

# Development at the Speed and Scale of Google



Ashish Kumar  
Engineering Tools





# The Challenge



# Speed and Scale of Google



- More than 5000 developers in more than 40 offices
- More than 2000 projects under active development
- More than 50000 builds per day on average
- More than 100 million test cases run per day
- 20+ code changes per minute; 50% of the code changes every month
- Single monolithic code tree with mixed language code
- Development on head; all releases from source

# Single monolithic code tree ...

- Develop at head
- Build everything from source
- Extensive automated tests running at each changelist
- Need strong enforcement of coding style and guidelines
- Can make changes to kernel, gmail and buzz in the same changelist
- Complex dependency graph across products and libraries

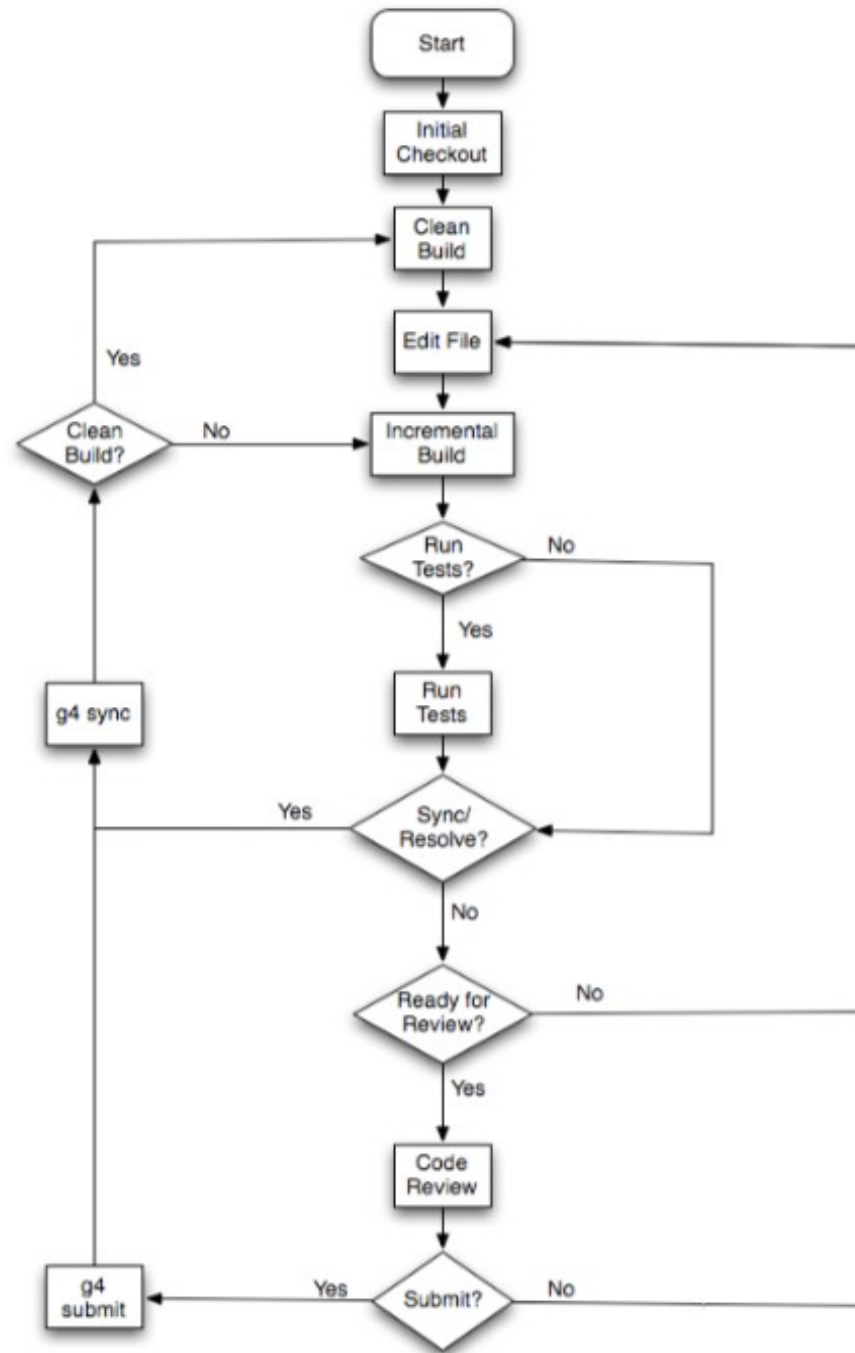




Why do we care?



# Rough developer workflow

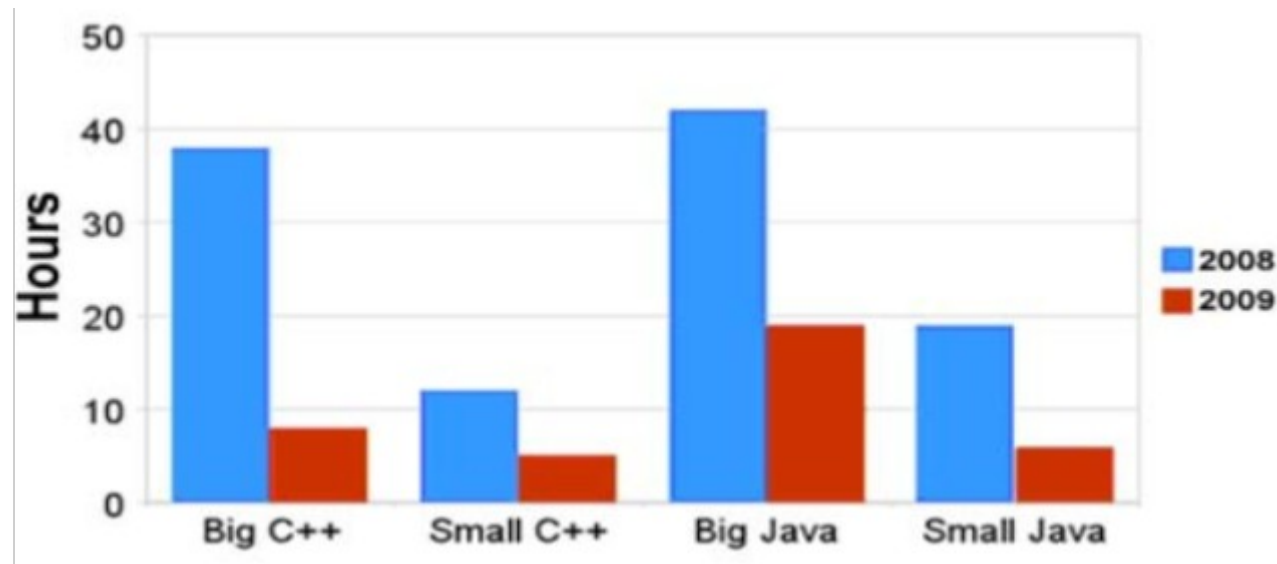


# Estimating build tools savings 2008 to 2009

- Rough use case estimates

ACTIVITY	INITIAL CHECK-OUT	CLEAN BUILD	BUILD AFTER EDIT	BUILD AFTER SYNC	RUN TESTS
FREQUENCY	2	4	160	20	60

- Estimated Time waiting on build tools



- Estimated Savings: ~600 person years



Who we are





# Engineering Tools and Engineering Productivity



- Google Focus Area: Engineering Productivity
  - Focus on Accelerating Google
  - Includes Test Engineering, Release Engineering, Engineering Docs and Education, ... , and Engineering Tools
- Engineering Tools
  - Focused on providing tools that accelerate Google engineers from idea to production
  - 100+ team of engineers spread across 4 major sites
  - Builds and manages tools related to Source Control, Developer Tools and IDEs, Test Infrastructure, Build Tools and Infrastructure, Project Management Tools, and others

