CS 4530: Fundamentals of Software Engineering
Lesson 6.3: Teams

Adeel Bhutta and Mitch Wand

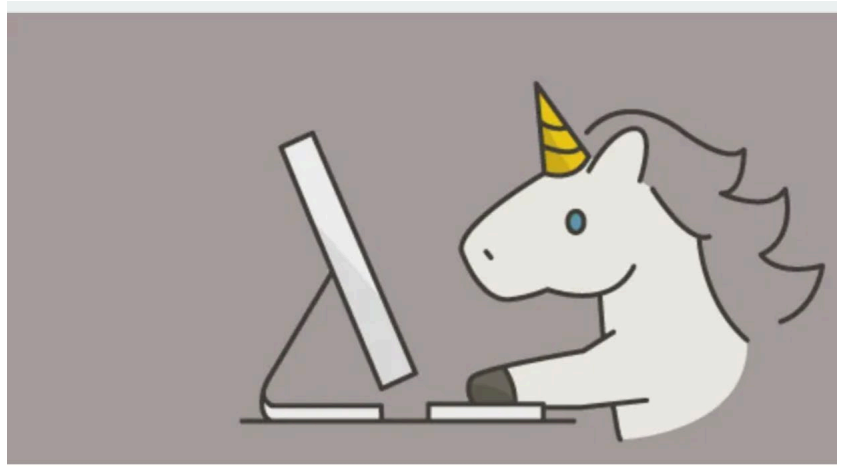
Khoury College of Computer Sciences

© 2023 Released under the CC BY-SA license

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 agile processes favor small teams
  - Apply root-cause analysis to construct a blameless post-mortem of a team project

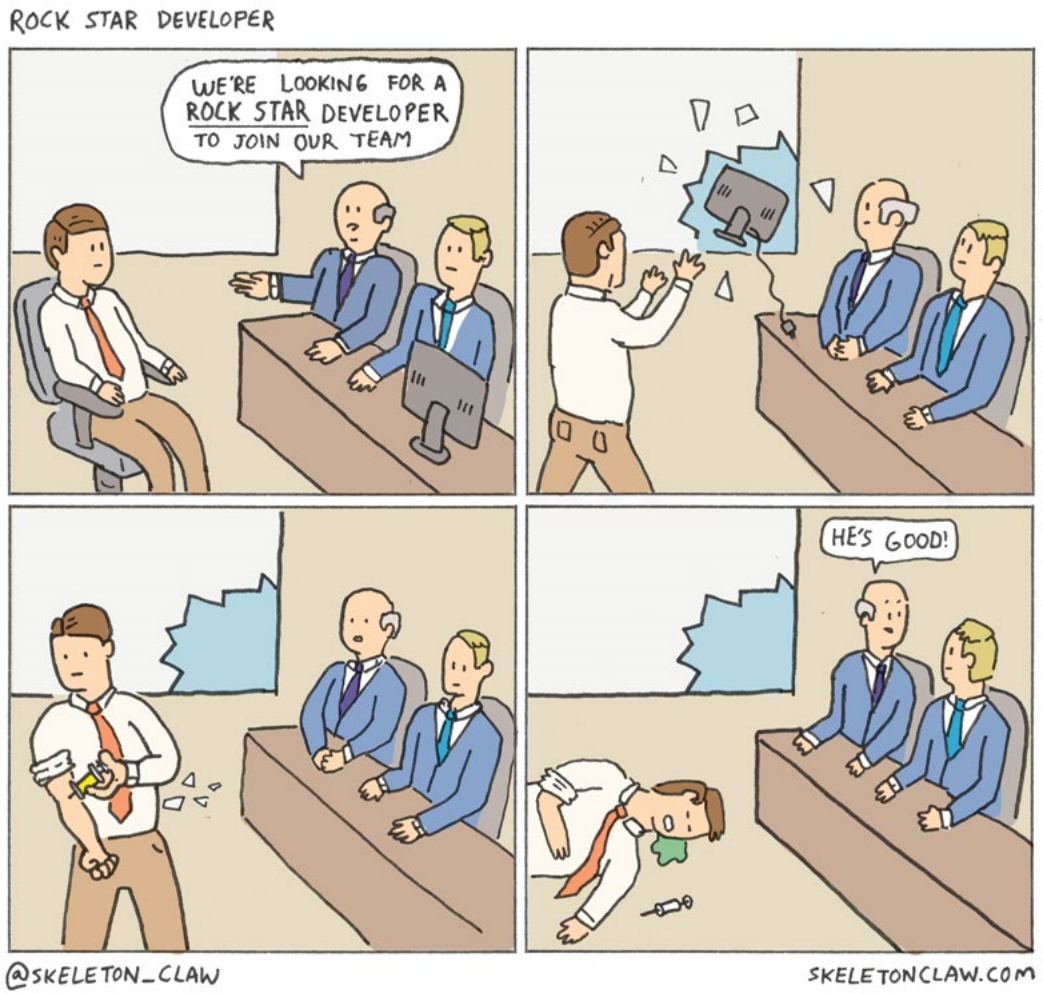
Why Teams? “The 10x Engineer”



