Intro to CI/CD

Tara Gu
Software Engineer at Google

About me

Duke undergrad BME 13’, masters ECE 15’

Google internship

–> IBM fulltime

–> briefly at Wish

–> back at IBM

–> Red Hat

–> Twitter

–> Google

Please feel free to connect with me:

tara.weiqing@gmail.com

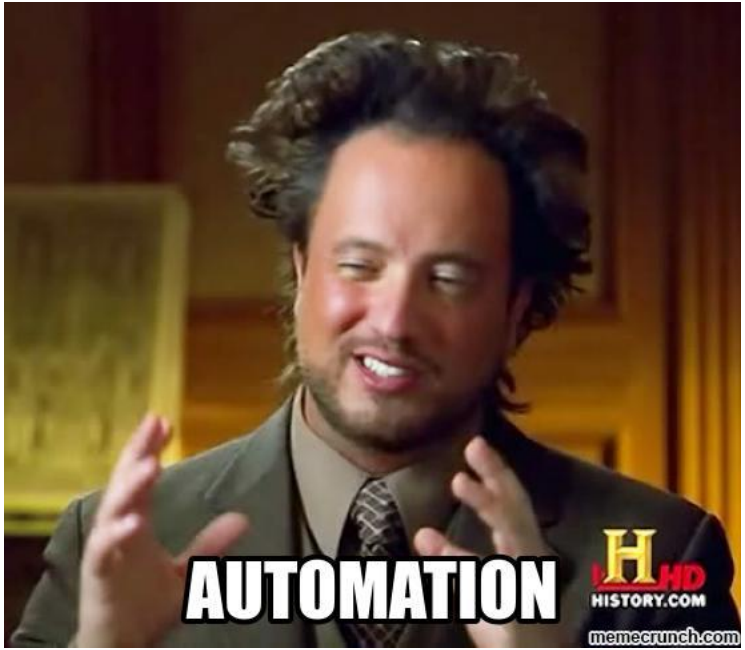
https://linkedin.com/in/taraguduke

Agenda

- What is CI?
- What is CD?
- Why does CI/CD matter to you?

What is CI (Continuous Integration)

- Goal: Quickly check if the changes you make will break anything
- Each "changeset" is verified by an automated build
  - Style checkers
  - Testing
    - Unit tests
    - Integration tests
  - Unit test change in coverage
- Saves time on code review

