CS 4530: Fundamentals of Software Engineering
## Lesson 7.1: Teams

Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

Learning Goals for this Lesson

- At the end of this lesson, you should be able to
  - Explain key advantages of working in a team and sharing information with your team
  - Describe the HRT pillars of social interaction
  - Understand why small teams are effective for agile processes
  - Apply root-cause analysis to construct a blameless post-mortem of a team project
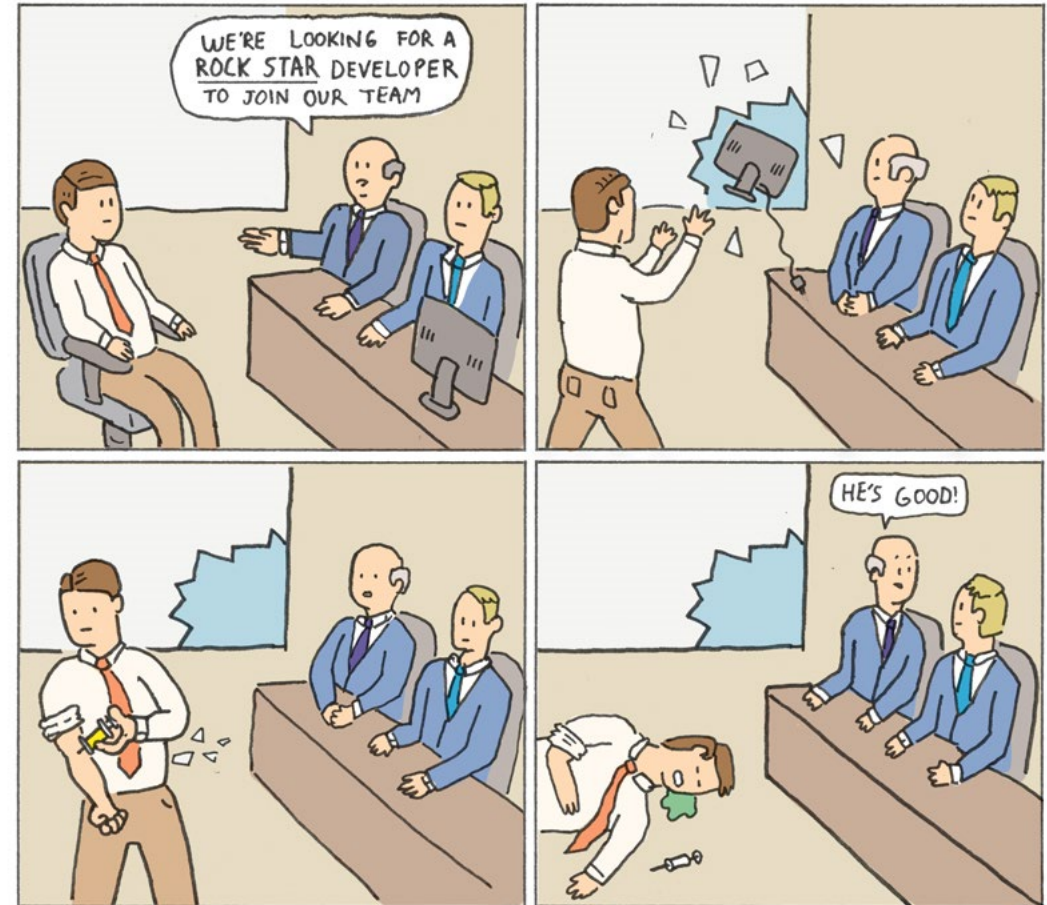
Why Teams? "The 10x Engineer"
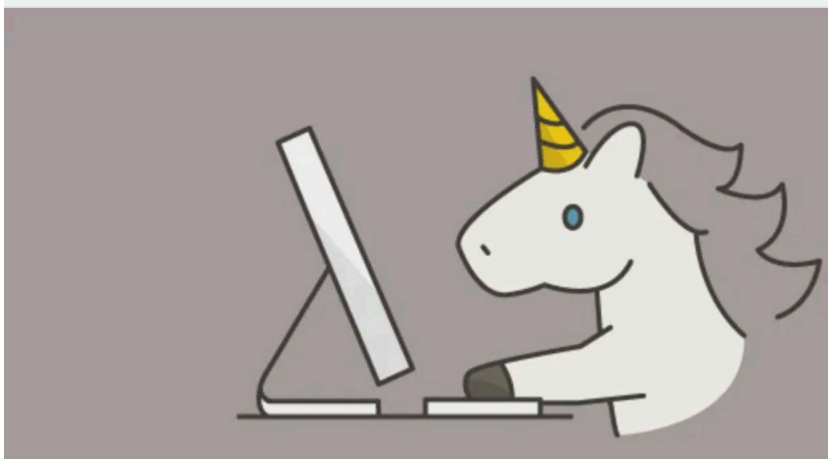
Why Teams? Software Engineering Draws on Many Skills