# What's Unique?



- Significant investment in infrastructure for developers
  - Core infrastructure technologies like GFS, BigTable etc. that developer can quickly build systems on
  - Core tools that developers can quickly build, test and release their products / projects with
  - Tools leverage the same production infrastructure that our products do
- Continuous Improvement with Tools
  - "We can't improve what we can't measure"
  - Data-driven culture: strong focus on metrics for improvement
  - Our goal: make the tools disappear from the workflow



How we do it



# Building for scale



# Our version



- "Free" infrastructure for all teams
  - Transparency of code changes through centralized code review service
  - Developers can run affected tests before submitting code
  - Run every affected test at every code change
  - Run tests on all major OS / browser combinations
  - Transparently store all build and test results (including build, code analysis, and linter warnings)
  - Provide comprehensive UI, API and notification
  - Move all "compute-intensive work" to the cloud

# Key Goals and Principles



- Speed: Developers spend lesser and lesser time waiting on tools e.g. builds, test systems, code analysis, ...
- High Quality Feedback: Deliver high quality feedback; more signal, less noise.
- Simplicity: Developers will ideally not need to know or understand how the underlying tools and systems work.

Measure everything



# Source code at scale ...



- How to allow 1000s of engineers to sync source code on a single tree with massive dependencies?
- A full checkout would take tens of minutes
  - Would easily choke any corporate network
  - Other companies create developer branches per feature
- Developers change < 10% of code they actually check out
  - Builds and tests often need the rest of the code to run
  - Deliver the rest of the code as a read-only copy, on demand
  - Implemented as a FUSE-based file system, tracks changes to main source depot and caches aggressively

# Keeping the code tree consistent

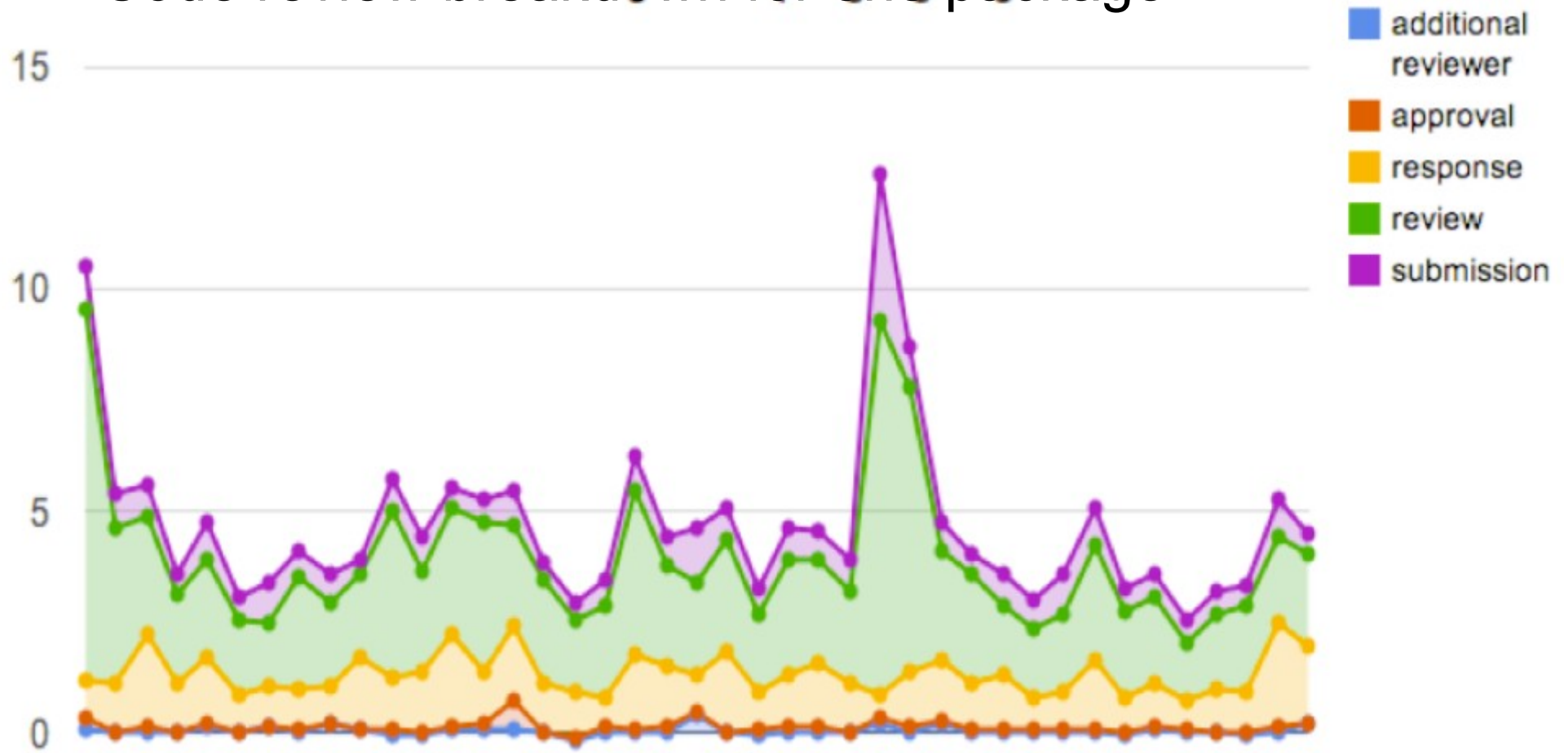


- Mandatory code reviews with central tool
  - Need code readability for languages (enforces style guide)
  - Need owners for code sub-tree that maintain consistency and correctness
  - Higher code transparency and code contributions across teams
- Reduce code review costs, provide lots of signals to reviewers
  - Lint errors
  - Code Analysis and Build warnings / errors
  - Code coverage data
  - Test results
  - Easy, web-based access - full graphical diffs available, easy to add comments
  - Future: integrate with IDEs