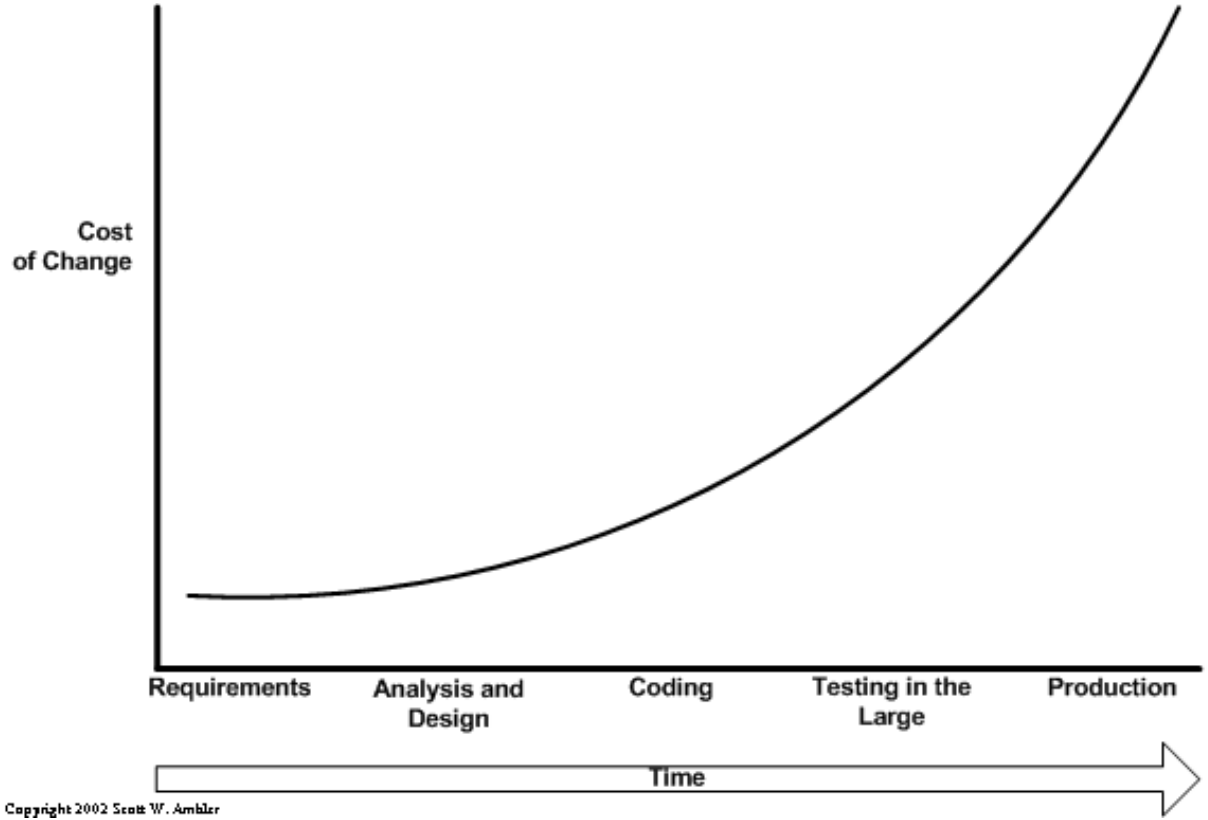


Testing

- Unit tests
  - Test individual functions
  - Code-level
- Integration tests
  - Also known as end-to-end or e2e tests
  - Test a user-facing functionality
  - System-level
- Unit test change in coverage
- Flaky tests: non-consistent failure
  - Why would flaky tests occur
    - Race conditions
    - An underlying dependency being unreliable

CI – Continuous Integration (cont.)

- Test automation is a pre-req
  - Not everything can be (easily) tested in an automated fashion (e.g. UI)
  - How consistent are the test results? (e.g. flaky test)
- Reducing cost of change



CI – Continuous Integration (cont.)

- Example pull request in an open source project (Kubernetes): https://github.com/kubernetes/kubernetes/pull/117043
  - What are "jobs"?
    - A collection of tests or verification
    - A Kubernetes concept
  - Job logs/history
    - Many of these jobs can be run locally (project README should include instructions)
    - Shortest jobs run first WHY?
    - Logs: helpful for debugging test failures and errors
  - Commands to interact with prow bot
    - Start running all jobs required to pass
    - Rerun certain test jobs
  - Dashboard showing flaky tests: https://testgrid.k8s.io/sig-release-master-blocking

Dashboards for CI (of an open source project)

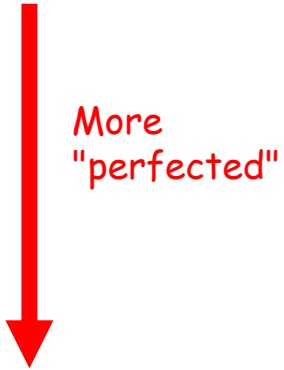
- Deck: prow.k8s.io
  - audience? developer, release manager that's tracking progress of a release
  - kubernetes project (not the product, but the kubernetes source repository)
- Testgrid: https://testgrid.k8s.io
  - what are all these blocks: sub projects or teams
  - Release blocking dashboard
  - Flaky tests

Up until this point, the code only exists on a feature branch…

When we want to deploy a code change, how do we make sure the change is problem-free and ready for customer usage?