- Nobody is an expert in *everything:*
  - Product management
  - Project management
  - System-level design and architecture
  - Unit-level design
  - Development
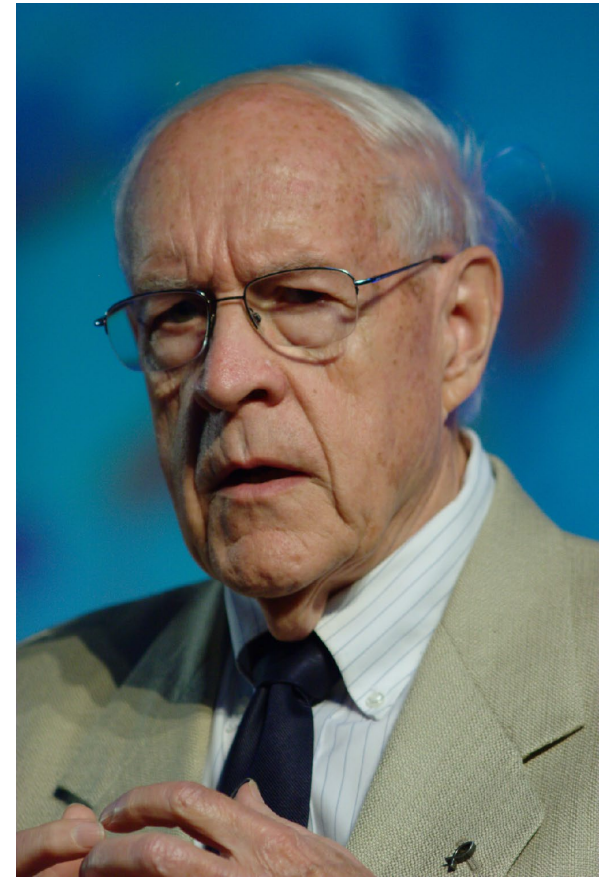 - Operations
  - Maintenance

Why Teams? The Bus Factor

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

Teams are hard: Brooks' Law

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

Fred Brooks, 1975

What goes wrong with teams in software development?

- How do you structure teams effectively?

- How do you encourage teams to share knowledge and collaborate?

