2020-11-10 @ UCLA CS188

Big Code: Shipping software at scale

Quinn Slack <quinn@slack.org>
Sourcegraph CEO/cofounder

What I've done/seen

- I love coding!
- Engineer @ Bleacher Report (2007)
- Stanford CS undergrad (2011)
- Built software inside Bank of America and JPMorgan (2011-2012)
- Founded Blend Labs (2012–)
- Founded Sourcegraph (2013–)
    - Product is code search ("Google for your code")
    - Seen up-close how software is built inside most major tech companies

Shipping at scale
is hard!

What Big Code means:
# of developers

	# devs From LinkedIn
Uber	5,600
Netflix	1,800
Airbnb	2,100
Lyft	1,700
Indeed	1,500
Cloudflare	325
Coinbase	300
Yelp	950
Apple	39,000
Palantir	1,050
CS188 team	4

What Big Code means:
amount of code

	# commits	# lines
Google	35,000,000	2,000,000,000
All CS188 teams	1,417	687,940

Source for Google data
Source for CS188 data

What Big Code means: ***problems you'll face***

- Lots of code
- Lots of developers
- Lots of complexity
- Multiple languages
- Many tools
- Maintenance
- Deprecation
- Uptime/availability requirements
- NIH (Not Invented Here) syndrome
- Terrible onboarding
- Microservices vs. monolith
- Tech debt
- People who left the company
- People who aren't working today
- External security risks
- Insider threats
- Outdated documentation
- Unclear requirements
- Business uncertainty/failure
- Distributed teams
- Company mergers/acquisitions
- "Reorgs" (team/leadership changes)
- Open-source usage prohibitions
- Constant refactoring

What Big Code means: ***tools you'll use***

- Tests
- Code review
- Monolith vs. microservices vs. library extraction
- Monorepo vs. many repositories
- Code search
- Large-scale automated changes

# devs	4
# dev teams	1
# customers	0

Day 0

Nikola founded!

# devs	4
# dev teams	1
# customers	0

Day 365

Ship Nikola v1.0!

# devs	4
# dev teams	1
# customers	1

Day 500

First customer!

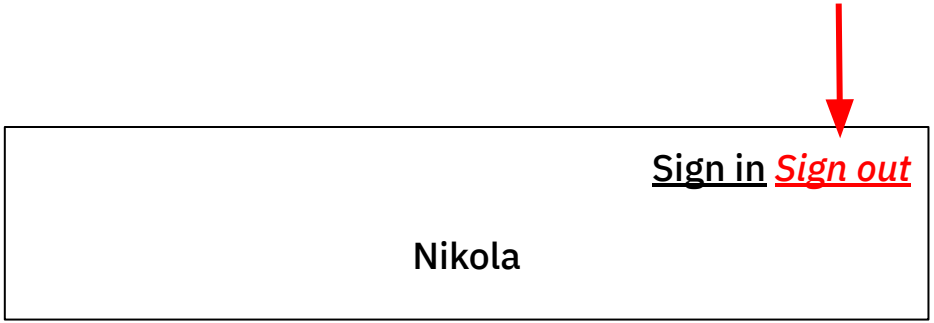
# devs	4
# dev teams	1
# customers	1

Day 500

Customer hits a bug

Fix the bug

"Oops! If the user isn't signed in, we shouldn't show the 'Sign out' button."




```
  @@ Header.tsx
  const Header: React.FunctionComponent = ({user}) =>
    <ul>
      {!user && <li><a href="/sign-in">Sign in</a></li>}
-     <li><a href="/sign-out">Sign out</a></li>
+     {user && <li><a href="/sign-out">Sign out</a></li>}
    </ul>
```

Avoid regressing (re-introducing the bug)

In the same commit that fixes the bug, also add a test!