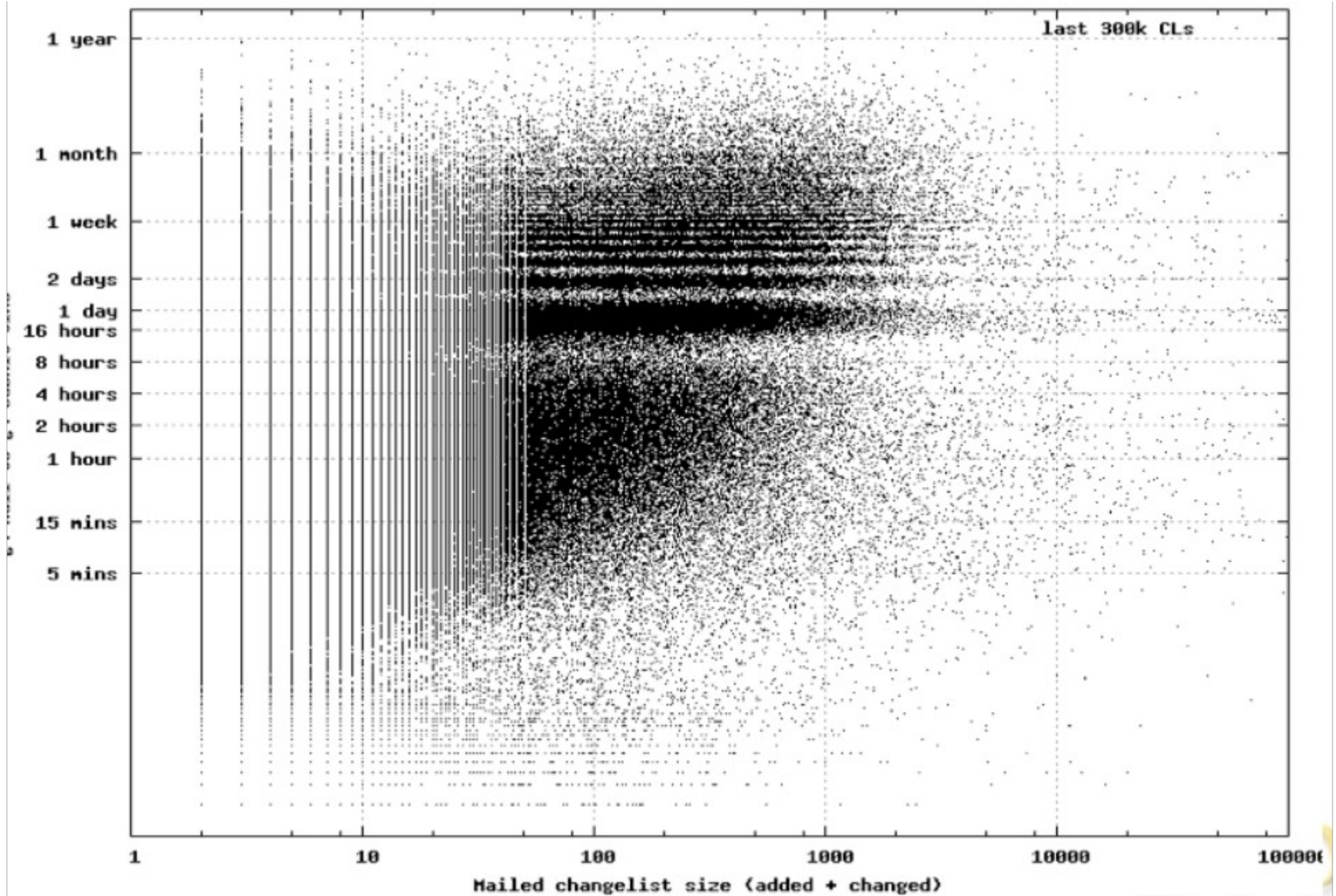


# Keep code reviews efficient

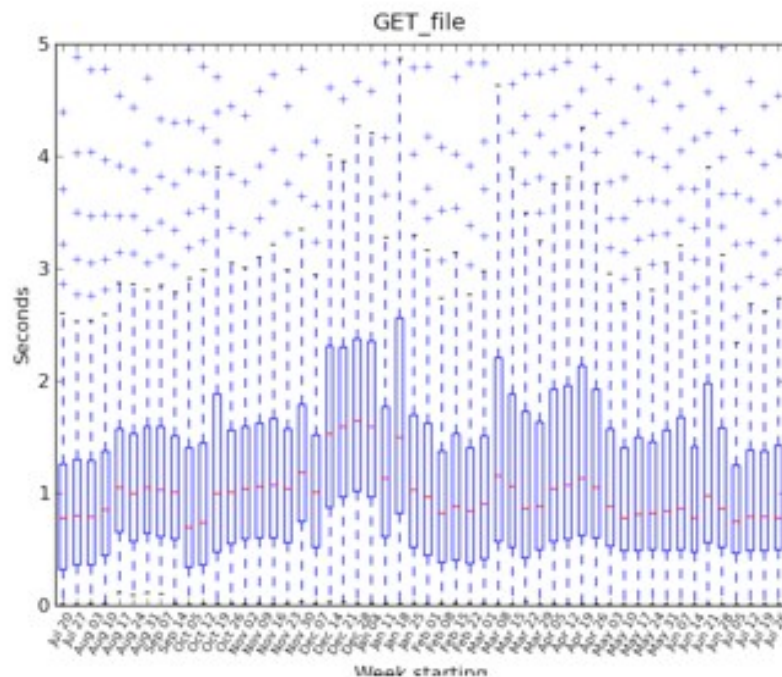
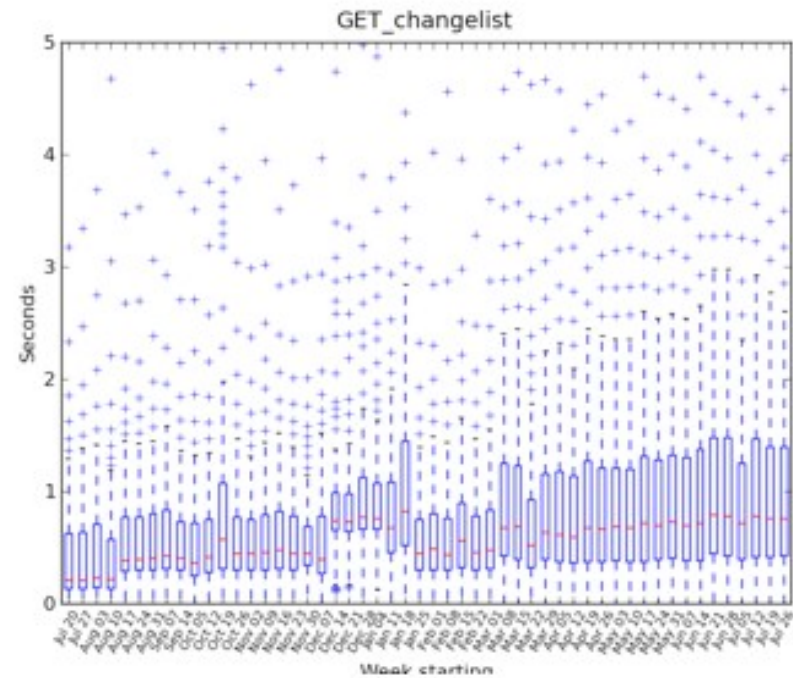
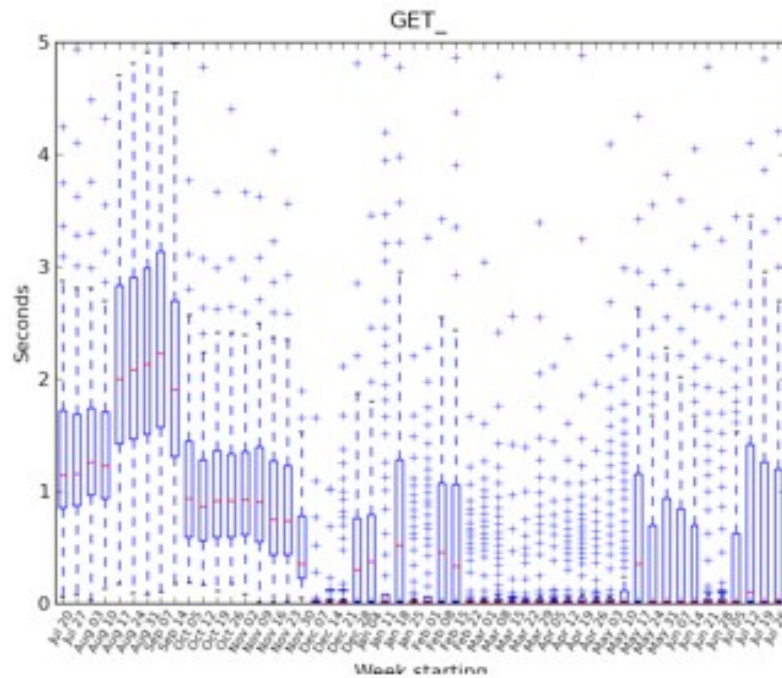
## Code review breakdown for one package