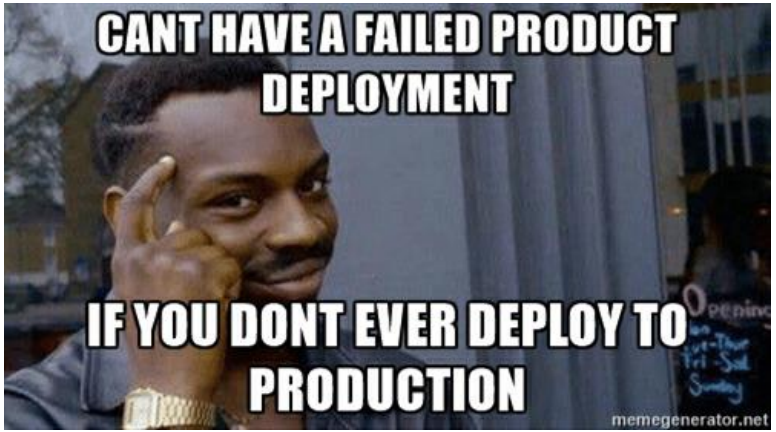
Environments

- What's an environment?
  - A group of machines that has its own:
    - Hardware
    - Software deployed
    - Permission of who/when can push new software versions
- Different kinds:
  - Development/dev/autopush
  - Staging
  - Pre-production
  - Canary
  - Production



CD - Continuous Delivery

- Once your code is merged (the end result of CI), actually deploy your code
  - Ensure it’s packaged with everything it needs (into a docker image for example) to deploy to any environment at any time
  - Deploy to pre-production environment or canary before production
- Feature flags: incomplete features do not affect customers in production
- Should be so routine and low-risk that the team is comfortable doing them anytime