This test checks that when the <Header> component is rendered for an anonymous viewer (`null` user), there is no Sign out link.

```
// Header.test.tsx
import React from 'react'
import { Header } from './Header
import { mount } from 'enzyme'

describe('Header', () => {
    test('no sign-out for anonymous viewer', () => {
        expect(
            mount(
                <Header user={null} />
            ).find('a[href="/sign-out"]').length
        ).toBe(0)
    })
})
```

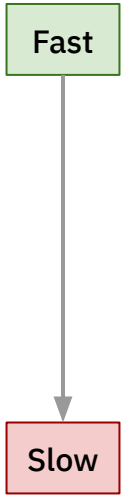
Tests

Benefits

- Avoid regressing on bugs
- Prevent bugs in the first place
- Make future changes easier for yourself and other team members
- Document code and edge cases

Kinds of tests

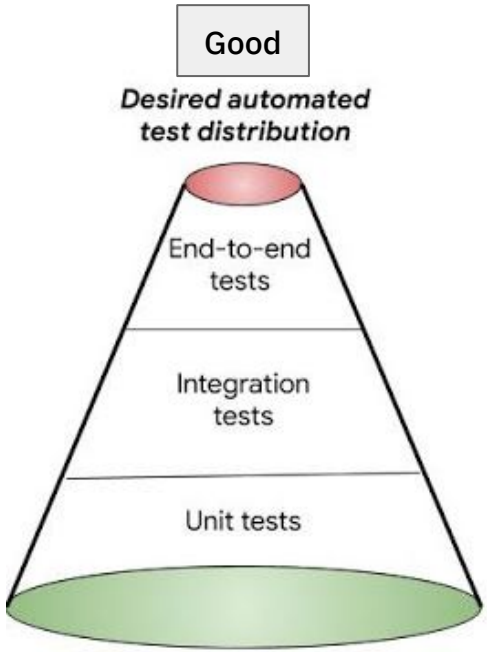
- *Unit test:* simple, isolated check of a single component (*`Header.test.tsx`*)
- *Integration test:* check of multiple components working together
- *End-to-end test:* run the full app and a browser, and pretend to click things like a real user



Tests should not slow you down

Write unit tests often.

Write end-to-end tests sparingly!



