

What goes wrong when thousands of engineers share the same continuous build?

Eran Messeri, Google
eranm@google.com

Goals

- Demonstrate feasibility of working from head
- Prove the importance of reliable, automated tests.
- Show how complex engineering tasks can be achieved with robust, basic tools.
- Convince you that releases doesn't have to be painful

Background

- Over 15,000 engineers in over 40 offices
- 4,000+ projects under active development
- 5500+ code submissions per day (20+ p/m)
- Over 75M test cases run daily
- 50% of code changes monthly
- Single source tree
- $|\text{DevInfra eng}| \ll |\text{Google eng}|$

Overview of dev. practices

- Single, searchable repository
- Each change requires a code review (ownership, readability)
- Unified build system (local/cloud).
- Continuous integration with presubmit capabilities.
- Single repository for test results (semi-structured).
- Integration testing

Developer workflow

- Check-out code
- Hack hack hack
- Build, test
- ... more hacking
- ... more building and testing
- Code out for review
- Code committed
- Pushed to production

Developer workflow

- Check-out code => Optimize with FUSE
- Hack hack hack => IDE support
- Build, test => In the cloud
- ... more hacking
- ... more building and testing
- Code out for review => Standardized tool
- Code committed => Triggers post-submit
- Pushed to production => Pick a green CL

Common scenarios

- Catching up with head
- Somebody else breaking your build
- Working with open-source & external code
- Good citizenship: codebase clean-up
- Pushing to production

Catching up with head

A simple matter of synchronizing...

- This is where merge happens (always rebasing)
- Cached build artifacts from the cloud.
- FUSE makes this fast

In practice, not very exciting..

Somebody broke your build

- Early detection mechanisms available (global presubmit)
- Have they announced the change?
 - Procedure for breaking changes
- Are your tests stable?
- Cultural commitment to keeping things green.
 - Short time window for fixing
 - Rollback if not feasible
 - No hard definitions

Working with external code

- Easy process for importing external open-source code.
 - Incl. open-source review
- Exactly one version of each library
 - No exceptions!
- “Public spaces” - shared maintenance burden.
 - Yes, it’s expensive
- Tools exist for open-source development

Codebase clean-ups

- Pre-requirements: good tools
- What will break if I change X?
- No need for individual project approval (global review)
- Tests transform fear to boredom

Appreciate and acknowledge such efforts

Pushing to production

- Code approved, submitted
- Post-submit triggers, test affected code.
- Good mix of small, medium and end-to-end tests.
- Separate method for bringing up systems in isolation.
- Easy deployment UI.

Release in hours instead of weeks

What we (think) we got right

- Getting started on the codebase
- New “checkout” and build.
- Effortless testing.
- Navigating around the code
- “Did that ever work?”

What doesn't work?

- Code change turn-around time: Bandwidth vs. change size
- Cost of test creation & maintenance
 - Mocks at different levels (class, module, system)
 - Creating hermetic tests is hard
 - Sometimes need specialists
- Resources consumption
- Churn - external and internal

Beyond the basics...

- Stack-trace analysis of failing tests
- Overcoming infrastructure failures
- Automated detection of dead code.
- Flakiness detection

Summary

- Collaborating over one source tree is possible, but non-trivial.
- Basic CI tools are hard to build at such a scale.
- Reliable automated tests will make your release easy.
- Nothing can replace good eng. citizenship.



Questions?

Additional resources

Talks:

“Continuous integration at Google Scale”

“Development at the Speed and Scale of Google”

“Tools for Continuous Integration at Google Scale”

Blog: Google Eng Tools blog