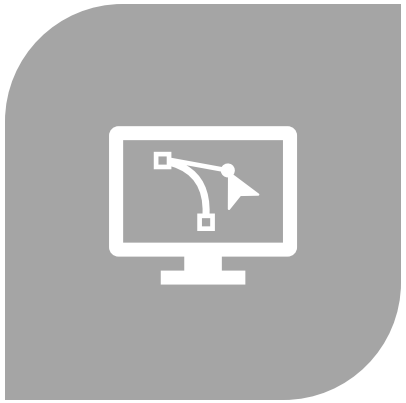
CS 4530: Fundamentals of Software Engineering
Lesson 4.3: Teams

Adeel Bhutta and Mitch Wand

Khoury College of Computer Sciences

software engineering must encompass 3 Ps:



PROGRAMS, PROCESSES, AND PEOPLE

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”

What makes a 10x Developer?

#10xdeveloper #productivity #beginners #career

Davide de Paolis Mar 11, 2019 · 6 min read

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
  - Testing
  - Operations
  - Maintenance

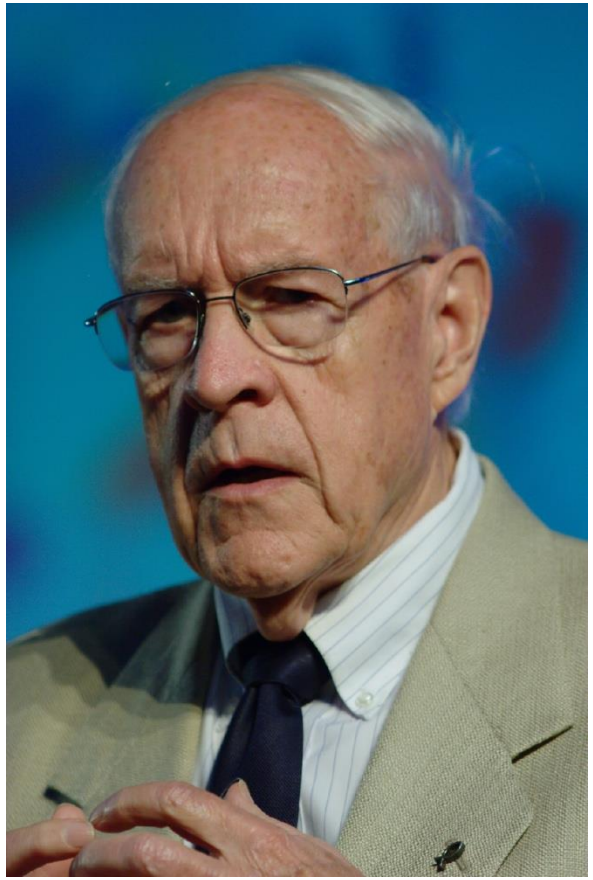
Why Teams? The Bus Factor

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?

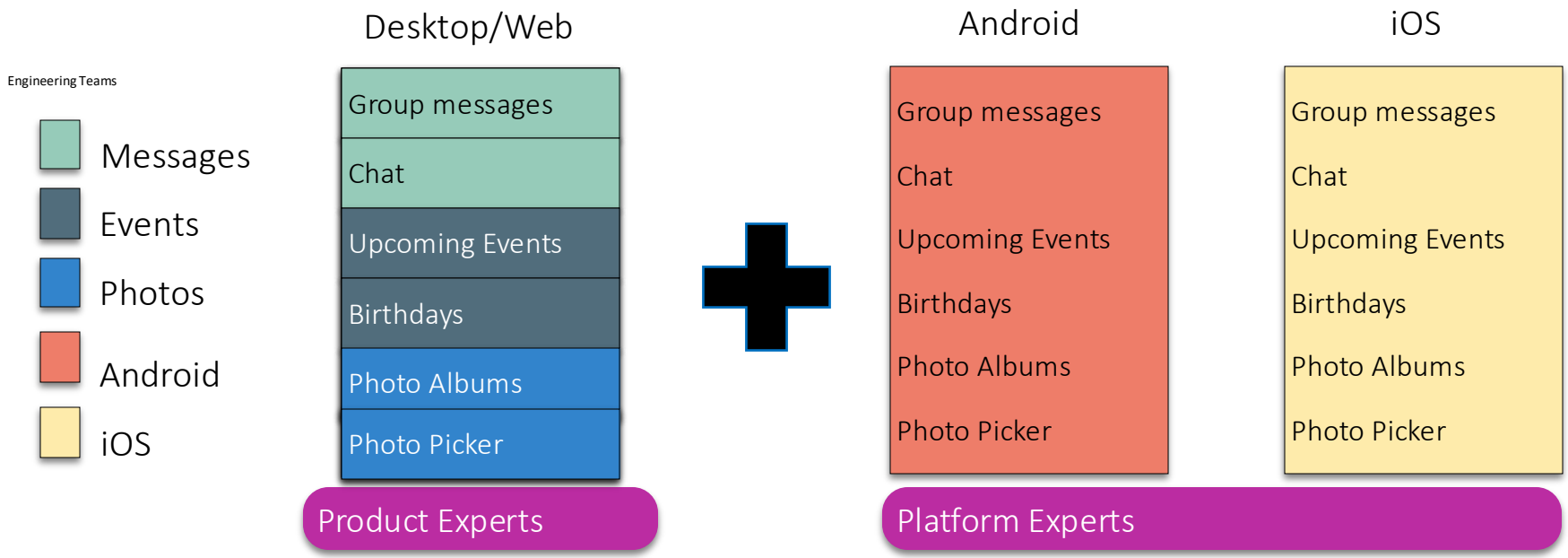
- How do you structure teams effectively?
- How do you encourage teams to share knowledge and collaborate?
- How do you encourage team-members to treat each other well?
- How do you respond to failures?