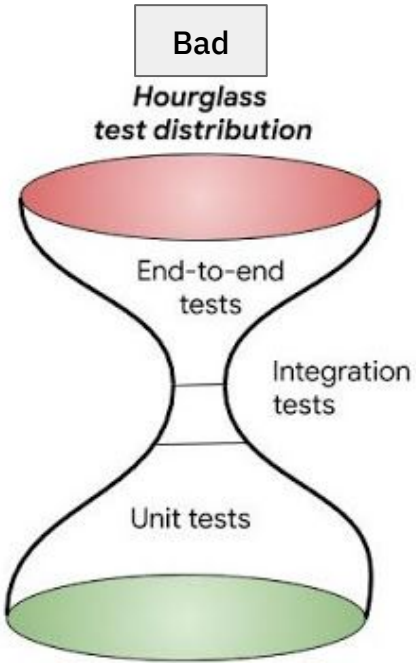


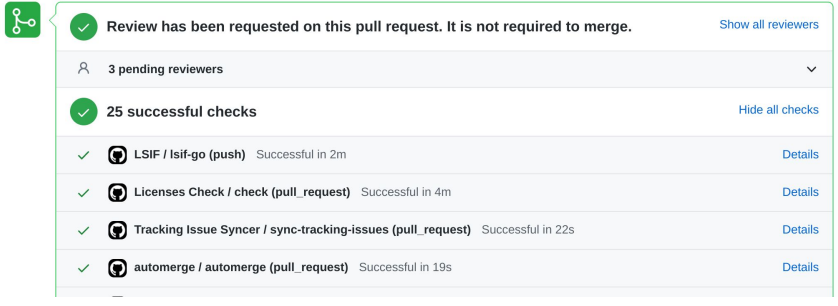
Image: Google Testing Blog

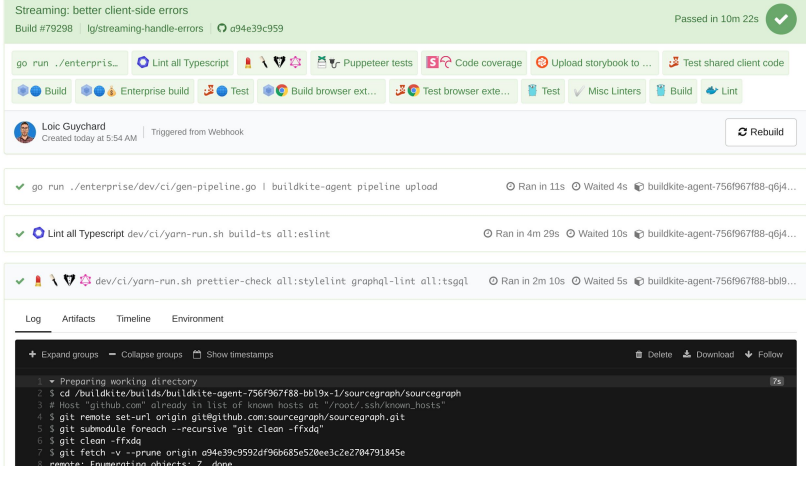
Running tests

Test watcher, for local development:

```
 PASS  web  src/enterprise/site-admin/dotcom/productSubscriptions/SiteAdminProductLicenseNode
 PASS  web  src/enterprise/campaigns/detail/changesets/ChangesetLabel.test.tsx
 PASS  web  src/components/externalServices/ExternalServiceForm.test.tsx
 PASS  web  src/enterprise/site-admin/dotcom/customers/SiteAdminCustomerBillingLink.test.tsx
 PASS  web  src/components/LoaderButton.test.tsx
 PASS  web  src/enterprise/campaigns/detail/CampaignSpecTab.test.tsx
 PASS  web  src/enterprise/site-admin/dotcom/productSubscriptions/SiteAdminGenerateProductLic
 PASS  web  src/enterprise/dotcom/productPlans/ProductSubscriptionUserCountFormControl.test.t
 PASS  web  src/components/diff/DiffStat.test.tsx
 PASS  web  src/search/results/SearchResults.test.tsx
 PASS  web  src/enterprise/campaigns/global/nav/CampaignsNavItem.test.tsx
 PASS  web  src/components/time/Timestamp.test.tsx
 PASS  web  src/enterprise/campaigns/global/marketing/CampaignsDotComPage.test.tsx
 PASS  web  src/enterprise/campaigns/detail/changesets/HiddenExternalChangesetNode.test.tsx
 PASS  web  src/enterprise/user/productSubscriptions/UserProductSubscriptionStatus.test.tsx
 PASS  web  src/search/input/interactive/FilterInput.test.tsx
 PASS  web  src/enterprise/campaigns/detail/changesets/CampaignChangesets.test.tsx
 PASS  web  src/enterprise/campaigns/list/CampaignListPage.test.tsx
 PASS  web  src/search/input/SearchPage.test.tsx
 PASS  web  src/enterprise/campaigns/detail/CampaignDetailsPage.test.tsx
 PASS  web  src/nav/GlobalNavbar.test.tsx
 PASS  web  src/search/queryBuilder/QueryBuilder.test.tsx
 PASS  web  src/search/results/SearchResultsList.test.tsx

Test Suites: 1 skipped, 86 passed, 86 of 87 total
Tests:       7 skipped, 328 passed, 335 total
Snapshots:   185 passed, 185 total
Time:        5.207s
Ran all test suites.
```





Testing

Other concepts

- *Test coverage:* what % of your code is checked by tests? (Goal: high, near 100%.)
- *Test-driven development:* when you start by writing tests *before* the code that makes them pass.

In practice

- The best software teams write tests for (almost) all major functionality.
- Pick {projects,libraries} that have good tests.
- Write code so it's easy to test.
- On important code, it's OK (and common) to spend 50% of your time writing tests.

Common testing tools

- *Test libraries and runners:* depends on language
- *Continuous integration:* Buildkite, CircleCI, GitHub Actions, Travis CI, Jenkins, Drone, AppVeyor

Recommended reading

- Google Testing Blog

# devs	5
# dev teams	1
# customers	1

Day 530

First non-founding engineer hired!

