Spring 6-16-2023

The Power of (Virtual) Convergence: The Unrealized Potential of Pair Programming and Remote Work

Mikayla Maki
Portland State University

The Power of (virtual) Convergence:

The Unrealized Potential of Pair Programming and Remote Work

By Mikayla Maki

For the last 6 months I've been working remotely on my PSU CS capstone. This should be a straightforward project: we're using a popular framework, we're building a straightforward application with plenty of prior art, we added a documentation generator in the first few weeks, and have even mandated that all PRs are reviewed. And yet this team has struggled to communicate and synthesize our work into a complete product. It wasn't until the second-to-last week that we actually integrated the most important component! Meanwhile, I've spent the past year working remotely at a startup, Zed, with colleagues around the globe. This is software development in hard mode: we're building a high performance, distributed system in our own closed source UI framework, with our own GPU render, custom data structures, and a custom network protocol and server. Yet somehow, the Zed team is able to iterate quickly, releasing new features every single week [1]. Why are these experiences so different?

To understand how Zed achieved this, we need to start at the basics of the work we do. Software development is the practice of turning ideas into a precise description of how a computer should act. This has long frustrated traditional management techniques because the process of turning specifications into running code can only be understood in retrospect. If we knew precisely what needed to be done, the problem would, by definition, have already been completed. Fred Brook's classic essay on the challenges of software development, *No Silver Bullet,* [2] describes the problem in terms of four essential difficulties:

- **Complexity:** Software involves dealing with vast interconnected systems that can have magnitudes more states than the other things we build.
- **Arbitrary:** Software has to interface with a complicated human and natural world that emerged with its own considerations. This requires programmers to understand not just how software works, but also the world beyond it.

- **Changeable:** Software is expected to change to reflect the world. Because it's so easy to recall or modify software, it can keep up with a changing world and users, and so it must.
- **Invisible:** Software is difficult to visually display because it has no natural relation to physical space. While you can attempt to portray it in 2 or 3 dimensional space, such rendering will always be a simplification. Keeping track of just what's going on in a software system is difficult because there is no natural map of the process at work.

Due to these essential difficulties, writing code is therefore a relatively small part of the work of programming. A much larger part is simply understanding the systems involved and how they interface with each other (be they social, environmental, technical) so that abstractions can be made that model or interlock with them. To make this problem even harder, programming is not done alone.

The process of building a shared understanding between people is described by Media Synchronicity Theory (MST) [3]. MST analyzes all communication in terms of two goals: conveyance of new information and convergence on an understanding of information. Asynchronous communication tends to be well suited for conveyance processes, where substantial time and care must be taken to create a new understanding of a situation, while synchronous communication is better for convergence processes, which require rapid, high-level interaction. As an example, pull requests (PRs) are a conveyance mechanism for enhancements to a project. They take a long time to put together, you only make a PR once you're ready, and it takes a lot of effort to review a colleague's PR. Video calls are a high convergence activity. Both participants are constantly communicating short messages with their voice, tone, and facial expressions, allowing participants to build up a mutual understanding of each other and the problem at hand. MST has already been used to analyze distributed software development and its

results have been shown to match how teams actually use communication tools [4]. So let's apply an MST based analysis to these two software teams.

Zed and the capstone team have a similar organizational structure. Both are small teams, eight and seven developers respectively, both use the same version control system (Git), both have a chat room, Slack and Discord, both have a culture of opening PRs, pushing commits, and leaving a few sentences of explanation as to what's been done and why, and both have an hour long meeting on Monday to talk about where people are at and what we're doing. But at the capstone, this was as far as communication could go. Everyone's availability was so disjoint that conversations on a specific feature or problem could go on for days or weeks before a solution would be found. Our ability to converge on a shared understanding was impeded by a total lack of tools for synchronizing our understanding beyond the Monday meetings. Sometimes we would go hours overtime at the Monday meeting, hashing out the last week's problems. This also resulted in wasted work, as approaches to problems like communicating with the database or implementing an account login page would be developed several times independently, without coordination between people. These challenges made it hard for the capstone to make as much progress as we wanted to.

