Collaboration & Knowledge Sharing

Advanced Software Engineering

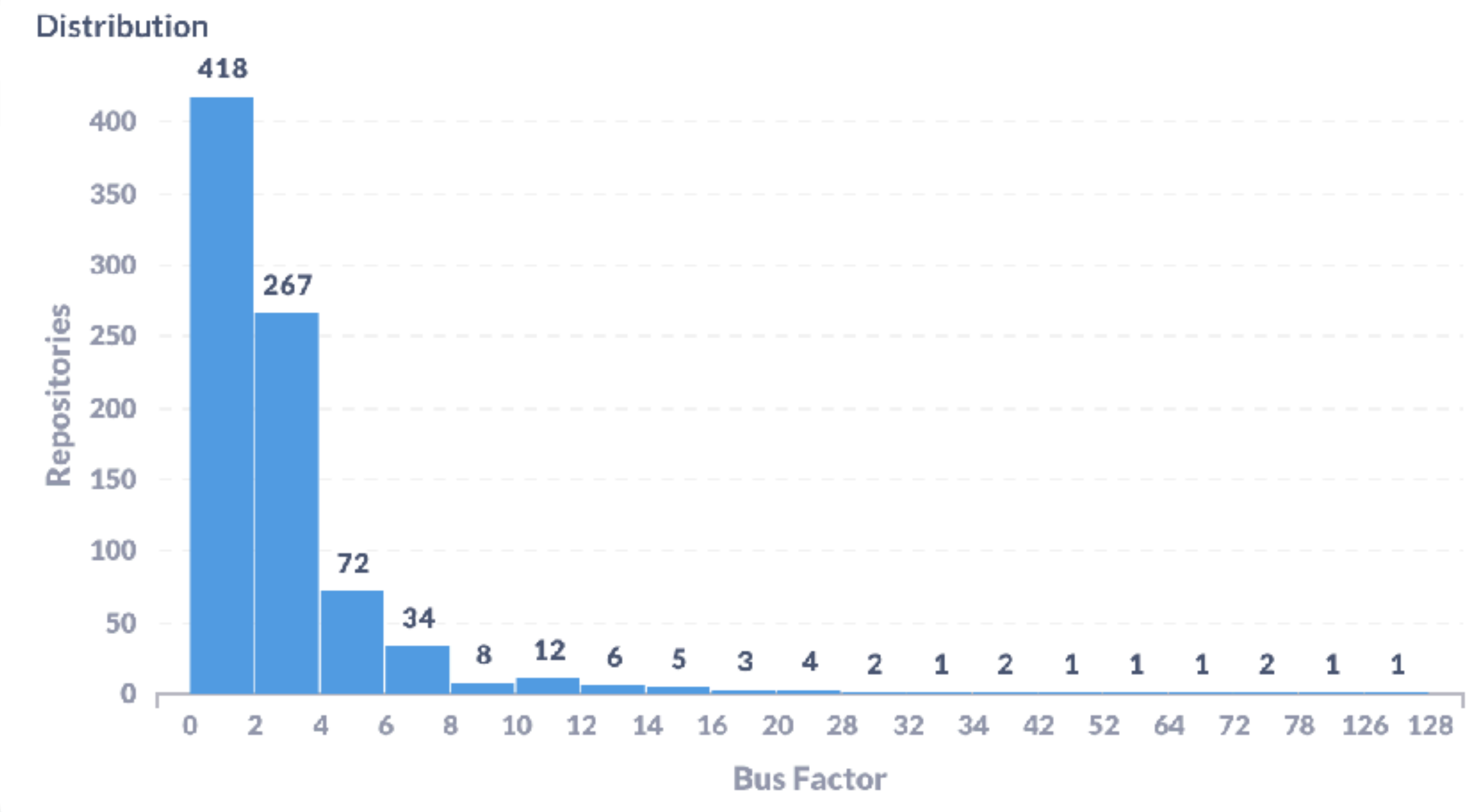
Spring 2023

Why Collaboration? The Bus Factor

Even if one person *can* own all of a project, they shouldn’t be relied to

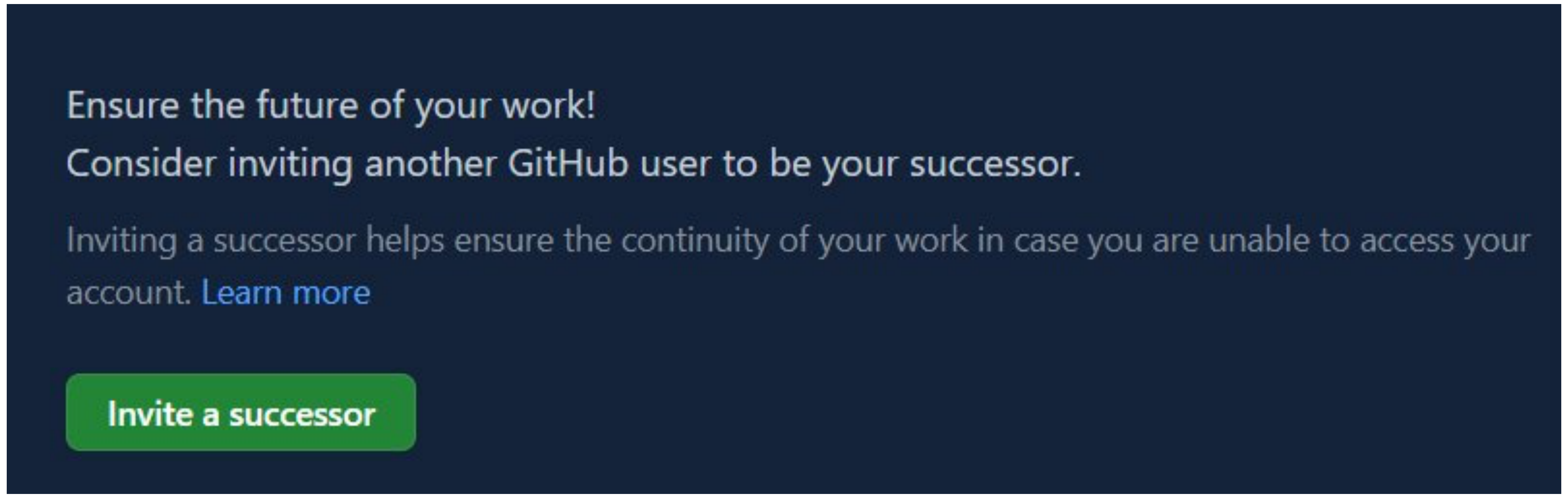
Consider Collaboration and Survivorship in Open Source, Too

Bus factor, top 1,000 projects on GitHub by stars (2023)



https://metabase-public.metabaseapp.com/public/dashboard/38598aee-ee65-4d6d-b459-5e046c3404d4

GitHub Successor Feature (added 2022)



Ensure the future of your work!
Consider inviting another GitHub user to be your successor.

Inviting a successor helps ensure the continuity of your work in case you are unable to access your account. Learn more

Why Collaboration? “The 10x Engineer”

No one person can complete the project alone (or if they can, the project might be too small!)

What makes a 10x Developer?

#10xdeveloper #productivity #beginners #career

Davide de Paolis Mar 11, 2019 · 6 min read

ROCK STAR DEVELOPER
WE'RE LOOKING FOR A ROCK STAR DEVELOPER TO JOIN OUR TEAM
HE'S GOOD!
@SKELETON_CLAW
SKELETONCLAW.COM

Why Collaboration? Software Engineering Draws on Many Skills

Nobody is an expert in everything

- Product management
- Project management
- System-level design and architecture
- Unit-level design
- Development
- Operations
- Maintenance

Collaboration: Why else?

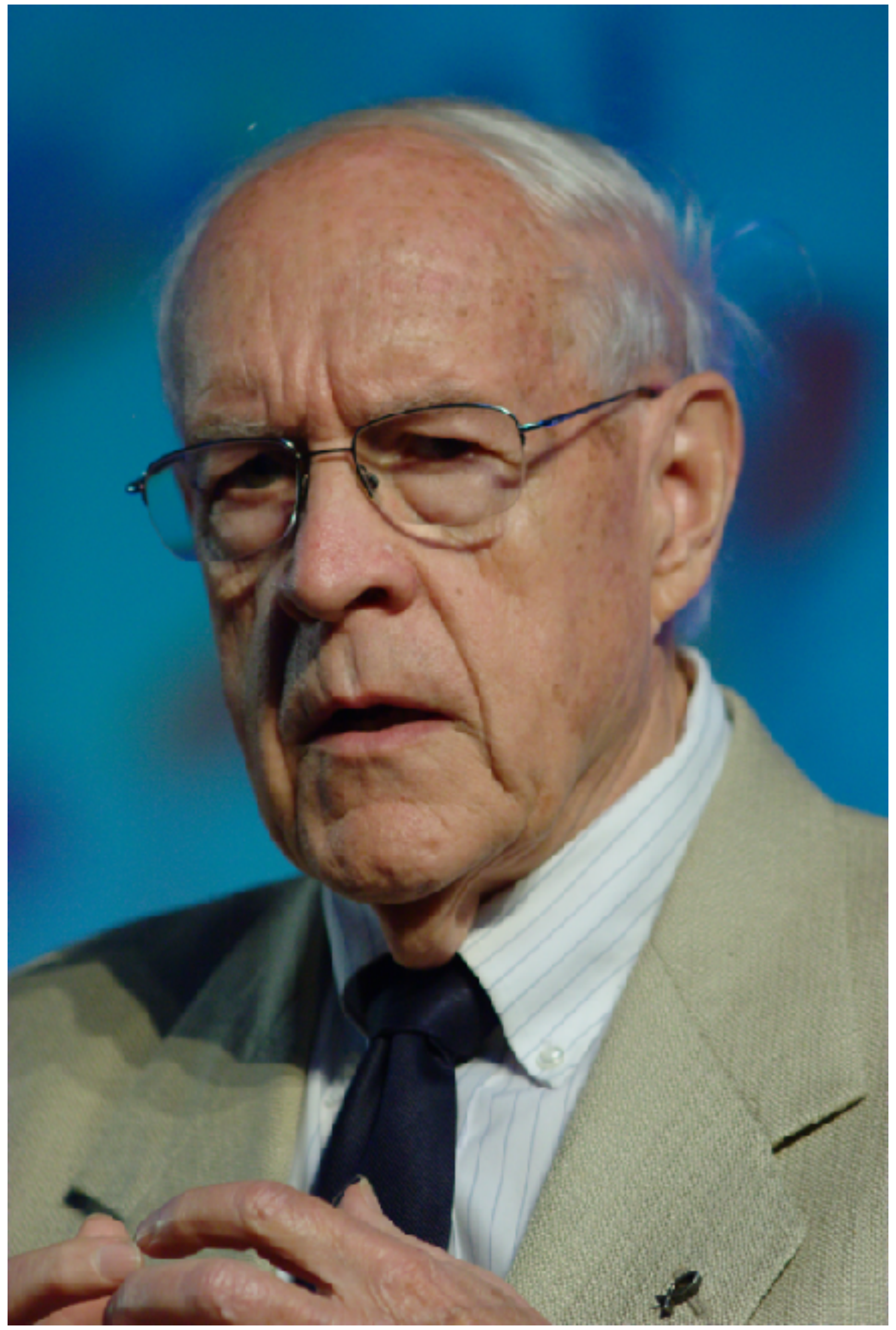
(Brainstorming from class)

- Brainstorming + ideation results in better results (both for the thing you are building, also for the processes)
- Better opportunities for detecting design flaws early on
- Specialization of skills (even within “development”)
- Diversity of expertise, particularly with project failure modes
- Operational resilience - and human resource retention
- Extending the working hours
- Diversity of team members, broadly construed (by geography, by background, by abilities)
- Forcing function for documentation/knowledge sharing

Collaboration is hard: Brooks' Law

"Adding manpower to a late software project makes it later"

Fred Brooks, 1975



Brooks’ Law: Historical Context

AMC’s Halt and Catch Fire, Season 1 Episode 5

How do we structure teams efficiently?