- How do you encourage team-members to treat each other well?
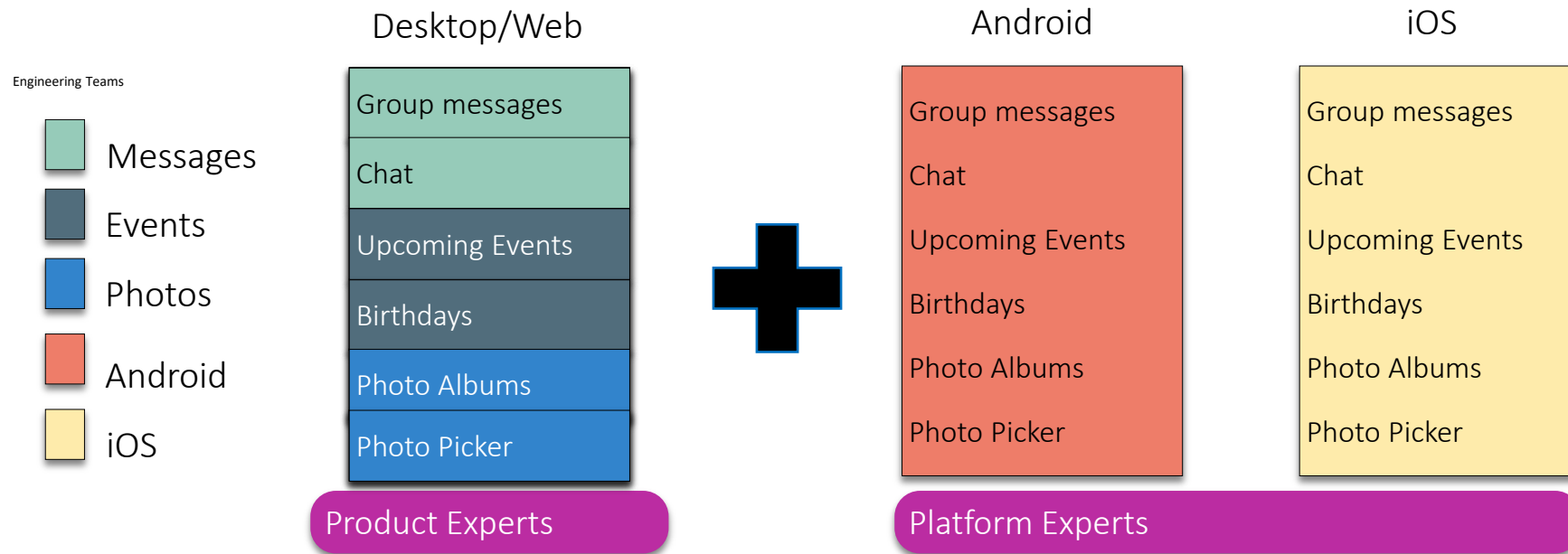
- How do you respond to failures?

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?

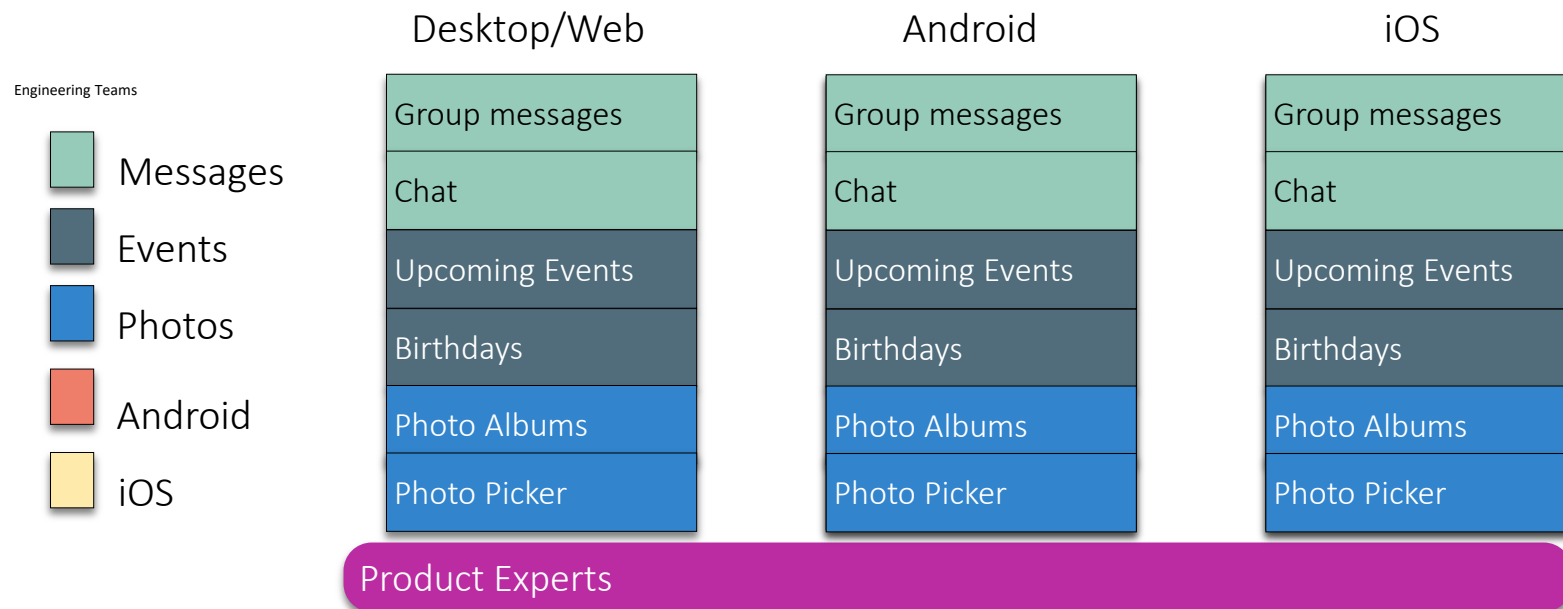- Self-organizing teams have proven more effective