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 each managed the app on their own platform only

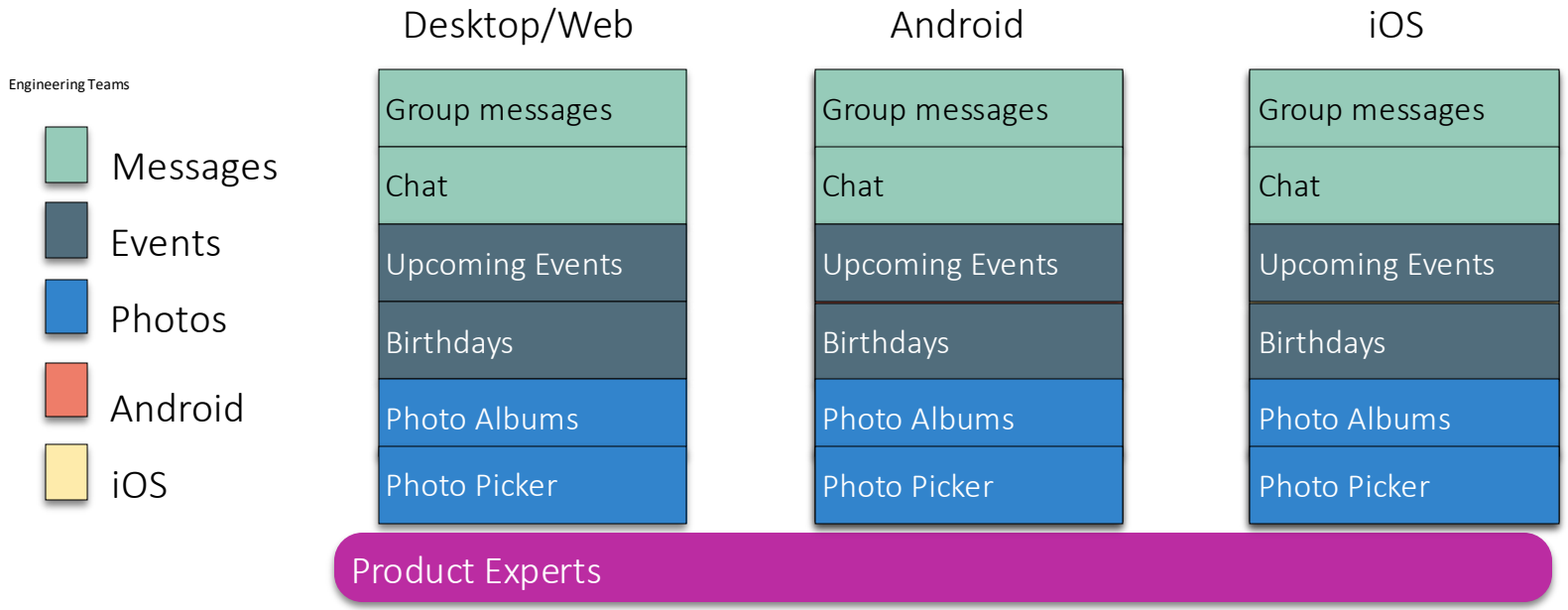
- If you work on the android teams, you work on putting all of the apps on android



