Software Teams and Communication

17-313 Fall 2026

Foundations of Software Engineering

https://cmu-17313q.github.io

Eduardo Feo Flushing

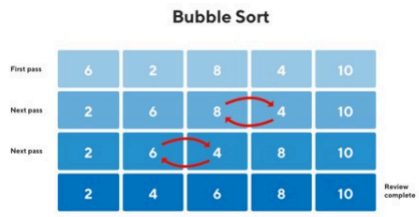
Administrivia

- Meet with your teams!
- P2A deadline has been extended until Tuesday, September 15
- Extra credit: Socialize with your teams outside work
  - Share a photo/screenshot of your team activity with your CA mentors before Tuesday night.
- Problem set is out
- Quiz#1: Sunday, September 20

Learning Goals

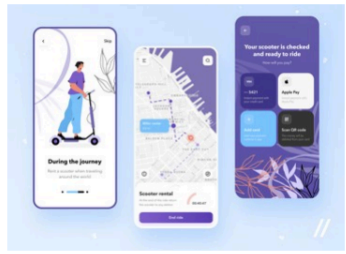
- Describe the pros and cons of working as a team
- Recognize the importance of communication in collaboration
- Recognize the need of having multiple communication channels
- Select an appropriate communication tool for a given communication goal
- Ask technical questions effectively
- Write clear and specific Github issues, pull requests, and comments

We all work in a team



Bubble Sort

Monopoly Game



Scooter App

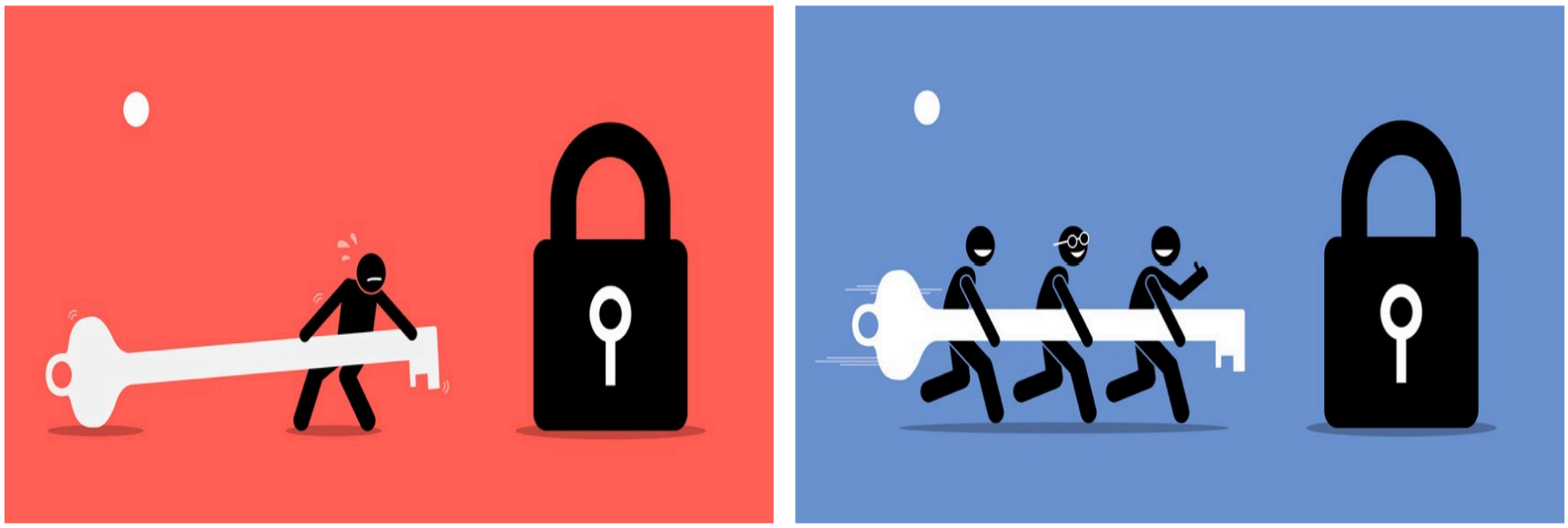
opencode

Autonomous Vehicle

NOTES

None of us built any of these systems alone — Bubble Sort, Monopoly, Teedy, a scooter app, Gmail, self-driving cars. We can't work alone, and that's exactly what we'll practice all semester.

Working solo vs. as a team



NOTES

Solo work pros: Full knowledge of the project; freedom to work at your own pace. Solo work cons: Limited perspectives and expertise; no one to share the workload or review your work. Teamwork pros: Shared workload; diverse skills and perspectives; peer support and review. Teamwork cons: Collaboration overhead; potential conflicts.

Teamwork in nature

NOTES

The Burj Khalifa is 479x taller than the humans who built it, impressive. But termites build mounds 2013x taller than themselves, with built-in climate control, and can coordinate across 200 million interconnected mounds. Nature's teamwork puts ours to shame. Nature favors teamwork.

Teamwork evolution

NOTES

Cooperation is a product of evolution, seen at every level from genomes to cells to human societies. It's what let unrelated individuals build something bigger together, and it plays a direct role in individual fitness/survival.

Working as a team

- Establish a collaboration process
- Meet with the team
- Divide work and integrate
- Share knowledge
- Resolve conflicts

NOTES

Notice that every one of these five steps comes down to communication. Cooperation is the organizing principle behind everything from hunter-gatherer bands to nation-states; it's how humans build new levels of organization.