# Code Review turnaround by size



# Measure the tool itself



Box-plots for the Code Review tool latencies

# The Build System is important

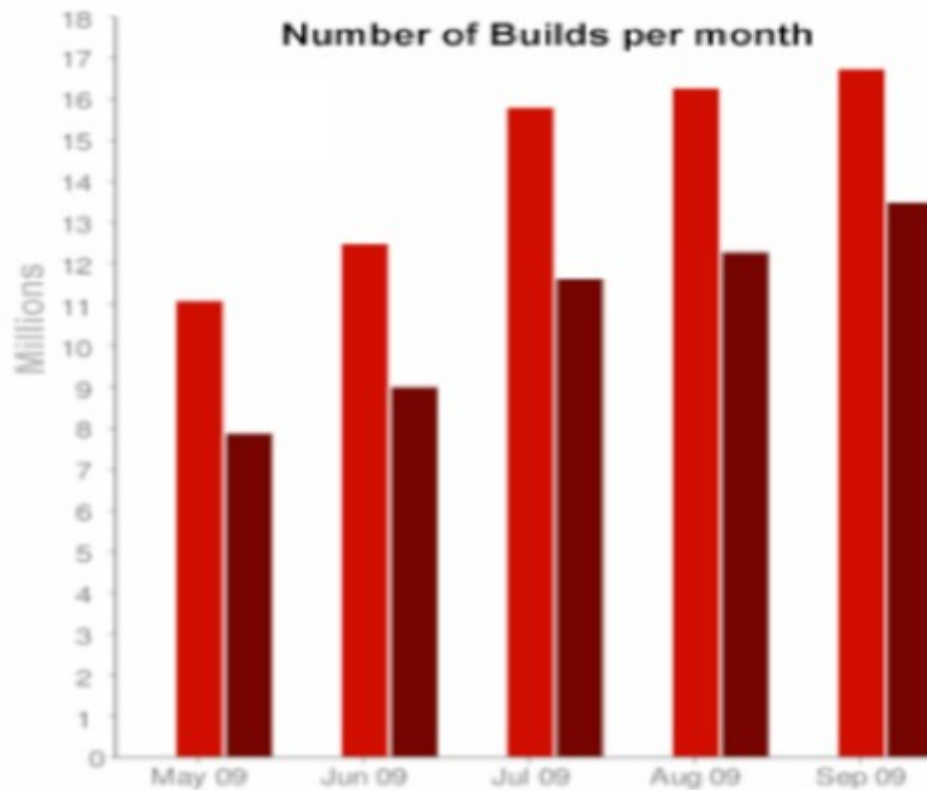


- Builds are glamour-less at most companies
- Problems with builds can result in huge productivity losses
  - Debugging build problems
  - Waiting for builds to finish
  - Feedback best attached to build systems; e.g. run tests, code analysis as part of builds
- Build metadata is equally important as source code
  - Needs to analyze and enforce dependencies, validate inputs
  - Needs to be correct and fast
  - Builds need to be hermetic to be distributed
  - Full knowledge of inputs, dependencies and outputs can allow massive parallelization of actions



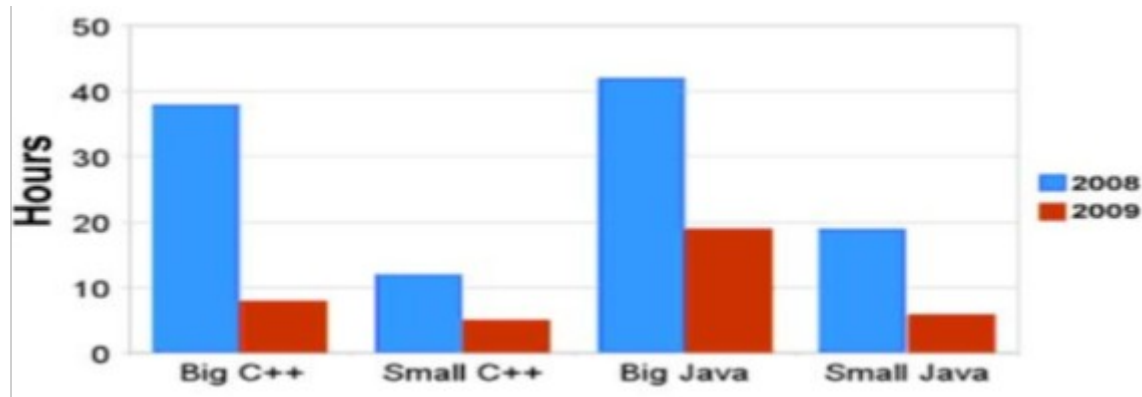
# Build Systems require strong CS skills

- Deal with massive scale
  - 20 Million+ builds per year
- Massive distributed execution
  - More than 10000 cores using > 50TB of memory
  - ~1 PB 7-day cached object output



# Durable metrics

- Remember this?