Examining Brooks' Law: "Adding manpower to a late software project makes it later"

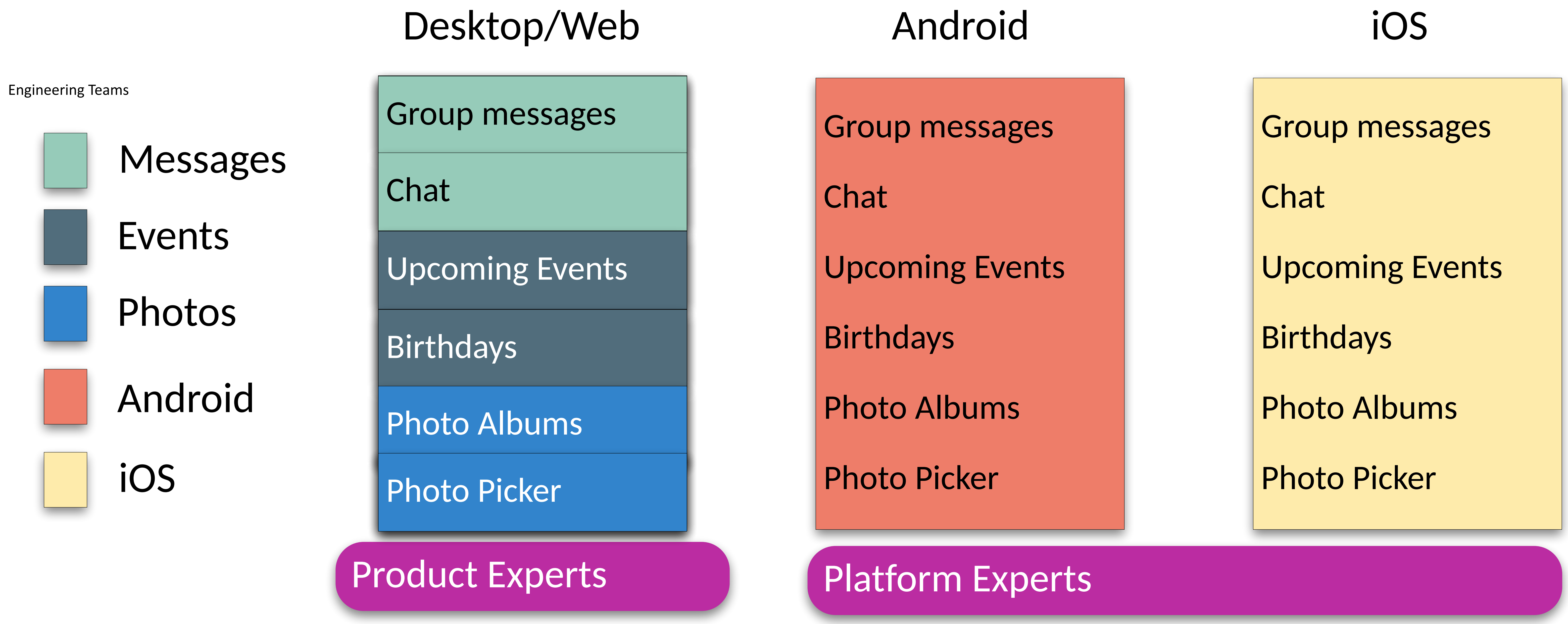
How many communication links are needed to finish a task?

Agile Favors “Two-Pizza” Teams

- Q: How many people on a team?
- A: “No more than you could feed with two pizzas”
- Rationale:
  - Decrease communication burdens
  - Focus conversations to relevant topics

Agile Favors “Product” teams, not ”Platform” teams

Example: Facebook mobile teams (with platform organization)



Engineering Teams

- 🟩 Messages
- ⬛ Events
- 🟦 Photos
- 🟥 Android
- 🟨 iOS

Desktop/Web	Android	iOS
Group messages	Group messages	Group messages
Chat	Chat	Chat
Upcoming Events	Upcoming Events	Upcoming Events
Birthdays	Birthdays	Birthdays
Photo Albums	Photo Albums	Photo Albums
Photo Picker	Photo Picker	Photo Picker

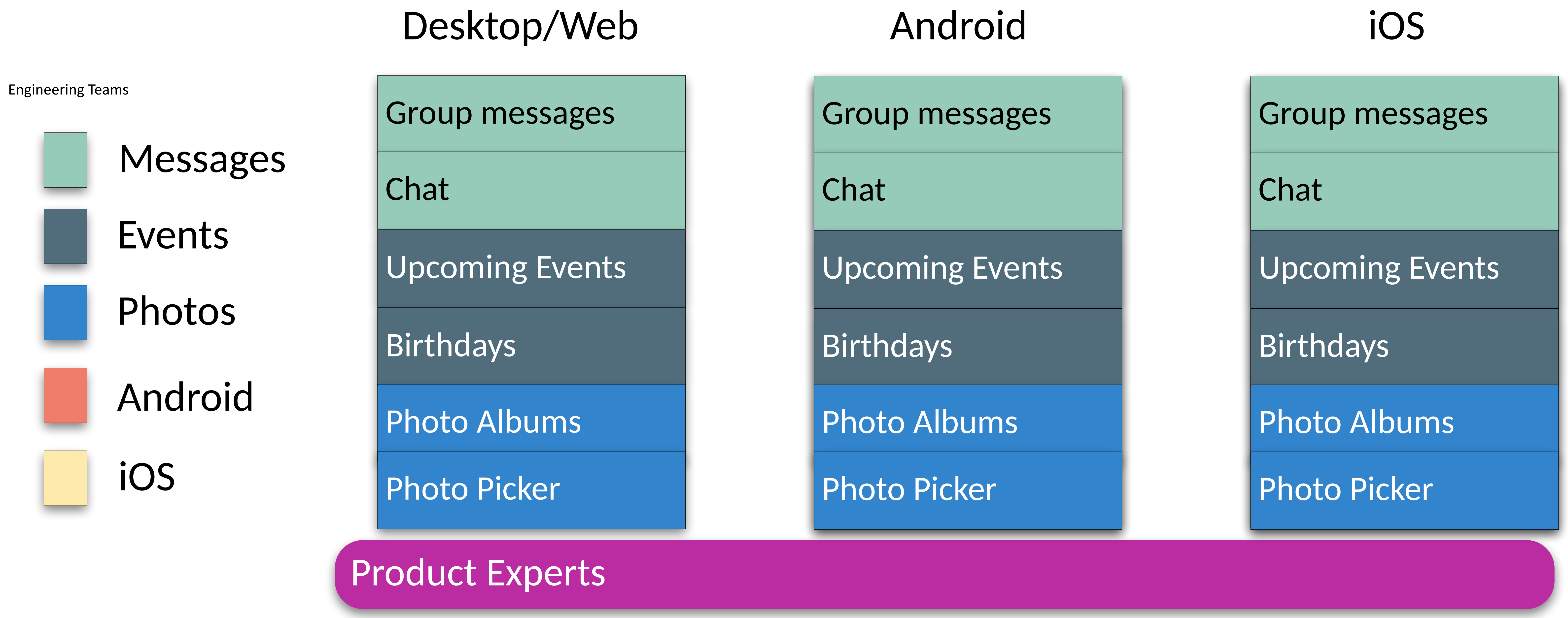
Product Experts

Platform Experts

https://www.youtube.com/watch?v=Nffzkkdq7GM

Agile Favors “Product” teams, not ”Platform” teams

Example: Facebook mobile teams (with platform organization)



Engineering Teams

- 🟩 Messages
- ⬛ Events
- 🟦 Photos
- 🟥 Android
- 🟨 iOS

Desktop/Web	Android	iOS
Group messages	Group messages	Group messages
Chat	Chat	Chat
Upcoming Events	Upcoming Events	Upcoming Events
Birthdays	Birthdays	Birthdays
Photo Albums	Photo Albums	Photo Albums
Photo Picker	Photo Picker	Photo Picker

Product Experts

https://www.youtube.com/watch?v=Nffzkkdq7GM

Create Intentional Opportunities for Knowledge Sharing

Ideally, scale linearly (or sub linearly) with org growth

(Brainstorming from class)

- Create onboarding processes, onboard multiple people concurrently
- Make One System To Rule Them All for wikis/chat/knowledge sharing, consider access-management
- Hackathons - group programming/development events to make “spike efforts”
- Recognize long-term impacts, not just short-term impacts (particularly around tool development)
- Put people in a room and make them explain stuff to each other (design conflict resolution meetings). Ensure that there is a process for “keeping everyone on the same page”
- Structured mentoring systems (including pair programming)
- Code review
- Asynchronous communication channels within teams, for troubleshooting + to create a searchable system
- Encourage organic, but disciplined meetings that have tangible and documented artifacts
- Experiment with workspace furniture

Pair Programming as a Mentoring Activity

- Two programmers work together at one computer, one “driving,” one “navigating”
- Survey of professional programmers (2001):
  - 90% “enjoyed collaborative programming more than solo programming”
  - 95% were “more confident in their solutions” when pair programmed
  - Provides long-term benefits: reduces defects by 15%, code size by 15%
  - Increases costs by 15% to 100% compared to single developer on the task

Cockburn and Williams. The Costs and Benefits of Pair Programming, (In: Extreme Programming Explained 2001)

Pair Programming Improves Tool Diffusion

Emerson Murphy-Hill & Gail C. Murphy, CSCW 2011

- Peer observation and recommendation shown to be more effective at discovering new tools than other knowledge sharing approaches

- Examples: Hot keys, especially for CLI; IDE tricks

- Most common in 2011 survey: “Open Type” feature in Eclipse, developer tools in web browser

