Teamwork and Professional Practices

Software Teams

- rarely just people writing code together
- most of a real project's code gets written independently
  - dividing work well, agreeing on interfaces, trusting a partner's piece are team skills
  - cooperative techniques like pair programming and swarming are for specific moments when tight coordination is needed
- there are many things besides writing correct code that a team must manage successfully
  - *roles* – there are different hats to be worn, and someone does more of different things at different moments
  - *conflict* – inevitable, and not necessarily a sign that something is broken
  - *communication* – how you say something is a skill with real consequences
  - *process* – how we organize our work together

Software Teams

- common roles
  - driver / navigator (moment-to-moment)
    - applies to pair programming
  - owner of a subsystem (over a project phase)
    - when tasks are split, the owner is not just who wrote the code but also who can explain how it works, who fixes it when it breaks, who others come to with questions about it
    - about responsibility, not territory – ownership doesn't mean off-limits to everyone else
  - reviewer (of a specific piece of work)
    - code review is a distinct skill with its own techniques
  - coordinator (rotating) – keeps track of deadlines, meeting schedule
    - the person actively using tools related to meeting structure and effort estimation
    - a housekeeping function rather than denoting who's in charge

Software Teams

- different moments call for different roles
  - in small teams each person wears multiple hats
- someone needs to own each responsibility, but roles rotate – not a permanent title
  - driver/navigator switch multiple times in a session
  - owner of a subsystem lasts for a project phase
  - reviewer lasts for a specific piece of work being reviewed
  - coordinator should rotate so no one is permanently saddled with the task

Advantages and Risks of Teams

- advantages
  - division of labor – more gets built
  - complementary skills / perspectives
  - built-in error-catching (multiple sets of eyes)
  - shared knowledge – together the team understands more of the system than any individual member would alone
- risks
  - coordination overhead – time spent syncing, not building
  - uneven contribution
  - communication overhead / misunderstanding
  - groupthink – bad ideas go unchallenged because no one pushes back
- teams can outperform individuals but only if the risks are actively managed
  - the benefits of teamwork are not automatic
  - the risks are not bad luck – they are the predictable cost of the same things that produce the benefits

Communication

- use the right form – each fits different needs
  - *synchronous* (meeting, call) – good for decisions, disagreements, design discussions
  - *asynchronous* (message, comment, doc) – good for status updates, quick questions
  - mismatch between the form and the task (e.g. a long design debate over asynchronous text) is a common source of friction
- writing clear messages
  - state what you need or what changed, not just “hey, question”
    - allows the recipient to respond without an extra round of communication just to find out what is being asked
  - be specific enough that the other person can act without a follow-up question
  - document decisions somewhere both of you can find them later – not just in someone’s memory or email

Diverse and Inclusive Teams

- different backgrounds bring different default communication styles and assumptions
- a diverse team’s advantage – more perspectives – only materializes if every voice actually gets heard
- actively invite quieter teammates’ input – ask “does anyone see this differently?”
  - silence can mean agreement, uncertainty, a communication-style difference, or simply not having been given room to speak – don’t assume it is agreement

Managing Other Risks

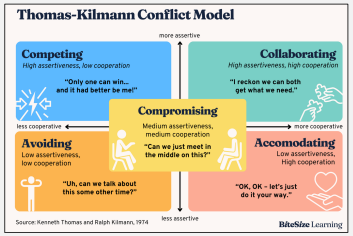
- *coordination overhead* – largely a cost to be minimized by design, not resolved after the fact
  - split tasks along clean interfaces, agreed on up front, so partners don’t need to sync constantly mid-task
  - reserve real-time sync (meetings, pair programming) for the moments that actually need it – not every small decision
  - some coordinate cost is unavoidable and worth paying – the goal is minimizing waste, not eliminating coordination entirely
- *uneven contribution* – prevent, don’t just resolve
  - split tasks with rough effort-parity in mind from the start
  - check in on progress before the deadline, not just at it
  - if it happens anyway, this is what conflict resolution is for

