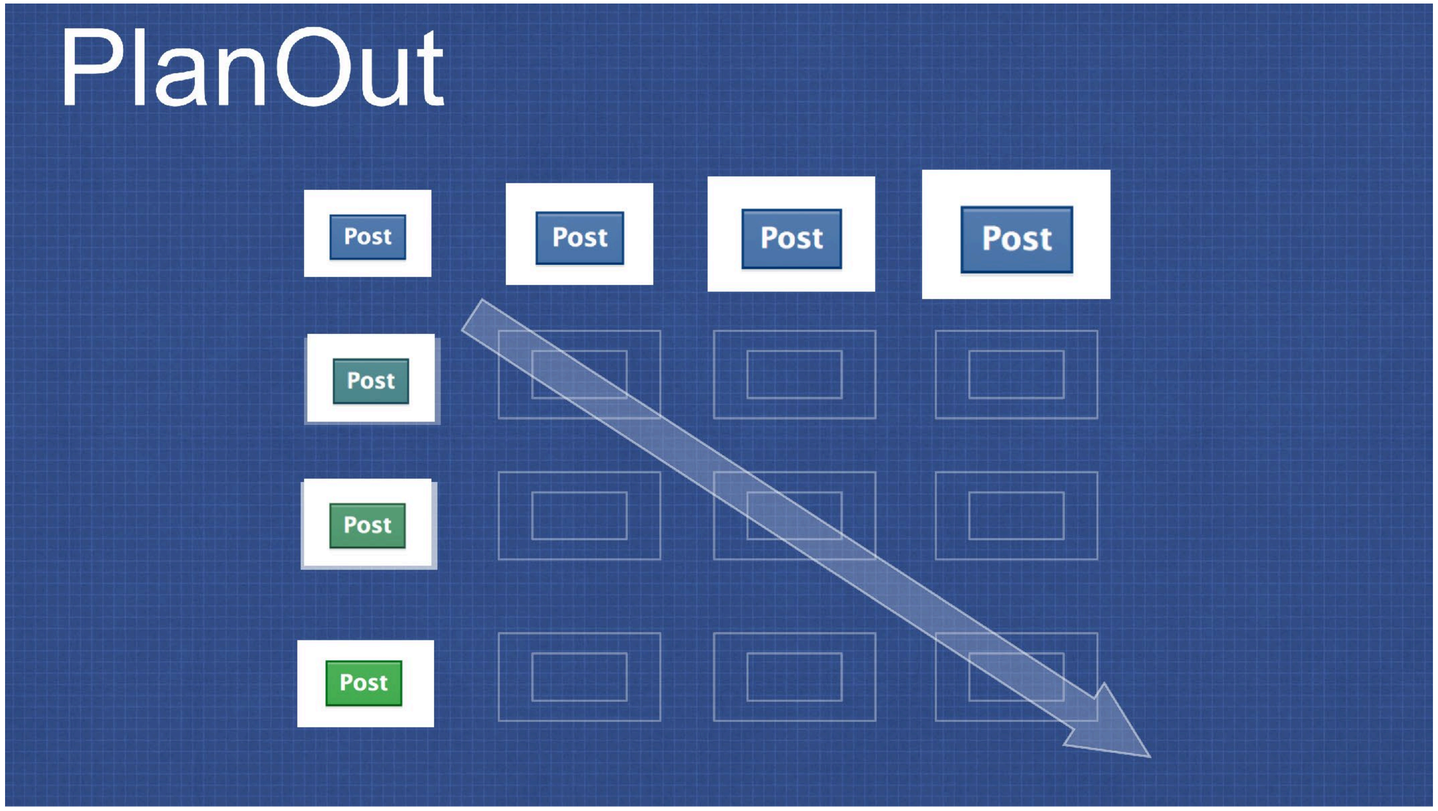


https://github.com/facebook/planout

https://www.slideshare.net/optimizely/opti-con-2014-automated-experimentation-at-scale

Usability Testing in Continuous Development

A/B Testing: PlanOut from Facebook (“N=10^9 user study”)