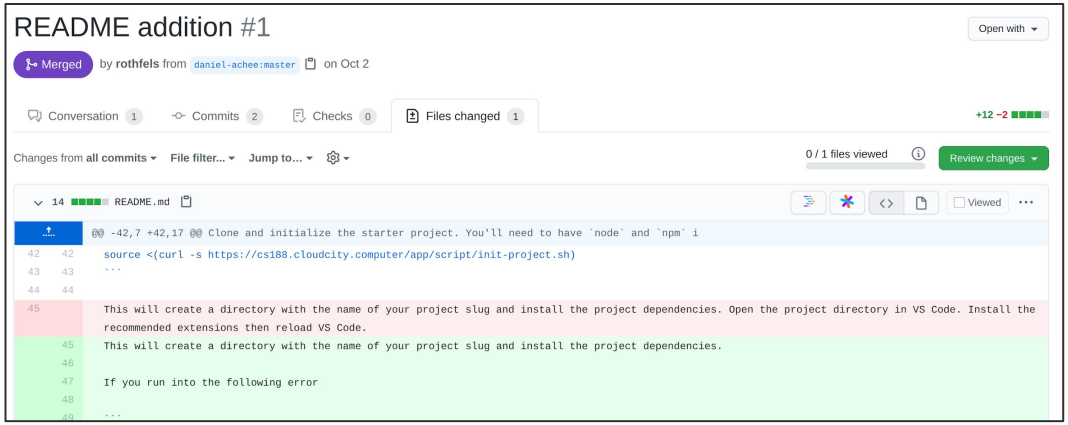
Code review

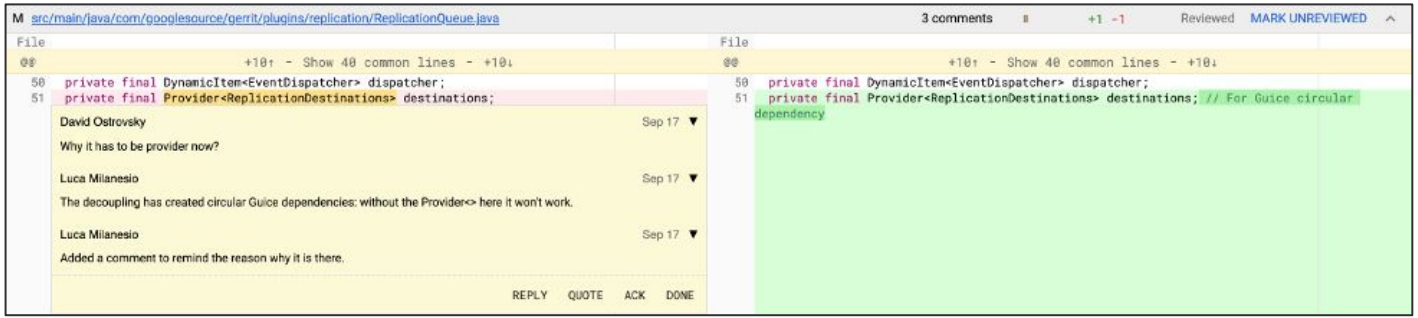
Benefits

- Share knowledge
- Improve quality

Tools

- GitHub pull requests
- GitLab merge requests
- Phabricator Differential (Facebook)
- Gerrit (Google)





# devs	20
# dev teams	5
# customers	10

Day 1,500

Our codebase is too big & complex!

On #dev-chat in Slack...

[9:32am] **@alice:** It's gotten so hard to just add a new feature. Our code is so interconnected and complex, and I can't juggle it all in my mind. Plus, my editor runs out of memory and crashes 5 times a day, and *`npm install`* and CI take too long!

Symptoms of a big & complex codebase

Every feature/component is intertwined with other unrelated things.

- Anxiety/stress when making changes: "It's hard to make a change without (potentially) breaking everything!"
- Unexpected side effects: "Why does changing a user's profile photo accidentally charge their credit card again?"