Facebook originally organized teams by platform

- If you work on the android teams, you work on putting all of the apps on android

Engineering Teams

- Messages
- Events
- Photos
- Android
- iOS

Desktop/Web

| Group messages |
| Chat |
| Upcoming Events |
| Birthdays |
| Photo Albums |
| Photo Picker |

Product Experts

+

Android

Group messages

Chat

Upcoming Events

Birthdays

Photo Albums

Photo Picker

iOS

Group messages

Chat

Upcoming Events

Birthdays

Photo Albums

Photo Picker

Platform Experts

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

But they eventually switched to "product" teams...

- If you are in the chat group, you work on the chat feature in all platforms

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

Good Teams create Intentional Opportunities for Knowledge Sharing

- Ideally, scale linearly (or sub linearly) with org growth
- What kind of approaches do you think could help improve knowledge sharing?
  - "Two Pizza" Teams
  - Pair Programming
  - Code Reviews
  - Multiple sharing channels

"Two-Pizza" Teams make knowledge sharing easier

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

Pair Programming is a Knowledge Sharing 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

- 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
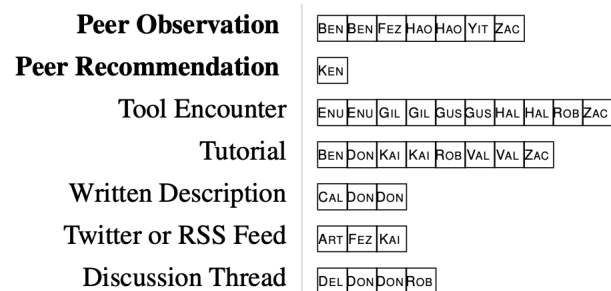
Peer Observation	Ben Ben Fez Hao Hao Yit Zac
Peer Recommendation	Ken
Tool Encounter	Enu Enu Gil Gil Gus Gus Hal Hal Rob Zac
Tutorial	Ben Don Kai Kai Rob Val Val Zac
Written Description	Cal Don Don
Twitter or RSS Feed	Art Fez Kai
Discussion Thread	Del Don Don Rob

Figure 2: Histogram of the most frequent discovery modes.

Peer Observation	Ben Cal Don Del Eli Enu Fez Hal Hao Ken Rob Val
Peer Recommendation	Ben Cal Fez Gil Gus Ken Hal Val Yit
Tool Encounter	Gil Hal Zac
Tutorial	Art Enu Kai
Written Description	Hao
Twitter or RSS Feed	Art Del Don Rob Zac
Discussion Thread	Eli Hal Yit
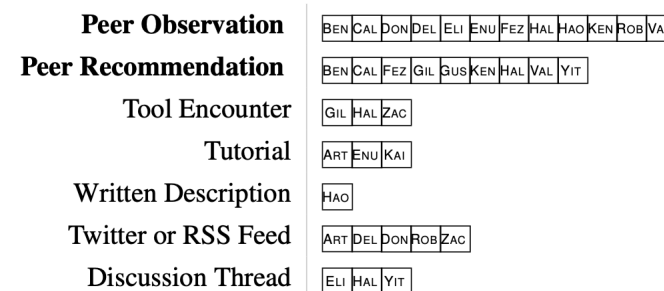
Figure 3: Histogram of the most effective discovery modes.