What makes a 10x Developer?

#10xdeveloper #productivity #beginners #career

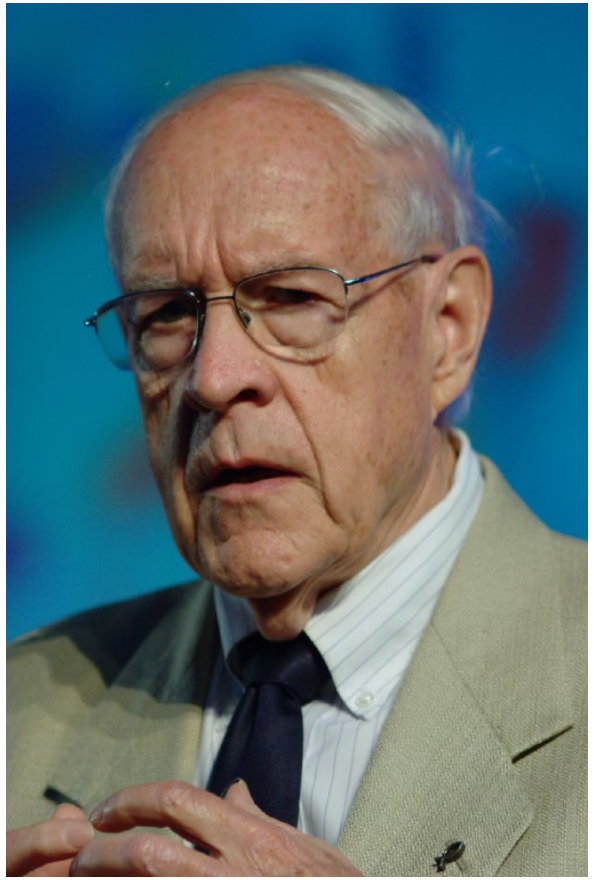
Davide de Paolis Mar 11, 2019 · 6 min read



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

Agile Favors “Two-Pizza” Teams

Q: How many people on a team?