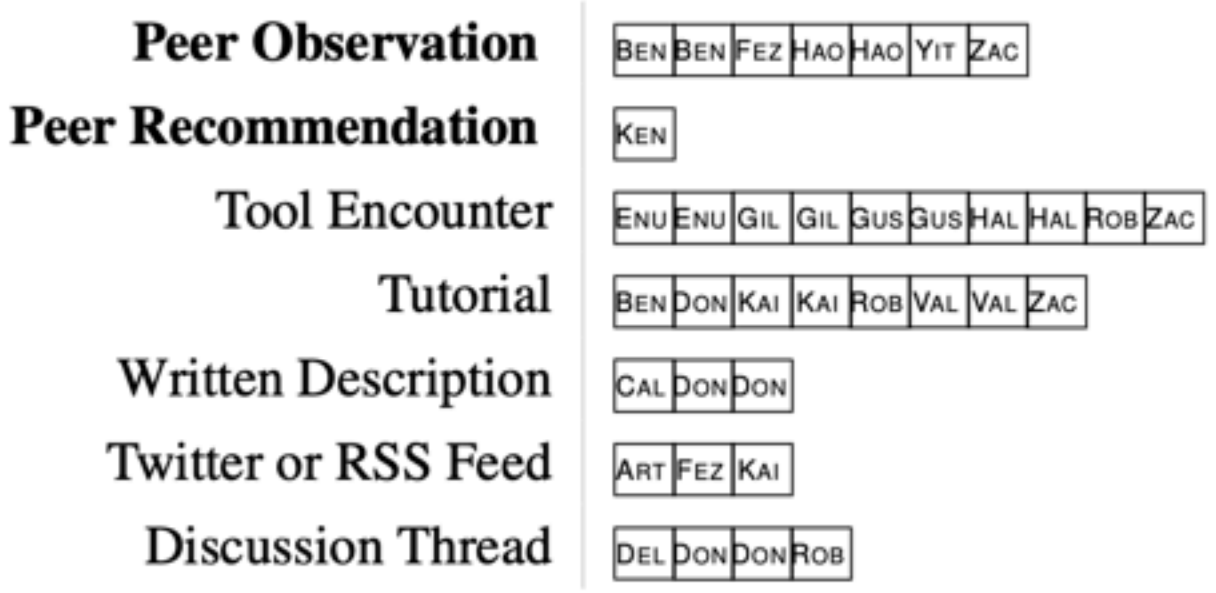


Figure 2: Histogram of the most frequent discovery modes.

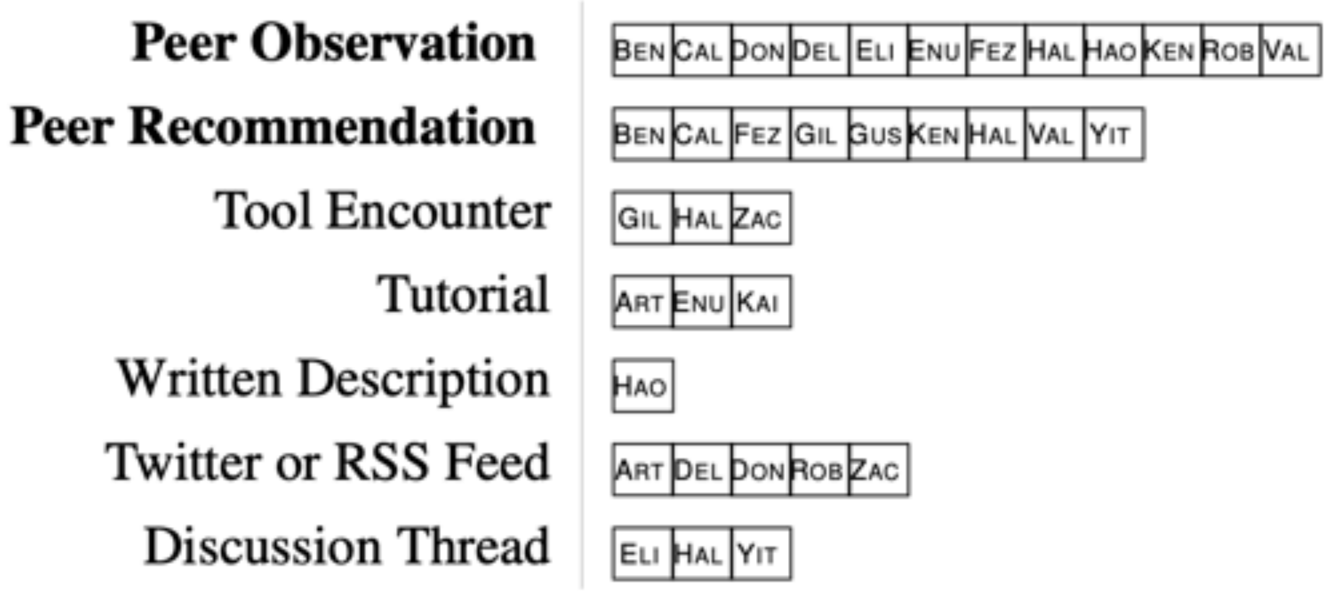
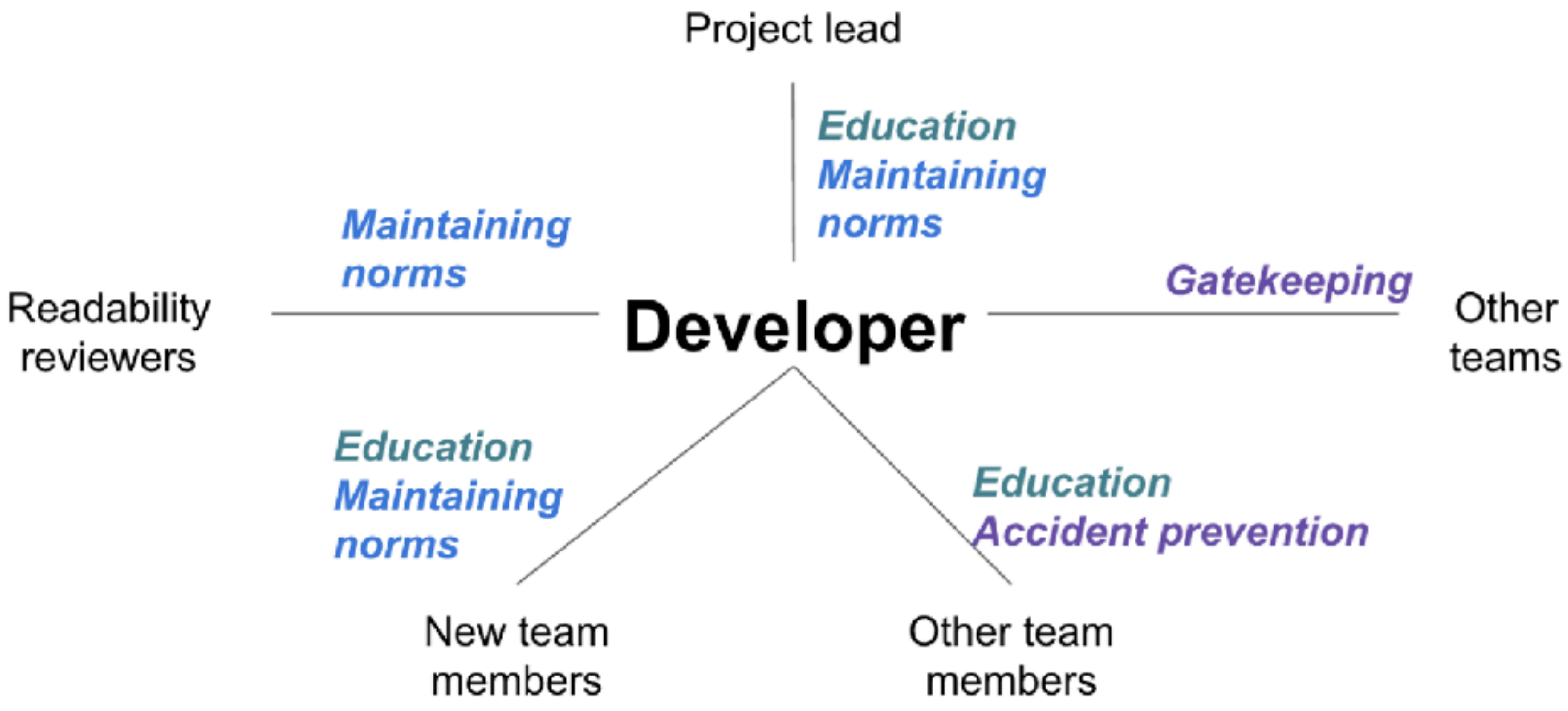


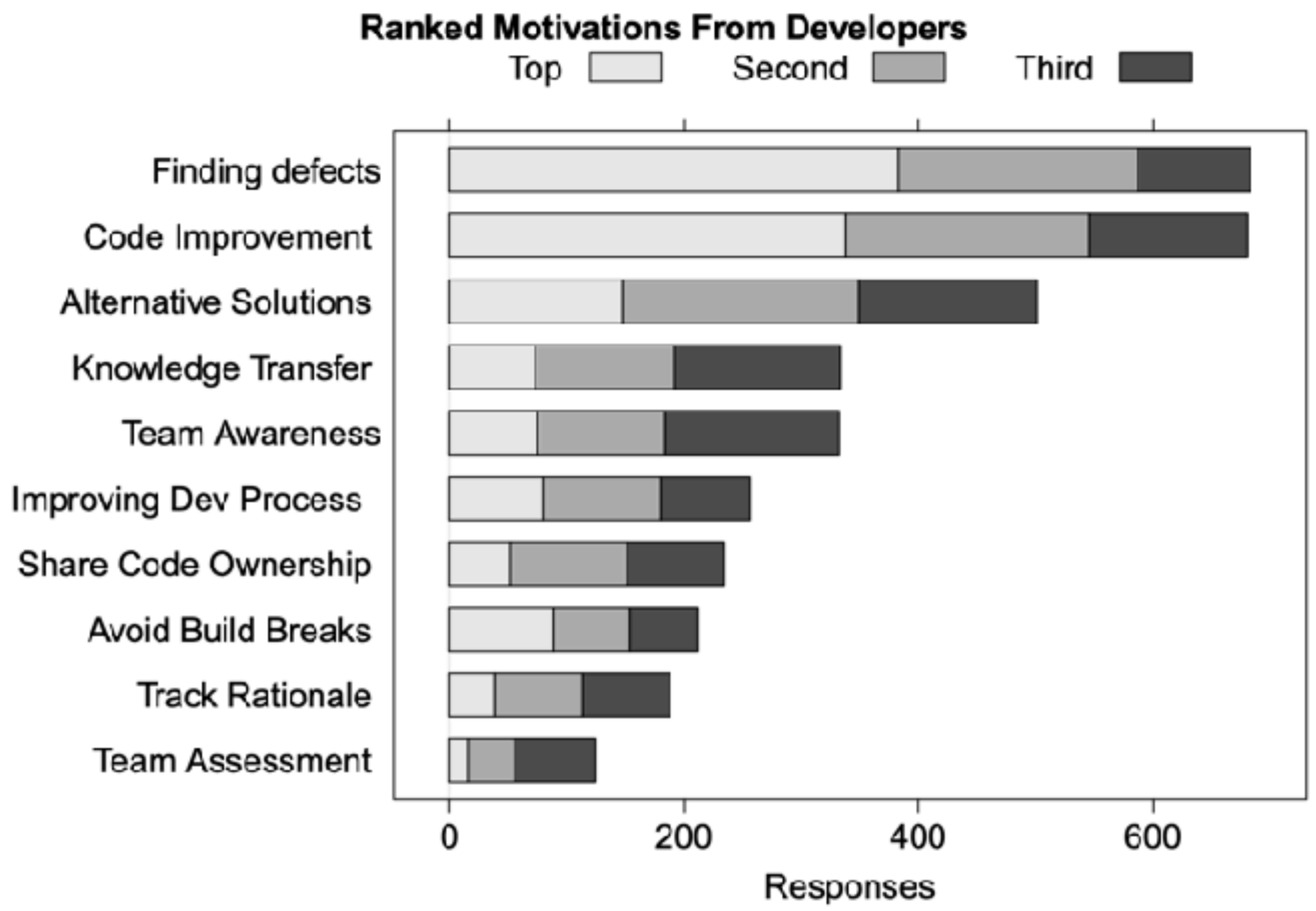
Figure 3: Histogram of the most effective discovery modes.