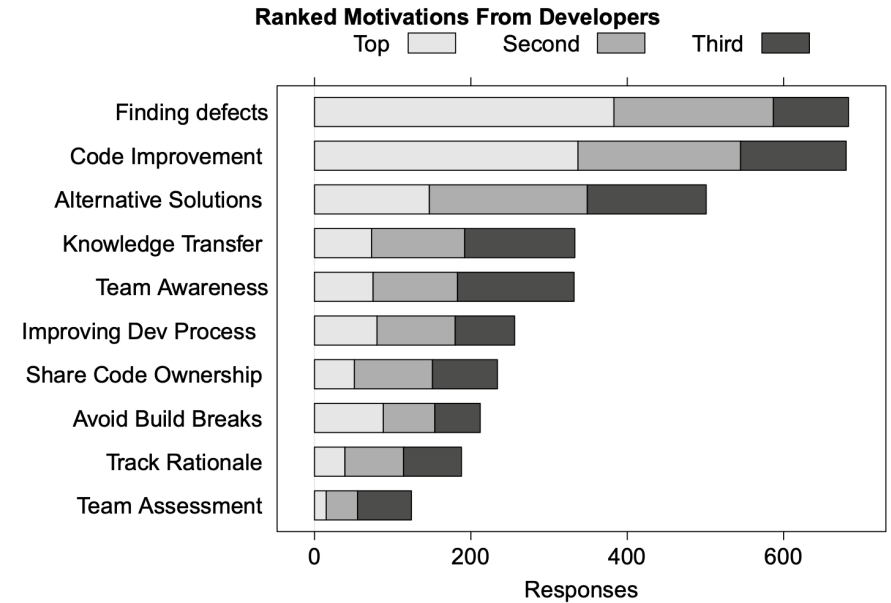"Peer interaction effectively, yet infrequently, enables programmers to discover new tools", Emerson Murphy-Hill & Gail C. Murphy, CSCW 2011

Code Review is a Knowledge Sharing Opportunity
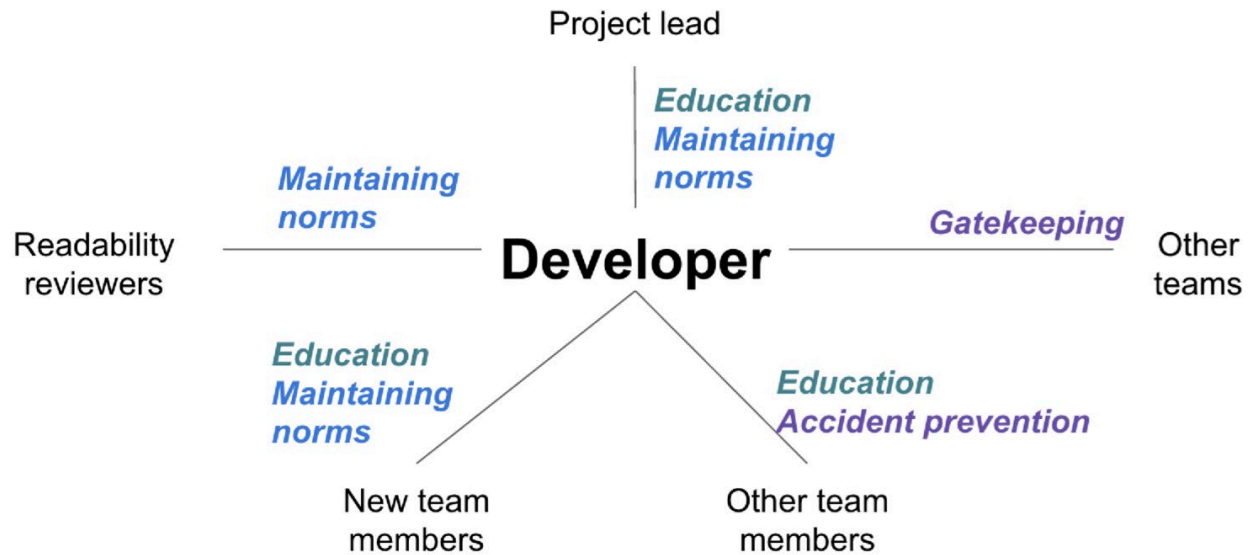
"Modern Code Review: A Case Study at Google", Sadowski et al, ICSE 2018

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

Scaling Communication linearly requires multiple channels

- Knowledge sharing needs to scale linearly (or sub linearly) with org growth:
  - Mentorship
  - Q&A
  - Mailing lists
  - Tech talks
  - Documentation

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)

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

Episode 284
April 30 2013

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 In The Toilet?

