**Some reasons to work on productivity and velocity**

A common topic of discussion among my close friends is where the bottlenecks are in our productivity and how we can execute more quickly. This is very different from what I see in my extended social circles, where people commonly say that velocity doesn't matter. In online discussions about this, I frequently see people go a step further and assign moral valence to this, saying that it is actually bad to try to increase velocity or be more productive or work hard (see appendix for more examples).

The top reasons I see people say that productivity doesn't matter (or is actually bad) fall into one of three buckets:

- Working on the right thing is more important than working quickly
- Speed at X doesn't matter because you don't spend much time doing X
- Thinking about productivity is bad and you should "live life"

I certainly agree that working on the right thing is important, but increasing velocity doesn't stop you from working on the right thing. If anything, each of these is a force multiplier for the other. Having strong execution skills becomes more impactful if you're good at picking the right problem and vice versa.

It's true that the gains from picking the right problem can be greater than the gains from having better tactical execution because the gains from picking the right problem can be unbounded, but it's also much easier to improve tactical execution and doing so also helps with picking the right problem because having faster execution lets you experiment more quickly, which helps you find the right problem.

A concrete example of this is a project I worked on to quantify the machine health of the fleet. The project discovered a number of serious issues (a decent fraction of hosts were actively corrupting data or had a performance problem that would increase tail latency by > 2 orders of magnitude, or both). This was considered serious enough that a new team was created to deal with the problem.

In retrospect, my first attempts at quantifying the problem were doomed and couldn't have really worked (or not in a reasonable amount of time, anyway). I spent a few weeks cranking through ideas that couldn't work and a critical part of getting to the idea that did work after "only" a few weeks was being able to quickly try out and discard ideas that didn't work. In part of a previous post, I described how long a tiny part of that process took and multiple people objected to that being impossibly fast in internet comments.

I find this a bit funny since I'm not a naturally quick programmer. Learning to program was a real struggle for me and I was pretty slow at it for a long time (and I still am in aspects that I haven't practiced). My "one weird trick" is that I've explicitly worked on speeding up things that I do frequently and most people have not. I view the situation as somewhat analogous to sports before people really trained. For a long time, many athletes didn't seriously train, and then once people started trying to train, the training was often misguided by modern standards. For example, if you read commentary on baseball from the 70s, you'll see people saying that baseball players shouldn't weight train because it will make them "muscle bound" (many people thought that weight lifting would lead to "too much" bulk, causing people to be slower, have less explosive power, and be less agile). But today, players get a huge advantage from using performance-enhancing drugs that increase their muscle-bound-ness, which implies that players could not get too "muscle bound" from weight training alone. An analogous comment to one discussed above would be saying that athletes shouldn't worry about power/strength and should increase their skill, but power increases returns to skill and vice versa.

Coming back to programming, if you explicitly practice and train and almost no one else does, you'll be able to do things relatively quickly compared to most people even if, like me, you don't have much talent for programming and getting started at all was a real struggle. Of course, there's always going to be someone more talented out there who's executing faster after having spent less time improving. But, luckily for me, relatively few people seriously attempt to improve, so I'm able to do ok.

Anyway, despite operating at a rate that some internet commenters thought was impossible, it took me weeks of dead ends to find something that worked. If I was doing things at a speed that people thought was normal, I suspect it would've taken long enough to find a feasible solution that I would've dropped the problem after spending maybe one or two quarters on it. The number of plausible-ish seeming dead ends was probably not unrelated to why the problem was still an open problem despite being a critical issue for years. Of course, someone who's better at having ideas than me could've solved the problem without the dead ends, but as we discussed earlier, it's fairly easy to find low hanging fruit on "execution speed" and not so easy to find low hanging fruit on "having better ideas". However, it's possible to, to a limited extent, simulate someone who has better ideas than me by being able to quickly try out and discard ideas (I also work on having better ideas, but I think it makes sense to go after the easier high ROI wins that are available as well). Being able to try out ideas quickly also improves the rate at which I can improve at having better ideas since a key part of that is building intuition by getting feedback on what works.

The next major objection is that speed at a particular task doesn't matter because time spent on that task is limited. At a high level, I don't agree with this objection because, while this may hold true for any particular kind of task, the solution to that is to try to improve each kind of task and not to reject the idea of improvement outright. A sub-objection people have is something like "but I spend 20 hours in unproductive meetings every week, so it doesn't matter what I do with my

other time". I think this is doubly wrong, in that if you then only have 20 hours of potentially productive time, whatever productivity multiplier you have on that time still holds for your general productivity. Also, it's generally possible to drop out of meetings that are a lost cause and increase the productivity of meetings that aren't a lost cause[1].