“Peer interaction effectively, yet infrequently, enables programmers to discover new tools”, Emerson Murphy-Hill & Gail C. Murphy, CSCW 2011

Code Review as a Knowledge Sharing Opportunity



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



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

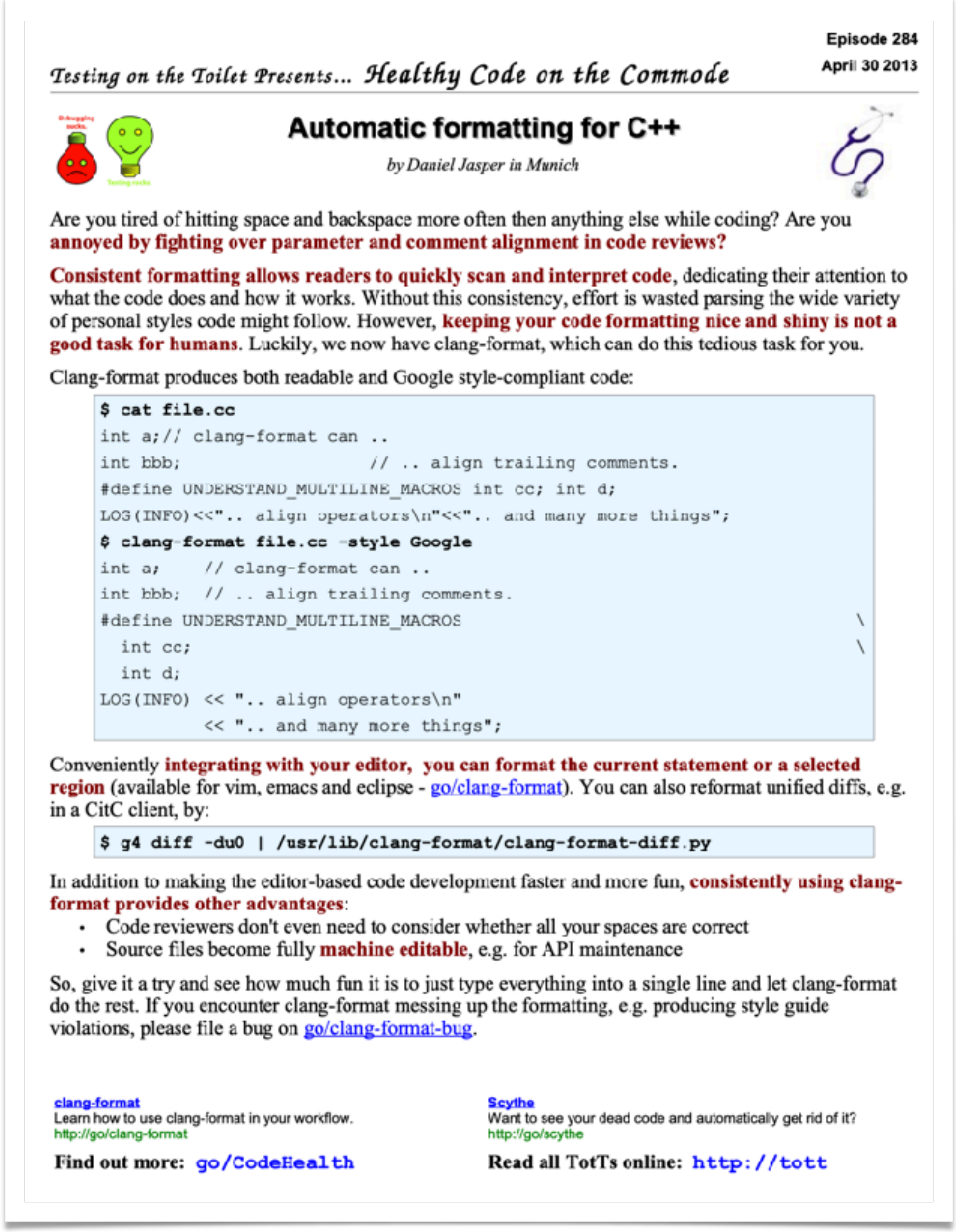
Standardize and Document Best Practices

Wikis, blogs, tech talks scale-out more than 1:1 mentoring

- Rule of thumb: once you have explained something to more than two people, maybe you should write a blog post

- Effective organizations cultivate programs to organically collect and share knowledge and best practices

- Example: Google “Testing on the Toilet” (c 2006)



Debugging sucks.
Testing rocks

Episode 284
April 30 2013

Testing on the Toilet Presents... Healthy Code on the Commode

Automatic formatting for C++

by Daniel Jasper in Munich

Are you tired of hitting space and backspace more often then anything else while coding? Are you **annoyed by fighting over parameter and comment alignment in code reviews?**

Consistent formatting allows readers to quickly scan and interpret code, dedicating their attention to what the code does and how it works. Without this consistency, effort is wasted parsing the wide variety of personal styles code might follow. However, **keeping your code formatting nice and shiny is not a good task for humans**. Luckily, we now have clang-format, which can do this tedious task for you.

Clang-format produces both readable and Google style-compliant code:

```
$ cat file.cc
int a;// clang-format can ..
int bbb;                  // .. align trailing comments.
#define UNDERSTAND_MULTILINE_MACROS int cc; int d;
LOG(INFO)<<".. align operators\n"<<".. and many more things";
$ clang-format file.cc -style Google
int a;    // clang-format can ..
int bbb;  // .. align trailing comments.
#define UNDERSTAND_MULTILINE_MACROS                                  \
  int cc;                                                            \
  int d;
LOG(INFO) << ".. align operators\n"
          << ".. and many more things";
```

Conveniently **integrating with your editor, you can format the current statement or a selected region** (available for vim, emacs and eclipse - go/clang-format). You can also reformat unified diffs, e.g. in a CitC client, by:

```
$ g4 diff -du0 | /usr/lib/clang-format/clang-format-diff.py
```

In addition to making the editor-based code development faster and more fun, **consistently using clang-format provides other advantages**:

- Code reviewers don't even need to consider whether all your spaces are correct
- Source files become fully **machine editable**, e.g. for API maintenance

So, give it a try and see how much fun it is to just type everything into a single line and let clang-format do the rest. If you encounter clang-format messing up the formatting, e.g. producing style guide violations, please file a bug on go/clang-format-bug.

clang-format
Learn how to use clang-format in your workflow.
http://go/clang-format

Scythe
Want to see your dead code and automatically get rid of it?
http://go/scythe

Find out more: go/CodeHealth

Read all TotTs online: http://tott

Do Developers Discover New Tools On The Toilet?

Murphy-Hill et al, ICSE 2019

- Exposure to the flyers significantly increased and sustained adoption of the tools advertised on them
- Provided more “memorability” compared to social media (location + curation)
- Limitations
  - Not evenly posted and updated globally (volunteer effort; minority tax)
  - Editorial curation is difficult
  - Not all episodes are relevant to all teams

How Developers use Social Media

“How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development”
Storey et al, TSE 2015

- On average, developers use *eleven* channels to stay up-to-date on development activities