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



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

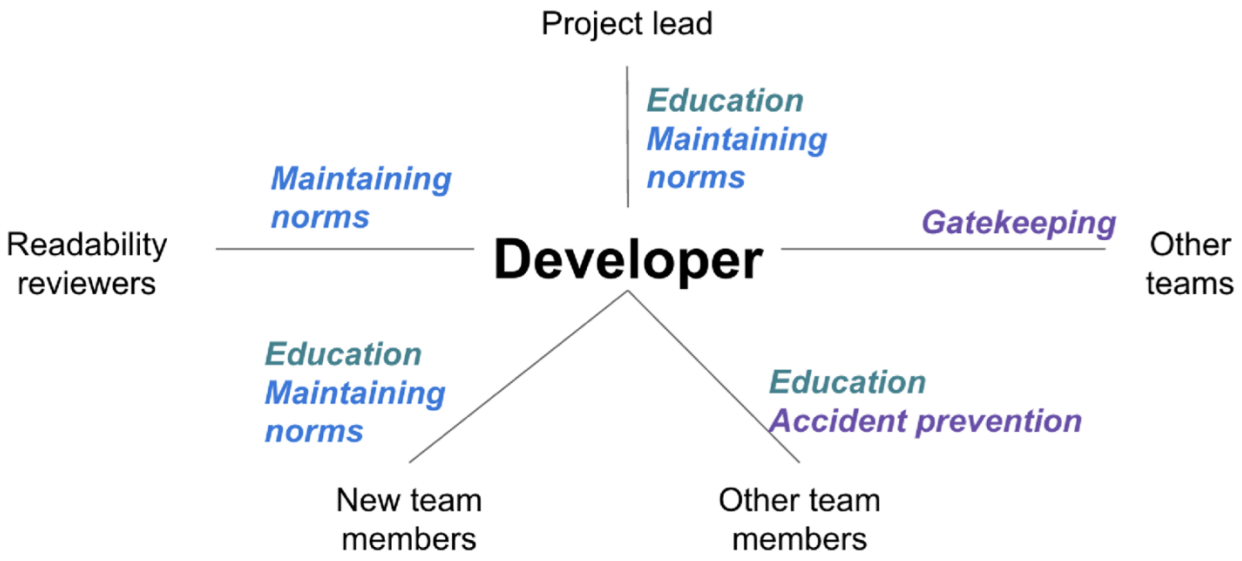
Q: How many people on a team?

A: “No more than you could feed with two pizzas”

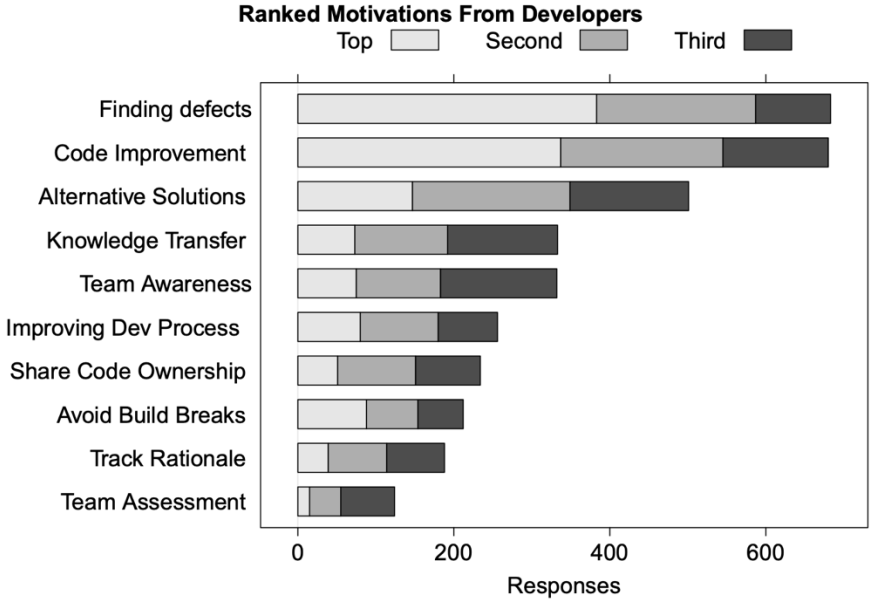
Rationale:

- Decrease communication burdens
- Focus conversations to relevant topics

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

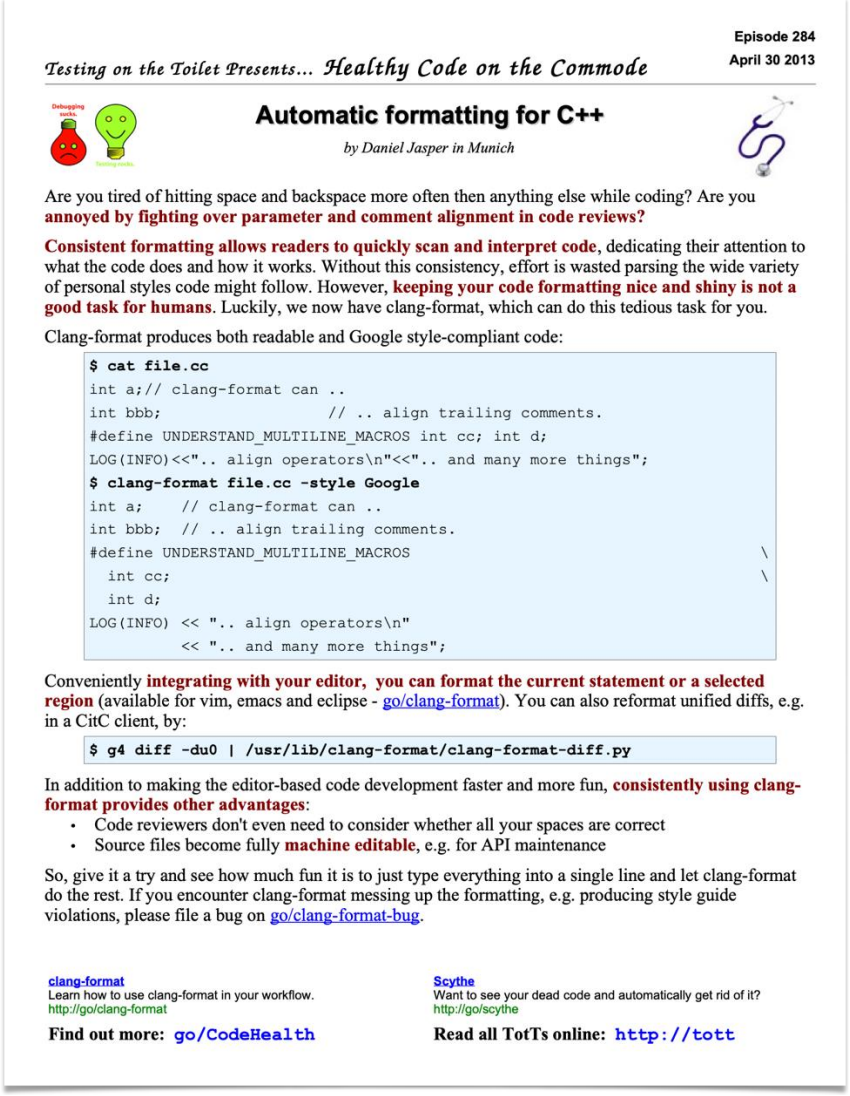
Scaling Communication linearly requires multiple channels

- 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)
  - significantly increased and sustained adoption of the tools advertised on flyers in toilets