Parts of the dev process that should be fast become painfully slow.

- "VS Code takes up 32 GB of memory and makes my laptop fans blast off."
- "Every time I switch a branch, I need to kill everything and wait 5 minutes for *npm install*."
- "CI takes 20 minutes to run, which makes a simple change take hours!"

On #dev-chat in Slack...

[9:45am] **@bob:** Let's split our code into microservices. Each service will be responsible for 1 thing and will be deployed completely independently. Each team just defines the contract for their service that other teams rely on, and we never need to talk to each other again!

Monolith vs. microservices

Monolith: when your entire application is written, tested, and deployed as a single unit

Microservices: when you break apart your application into smaller, independent services that communicate using a well-defined API

Example: There's a team at Uber that just maintains a simple "airport doors" API service that lets any app (Uber iOS/Android/etc.) query a list of airport doors for a user's location.

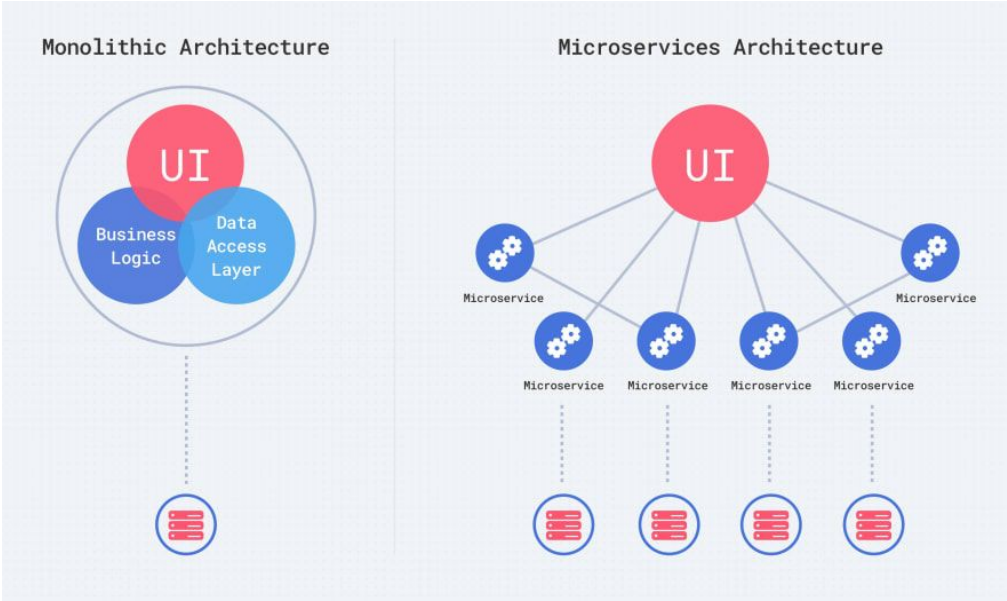


Image: Alex Barashkov

On #dev-chat in Slack...

[9:53am] **@carol:** No to microservices! Microservices will add too much deployment complexity and overhead at runtime. Let's just extract standalone components into libraries, and use them as dependencies. And then we can even open-source them!

Monolith vs. library extraction

Monolith: when your entire application's functionality is implemented inline

Library extraction: when you extract code (that isn't strictly tied to your application) to a separate library and repository that could (theoretically) be useful to other applications

Example: `@reach/router` started as a bunch of helper code in someone's application, then they extracted it to a library that you all use now.

Tip: Starting with library extraction is easy! First, just move a commonly used function and its tests to their own directory.

On #dev-chat in Slack...

[9:58am] **@david:** We need to split our code into multiple repositories to truly let our teams work independently and not need to juggle the entire codebase in their heads and editors. Otherwise, it's just too easy for someone to accidentally entangle 2 components.

Monorepo vs. many repositories

Monorepo: when your entire application is in a single Git repository

Many repositories: when it's split into multiple Git repositories that depend on each other

(Simple idea, but with many consequences!)