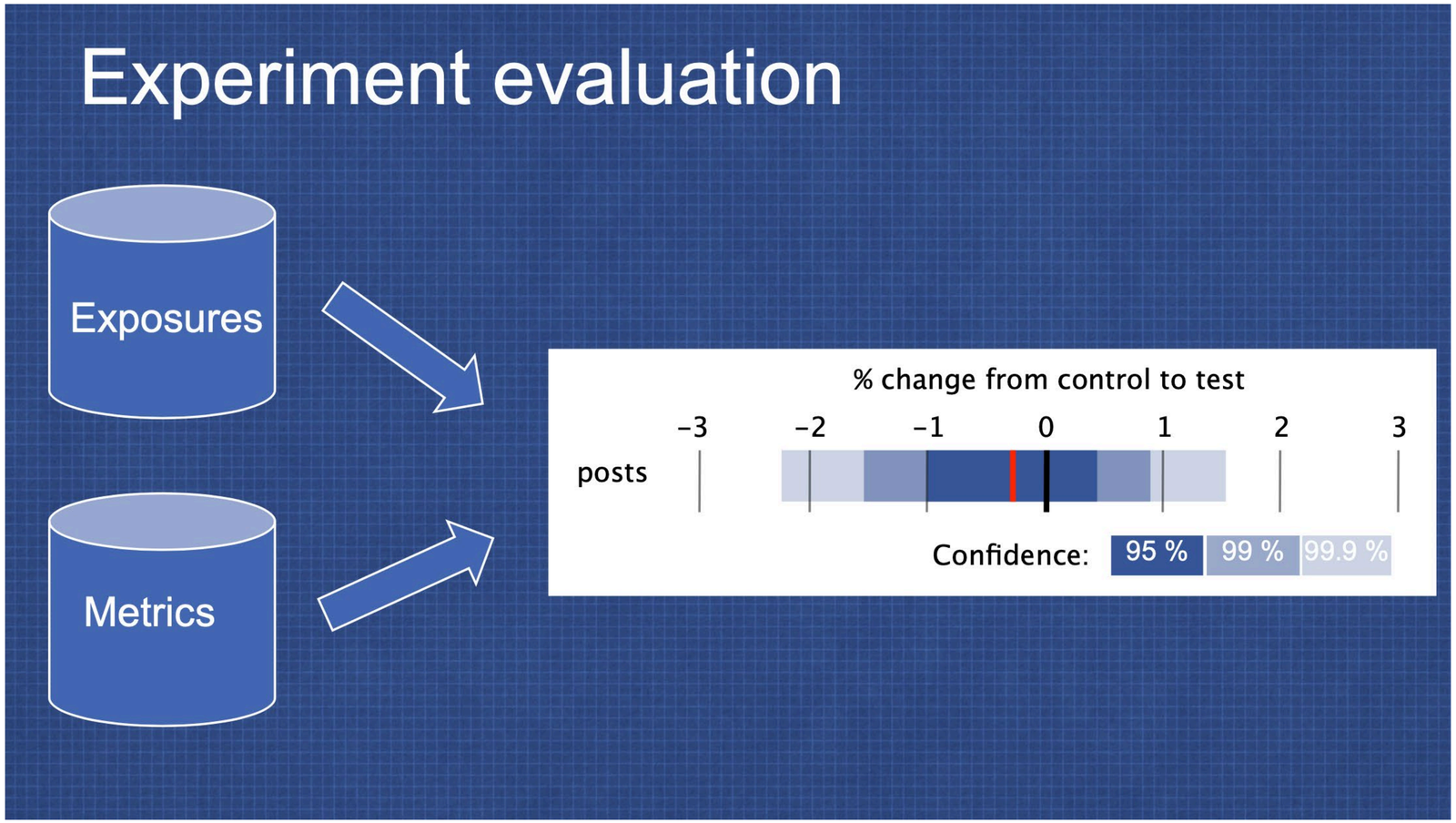


https://github.com/facebook/planout

https://www.slideshare.net/optimizely/opti-con-2014-automated-experimentation-at-scale

Usability Testing in Continuous Development

A/B Testing: PlanOut from Facebook ("N=10^9 user study")



https://github.com/facebook/planout

https://www.slideshare.net/optimizely/opti-con-2014-automated-experimentation-at-scale

Beware of Metrics

McNamara Fallacy

- Measure whatever can be easily measured
- Disregard that which cannot be measured easily
- Presume that which cannot be measured easily is not important
- Presume that which cannot be measured easily does not exist

What Could Knight Capital Have Done Better?

- Use capture/replay testing instead of driving market conditions in a test
- Avoid including “test” code in production deployments
- Automate deployments
- Define and monitor risk-based KPIs
- Create checklists for responding to incidents

Learning Objectives for this Lesson

By the end of this lesson, you should be able to…

- Describe how continuous integration helps to catch errors sooner in the software lifecycle
- Describe the benefits of a culture of code review
- Describe strategies for performing quality-assurance on software as and after it is delivered