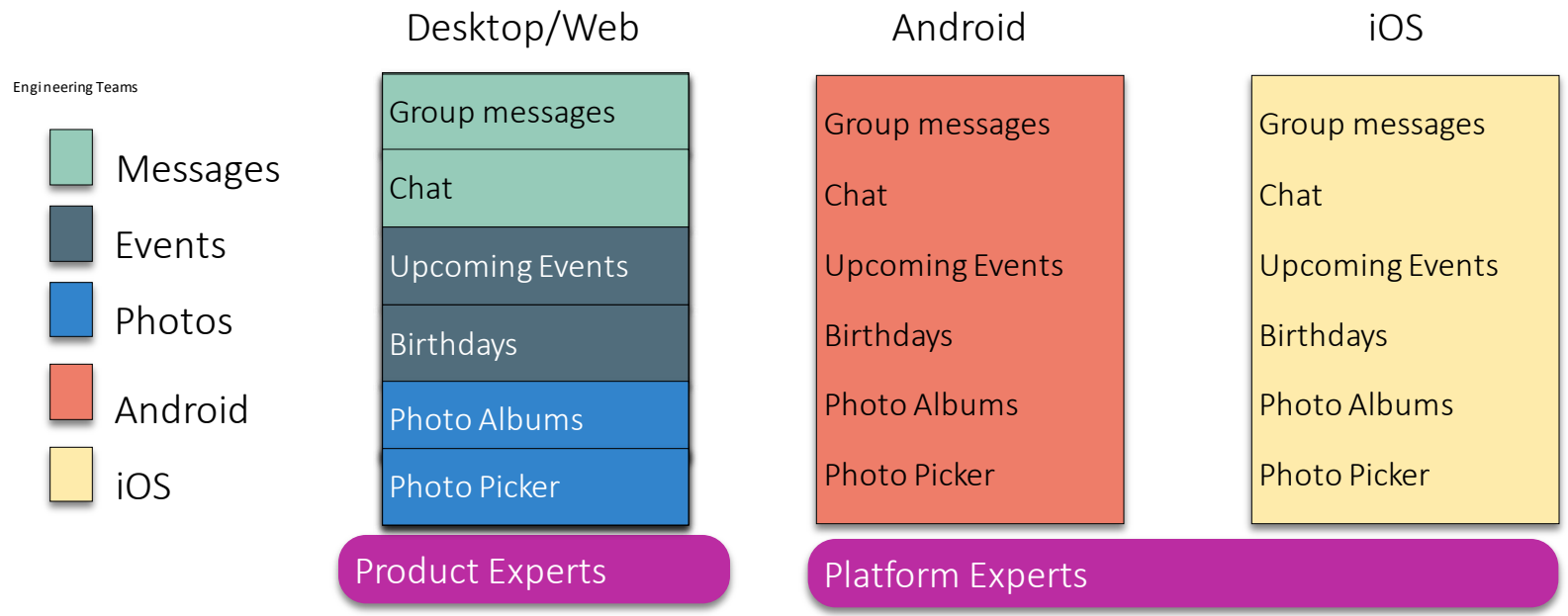
A: “No more than you could feed with two pizzas”

Rationale:

- Decrease communication burdens
- Focus conversations to relevant topics

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

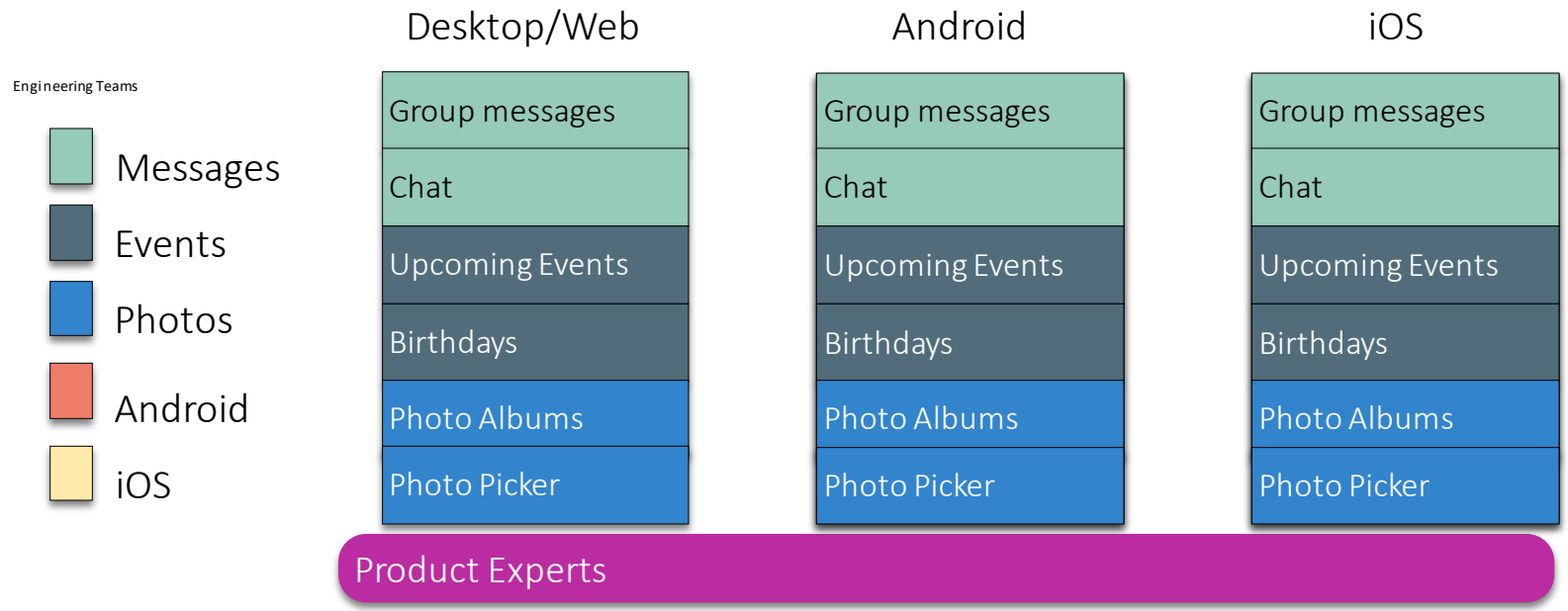
Example: Facebook mobile teams (with platform organization)