Combining microservices, library extraction, multi-repo

Can you...

- use microservices in a monorepo? Why?
- extract libraries but keep a monorepo? Why?
- use microservices and library extraction together? What belongs as a microservice and what as a library?

The many ways to manage code size & complexity

Start with a monolith. Then you can consider:

1. Library extraction

   ...then make sure it's really necessary before continuing...

2. Microservices
3. Many repositories

🔥 Hot take: monoliths, with good code practices, remain the simpler alternative for longer than most people think!

# devs	40
# dev teams	10
# customers	50

Day 1,800

I don't even know where to start!

On #dev-chat in Slack...

[3:12pm] **@andy:** Who can screen-share today to help me understand how we store user profile data?

...

@andy: Anyone? 😟

On #dev-chat in Slack...

[3:12pm] **@andy:** What's the current best way to fetch a user's profile photo in our app?

...

@andy: Anyone? 😟

On #dev-chat in Slack...

[3:12pm] **@andy:** Is it safe for me to remove the confetti banner code now that we switched to the new animation?

...

@andy: Anyone? 😟

On #dev-chat in Slack...

[3:12am] **@andy:** I just got paged about a problem with our app's listSyncedPhotos API. Haven't worked on this API in a year. Anyone have ideas why it's broken or who can help?????

...

@andy: Anyone? 😟😟😟😟😟😟😟😟😟😟😟😟😟😟😟

Code search

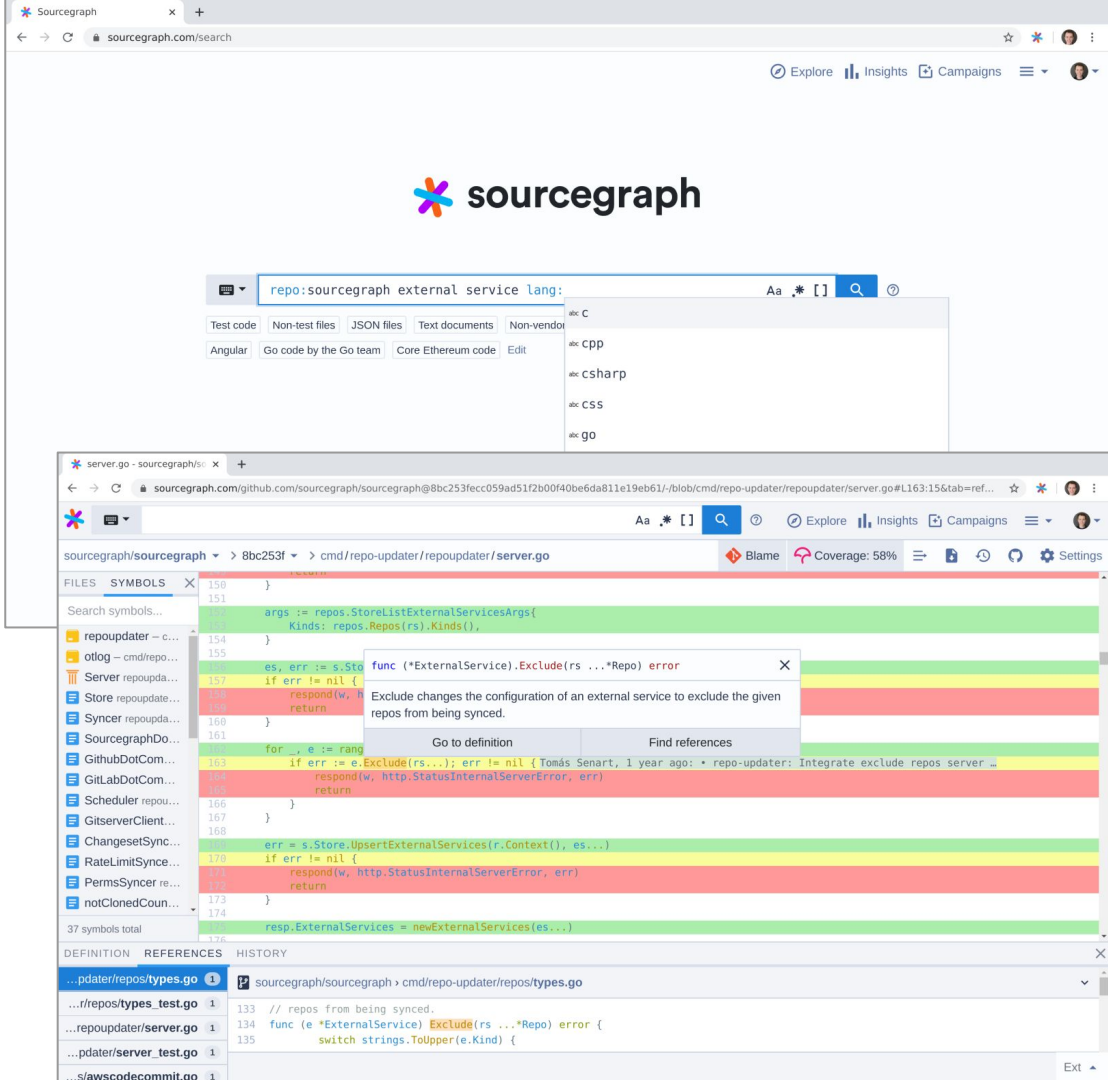
"Google for your code"