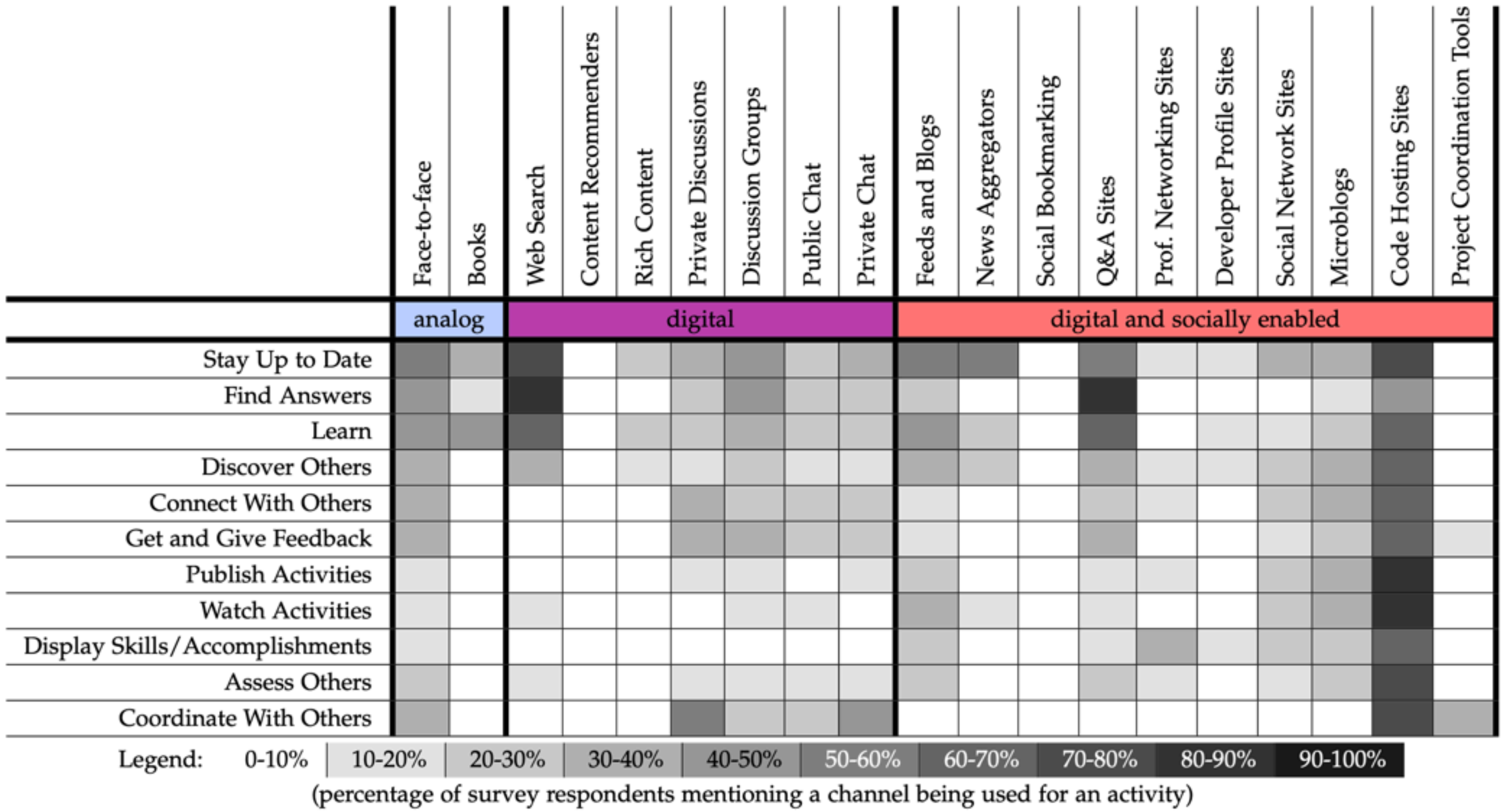
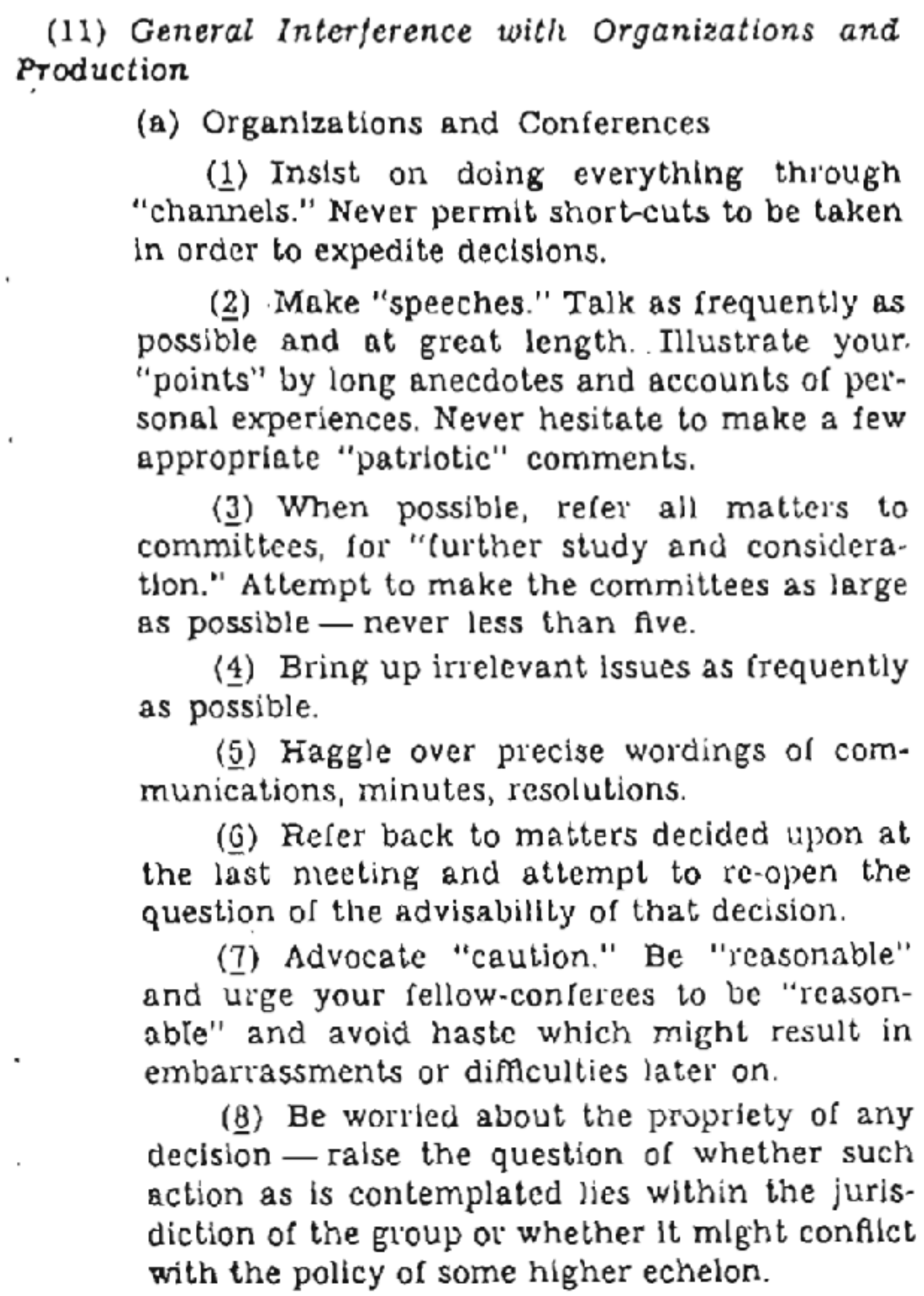


TABLE 4
Channels used by our respondents and the activities they support.

Anti-Patterns for Teams

(CIA Sabotage Guide c 1944)



(11) *General Interference with Organizations and Production*

(a) Organizations and Conferences

(1) Insist on doing everything through "channels." Never permit short-cuts to be taken in order to expedite decisions.

(2) Make "speeches." Talk as frequently as possible and at great length. Illustrate your "points" by long anecdotes and accounts of personal experiences. Never hesitate to make a few appropriate "patriotic" comments.

(3) When possible, refer all matters to committees, for "further study and consideration." Attempt to make the committees as large as possible — never less than five.

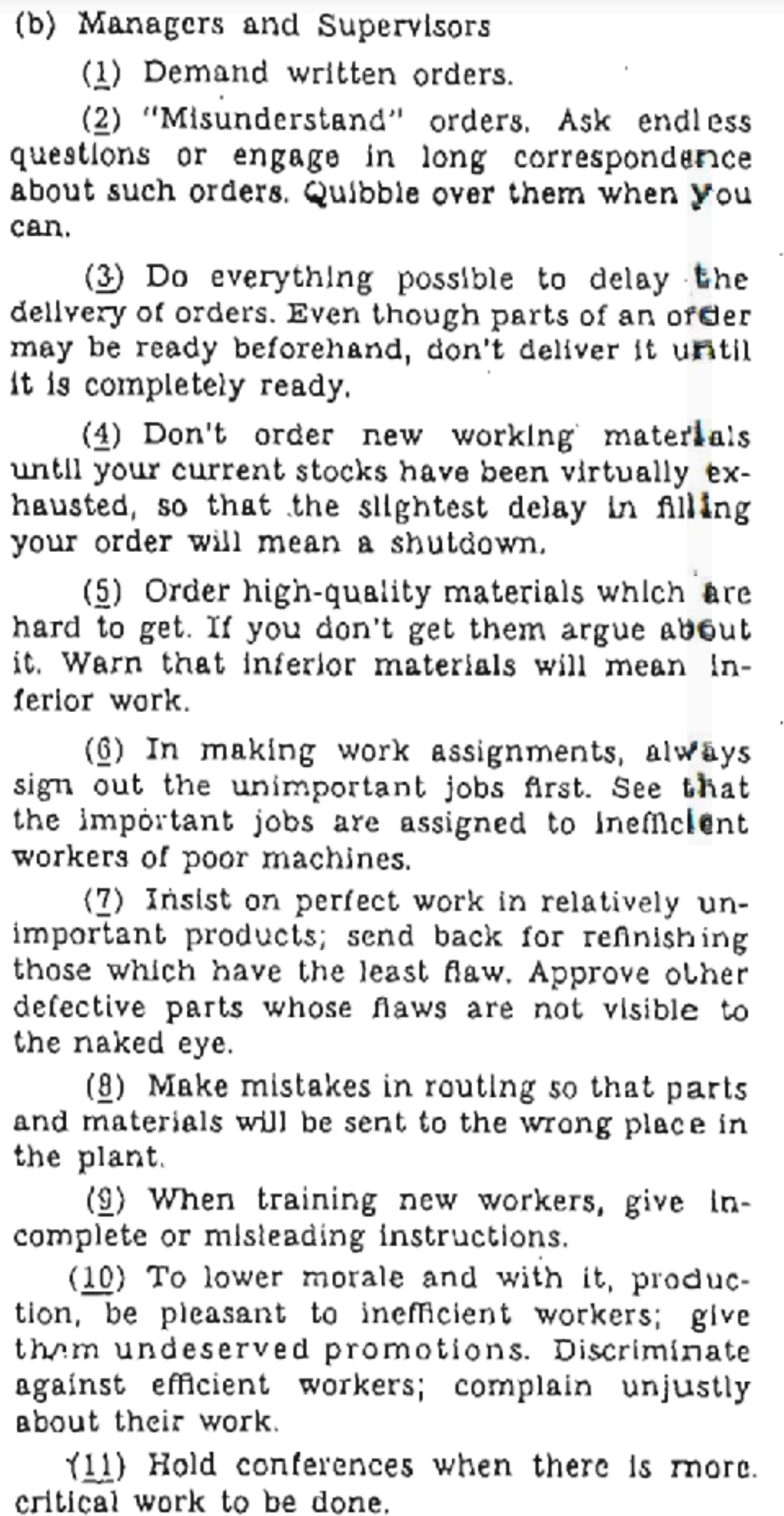
(4) Bring up irrelevant issues as frequently as possible.

(5) Haggle over precise wordings of communications, minutes, resolutions.

(6) Refer back to matters decided upon at the last meeting and attempt to re-open the question of the advisability of that decision.

(7) Advocate "caution." Be "reasonable" and urge your fellow-conferees to be "reasonable" and avoid haste which might result in embarrassments or difficulties later on.

(8) Be worried about the propriety of any decision — raise the question of whether such action as is contemplated lies within the jurisdiction of the group or whether it might conflict with the policy of some higher echelon.



(b) Managers and Supervisors

(1) Demand written orders.

(2) "Misunderstand" orders. Ask endless questions or engage in long correspondence about such orders. Quibble over them when you can.

(3) Do everything possible to delay the delivery of orders. Even though parts of an order may be ready beforehand, don't deliver it until it is completely ready.

(4) Don't order new working materials until your current stocks have been virtually exhausted, so that the slightest delay in filling your order will mean a shutdown.

(5) Order high-quality materials which are hard to get. If you don't get them argue about it. Warn that inferior materials will mean inferior work.

(6) In making work assignments, always sign out the unimportant jobs first. See that the important jobs are assigned to inefficient workers of poor machines.