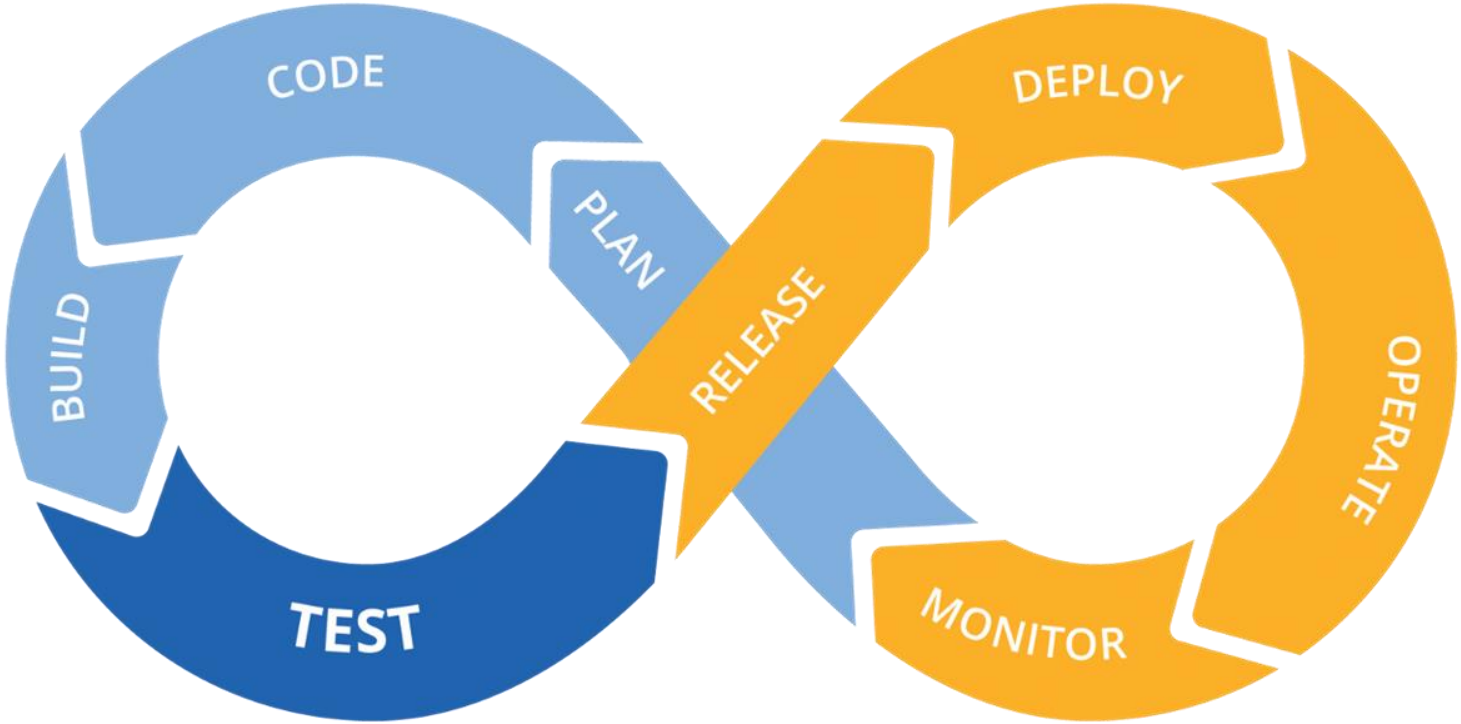


The non-code aspects of "delivery"

- If it's a user facing product, may need to coordinate with media/press department for a public announcement and blog posts
- Documentation needs to be up-to-date
- A plan to quickly rollback if things go wrong
- How to tell the state of the app?
  - Monitoring dashboards
  - Collect metrics in the app (next slide)
  - Log collection

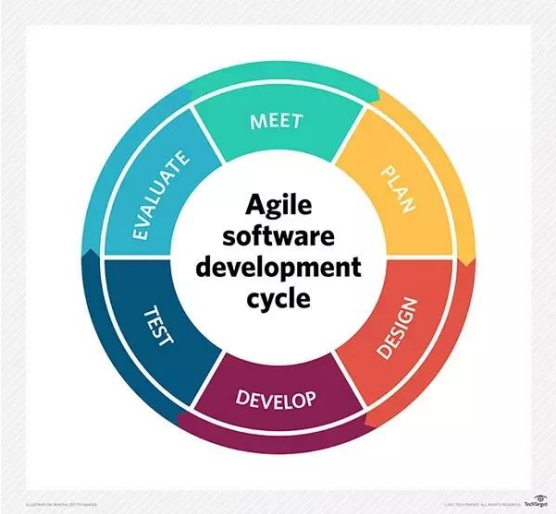
Monitoring

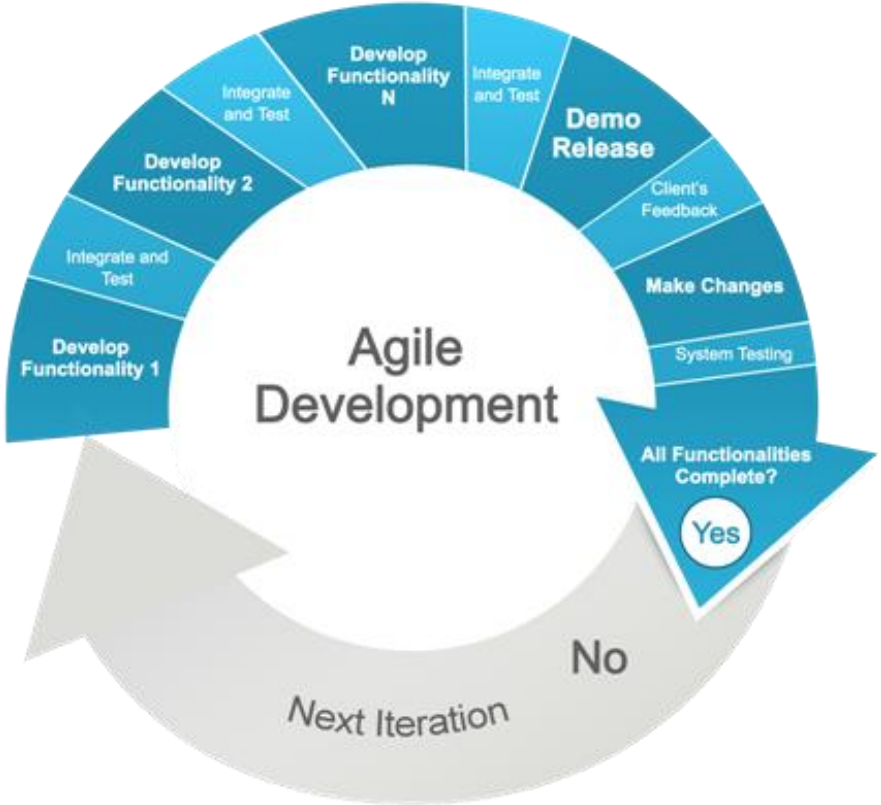
- Collect meaningful metrics that tells the state of the application
- Example metrics
  - CPU and memory, network traffic
  - Calling an API:
    - Ratio of request error count / total number of requests
    - Request/response time duration
  - Dependency on other libraries:
    - Number of instance running which versions of a dependent software
- Metrics + alerts
  - Each metric have a threshold for when alerts will be triggered
    - Oncall