A way to search and navigate across all of your code to:

- find usage examples
- see the impact of a change
- fix bugs and see who can help

Examples:

tsx files
router changes
getApolloClient



[tsx files]: #
[router changes]: #
[getApolloClient]: #

Table 1: Categories of answers to fourth survey question, "What question are you trying to answer?"

Category	Example	Count	Percent
Example Code Needed (How)		**87**	**33.5%**
API consumer needs help	"I want to know how a function should be called."	57	22%
Discover correct library for task	"Best way to convert output stream into a string of limited length."	14	5%
Example to build off of	"Just want to copy-and-paste some code I'm changing"	9	3.5%
How to do something	"How to write a hash function"	7	3%
Exploring or Reading Code (What)		**67**	**26%**
Check implementation details	"What a particular script does"	51	20%
Browsing	"Re-familiarizing myself with some code referenced in a CL under review."	11	4%
Check common style	"Where are friend classes usually declared?"	3	1%
Name completion	"I'm looking for an enum member that I think begins with a particular prefix."	2	1%
Code Localization (Where)		**41**	**16%**
Reachability	"Where a class is instantiated"	22	8.5%
Showing to someone else	"I'm trying to create a link to a known piece of code, to provide to someone else."	10	4%
Location in source control	"Where are all the boxed environment configurations?"	9	3.5%
Determine Impact (Why)		**42**	**16%**
Why is something failing	"I'm wondering why a CL I wrote earlier did not fix a currently-occurring production problem and am reading the code to diagnose."	26	10%
Understanding dependencies	"Looking for dependencies of a build file"	12	4.5%
Side effects of a proposed change	"Am I about to blow up production with my CL"	4	1.5%
Metadata (Who and When)		**22**	**8.5%**
Trace code history	"Who last touched some code."	13	5%
Responsibility	"Who is allowed to approve changelists for a particular file."	9	3.5%
Total		259	100.00%

Sadowski, Caitlin, Kathryn T. Stolee, and Sebastian Elbaum. "**How developers search for code: a case study**." Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015. PDF

# devs	80
# dev teams	15
# customers	100

Day 2,100

Our codebase is constantly in motion

On #dev-chat in Slack...

[12:31pm] **@abigail:** I've seen a LOT of bugs from calling useEffect without the right dependency array, like:

```
useEffect(() => addToastListener(user), [])
```

which should be:

```
useEffect(() => addToastListener(user), [user])
```

I'm going to push branches and pull requests to automatically fix this for all of our code (and check it in CI going forward)!