More generally, when people say that optimizing X doesn't help because they don't spend time on X and are not bottlenecked on X, that doesn't match my experience as I find I spend plenty of time bottlenecked on X for commonly dismissed Xs. I think that part of this is because getting faster at X can actually increase time spent on X due to a sort of virtuous cycle feedback loop of where it makes sense to spend time. Another part of this is illustrated in this comment by Fabian Giesen:

> It is commonly accepted, verging on a cliche, that you have no idea where your program spends time until you actually profile it, but the corollary that you also don't know where *you* spend your time until you've measured it is not nearly as accepted.

When I've looked how people spend time vs. how people think they spend time, it's wildly inaccurate and I think there's a fundamental reason that, unless they measure, people's estimates of how they spend their time tends to be way off, which is nicely summed in by another Fabian Giesen quote, which happens to be about solving rubik's cubes but applies to other cognitive tasks:

> Paraphrasing a well-known cuber, "your own pauses never seem bad while you're solving, because your brain is busy and you know what you're thinking about, but once you have a video it tends to become blindingly obvious what you need to improve". Which is pretty much the usual "don't assume, profile" advice for programs, but applied to a situation where you're concentrated and busy for the entire time, whereas the default assumption in programming circles seems to be that as long as you're actually doing work and not distracted or slacking off, you can't possibly be losing a lot of time

Unlike most people who discuss this topic online, I've actually looked at where my time goes and a lot of it goes to things that are canonical examples of things that you shouldn't waste time improving because people don't spend much time doing them.

An example of one of these, the most commonly cited bad-thing-to-optmize example that I've seen, is typing speed (when discussing this, people usually say that typing speed doesn't matter because more time is spent thinking than typing). But, when I look at where my time goes, a lot of it is spent typing.

A specific example is that I've written a number of influential docs at my current job and when people ask how long some doc took to write, they're generally surprised that the doc only took a day to write. As with the machine health example, a thing that velocity helps with is figuring out which docs will be influential. If I look at the docs I've written, I'd say that maybe 15% were really high impact (caused a new team to be created, changed the direction of existing teams, resulted in significant changes to the company's bottom line, etc.). Part of it is that I don't always know which ideas will resonate with other people, but part of it is also that I often propose ideas that are long shots because the ideas sound too stupid to be taken seriously (e.g., one of my proposed solutions to a capacity crunch was to, for each rack, turn off 10% of it, thereby increasing effective provisioned capacity, which is about as stupid sounding an idea as one could come up with). If I was much slower at writing docs, it wouldn't make sense to propose real long shot ideas. As things are today, if I think an idea has a 5% chance of success, in expectation, I need to spend ~20 days writing docs to have one of those land.

I spend roughly half my writing time typing. If I typed at what some people say median typing speed is (40 WPM) instead of the rate some random typing test clocked me at (110 WPM), this would be a 0.5 + 0.5 * 110/40 = 1.875x slowdown, putting me at nearly 40 days of writing before a longshot doc lands, which would make that a sketchier proposition. If I hadn't optimized the non-typing part of my writing workflow as well, I think I would be, on net, maybe 10x slower[2], which would put me at more like ~200 days per high impact longshot doc, which is enough that I think that I probably wouldn't write longshot docs[3].

More generally, Fabian Giesen has noted that this kind of non-linear impact of velocity is common:

> There are "phase changes" as you cross certain thresholds (details depend on the problem to some extent) where your entire way of working changes. ... There's a lot of things I could in theory do at any speed but in practice cannot, because as iteration time increases it first becomes so frustrating that I can't do it for long and eventually it takes so long that it literally drops out of my short-term memory, so I need to keep notes or otherwise organize it or I can't do it at all.
>
> Certainly if I can do an experiment in an interactive UI by dragging on a slider and see the result in a fraction of a second, at that point it's very "no filter", if you want to try something you just do it.
>
> Once you're at iteration times in the low seconds (say a compile-link cycle with a statically compiled lang) you don't just try stuff anymore, you also spend time thinking about whether it's gonna tell you anything because it takes long enough that you'd rather not waste a run.