CODE
BUILD
TEST
PLAN
RELEASE
DEPLOY
OPERATE
MONITOR

How does CI/CD relate to agile?

- Agile focuses on fast and continuous deliver to customers
- Perform each stage efficiently





Why does CI/CD matter to you?

swe hour

Also known as: SWEh, SWEhr

A convenient and somewhat arbitrary unit for comparing the costs of different resources. This is not a real hour of SWE time (a legacy decision set it at 9 hours per day), but it's in the ballpark.

Better CI/CD

== More SWE hours saved

== Money saved for your employer

== More impact

== Potential promotion

Useful tools to learn

- Source control: Git
- CI tool: Docker, Gitlab CI/CD, Github Action, Kubernetes
- Automated provisioning tools: Ansible, Terraform
- Monitoring tools: Grafana

How to learn:

- Try them out in a Hackathon (and put in your resume)
- Side projects

Useful tools to learn

- Source control: Git
- CI tool: Docker, Gitlab CI/CD, Github Action, Kubernetes
- Automated provisioning tools: Ansible, Terraform
- Monitoring tools: Grafana

But why should I learn these instead of doing leetcode?

How to learn:

- Try them out in a Hackathon (and put in your resume)
- Side projects

Alex Chiou (He/Him) • 1st
Co-Founder @ joinTaro.com, Making Software Engineers Better At Their Jobs
11h •

A weird phenomenon I have noticed across software engineers is that they develop Stockholm Syndrome with LeetCode.

Even after getting the job, they'll continue regularly doing data structures and algorithms (DSA) problems to "keep their skills sharp".

LeetCode is not a real skill. It's parasitic activity that wastes your time and isn't relevant towards the actual craft of software engineering.

True growth comes from staying on a job for a prolonged period of time, fully integrating yourself into the team and learning how to add value to others.

This means that you shouldn't be on your toes constantly, grinding DSA to preemptively prepare for your next interview with an eye on the exit door.

To learn how you can move past LeetCode to actually level up as a software engineer, check out our in-depth explainer which contains great insights from **Xue Hua** and **Josh Yu**: **https://lnkd.in/gGFSTSRp**

Questions to ask your future employer

- How often do you need to manually test your code change?
- How and how often are code changes deployed to production?
- Is there any manual config/button pushing to deploy to production?
- Do you have a team of infrastructure engineers you work with to design/improve developer workflow and deployment process?
- Are all configurations stored in code repositories and are used directly for deployment without manual changes?
- Do you do test-driven development?
- How long does it take for the longest pre-submit pull request job to finish?
- How long does the CD pipeline take?
- What’s the longest it took for your pull request to be merged?
- How is agile development integrated in your team/company?
- What collaboration tools do you use for development?


That's all Folks!