To understand why Zed doesn't struggle with these same problems, we first have to understand Pair Programming. Pair programming is a long standing agile development practice where two developers sit down together, at the same workstation, and write code together. There's usually a split between a 'driver', the person writing the code, and the 'navigator', the one reviewing that code and thinking of the big picture. These two roles swap continuously throughout the pair programming session, so that the knowledge relevant to each role can be distributed between the two participants. This results in significantly less defects and a number

of intangible benefits, such as improving the pair's problem solving abilities and code quality [5]. But pair programming practices haven't been able to translate into a remote work environment.

D. Smite, et al. [6] surveyed pair programmers who went remote during the pandemic in Norway and found that they were unable to maintain their normal practices over the first year of the pandemic. Existing pair programming tools, like Tuple, VSCode's LiveShare, and IntelliJ's "Code with me" were frustrating and limited, so most developers were restricted to video calls. However, screen sharing and video calls can't support pair programming's continual swapping of driver and navigator. This is exhausting for everyone involved, and so most developers surveyed ended up writing code separately, with periodic video-call based check-ins. While this kind of synchronization would have certainly helped the capstone team, even in industry the technical challenges of remote pair programming haven't been solved. According to the The 2022 Stack Overflow Developer Survey, the most popular synchronous development tools people use are chat and video apps like Zoom, Microsoft Teams, and Slack [7], none of the pair programming tools are even on the list

Zed is solving the remote pair programming problem by building a new code editor, from the ground up. It uses cutting edge research into Conflict Free Replicated Data Types [8] to create a smooth, back-and-forth pairing experience that no other implementation has. With this strong foundation, Zed is able to have a remote pair programming culture. Whenever I have a problem with a component, I can message the person who built it and within minutes have them appear in my editor so they can walk me through how each piece fits together. This technology and the broader company culture of pair programming is so successful at enabling convergence that, within 3 months of joining the company, I went from never having touched a local

application to building a terminal emulator into Zed. But the benefits of frictionless, real-time collaboration go beyond just higher productivity; they also help develop the relationships that make up an organization.

Organizations are made up of large networks of people with both formal and informal interactions, collaborating to produce the organization's output. In the classic 1973 paper *The Strength of Weak Ties*, Mark S. Granovetter provides a framework for understanding how the character of these relationships affects a network's information flows. The strength of a tie is made up of "the amount of time, the emotional intensity, the intimacy, (...) and [reciprocity]" [9] characteristic to the tie. The strength of these ties corresponds to the similarity between the people involved as it's difficult to have strong ties with conflicting perspectives without "psychological strain" [9]. This coherency allows strongly connected people to work together easily due a shared understanding of each other, at the cost of homogeneity in viewpoints.

Weak ties act as a way to break out of this dynamic because they don't demand the same degree of cohesion between their members. They allow information to transfer between heterogeneous groups, improving knowledge transfer for everyone involved. This theory applies to the informal communication graph of an organization as much as any other group of people. As an example, senior developers sitting next to junior developers and graphic designers at lunch is essential for building up an organization-wide awareness of relevant information. But the onset of mass remote work in 2020 has disrupted the formation of strong and weak ties.

In a recent study, Emanuel Et. Al. observed how junior developers struggled with this shift at an unnamed Fortune 500 company. New developers, particularly women, were unable to get the feedback they needed, leading to up to a 5x larger quit rate [10]. In the language of relational ties, these junior developers were unable to develop their initial weak ties with senior

developers into the kinds of strong ties that enable them to participate autonomously in the organization's work.

This problem is corroborated by similar studies on MIT faculty [11] and on Microsoft employees [12] conducted during COVID. In both organizations there was a decline in communication with weak ties during the pandemic. People increasingly collaborated with pre-existing strong ties, largely disconnecting from their weak ties. Particularly concerning in the Microsoft study was that they found that there was a decline in the number of new weak ties formed – as the study put it the organization saw an "ossification" as a result of the move to remote work.

To understand why this change occurred, we can look at a more fine-grained study which surveyed 20 Taiwanese remote workers during the pandemic [13]. This study found that it was difficult for these remote workers to gauge each other's engagement level with the asynchronous tools they used to interact with their weak ties. With a lack of solid information, these workers tended to "develop polarized perceptions" of their coworkers and so were also less inclined to collaborate with weak ties. This particularly hurt new hires who had no pre-existing strong ties and lacked an understanding of how the implicit structure of the company worked. The result was that new hires to the company were left isolated, just as observed in the larger studies above.