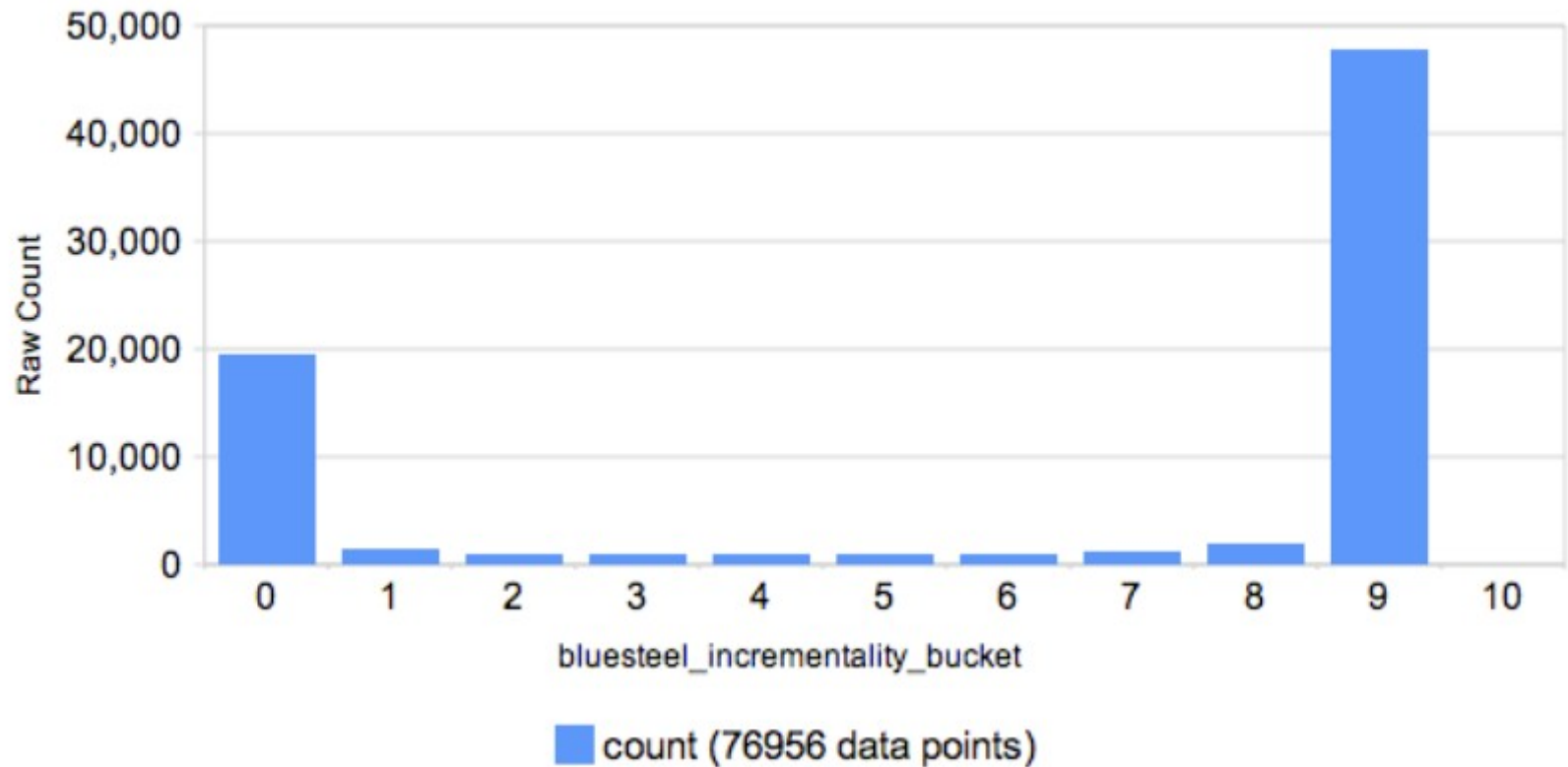


- Mostly flat between 2009 and 2010
  - Files for each (measured) target grew by 54% to 191%
  - Doing significant more work in the same time
- Needed durable metrics across time; bucket builds by:
  - Count of discrete actions and inputs
  - Office
  - Incrementality
  - ...

# Builds by incrementality

- Many builds are clean, but most are in the 90-100% incrementality range!

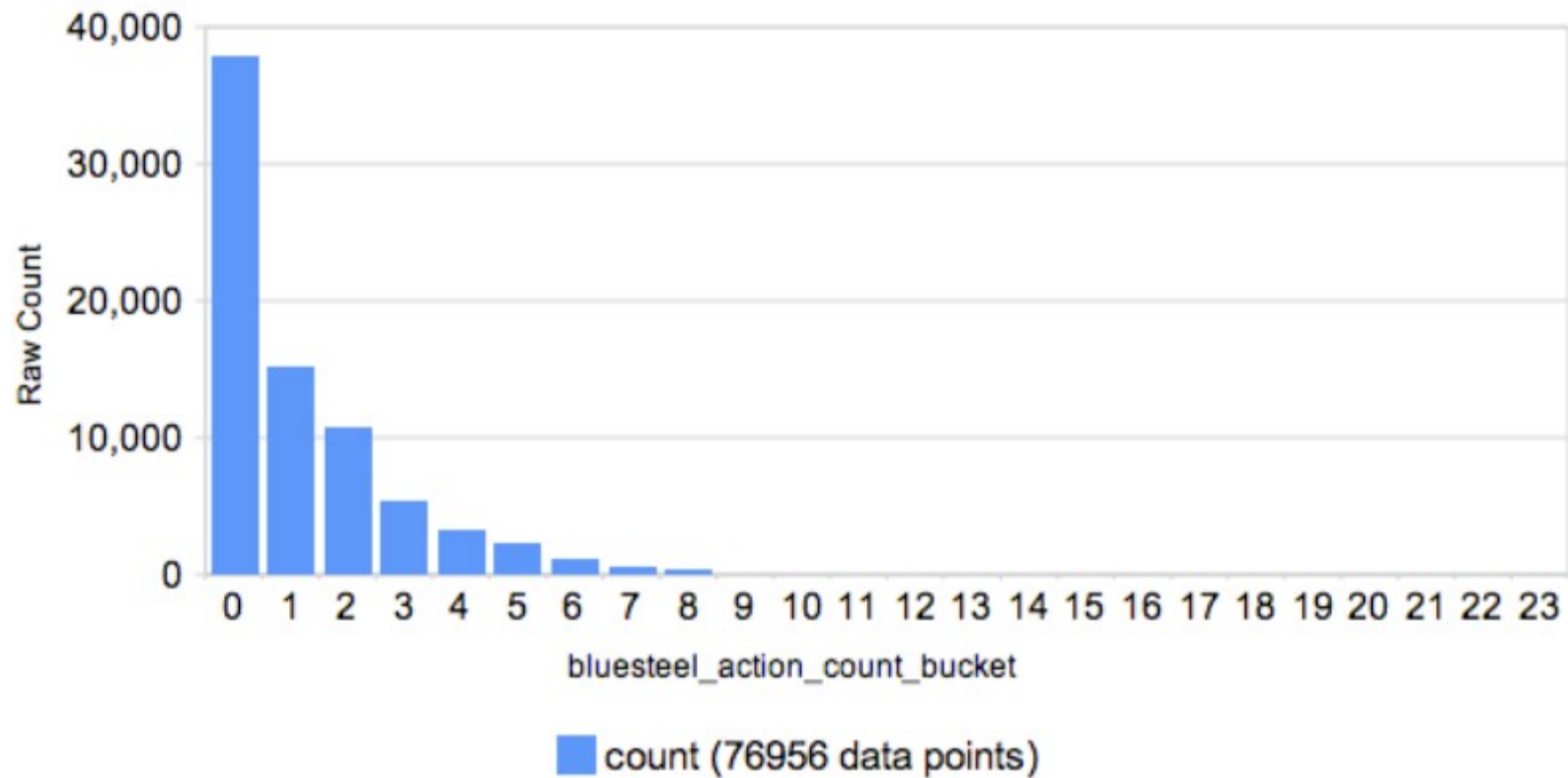
Incrementality Distribution (10% buckets)