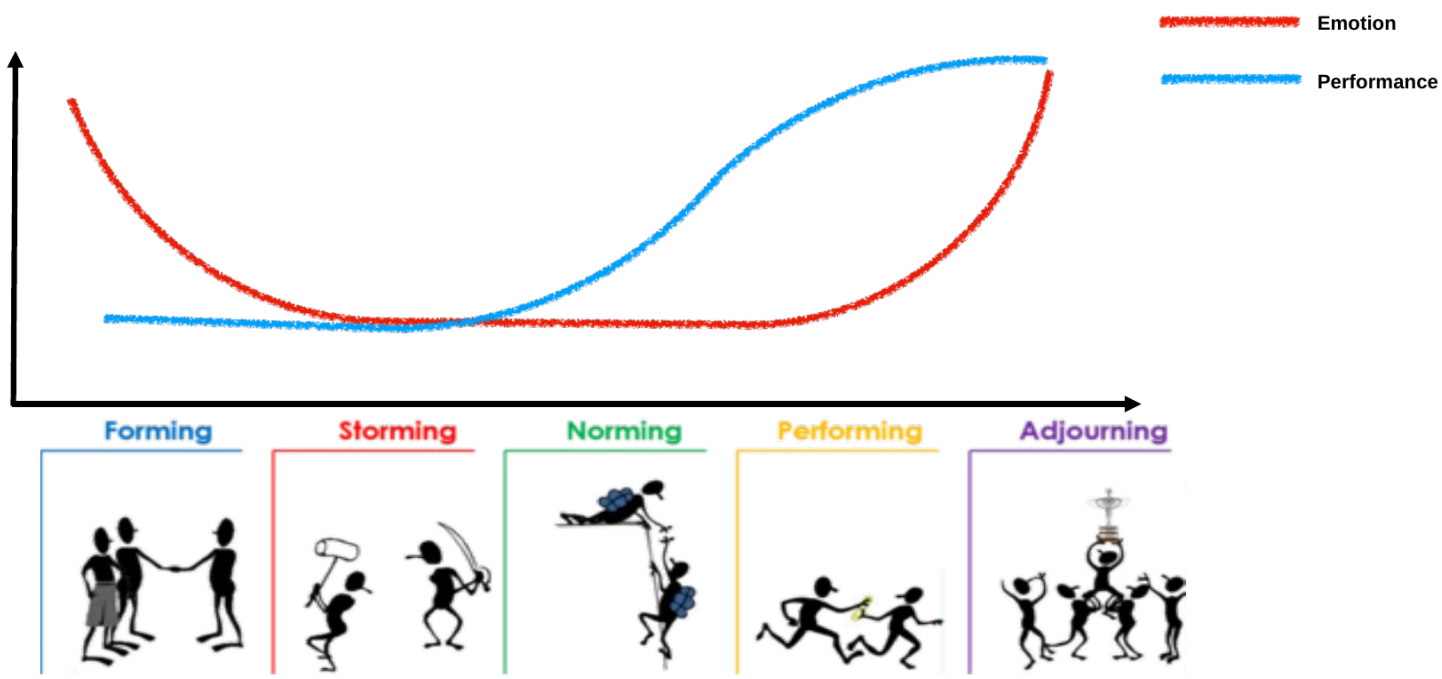
Working as a team

- **Establish a collaboration process**
- Meet with the team
- Divide work and integrate
- Share knowledge
- Resolve conflicts

NOTES

Everything on this list really comes down to communication.

Stages of Team Formation



NOTES

Keep this simple model in the back of your mind as your team evolves. - Forming: you meet, learn the challenge, agree on goals. - Storming: people voice opinions and conflict can surface as roles and status get sorted out. - Norming: the team tolerates differences and starts pulling together — the risk here is avoiding conflict so much that good ideas go unsaid. - Performing: with norms set, the team hits its stride and often exceeds expectations.

Norming

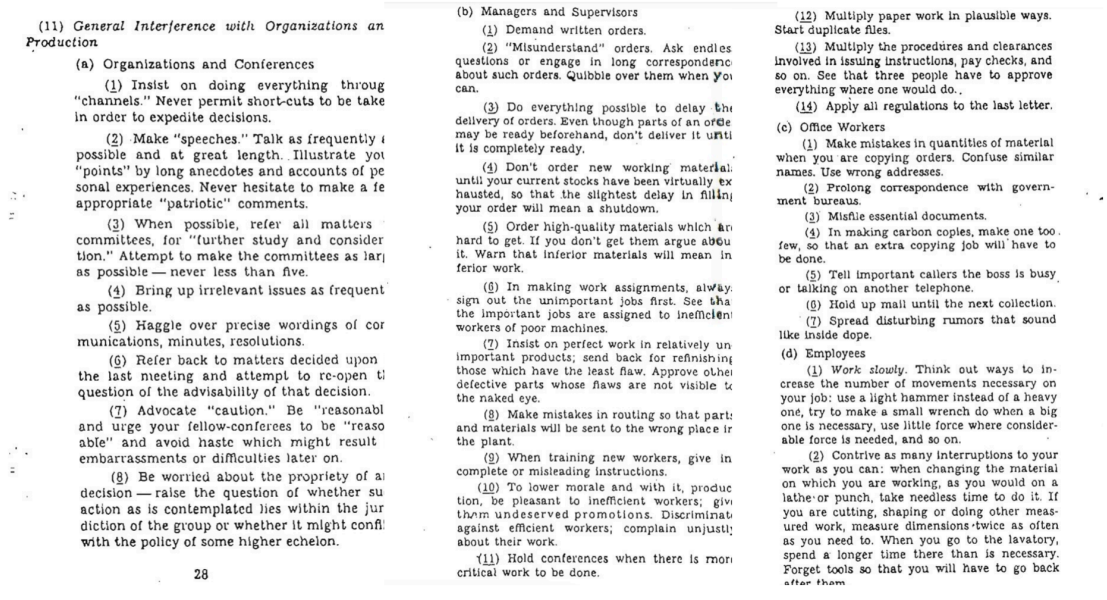
- When are you generally available to work outside class?
- How will you let the team know when you are busy or unavailable?
- How quickly should team members respond to messages?
- What should someone do when they are blocked or falling behind?
- What practices from your previous team experiences would you like to adopt or avoid?

Establishing norms is an important part of team formation.

NOTES

Norms don't have to be explicitly written down — sometimes they're discovered, sometimes formalized, like an employment contract or your own team contract.

A humorous aside



(11) *General Interference with Organizations an Production*

(a) Organizations and Conferences

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

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

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

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

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

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

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

(8) Be worried about the propriety of a decision — raise the question of whether su action as is contemplated lies within the jur diction of the group or whether it might confl with the policy of some higher echelon.

28

(b) Managers and Supervisors

(1) Demand written orders.

(2) "Misunderstand" orders. Ask endles questions or engage in long correspondenc about such orders. Quibble over them when yo can.

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

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

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

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

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

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

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

(10) To lower morale and with it, produc tion, be pleasant to inefficient workers; giv them undeserved promotions. Discriminat against efficient workers; complain unjustly about their work.

(11) Hold conferences when there is mor 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.