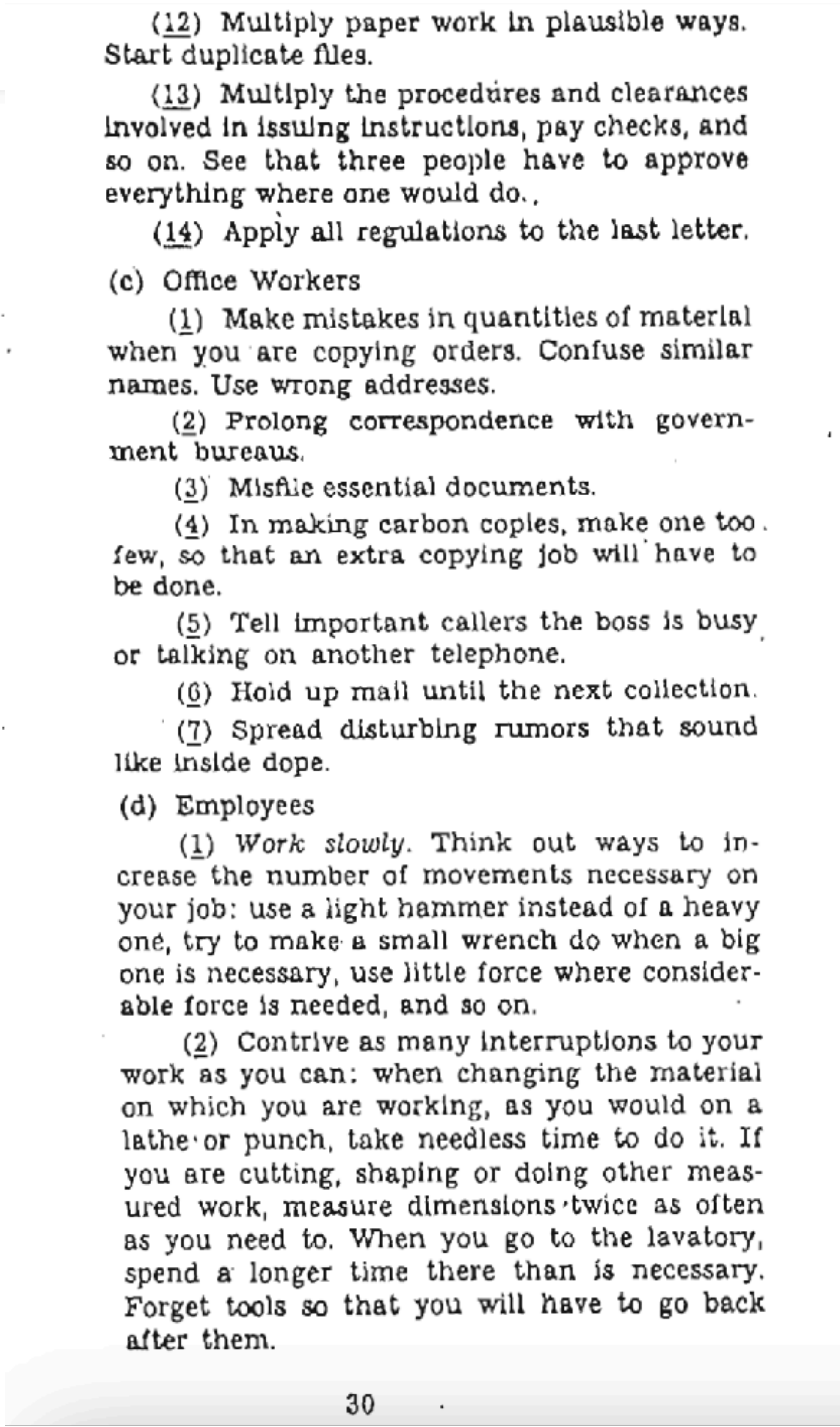
(7) Insist on perfect work in relatively unimportant products; send back for refinishing those which have the least flaw. Approve other defective parts whose flaws are not visible to the naked eye.

(8) Make mistakes in routing so that parts and materials will be sent to the wrong place in the plant.

(9) When training new workers, give incomplete or misleading instructions.

(10) To lower morale and with it, production, be pleasant to inefficient workers; give them undeserved promotions. Discriminate against efficient workers; complain unjustly about their work.

(11) Hold conferences when there is more critical work to be done.



(12) Multiply paper work in plausible ways. Start duplicate files.

(13) Multiply the procedures and clearances involved in issuing instructions, pay checks, and so on. See that three people have to approve everything where one would do.

(14) Apply all regulations to the last letter.

(c) Office Workers

(1) Make mistakes in quantities of material when you are copying orders. Confuse similar names. Use wrong addresses.

(2) Prolong correspondence with government bureaus.

(3) Misfile essential documents.

(4) In making carbon copies, make one too few, so that an extra copying job will have to be done.

(5) Tell important callers the boss is busy or talking on another telephone.

(6) Hold up mail until the next collection.

(7) Spread disturbing rumors that sound like inside dope.

(d) Employees

(1) *Work slowly.* Think out ways to increase the number of movements necessary on your job: use a light hammer instead of a heavy one, try to make a small wrench do when a big one is necessary, use little force where considerable force is needed, and so on.

(2) Contrive as many interruptions to your work as you can: when changing the material on which you are working, as you would on a lathe or punch, take needless time to do it. If you are cutting, shaping or doing other measured work, measure dimensions twice as often as you need to. When you go to the lavatory, spend a longer time there than is necessary. Forget tools so that you will have to go back after them.

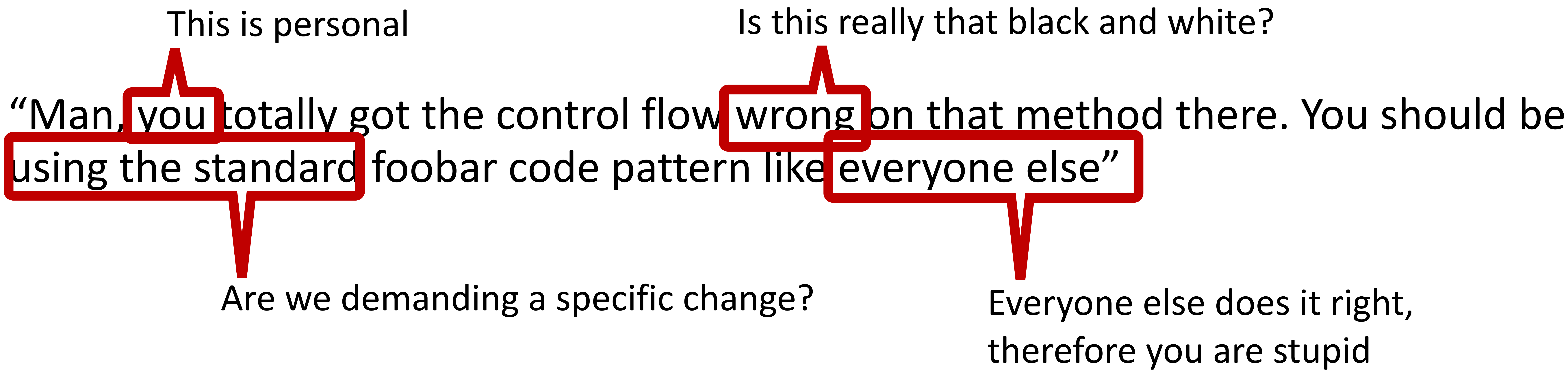
Three Pillars of Social Skills for Collaboration

A pattern for effective teams

- Pillar 1: Humility: You are not the center of the universe (nor is your code!). You're neither omniscient nor infallible. You're open to self-improvement.
- Pillar 2: Respect: You genuinely care about others you work with. You treat them kindly and appreciate their abilities and accomplishments.
- Pillar 3: Trust: You believe others are competent and will do the right thing, and you're OK with letting them drive when appropriate.

From "Debugging Teams" by Ben Collins-Sussman and Brian Fitzpatrick

HRT Example: Code Review



This is personal

Is this really that black and white?

“Man, you totally got the control flow wrong on that method there. You should be using the standard foobar code pattern like everyone else”

Are we demanding a specific change?

Everyone else does it right, therefore you are stupid

From “Debugging Teams” by Ben Collins-Sussman and Brian Fitzpatrick

HRT Example: Code Review

“Man, you totally got the control flow wrong on that method there. You should be using the standard foobar code pattern like everyone else”

“Hmm, I’m confused by the control flow in this section here. I wonder if the foobar code pattern might make this clearer and easier to maintain?

Humility! This is about *me,* not you

From “Debugging Teams” by Ben Collins-Sussman and Brian Fitzpatrick

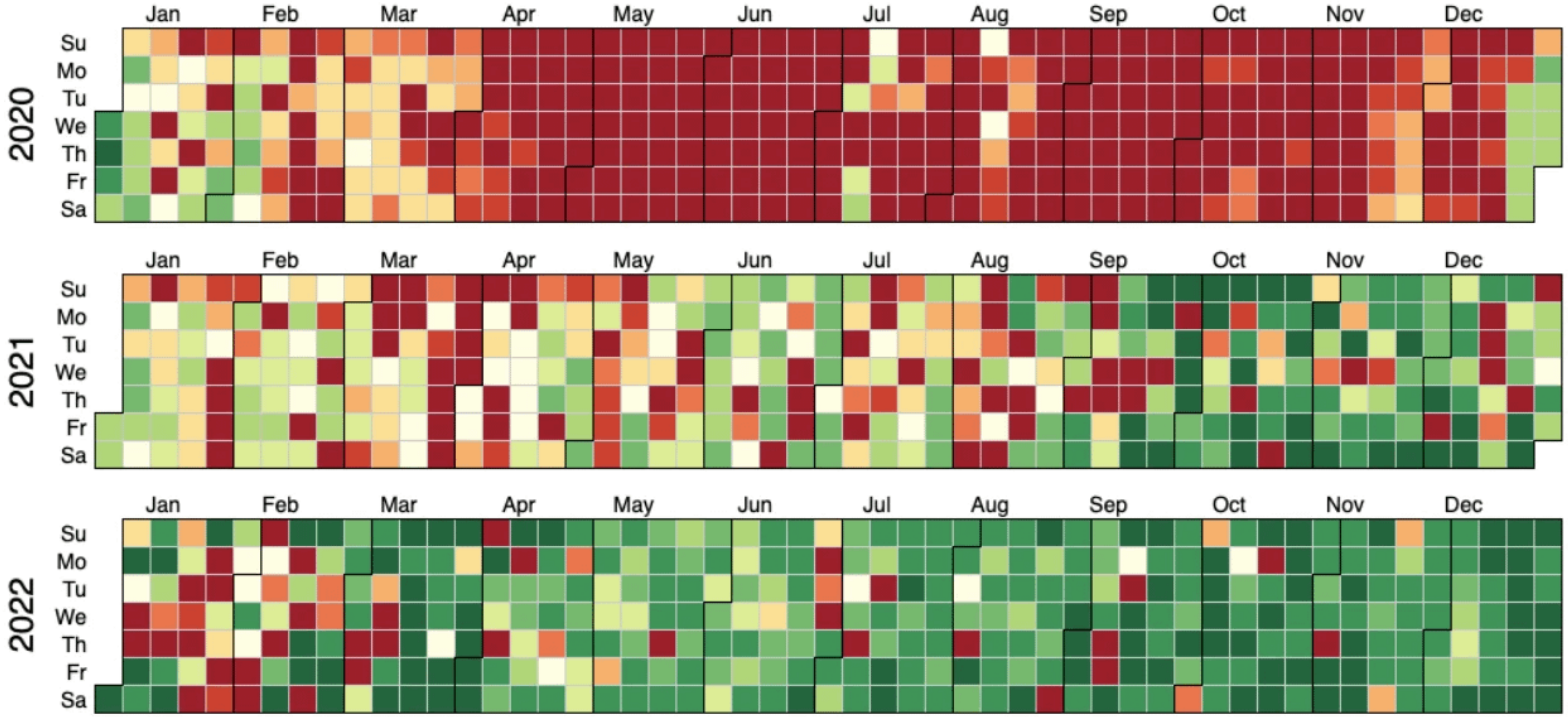
HRT Example: Asking and Answering Questions

- Questions:
  - “I can't get the code from project foo to compile. Why is it broken?” Vs
  - “The code from project foo doesn't compile under Nulix version 6.2. I've read the FAQ, but it doesn't have anything in it about Nulix-related problems. Here's a transcript of my compilation attempt; is it something I did?”
- Answers:
  - Help others learn from the question: “How would that FAQ need to be updated so that nobody has to answer this question gain?”
  - Demonstrate your skills (how you reached the answer) rather than your omniscience (providing the minimal, but correct answer)