# Builds by action size

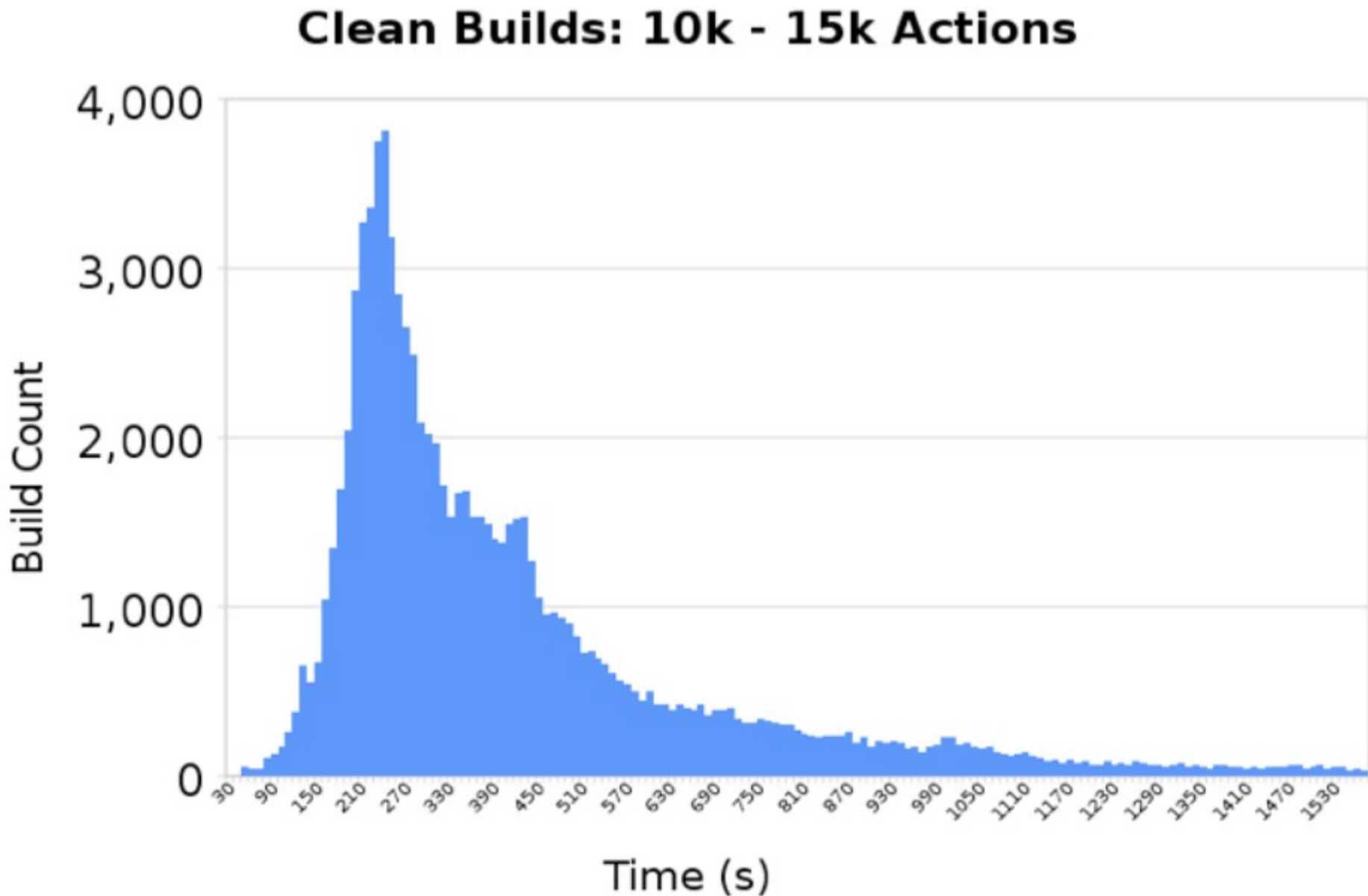
- Most builds are small, but long tail (mostly by our own automated systems)

Action-Size Distribution (5k buckets)