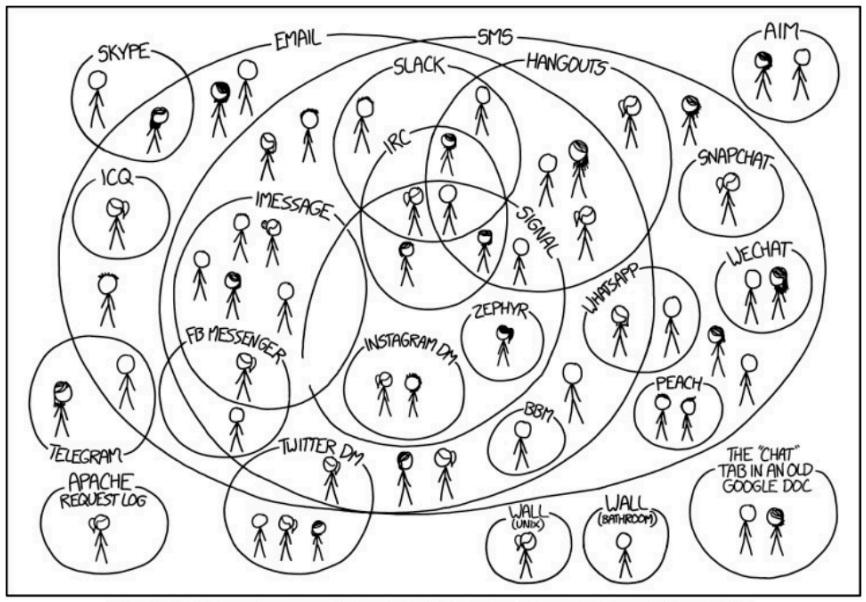
(2) Contrive as many interruptions to your work as you can: when changing the material on which you are working, as you would on a lathe or punch, take needless time to do it. If you are cutting, shaping or doing other measured work, measure dimensions twice as often as you need to. When you go to the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

---

NOTES

A scanned excerpt from the CIA's WWII "Simple Sabotage Field Manual" (office sabotage tactics).

Team communication

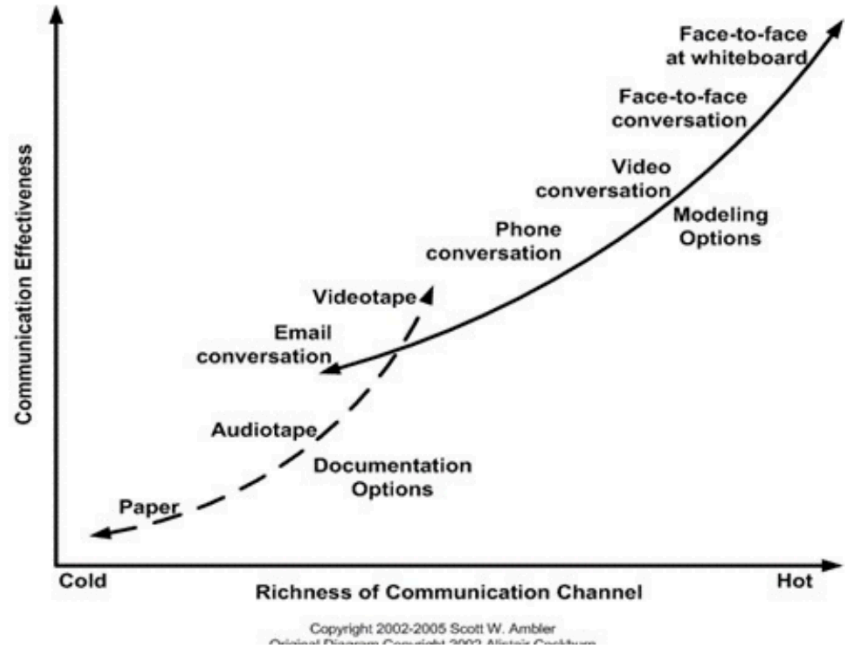


I HAVE A HARD TIME KEEPING TRACK OF WHICH CONTACTS USE WHICH CHAT SYSTEMS.

NOTES

Most of us lose count of how many communication tools we use. Sometimes, less is more. Be selective about which tools your team uses to avoid confusion and ensure everyone knows where to find important information.

Select the right comm. tools



NOTES

You usually don't get to pick your communication tools from scratch — we already did that for 17-313. But the principle matters: different channels rank differently on effectiveness vs. cost, and you need more than one. If the server goes down at 11pm, you don't file a GitHub issue and wait — but you also don't wake someone up for a minor bug.

Ambler, S. (2002). Agile modeling: effective practices for extreme programming and the unified process. John Wiley & Sons.

Establish communication patterns

- Trello, Microsoft Projects, …
- Github Wiki, Google Docs, Notion, …
- Github Issues, Jira, …
- Email, Slack, Facebook groups, …
- Zoom, Microsoft Teams, Skype, Phone call, …
- Face-to-face meetings

NOTES

Agree on which tool to use for each purpose and avoid sharing the same type of information across multiple places.

17-313 Communication channels

- Slack
- Regular meeting (Lectures, Recitations)
- Office Hours
- Gradescope
- Webpage

NOTES

For this course, we've already set up how we communicate: Canvas/Gradescope for assigning tasks, the webpage for stable reference info, Slack for everything live. Your team needs to have this same conversation and set up your own process.

Check out other projects

Communication

- Forums: Discuss implementations, research, etc. https://discuss.pytorch.org
- GitHub Issues: Bug reports, feature requests, install issues, RFCs, thoughts, etc.
- Slack: The PyTorch Slack hosts a primary audience of moderate to experienced PyTorch users and developers for general chat, online discussions, collaboration, etc. If you are a beginner looking for help, the primary medium is PyTorch Forums. If you need a slack invite, please fill this form: https://goo.gl/forms/PP1AGvNHpSaJP8to1
- Newsletter: No-noise, a one-way email newsletter with important announcements about PyTorch. You can sign-up here: https://eepurl.com/cbG0rv
- Facebook Page: Important announcements about PyTorch. https://www.facebook.com/pytorch
- For brand guidelines, please visit our website at pytorch.org

NOTES

Do not reinvent the wheel. Look at successful projects to see which communication tools they use and how they use them. Adapt practices that fit your team and project.

Communication expectation

- Quality of service guarantee
  - How soon will you get back to your teammates?
  - Weekend? Evening?
- Emergency
  - Tag w/ 911
  - Notify everyone with @channel

Working as a team

- Establish a collaboration process
- **Meet with the team**
- Divide work and integrate
- Share knowledge
- Resolve conflicts

NOTES

Everything on this list really comes down to communication.

Running a (good) meeting

How to run a meeting

The Three Rules of Running a Meeting

1. Set the agenda
2. Start on time. end on time.
3. End with action items (and share them)
    - Github Issues, Meeting Notes, …

NOTES

Running effective meetings is a superpower. A good meeting has a clear purpose, stays focused, and ends with concrete action items. Each action item should have an owner and a deadline, setting clear expectations for what happens next. Start every meeting by reviewing the previous action items and checking progress. End by summarizing decisions, new action items, owners, and deadlines.