My experience at the capstone matched this trend. I've only developed a rapport with my team lead, Zach. I don't have nearly the same relationship with my other teammates and it makes it difficult to interact with each other's PRs beyond just approving them. However, I had no difficulty forming strong ties while working at Zed. Because of Zed's reliance on pair programming, I was able to build up strong relationships with the senior engineers who in turn offered advice and mentorship. These relationships are why I could ship major features, at all, let

alone within just a few months of joining. Most other new hires at Zed have had a similar experience. As long as the new developer is able to ask questions and put themselves out there for pairing, they can get a lot of support. This social element of pair programming is also corroborated by the study from Smite Et. Al. of the norwegian programmers:

> “We [pair programmed] both with regard to quality, but not the least with regard to people's need for seeing each other and to get a feel of working together [...] it adds something positive in terms of more contact with the other team members" [6]

Pair Programming is highly effective at enabling strong tie formation and retaining junior developers, offsetting the costs that remote work levies on weak ties.

Strong ties require intimacy and mutual confidence in how each participant will perceive and respond to the world. Building up this understanding is fundamentally a convergence problem, and so requires fast, synchronous, reciprocal interaction to form. But in analyzing the software used to enable remote work I find that existing tools often frustrate convergence, and the few synchronous tools available are incapable of providing the rich back-and-forth necessary for reciprocal techniques like pair programming. However my experience at Zed shows that this is, at least in part, a technical problem. Frequent, remote pair programming using our own software created the opportunities for organic convergence that helps foster the strong and weak ties that are necessary for a functional organization. Maybe remote work doesn’t have to be quite so expensive

Citations

[1] “Zed Stable Releases,” Zed, https://zed.dev/releases/stable (accessed May 26, 2023).

[2] Brooks, “No silver bullet essence and accidents of software engineering,” Computer, vol. 20, no. 4, pp. 10–19, 1987. doi:10.1109/mc.1987.1663532

[3] A. R. Dennis, R. M. Fuller, and J. S. Valacich, “Media, tasks, and Communication Processes: A theory of media synchronicity,” MIS Quarterly, vol. 32, no. 3, p. 575, 2008. doi:10.2307/25148857

[4] T. Nii, A. Piri, C. Lassenius, and M. Paasivaara, “Reflecting the choice and usage of communication tools in GSD projects with media synchronicity theory,” *2010 5th IEEE International Conference on Global Software Engineering*, 2010. doi:10.1109/icgse.2010.11

[5] A. Cockburn and L. Williams, “The costs and benefits of pair programming: Extreme Programming examined,” ACM Digital Library, https://dl.acm.org/doi/abs/10.5555/377517.377531 (accessed May 26, 2023).

[6] D. Smite, M. Mikalsen, N. B. Moe, V. Stray, and E. Klotins, “From collaboration to solitude and back: Remote Pair programming during COVID-19,” arXiv.org, https://arxiv.org/abs/2105.05454v1 (accessed May 26, 2023).

[7] “Stack overflow developer survey 2022,” Stack Overflow, https://survey.stackoverflow.co/2022/ (accessed May 26, 2023).

[8] N. Sobo, “How CRDTs make multiplayer text editing part of Zed’s DNA - zed blog,” Zed, https://zed.dev/blog/crdts (accessed May 26, 2023).

[9] M. S. Granovetter, “The strength of weak ties,” *Social Networks*, pp. 347–367, 1977. doi:10.1016/b978-0-12-442450-0.50025-0

[10] N. Emanuel, E. Harrington, and A. Pallais, The Power of Proximity: Office Interactions Affect Online Feedback and Quits, Especially for Women and Young Workers. [Online]. Available: https://scholar.harvard.edu/pallais/publications/power-proximity-office-interactions-affect-online-feedback-and-quits-especially

[11] L. Yang et al., "The effects of remote work on collaboration among information workers," Nature Human Behaviour, vol. 6, no. 1, pp. 43–54, 2021. doi:10.1038/s41562-021-01196-4

[12] D. Carmody *et al.*, "The effect of co-location on Human Communication Networks," *Nature Computational Science*, vol. 2, no. 8, pp. 494–503, 2022. doi:10.1038/s43588-022-00296-z

[13] C.-L. Yang, N. Yamashita, H. Kuzuoka, H.-C. Wang, and E. Foong, "Distance matters to weak ties," Proceedings of the ACM on Human-Computer Interaction, vol. 6, no. GROUP, pp. 1–26, 2022. doi:10.1145/3492863