- Researchers studied the efficacy of the flyer
- 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

"Do Developers Disocver New Tools on The Toilet?", Emerson Murphy-Hill et al, ICSE 2019

Communicate Development Activities with Different Channels

- On average, developers use *eleven* channels to stay up-to-date on development activities
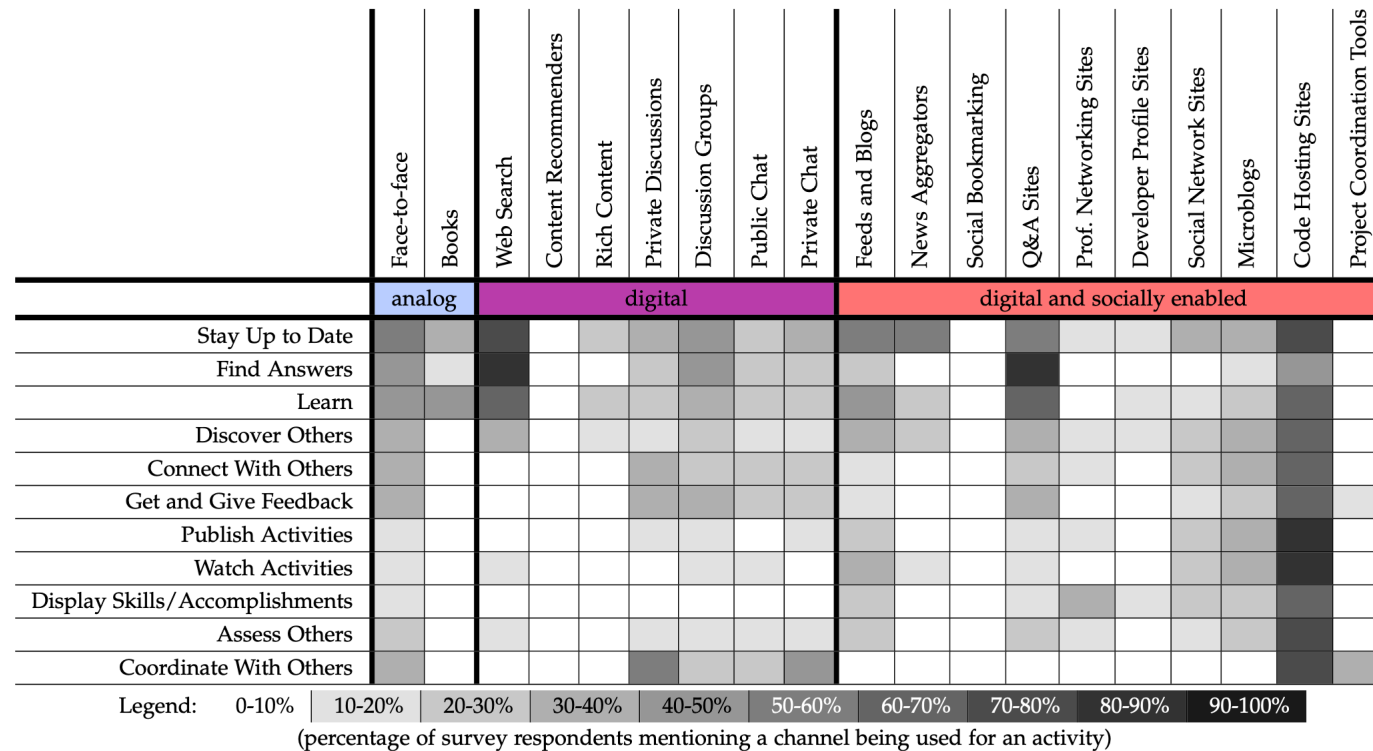
TABLE 4
Channels used by our respondents and the activities they support.

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

How do you encourage team members to treat each other well?

Three Pillars of Social Skills

- 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.

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

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

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

Learning Goals for this Lesson

- At the end of this lesson, you should be able to
  - Explain key advantages of working in a team and sharing information with your team
  - Describe the HRT pillars of social interaction
  - Understand why small teams are effective for agile processes
  - Apply root-cause analysis to construct a blameless post-mortem of a team project