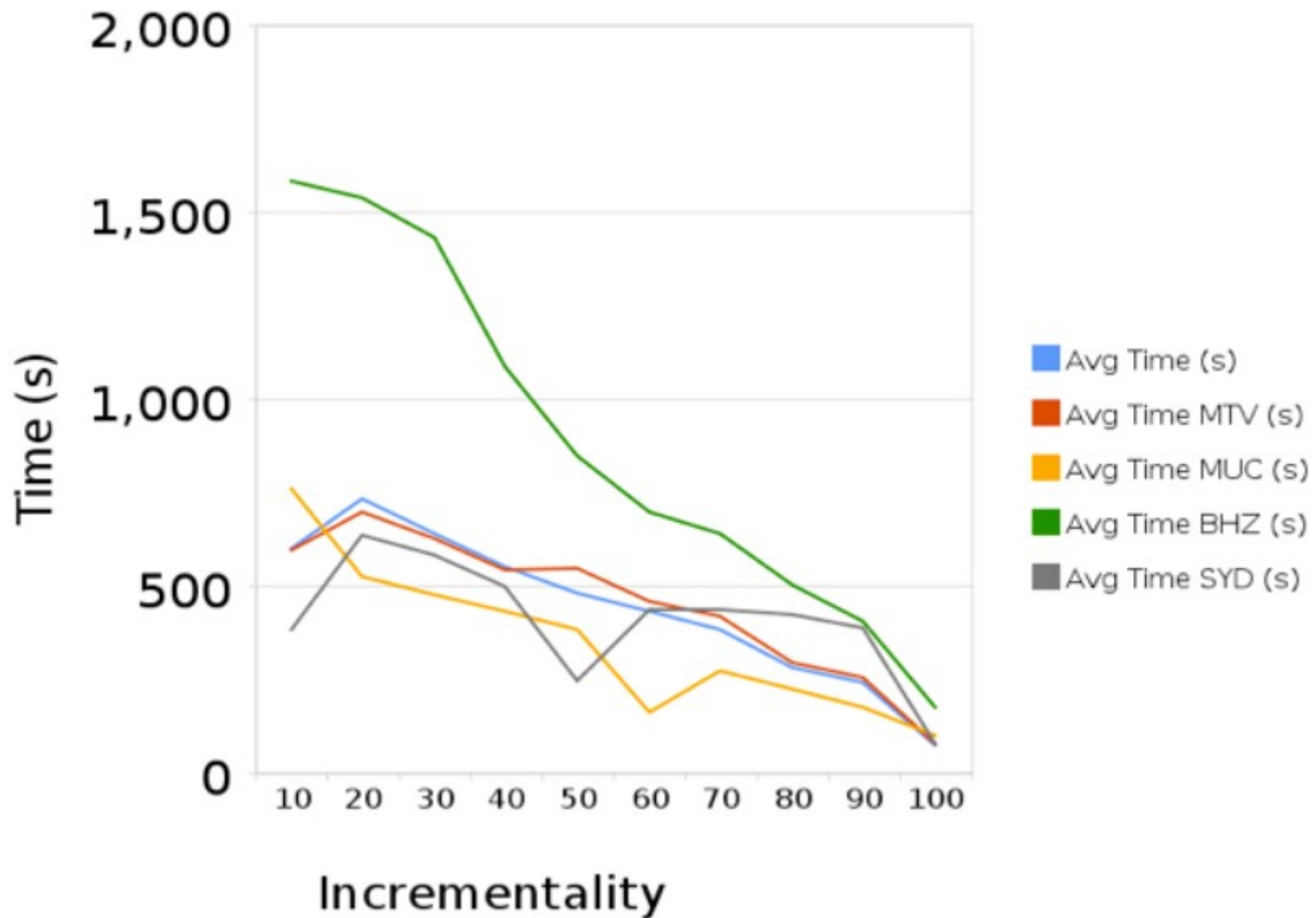




# Clean Build times



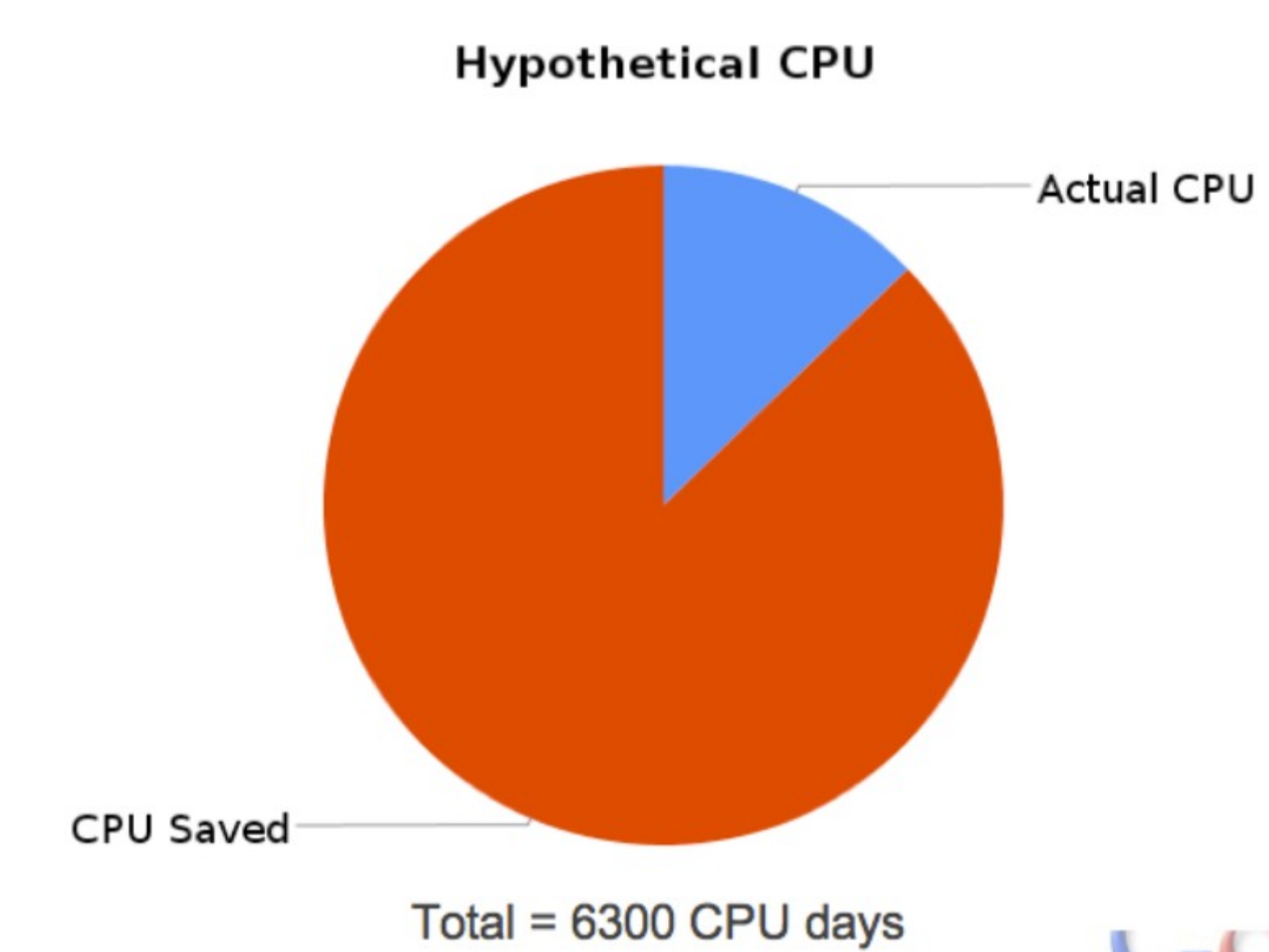
# Build times by office



# Action Cache



# How much did we save?



# Object caching wins



## Statistics from a single day

- ~ 500M build actions
- 94% action cache hit rate
- 30M cache misses
- 800 CPU days (just build and test)
- 66% of actions from automated builds

# Building in the cloud has costs ...



- Large builds have large outputs
- Corp-Cloud network is not as efficient as Cloud-Cloud network, transferring bits can be a significant time sink and network hog
- Solution: don't send the build outputs to the workstation till they are actually needed or read.
  - Implemented as a Fuse-based file system that allows directory operations on the output.
  - Aggressive caching for build outputs by office and workstation

# Distributed builds have costs ...



- Link actions require all the input object files
  - Requires moving all object files that are built on different distributed nodes to the one node where the link action occurs
  - Can be expensive and on the critical path
- Solution: Incremental link
  - Store additional information in a binary
  - Use old binary + modified object files to build new binary
  - Only process modified object files symbol tables and relocations
  - expected 10x improvement in link speed

# Continuous Integration at Scale



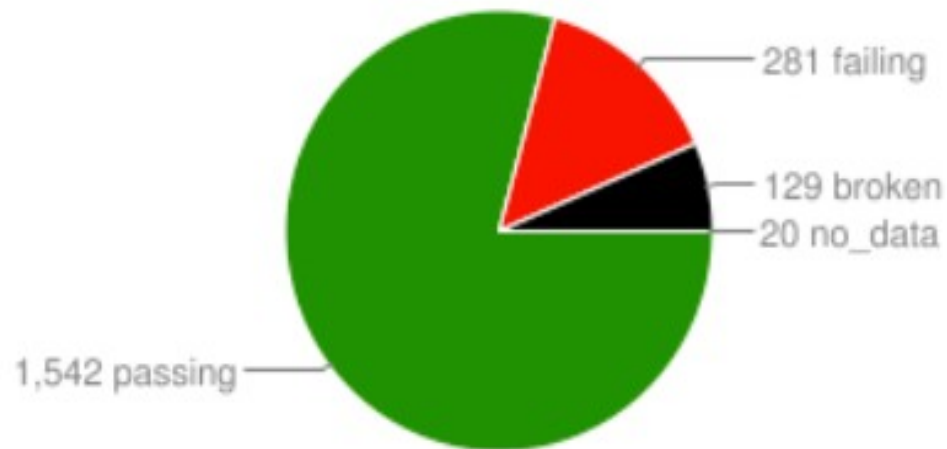
- Fail fast, report clearly, root cause
- Test early at every stage
- Reduce defect identification to fix time
- Use feedback and data to stay healthy
- Reduce complexity