How to run a meeting

- Set and document clear responsibilities and expectations
- Make everyone contribute
  - Possible Roles: Coordinator, Scribe, Checker
  - Manage Personalities
  - Be Vulnerable

NOTES

Being vulnerable means admitting you don't have all the answers — manage your own personality here. Sharing your mistakes openly builds a culture of continuous learning for the whole team.

Random Advice

- Note takers have a lot of power to steer the meeting
  - Collaborative notes are even better!
- Different meeting types have different best practices
  - Decision-making meeting
  - Brainstorming meeting
  - One-on-one meeting
  - Working sessions

NOTES

Brainstorming meetings should stay creative and exploratory — don't shut ideas down early. One-on-ones are exactly why we have office hours. And a practical tip: whoever takes notes has real influence over how the meeting is remembered. So it is good to rotate roles.

Avoid unnecessary meetings

NOTES

Use the flowchart mentally before scheduling anything: could this be an email instead?

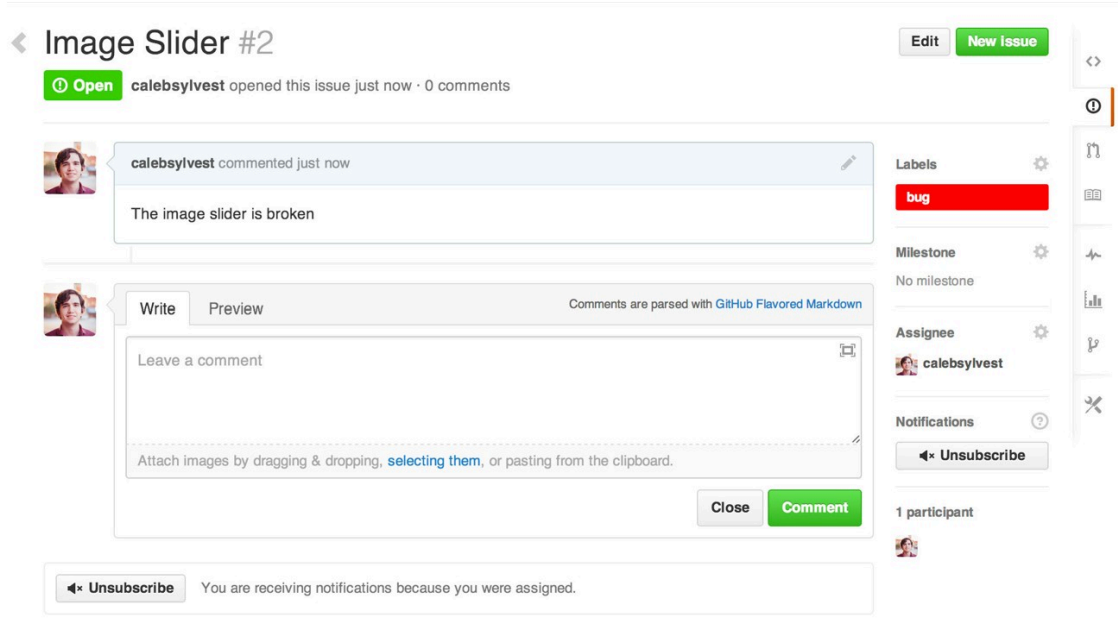
Working as a team

- Establish a collaboration process
- Meet with the team
- **Divide work and integrate**
- Share knowledge
- Resolve conflicts

NOTES

Everything on this list really comes down to communication — and you can't work together on every single task; dividing and reintegrating work is unavoidable.

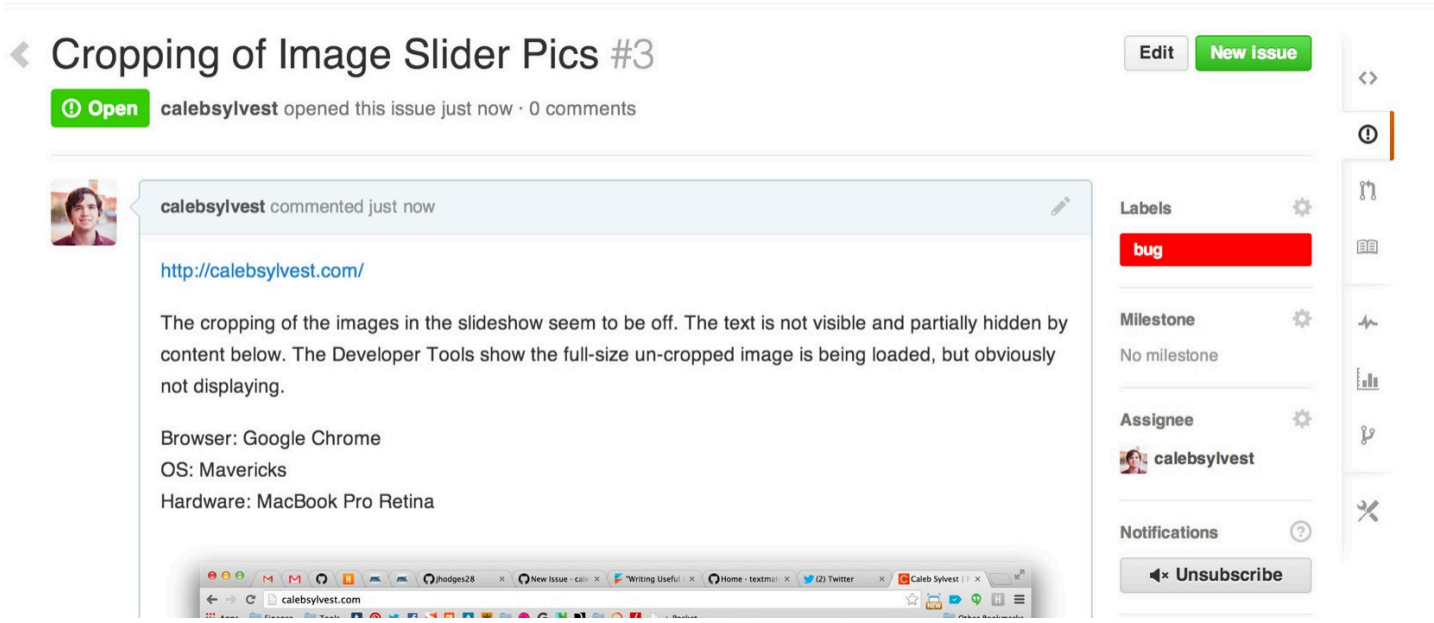
Is this issue useful?