Examples via the interesting site: “How To Ask Questions The Smart Way” by Eric S Raymond & Rick Moen

Knowledge Sharing Case Study: Reddit 3/14/2023

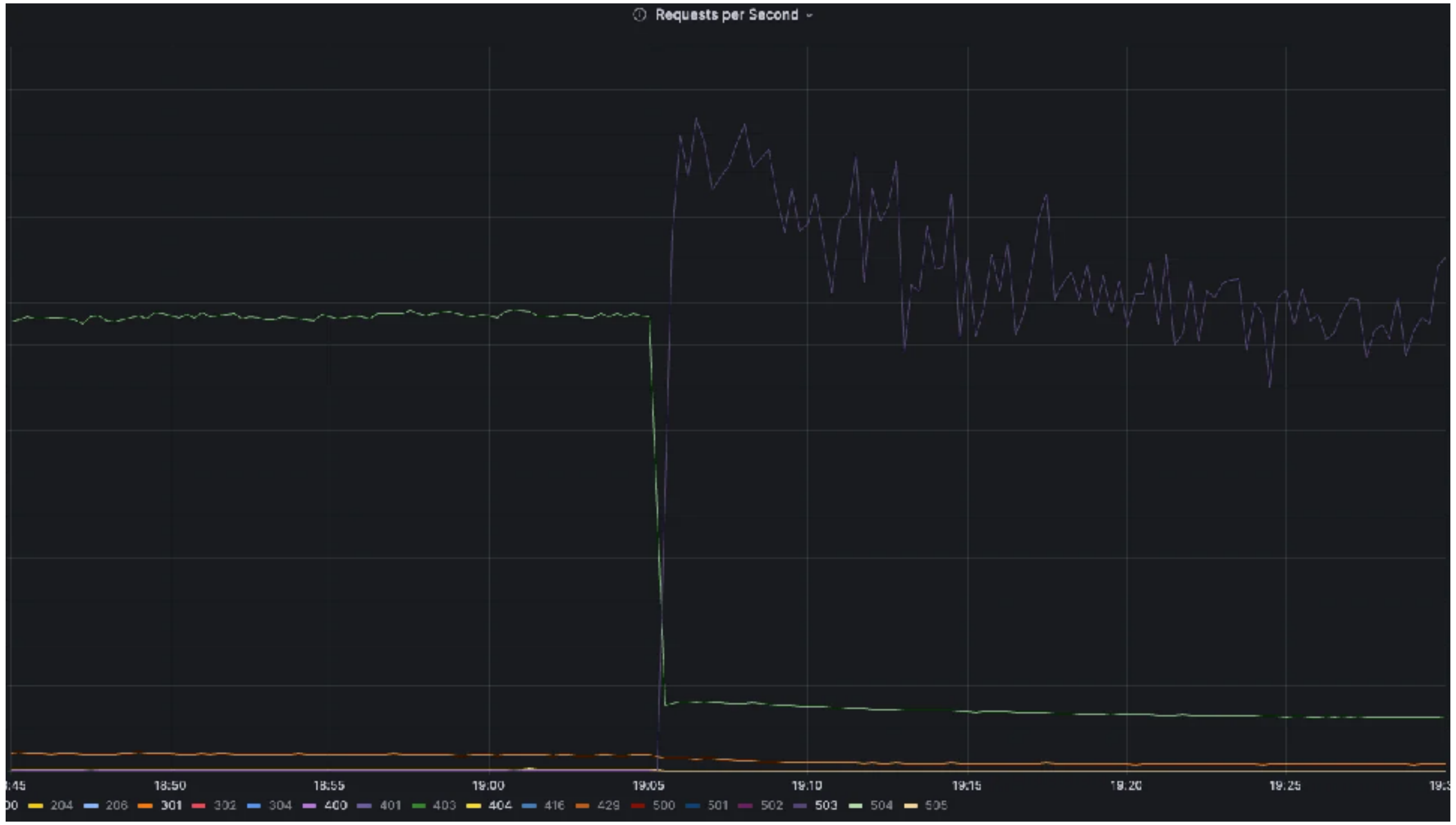
Context: Reddit has been working on standardizing and improving infrastructure for reliability



https://www.reddit.com/r/RedditEng/comments/11xx5o0/you_broke_reddit_the_piday_outage/

Knowledge Sharing Case Study: Reddit 3/14/2023

“**The route reflectors were set up several years ago by the precursor to the current Compute team. Time passed, and with attrition and growth, everyone who knew they existed moved on to other roles or other companies.** Only our largest and most legacy clusters still use them. So there was nobody with the knowledge to interact with the route reflector configuration to even realize there could be something wrong with it or to be able to speak up and investigate the issue. Further, Calico’s configuration doesn’t actually work in a way that can be easily managed via code. Part of the route reflector configuration requires fetching down Calico-specific data that’s expected to only be managed by their CLI interface (not the standard Kubernetes API), hand-edited, and uploaded back. To make this acceptable means writing custom tooling to do so. Unfortunately, we hadn’t. The route reflector configuration was thus committed nowhere, leaving us with no record of it, and no breadcrumbs for engineers to follow. One engineer happened to remember that this was a feature we utilized, and did the research during this postmortem process, discovering that this was what actually affected us and how.”

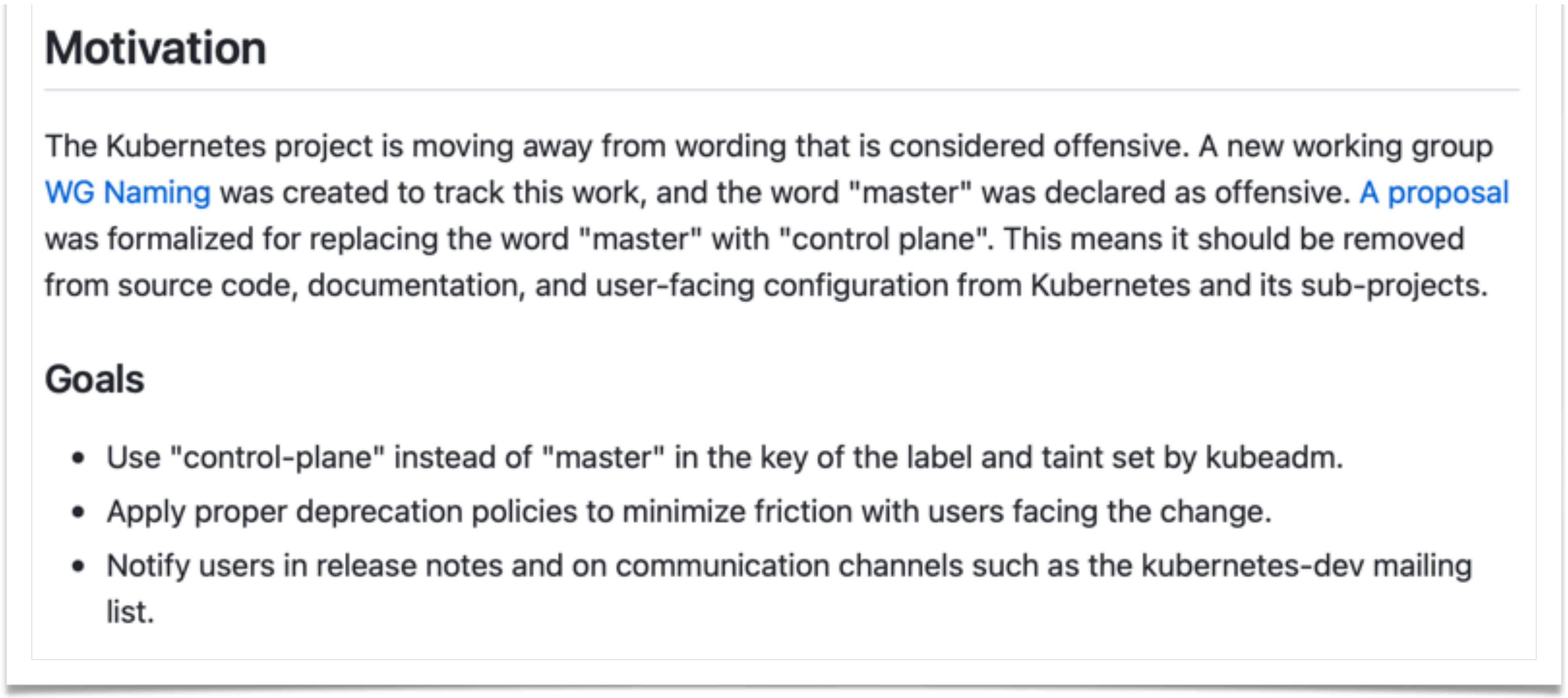


https://www.reddit.com/r/RedditEng/comments/11xx5o0/you_broke_reddit_the_piday_outage/