"... the key is to practice continual improvement and think of (it) as a system, not as bits and pieces." -  
*Dr. W. Edwards Deming*



# Continuous Integration at Scale

- 120K test suites in the code base
- Run 7.5M test suites per day
- 120M individual test cases / day and growing
- 1800+ continuous integration builds



Mountains of data == Opportunity for data mining and research

# Scale requires Search



The screenshot shows the Google Sponge search interface. The search bar contains the query 'status:failed label:tap target:gws'. Below the search bar, there are buttons for 'Search', 'I'm Feeling Lucky', and 'Advanced Search'. The search results are displayed in a table with the following columns: Status, Targets, Labels, Run Date (GMT-7:00), Elapsed / Test Time (h:mm:ss), User@hostname, Client, and Change.

Status	Targets	Labels	Run Date (GMT-7:00)	Elapsed / Test Time (h:mm:ss)	User@hostname	Client	Change
 188 Failed 13 Passed	various targets	blaze, tap, test	2010-07-18 1:47 AM	04:39 / 06:46	tap-prod@ybhw33		<a href="#">164749</a>
 191 Failed 10 Passed	various targets	blaze, tap, test	2010-07-17 2:09 PM	06:38 / 03:20	tap-prod@ybkw37		<a href="#">164738</a>
 2 Broken 2 Failed 154 Passed	various targets	blaze, tap, test	2010-07-17 9:42 AM	12:49 / 1:23:23	tap-prod@prau6		<a href="#">164732</a>
 52 Failed 33 Passed	various targets	blaze, tap, test	2010-07-17 9:39 AM	09:00 / 28:01	tap-prod@pre31		<a href="#">164733</a>

Also provides a SQL interface to query build and test results for further analysis

# Test results repository

Detail for test run on 2010-...  
http://sponge/targetResult?id=4b692964-433c-4e23-afab-82e855d2f2cd&index=54&show=FAILED&searchFor=

Failed  
On February 26, 2010 5:10 PM, pepstein ran [junittests.com/google/testing/metricstore/server/web/Large Tests].  
See [all target results](#) for this invocation.

Details Build Log Test Log Test Cases Timing Breakdown Comments (8)  
235 test cases Show passed (234)

- top-level 235 test cases, 1 error in 37.307s
- Large Tests (com.google.testing.metricstore.server.web) 235 test cases, 1 error in 37.307s
  - Parallel (com.google.testing.metricstore.server.web) 263 test cases, 1 error in 37.856s
    - InvocationDetailPageTest (com.google.testing.metricstore.server.web) 67 test cases, 1 error in 18.489s
      - testHeaderRenderingWithEmptyCI error in 17.270s Attached tests: [ [Can-keep Out](#) ]

```
com.gargoylesoftware.htmlunit.FailingHttpDriverCodeException: 500 Internal Server Error for http://gsl102146026-1.companion-test1119641200130-10e2-4701-b0e7-8c30002ae001
at com.gargoylesoftware.htmlunit.WebClient.showOutgoingHttpDriverCodeExceptionIfNeeded(WebClient.java:130)
at com.gargoylesoftware.htmlunit.WebClient.getPage(WebClient.java:311)
at com.gargoylesoftware.htmlunit.WebClient.getPage(WebClient.java:317)
at com.gargoylesoftware.htmlunit.WebClient.getPage(WebClient.java:371)
at com.google.testing.metricstore.testing.SpongePage.<init>(SpongePage.java:67)
at com.google.testing.metricstore.testing.InvocationDetailPage.<init>(InvocationDetailPage.java:52)
at com.google.testing.metricstore.server.web.InvocationDetailPageTest.<init>(InvocationDetailPageTest.java:127)
at com.google.testing.metricstore.server.web.InvocationDetailPageTest.testHeaderRenderingWithEmptyCI(InvocationDetailPageTest.java:152)
at sun.reflect.NativeMethodAccessorImpl.invoke0
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:116)
at junit.framework.TestCase.runTest(TestCase.java:109)
at junit.framework.TestCase.runBare(TestCase.java:134)
at junit.framework.TestRunner$1.runTest(TestRunner.java:118)
at com.google.testing.junit.runner.OutputRedirectingTestResultCollector.collect(OutputRedirectingTestResult.java:411)
at com.google.testing.junit.junit4.AdvancedTestRunner$TimingInterceptor.collect(AdvancedTestResult.java:120)
at junit.framework.TestRunner$runProtected/TestRunner.java:122)
at com.google.testing.junit.junit4.AdvancedTestRunner$runProtected(AdvancedTestRunner.java:124)
at com.google.testing.junit.runner.OutputRedirectingTestResult.runProtected(OutputRedirectingTestResult.java:137)
at junit.framework.TestRunner.run(TestRunner.java:118)
at com.google.testing.junit.junit4.AdvancedTestRunner.actualRun(AdvancedTestResult.java:119)
at com.google.testing.junit.junit4.AdvancedTestRunner.access$101(AdvancedTestRunner.java:19)
at com.google.testing.junit.junit4.AdvancedTestRunner$Finalizer.run(Finalizer.java:74)
```







# Faster time to fix

The screenshot displays the Google Test Infrastructure (TAP) interface. On the left, a sidebar shows the test status for 'zzz\_for\_real2\_0257 broken at CL', indicating 14 tests failed. The main window shows the details for the failed test '://borg/simulator:get pending reason test' at changelist 14686262. Below the test name, there is a 'Test Target Output' section with a 'View logs in Sponge' link. The logs show the test command and its execution environment, including various environment variables like 'BORG\_CELL', 'GOV\_PREFIX', and 'GOV\_PREFIX\_STRIP'. The test command is 'java -Xmx256m -Xmx1024m -XX:ErrorFile=/dev/stderr'. Below the logs, there is a 'Test History' section showing a sequence of test results (P for pass, F for fail) for the test, with the current failure at changelist 14686262. Finally, there is a 'Changelist Description' section for changelist 14686262, which describes the change: 'Change the Python Protocol/Sobby to let you pass in a response protocol buffer instead of automatically creating a new one.' The description lists the author 'mcquire' and the reviewers 'petar, jrobbins, robinson'.

# And of course, we need more ...



- IDEs that can work at scale
- Code visualization and search
- Code Analysis and Documentation
- ... many more



# Summary





# What we do different



- Invest in our developer infrastructure
  - Developers can build upon common technologies
  - Significant investment in central tools team results in a measurable boost in engineer productivity
- Parallelize and Distribute where possible
  - Compute intensive operations leverage the cloud, while UI-sensitive work stays closer to the developer
- Hire the best / Design for scale
  - Developer Tools and Build Systems are tough computer science and systems problems; they need the best developers
- Measure Everything
  - Cannot improve what we don't measure