NOTES

We asked the class. Some answers: It is not useful. We do not know which page or feature is affected, what “broken” means, what behavior was expected, how to reproduce the problem, or which browser, device, or runtime environment was used. A useful issue should provide enough context for another team member to reproduce and investigate the problem.

Writing useful Github issues



NOTES

Add context and be specific. You're writing this today, but someone — maybe even future you — might read it a year from now with zero memory of the situation.

Writing useful Github issues

- Issue should include
  - Context: explain the conditions which led you to write the issue
  - Problem or idea: the context should lead to something
  - Previous attempts to solve
  - Solution or next step (if possible)
- Be specific!
  - Include environment settings, versions, error messages, code examples when necessary

NOTES

Take yourself out of the loop when you write an issue — write it for whoever picks it up next, not for someone who already has your context.

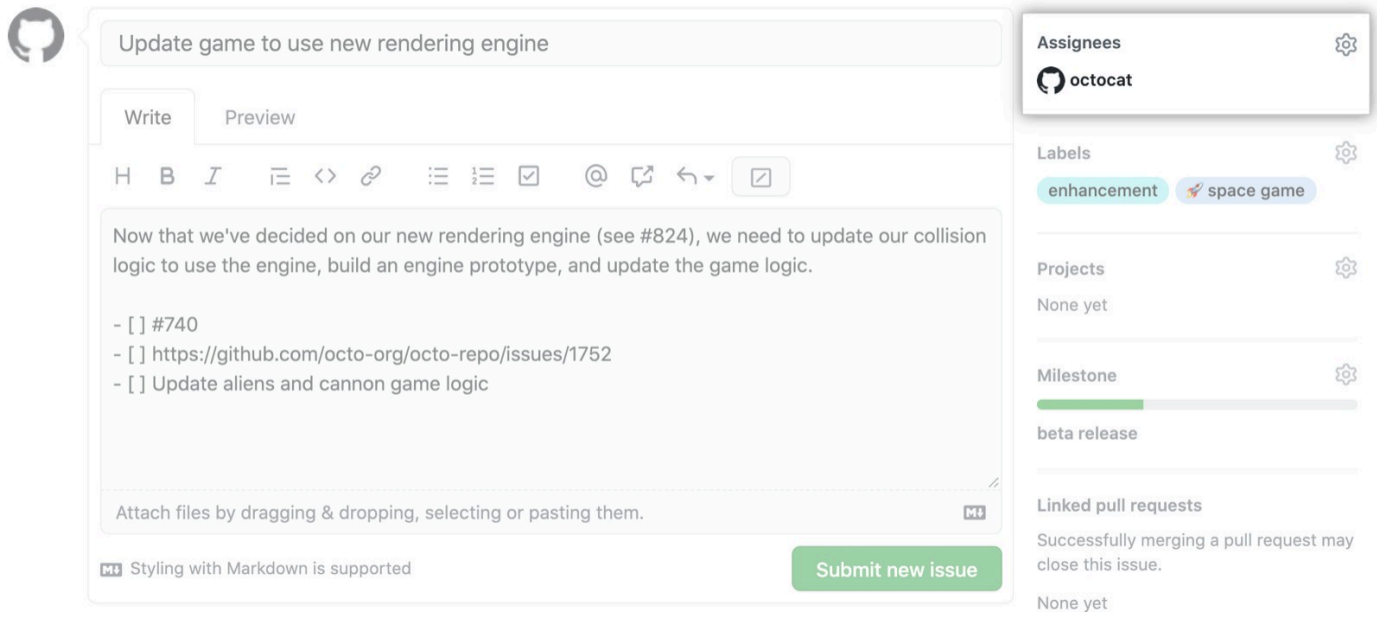
Writing useful Github issues

- Check out guidelines
  - Google: developers.google.com/issue-tracker
  - Rust: rustc-dev-guide.rust-lang.org
- Don’t assume the solution
- One issue per issue
- Keep titles short and descriptive
- Format your messages

NOTES

One issue per issue — think of the INVEST criteria from user stories; the same logic applies here.

@Mention or assign appropriate people



NOTES

Assigning an issue to a specific person is how you communicate ownership and responsibility clearly.

Use labels

- Break the project down by areas of responsibility
- Mark non-triaged issues
- Isolate issues that await additional information from the reporter
- Example:
  - Bug / Duplicate / Documentation / Help Wanted / Invalid / Enhancement
  - status: wip, status: ready to implement, status: needs discussion

---

NOTES

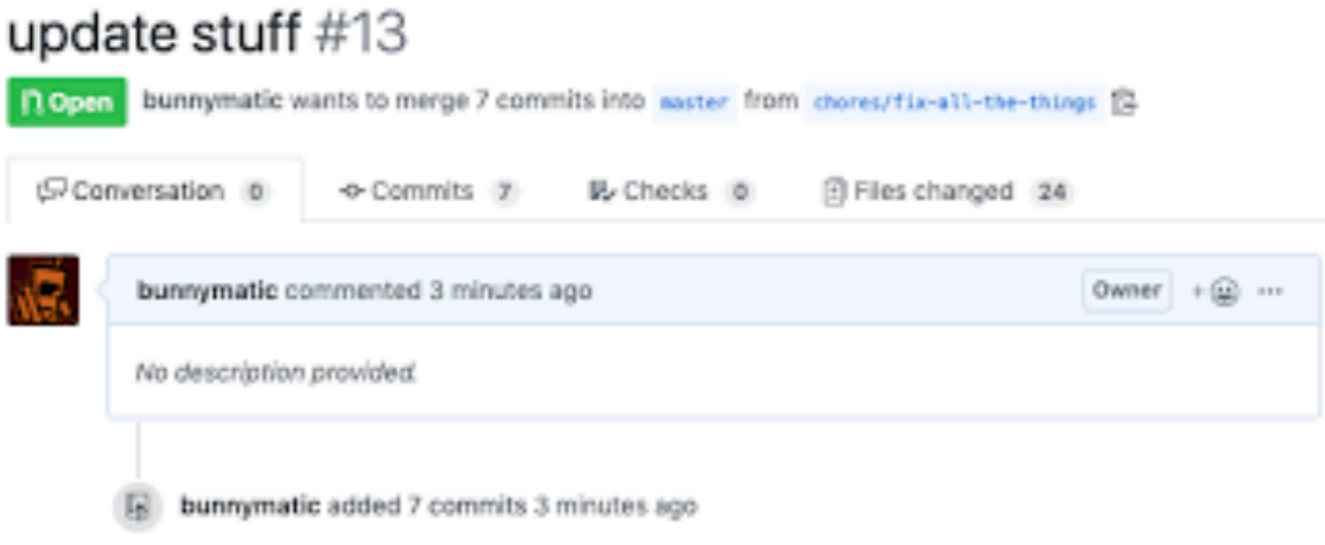
"WIP" means someone's actively working it. "Ready to implement" means it's scoped and actionable. "Needs discussion" means it's not ready to be worked yet. (Real-world example: the mislabeled issue connected to the 737 MAX case — labels matter.) In class, we looked at several active GitHub projects and examined how they use labels to organize issues.