How to run that React hook fix across all code

1. Set up ESLint for each project
2. Use the eslint-plugin-react-hooks ESLint plugin
3. For each repository:
   a. `eslint --fix`
   b. `git commit -a -m 'fix hooks'`
   c. `git push`
   d. `# create pull request`

Very tedious and slow!

Large-scale, automated change tools

"Find-replace across all of your code"

A tool that makes it easy to make a programmatic change across multiple repositories.

1. Run a script in every repository
    a. Record the file changes made by the script
    b. Push a branch and create a PR with the changes
2. Track the progress of all of the PRs

```
# fix-react-hooks.campaign.yaml

name: fix-react-hooks

on:
  - repositoriesMatchingQuery: lang:typescript

steps:
  - run: yarn add -D eslint-plugin-react-hooks && eslint --fix
```

Examples: gofmt

Large-scale change tools are very new/rare

- In use at: Google, Facebook, Lyft, NerdWallet, and just a few other companies I'm aware of
- When available, the # of "bot" commits grows quickly (see chart)

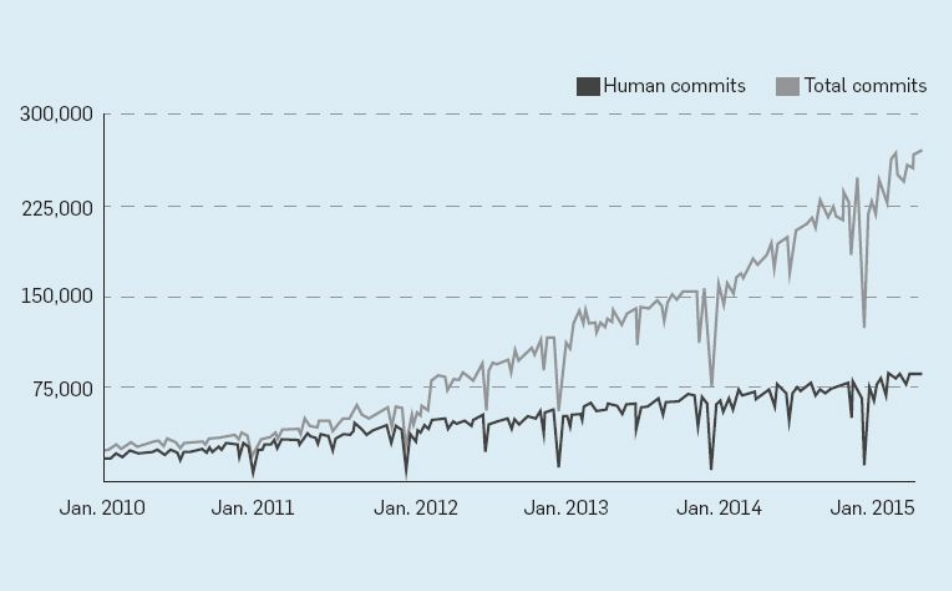


Chart: Google/ACM

Further reading

- Google's Rosie
- Google's ClangMR
- "Large-scale Changes" chapter of SWE at Google book
- NerdWallet's shepherd

# devs	. . .
# dev teams	. . .
# customers	. . .

Day ...

Uncharted territory!

Summary

- Get the right tools and practices in place before you need them!
  - Tests
  - Code review
  - *Code* ownership
  - Monolith vs. microservices vs. lib extract
  - Monorepo vs. many repositories
  - Code search
  - Large-scale automated changes
  - *Observability*
  - *Deployment*
- Improving dev tooling is often the highest-impact thing you can do.
- Fierce competitors can still collaborate on better tooling. Share what works!
- Big Code is inevitable as your company grows.

Sourcegraph

Join us and bring universal code search to every dev & company! Amazing team of ~75 and growing quickly, huge market, massive traction, world-class customers, open-source code.

Work on a product you will use (along with every dev at Uber, Airbnb, Lyft, Twitter, Netflix, Coinbase, etc.)!

Roles: https://about.sourcegraph.com/company/careers
Internships available for any full-time eng/product roles posted there.

Interested?
Email me <sqs@sourcegraph.com>.
I'll connect you to the right team.