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

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

Example: Facebook mobile teams (with platform organization)



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

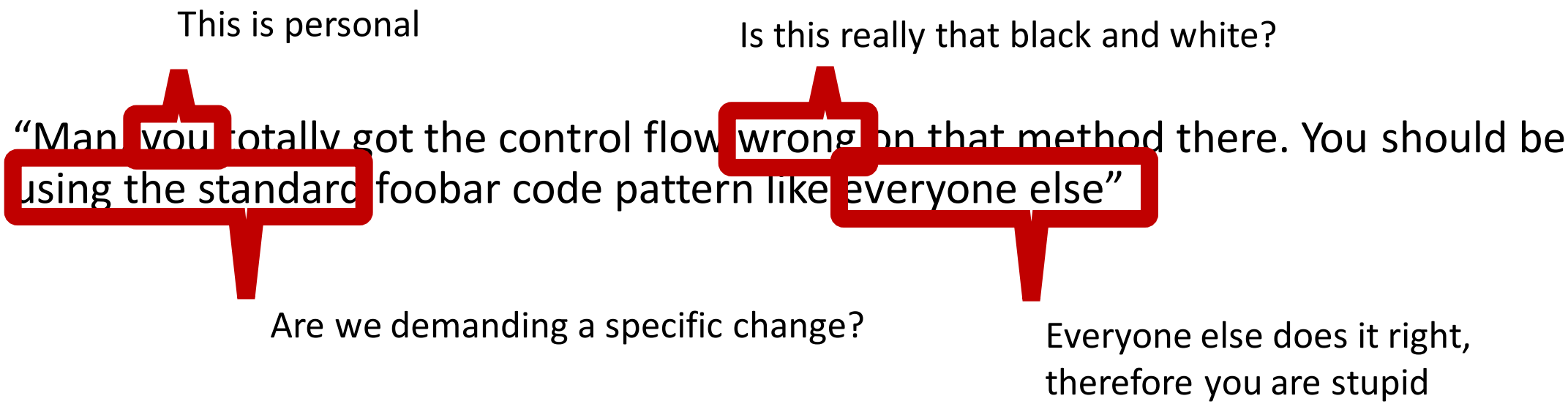
How do you encourage team members to treat each other well?

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

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

Scaling Communication and Knowledge Sharing

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

Bus Factor & Importance of Information Sharing

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

Invite a successor

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 agile processes favor small teams
  - Apply root-cause analysis to construct a blameless post-mortem of a team project