Don’t forget to follow-up and close issues

- closes/resolves #issue_number

NOTES

Pull requests

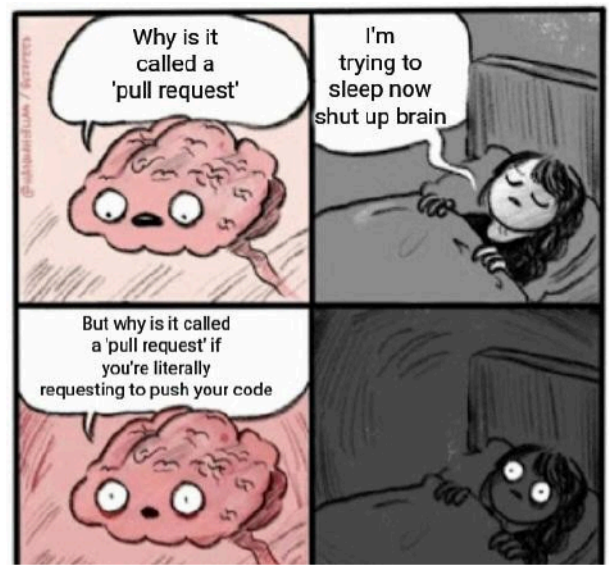


NOTES

This is what a bad PR looks like — no description, just a pile of commits.

How to write good pull requests

- `## What?`
- `## Why?`
- `## How?`
- `## Testing?`
- `## Screenshots (optional)`
- `## Anything Else?`



NOTES

Highlight important design decisions, tradeoffs, and any technical debt so reviewers do not have to guess. Using a PR template is a good practice because it helps ensure that important information is included. You used a predefined template in P1B.

How to write good pull requests

- Remember that anyone (in the company) could be reading your PR
- Be explicit about what/when feedback you want
- @mention individuals that you specifically want to involve in the discussion, and mention why.
  - "/cc @jesseplusplus for clarification on this logic"

NOTES

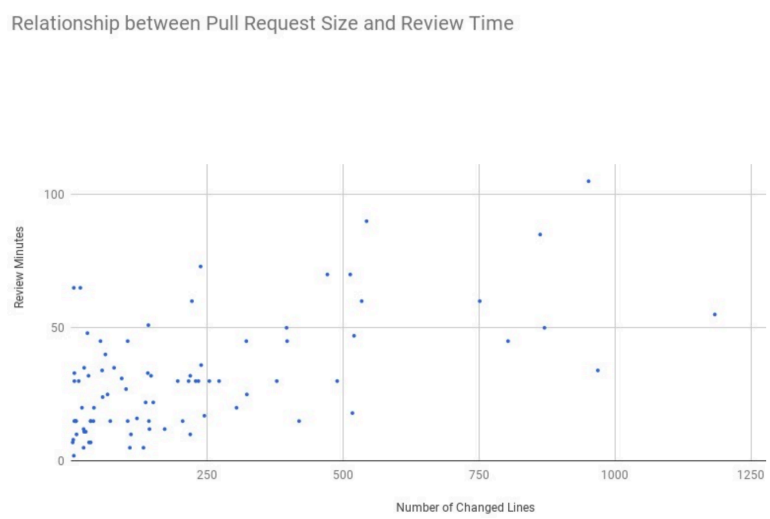
If a PR isn't finished, say so — a "[WIP]" prefix is a simple, common signal. PR quality genuinely correlates with whether it gets accepted, so it's worth the extra effort.

Keep your PRs small

NOTES

Studies consistently show bugs are harder to catch in large diffs, and big PRs block anyone whose work depends on that code. Tying a PR back to a specific user story also makes it much easier for a reviewer to evaluate.

With this number in mind, a good pull request should not have more than 250 lines of code changed



Based on a chart relating pull request size to review time — larger PRs get slower, shallower reviews.

NOTES

Would you be happy reviewing a pull request with thousands of changed lines? Would you review it carefully or just approve it? Large pull requests are difficult to review thoroughly, so important defects are more likely to be missed. Keep pull requests small and focused whenever possible. We will discuss effective code reviews in more detail later.

Offer useful feedback

- If you disagree strongly, consider giving it a few minutes before responding; think before you react.
- Ask, don’t tell. ("What do you think about trying…?" rather than "Don’t do…")
- Explain your reasons why code should be changed. (Not in line with the style guide? A personal preference?)
- Be humble. ("I’m not sure, let’s try…")
- Avoid hyperbole. ("NEVER do…")
- Be aware of negative bias with online communication.