Conflict

- conflict is normal, not a failure
  - every team experiences disagreement
  - something to learn to handle well rather than only trying to avoid
    - the measure of a good team isn't zero conflict, but conflict that gets surfaced early and handled constructively
- common causes
  - uneven workload / perceived unfairness
  - technical disagreement – different ideas of the "right" design
  - unclear roles/expectations
  - communication style differences
  - missed commitments

Responses to Conflict

- five ways people respond to conflict
  - competing – push your position
  - accommodating – give in
  - avoiding – ignore it
  - compromising – split the difference
  - collaborating – find a solution addressing the concerns on both sides



 https://www.bitesizelearning.co.uk/resources/thomas-kilmann-conflict-model

Responses to Conflict

- none of these responses is always wrong – or always right
  - competing – push your position
    - correct when correctness or safety genuinely isn't up for negotiation or there isn't time to do anything else e.g. an ethical boundary is being crossed or there's a time-sensitive decision
    - toxic when it is the default
  - accommodating – give in
    - generous when it is chosen – you don't have a strong preference when another does, preserving the relationship matters more than winning, you realize that the other person is right
    - problematic when it is compulsive – someone always defers, always agrees – builds up resentment and silences a voice
  - avoiding – ignore it
    - a common personality trait in software teams
    - right when an issue is trivial or the timing is bad e.g. the argument needs a cooling-off period or time is needed to gather more information
    - detrimental to team performance when chronic – things which should be challenged aren't
  - compromising – split the difference
    - reasonable and fair – right when both sides have legitimate claims but there isn't time or trust available for the deeper work of collaboration
    - the wrong answer when problems don't have a middle ground – true of many design decisions
  - collaborating – find a solution that addresses the concerns on both sides
    - expensive – takes time, effort, emotional labor and requires vulnerability – and requires trust that others will engage in good faith
    - right when the issue is important enough to justify the cost and there's sufficient trust

Choosing a Mode

- choose what is called for, rather than just defaulting to your personal style

A rough guide, not an algorithm –

- how much does this actually matter?
  - trivial → avoiding is fine
  - high-stakes → worth more effort
- how much time is there?
  - very little → competing or a fast compromise
  - plenty → collaborating becomes viable
- how much trust exists with this person right now?
  - low → collaborating is risky and may not be reciprocated
  - higher → collaborating pays off
- whose issue is it more, really?
  - mostly theirs → accommodating might be right
  - mostly yours → don't just avoid it

Conflict Case Study

- For your scenario, what's the likely underlying cause and which mode is each person current defaulting to?
- As a group, decide which mode actually fits this situation best. Be ready to state your answer and your reasoning.

Scenario A

You and your partner agreed to split responsibilities: … Two days before the deadline, you check the shared repository and see almost nothing has been committed on their end. They say they've "been busy" and will "get to it".

Scenario B

You and your partner disagree on how to structure a particular component – you want a simpler approach that works for now; they want to build something more elaborate "because we'll need it later". Neither of you will budge, and you're a week behind.

Scenario C

Your partner keeps rewriting parts of your code without discussing it first, saying they are "just improving it". You feel like your work isn't respected, but you haven't said anything yet.

Scenario D

You and your partner are supposed to meet twice a week to sync up, but your partner keeps showing up late or canceling last-minute, always with a reasonable-sounding excuse. Individually none of it seems like a big deal, but you've fallen behind on decisions that need both of you and you are starting to resent bringing it up again.

Collaborating Well

- resolution process
  - name the issue concretely and early
  - state your interest, not just your position
  - listen for their underlying interest too
  - look for an option addressing both
  - if stuck, bring in a third party
- what not to do
  - silently resent a partner instead of raising the issue
  - escalate to accusation instead of describing behavior
  - wait until the deadline to bring up a problem that's been building for a week
  - choose avoiding or accommodating out of habit, not because it actually fits

Conflict Case Study

- For your scenario, what's the likely underlying cause and which mode is each person current defaulting to?
- As a group, decide which mode actually fits this situation best. Be ready to state your answer and your reasoning.
- As a group, draft 2-3 sentences of what the response would look like in your chosen mode. Be ready to share your answer.