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

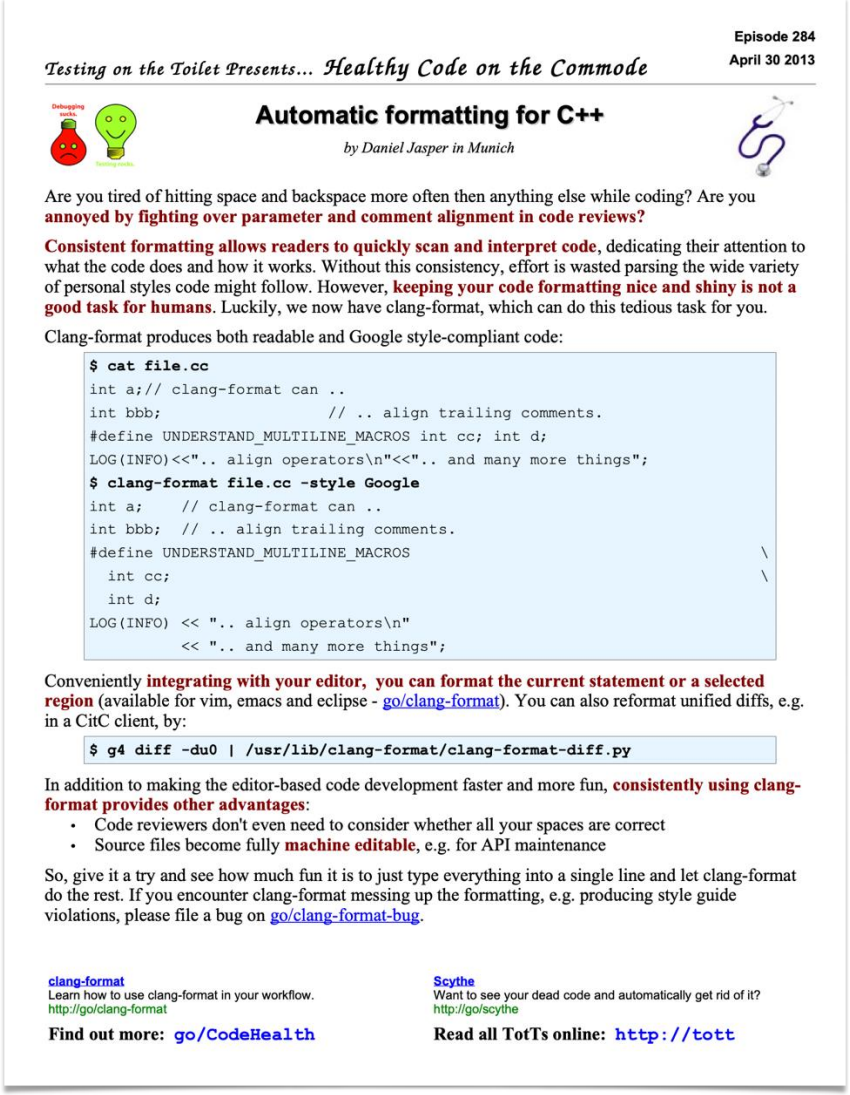
Scythe
Want to see your dead code and automatically get rid of it?
http://go/scythe

Find out more: go/CodeHealth

Read all TotTs online: http://tott

Do Developers Discover New Tools 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