Knowledge Sharing Case Study: Reddit 3/14/2023

Technical cause for the failed upgrade: configuration parameter renamed

KEP-2067: Rename the kubeadm "master" label and taint

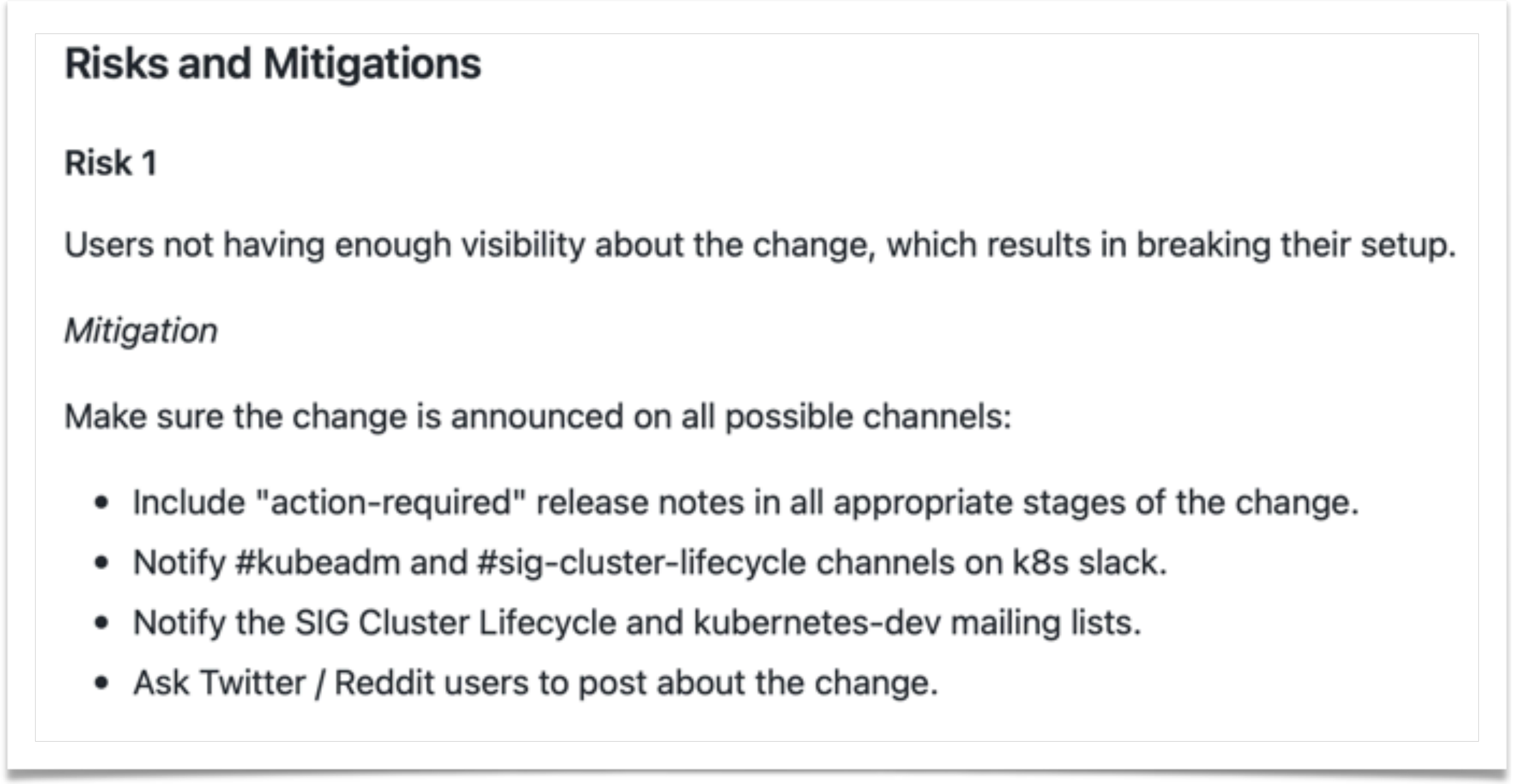


Motivation

The Kubernetes project is moving away from wording that is considered offensive. A new working group WG Naming was created to track this work, and the word "master" was declared as offensive. A proposal was formalized for replacing the word "master" with "control plane". This means it should be removed from source code, documentation, and user-facing configuration from Kubernetes and its sub-projects.

Goals

- Use "control-plane" instead of "master" in the key of the label and taint set by kubeadm.
- Apply proper deprecation policies to minimize friction with users facing the change.
- Notify users in release notes and on communication channels such as the kubernetes-dev mailing list.



Risks and Mitigations

Risk 1

Users not having enough visibility about the change, which results in breaking their setup.

Mitigation

Make sure the change is announced on all possible channels:

- Include "action-required" release notes in all appropriate stages of the change.
- Notify #kubeadm and #sig-cluster-lifecycle channels on k8s slack.
- Notify the SIG Cluster Lifecycle and kubernetes-dev mailing lists.
- Ask Twitter / Reddit users to post about the change.

https://github.com/kubernetes/enhancements/blob/master/keps/sig-cluster-lifecycle/kubeadm/2067-rename-master-label-taint/README.md

Root cause for the failed upgrade: Inconsistent, undocumented configurations

> "Nearly every critical Kubernetes cluster at Reddit is bespoke in one way or another. Whether it's unique components that only run on that cluster, unique workloads, only running in a single availability zone as a development cluster, or any number of other things. This is a natural consequence of organic growth, and one which has caused more outages than we can easily track over time."

https://www.reddit.com/r/RedditEng/comments/11xx5o0/you_broke_reddit_the_piday_outage/

Responding to Failures

In software, in humans, and in processes.

How do we learn:

- What went well?
- What went wrong?
- Where we got lucky?
- How do we prevent it from happening again?

How Not to Respond to Failures

1. Some engineer contributes to failure or incident
2. Engineer is punished/shamed/blamed/retrained
3. Engineers as a whole become silent on details to management to avoid being scapegoated
4. Management becomes less informed about what actually is happening, do not actually find/fix root causes of incidents
5. Process repeats, amplifying every time

Blameless Post-Mortems

- What actions did you take at the time?
- What effects did you observe at the time?
- What were the expectations that you had?
- What assumptions did you make?
- What is your understanding of the timeline of events as they occurred?

Lessons Learned

What went well

- Monitoring quickly alerted us to high rate (reaching ~100%) of HTTP 500s
- Rapidly distributed updated Shakespeare corpus to all clusters

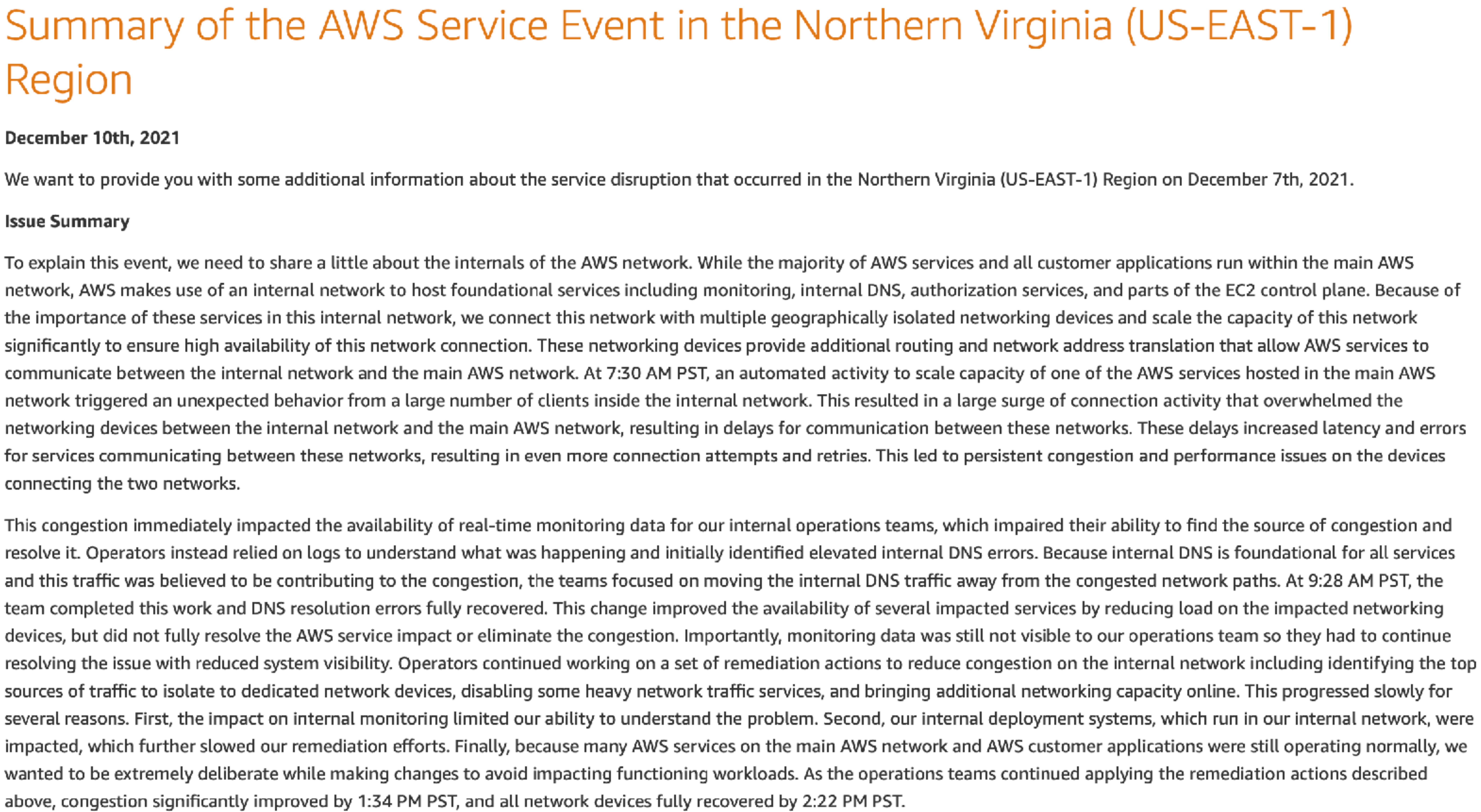
What went wrong

- We're out of practice in responding to cascading failure
- We exceeded our availability error budget (by several orders of magnitude) due to the exceptional surge of traffic that essentially all resulted in failures

Where we got lucky[166]

- Mailing list of Shakespeare aficionados had a copy of new sonnet available
- Server logs had stack traces pointing to file descriptor exhaustion as cause for crash
- Query-of-death was resolved by pushing new index containing popular search term

Blameless Post-Mortems: Real World Example



Summary of the AWS Service Event in the Northern Virginia (US-EAST-1) Region

December 10th, 2021

We want to provide you with some additional information about the service disruption that occurred in the Northern Virginia (US-EAST-1) Region on December 7th, 2021.

Issue Summary

To explain this event, we need to share a little about the internals of the AWS network. While the majority of AWS services and all customer applications run within the main AWS network, AWS makes use of an internal network to host foundational services including monitoring, internal DNS, authorization services, and parts of the EC2 control plane. Because of the importance of these services in this internal network, we connect this network with multiple geographically isolated networking devices and scale the capacity of this network significantly to ensure high availability of this network connection. These networking devices provide additional routing and network address translation that allow AWS services to communicate between the internal network and the main AWS network. At 7:30 AM PST, an automated activity to scale capacity of one of the AWS services hosted in the main AWS network triggered an unexpected behavior from a large number of clients inside the internal network. This resulted in a large surge of connection activity that overwhelmed the networking devices between the internal network and the main AWS network, resulting in delays for communication between these networks. These delays increased latency and errors for services communicating between these networks, resulting in even more connection attempts and retries. This led to persistent congestion and performance issues on the devices connecting the two networks.

This congestion immediately impacted the availability of real-time monitoring data for our internal operations teams, which impaired their ability to find the source of congestion and resolve it. Operators instead relied on logs to understand what was happening and initially identified elevated internal DNS errors. Because internal DNS is foundational for all services and this traffic was believed to be contributing to the congestion, the teams focused on moving the internal DNS traffic away from the congested network paths. At 9:28 AM PST, the team completed this work and DNS resolution errors fully recovered. This change improved the availability of several impacted services by reducing load on the impacted networking devices, but did not fully resolve the AWS service impact or eliminate the congestion. Importantly, monitoring data was still not visible to our operations team so they had to continue resolving the issue with reduced system visibility. Operators continued working on a set of remediation actions to reduce congestion on the internal network including identifying the top sources of traffic to isolate to dedicated network devices, disabling some heavy network traffic services, and bringing additional networking capacity online. This progressed slowly for several reasons. First, the impact on internal monitoring limited our ability to understand the problem. Second, our internal deployment systems, which run in our internal network, were impacted, which further slowed our remediation efforts. Finally, because many AWS services on the main AWS network and AWS customer applications were still operating normally, we wanted to be extremely deliberate while making changes to avoid impacting functioning workloads. As the operations teams continued applying the remediation actions described above, congestion significantly improved by 1:34 PM PST, and all network devices fully recovered by 2:22 PM PST.

Conducting Postmortems

- Apply this technique after any event you would like to avoid in the future
- Apply this to technical and non-technical events
- Focus on improvement, resilience, and collaboration: what could any of the actors have done better?
- Google’s generic postmortem template