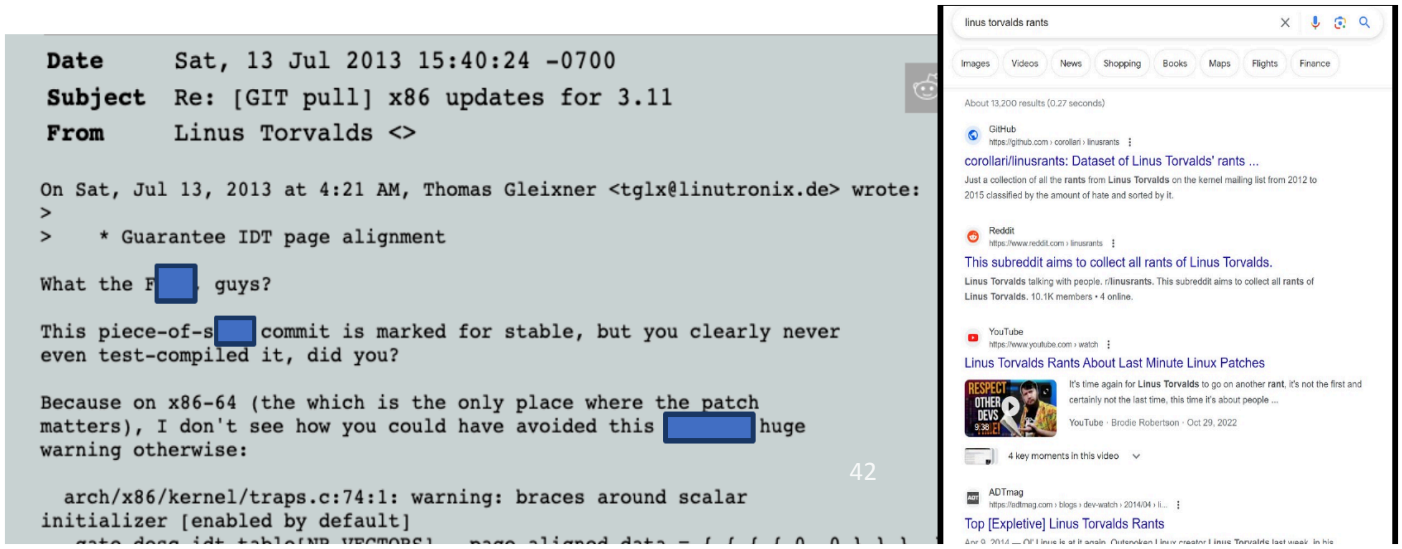
NOTES

People tend to read neutral text as negative by default, so lean positive on purpose. No one should ever get torn apart in a code review.

Avoid Duplicates

- "Duplicate of" issue/pull request number

Be a nice person



NOTES

Linus Torvalds is not always the best example of being nice in collaborative software projects. There are even datasets containing hundreds of his comments, though who knows what machine-learning tasks use them. The point is simple: when you are in a leadership position or reviewing a teammate's work, be respectful and focus your comments on the work, not the person. At the same time, in some projects, open-source projects, or workplaces, you cannot control how others communicate themselves. You may need to develop a thick skin, avoid taking comments personally, and focus on any useful feedback they contain.

Working as a team

- Establish a collaboration process
- Meet with the team
- Divide work and integrate
- **Share knowledge**
- Resolve conflicts

NOTES

Everything on this list really comes down to communication.

No matter the format, documentation is important

NOTES

Documentation serves different readers, including your future self, teammates, TAs, other developers, and end users. Each audience has different needs and background knowledge, so the same documentation will not work equally well for everyone. Before writing, ask who will use it, what they already know, and what they need to accomplish.

Know your audience

- Internal document for your team (e.g., meeting note)
- Documentation for project contributors
- Documentation for non-developer collaborators (e.g., UX researchers)
- Documentation for developer users
- Documentation for clients with no software knowledge
- User manual for end users

NOTES

Match your language to who's reading — domain terms for developers, plain explanations for everyone else. Different audiences care about different parts of the same concept.

What is wrong with this question?

New To Coding. Can anyone assist me?

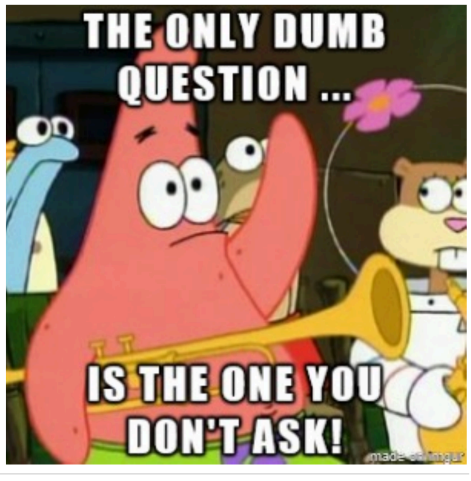
Asked 7 years, 1 month ago Modified 7 years, 1 month ago Viewed 47 times

-4

I am trying to make a word counter and I just cant seem to get it. Can

```
import re
print("Welcome To This Software Made By Aaron!")
word = raw_input("Enter Your Words: ")
Check = 0
Right = 0
Length = len(word)
while True:
    if Right == 1:
        if Length < Check:
            Check = Check + 1
            print(Check)
    if Length == Check:
        Right = 1

print("Your Word Count Is " +Check)
```



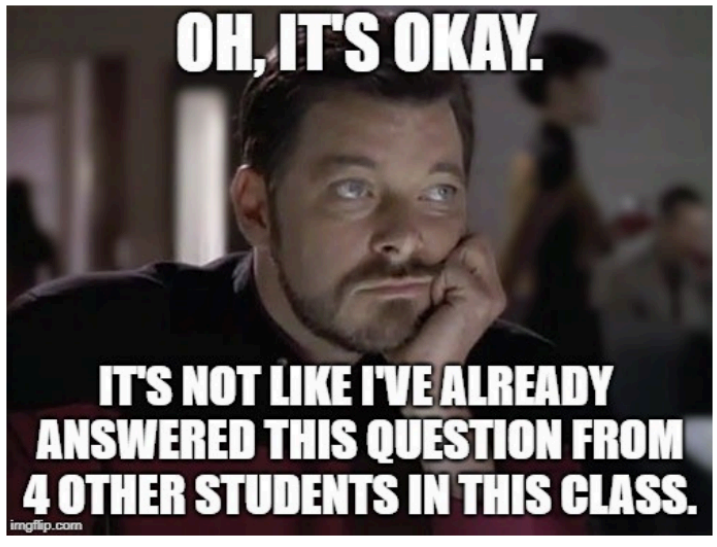
NOTES

Don't hesitate to ask questions — of instructors, teammates, or open-source maintainers — just learn to ask them well.

Make it easy for people to help you

- I am trying to ___, so that I can ___. I am running into ___. I have looked at ___ and tried ___.
  - I’m using this tech stack: ___.
  - I’m getting this error/result: ___.
  - I think the problem could be ___.