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

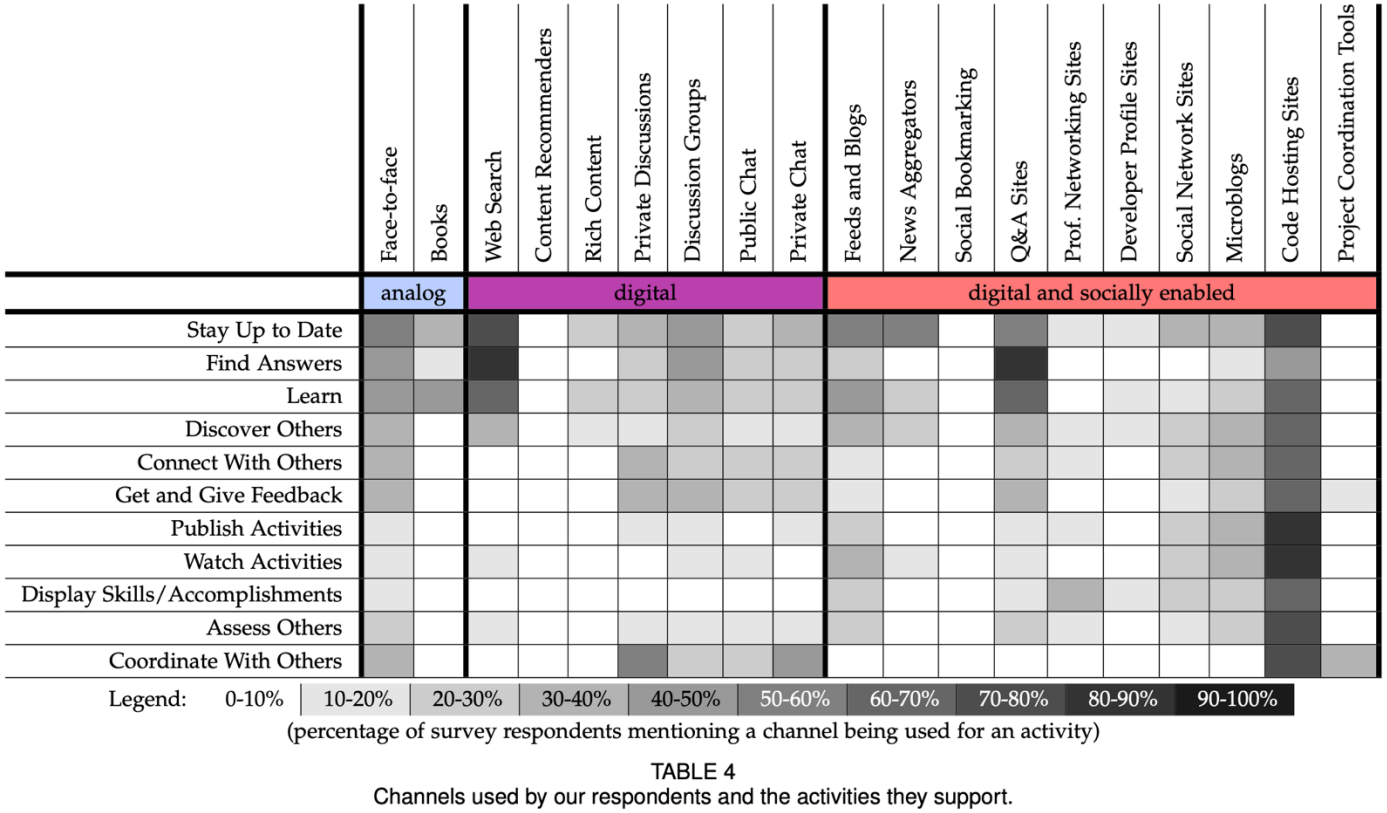
Find out more: go/CodeHealth

Read all TotTs online: http://tott

“Do Developers 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



	Face-to-face	Books	Web Search	Content Recommenders	Rich Content	Private Discussions	Discussion Groups	Public Chat	Private Chat	Feeds and Blogs	News Aggregators	Social Bookmarking	Q&A Sites	Prof. Networking Sites	Developer Profile Sites	Social Network Sites	Microblogs	Code Hosting Sites	Project Coordination Tools
	analog		digital							digital and socially enabled									
Stay Up to Date																			
Find Answers																			
Learn																			
Discover Others																			
Connect With Others																			
Get and Give Feedback																			
Publish Activities																			
Watch Activities																			
Display Skills/Accomplishments																			
Assess Others																			
Coordinate With Others																			

Legend: 0-10% | 10-20% | 20-30% | 30-40% | 40-50% | 50-60% | 60-70% | 70-80% | 80-90% | 90-100%

(percentage of survey respondents mentioning a channel being used for an activity)

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.

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

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

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.

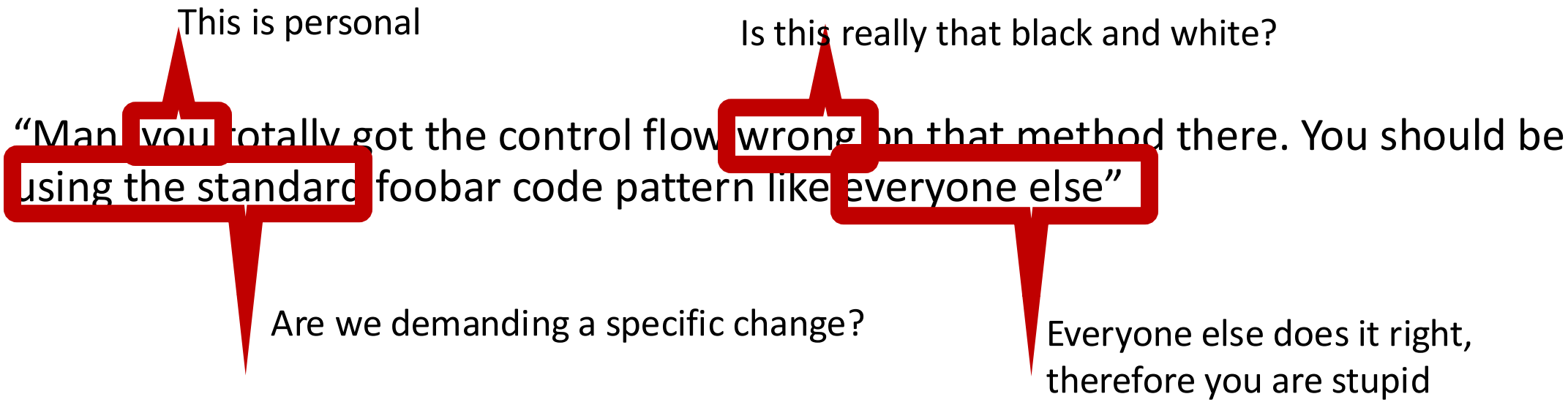
(7) Spread disturbing rumors that sound like inside dope.

(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.

HRT Example: Code Review



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

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