Avoid Duplication

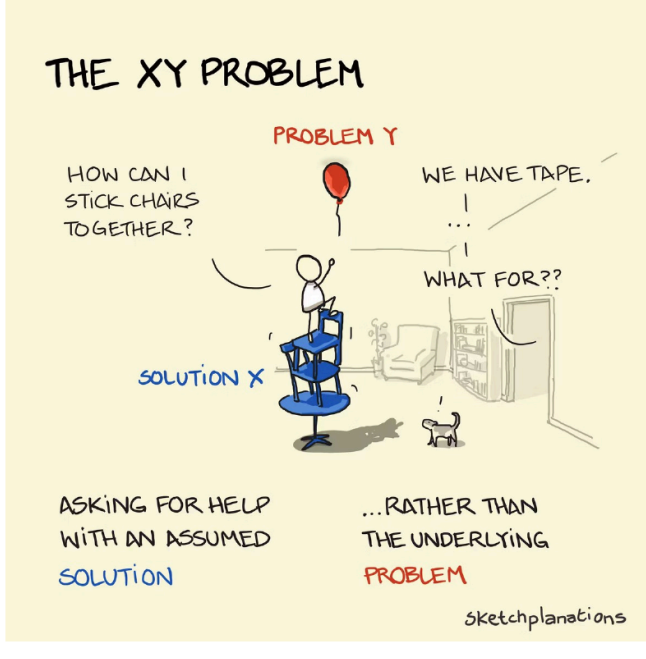


See also: research on mining/identifying duplicate questions and issues.

NOTES

This is a real research area — people build tooling specifically to detect duplicate questions and issues automatically.

Avoid the XY Problem



Credit: sketchplanations.com/the-xy-problem

NOTES

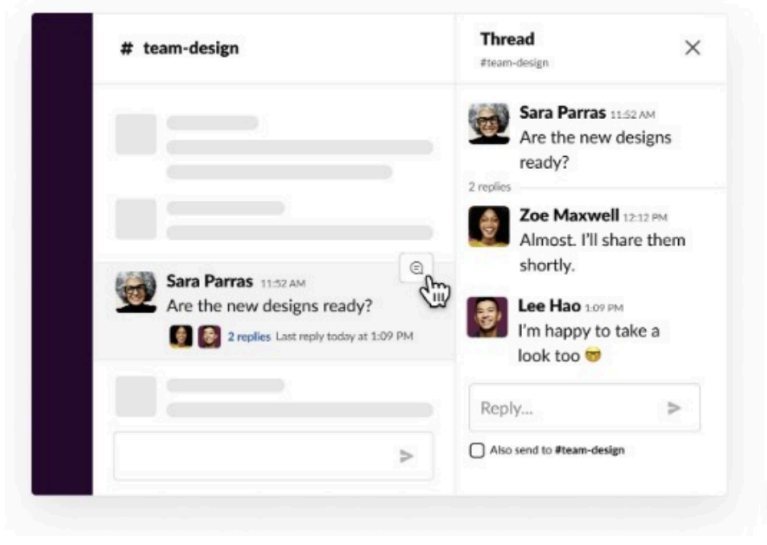
The XY problem occurs when someone asks for help with their attempted solution, X, instead of explaining the actual problem, Y. The helper may spend time fixing X without realizing that it is unnecessary or that a better approach exists.

Avoid Duplication - Slack

- Add quotation marks to search a specific phrase
  - "Connection refused errors" will find results containing the entire phrase
- Add `from:` in front of a display name to search for information shared by someone specific
  - `HW1 from:@Eduardo Feo`
- Add `is:thread` to search within threads
  - `WSL is:thread`
- Recap problem in caption to enable searching (if using screenshots)

Use threads

- Threads help us create organized discussions around specific messages, without adding clutter to a channel.
- You can manage thread notifications.



NOTES

Turning on notifications for a specific thread lets you follow up on exactly the question you asked, without tracking the whole channel. It also keeps the reasoning searchable for whoever hits the same issue later.

Use channels properly

- `#announcements`: Class / homework announcements
- `#general`: Administrative / logistics questions
- `#random`: Anything! Useful links, memes, …
- `#technicalsupport`: Technical issues (e.g., env setup, errors)

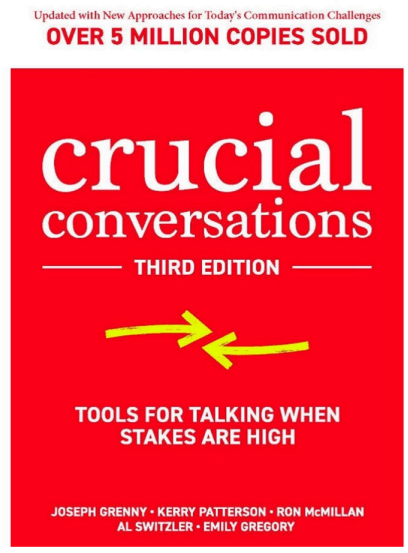
Archive and share the answers

- Avoid duplication!
- You’re probably not the only one who’s wondering.
- For 313, post your questions in public channels if possible.
  - Feel free to answer too!
- For your team, create a team wiki (e.g., Github project wiki) or shared google document.

NOTES

Pick the right medium for the right audience: public wikis or GitHub project wikis for teams, personal notes for yourself — the goal either way is not losing the answer.

Resolve Conflicts



NOTES

Consider a real 17-313 example: a student who stopped attending and contributing turned out to be juggling job interviews; the team spoke up, staff intervened, and the student got a second chance and improved. The lesson: the team being brave enough to communicate directly is what made the intervention possible — and it could have been avoided entirely with an upfront conversation. Real-world conflicts get harder with age/education/language gaps, time zones, and remote work — there's no universal fix, just communication. It could be you causing the friction just as easily as someone else.

You can't solve any problem without communication!

Conflict Resolution

- Your goal: Find a solution to the problem and move forward.
- Make sure that everybody works from the same set of facts.
- Establish ground rules for your team's discussion.
  - Talk about how the situation made you feel. Never presume anything about anyone else.
- Remain calm and rational. If you feel triggered or threatened, extract yourself from the situation, wait an hour to chill out, and then try again.
- If you reach an impasse, talk to your team leader.
- If your team remains in conflict, escalate to your mentor CA.
  - Your mentor CA will not solve your problem. They will help you to solve your own problems.

Team survey

RESEARCH-ARTICLE

Identifying Struggling Teams in Software Engineering Courses Through Weekly Surveys

Authors: Kai Presler-Marshall, Sarah Heckman, Kathryn T. Stolee Authors Info & Claims

SIGCSE 2022: Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 • February 2022 • Pages 126–132 • https://doi.org/10.1145/3478431.3499367

Reference: Presler-Marshall, K., Heckman, S., & Stolee, K. T. (2022). Identifying Struggling Teams in Software Engineering Courses Through Weekly Surveys. SIGCSE 2022.

NOTES

The weekly team survey helps course staff spot struggling teams early — it's confidential, not anonymous, and meant to trigger help, not punishment. 64.4% of students found it useful for keeping their team on track, and communication skills were consistently the top concern raised.