> Once you get into several-minute or multi-hour iteration times there's a lot of planning to not waste runs, and context switching because you do other stuff while you wait, and note-taking/bookkeeping; also at this level mistakes are both more expensive (because a wasted run wastes more time) and more common (because your attention is so divided).
>
> As you scale that up even more you might now take significant resources for a noticeable amount of time and need to get that approved and budgeted, which takes its own meetings etc.

A specific example of something moving from one class of item to another in my work was this project on metrics analytics. There were a number of proposals on how to solve this problem. There was broad agreement that the problem was important with no dissenters, but the proposals were all the kinds of things you'd allocate a team to work on through multiple roadmap cycles. Getting a project that expensive off the ground requires a large amount of organizational buy-in, enough that many important problems don't get solved, including this one. But it turned out, if scoped properly and executed reasonably, the project was actually something a programmer could create an MVP of in a day, which takes no organizational buy-in to get off the ground. Instead of needing to get multiple directors and a VP to agree that the problem is among the org's most important problems, you just need a person who thinks the problem is worth solving.

Going back to Xs where people say velocity doesn't matter because they don't spend a lot time on X, another one I see frequently is coding, and it is also not my personal experience that coding speed doesn't matter. For the machine health example discussed above, after I figured out something that would work, I spent one month working on basically nothing but that, coding, testing, and debugging. I think I had about 6 hours of meetings during that month, but other than that plus time spent eating, etc., I would go in to work, code all day, and then go home. I think it's much more difficult to compare coding speed across people because it's rare to see people do the same or very similar non-trivial tasks, so I won't try to compare to anyone else, but if I look at my productivity before I worked on improving it as compared to where I'm at now, the project probably would have been infeasible without the speedups I've found by looking at my velocity.

Amdahl's law based arguments can make sense when looking for speedups in a fixed benchmark, like a sub-task of SPECint, but when you have a system where getting better at a task increases returns to doing that task and can increase time spent on the task, it doesn't make sense to say that you shouldn't work on something because you spend a lot of time doing it. I spend time on things that are high ROI, but those things are generally only high ROI because I've spent time improving my velocity, which reduces the "I" in ROI.

The last major argument I see against working on velocity assigns negative moral weight to the idea of thinking about productivity and working on velocity at all. This kind of comment often assigns positive moral weight to various kinds of leisure, such as spending time with friends and family. I find this argument to be backwards. If someone thinks it's important to spend time with friends and family, an easy way to do that is to be more productive at work and spend less time working.

Personally, I deliberately avoid working long hours and I suspect I don't work more than the median person at my company, which is a company where I think work-life balance is pretty good overall. A lot of my productivity gains have gone to leisure and not work. Furthermore, deliberately working on velocity has allowed me to get promoted relatively quickly[4], which means that I make more money than I would've made if I didn't get promoted, which gives me more freedom to spend time on things that I value.

For people that aren't arguing that you shouldn't think about productivity because it's better to focus on leisure and instead argue that you simply shouldn't think about productivity at all because it's unnatural and one should live a natural life, that ultimately comes down to personal preference, but for me, I value the things I do outside of work too much to not explicitly work on productivity at work.

As with this post on reasons to measure, while this post is about practical reasons to improve productivity, the main reason I'm personally motivated to work on my own productivity isn't practical. The main reason is that I enjoy the process of getting better at things, whether that's some nerdy board game, a sport I have zero talent at that will never have any practical value to me, or work. For me, a secondary reason is that, given that my lifespan is finite, I want to allocate my time to things that I value, and increasing productivity allows me to do more of that, but that's not a thought i had until I was about 20, at which point I'd already been trying to improve at most things I spent significant time on for many years.

Another common reason for working on productivity is that mastery and/or generally being good at something seems satisfying for a lot of people. That's not one that resonates with me personally, but when I've asked other people about why they work on improving their skills, that seems to be a common motivation.

A related idea, one that Holden Karnofsky has been talking about for a while, is that if you ever want to make a difference in the world in some way, it's useful to work on your skills even in jobs where it's not obvious that being better at the job is useful, because the developed skills will give you more leverage on the world when you switch to something that's more aligned with you want to achieve.

## Appendix: one way to think about what to improve

Here's a framing I like from Gary Bernhardt (not set off in a quote block since this entire section, other than this sentence, is his).

People tend to fixate on a single granularity of analysis when talking about efficiency. E.g., "thinking is the most important part so don't worry about typing speed". If we step back, the response to that is "efficiency exists at every point on the continuum from year-by-year strategy all the way down to millisecond-by-millisecond keystrokes". I think it's safe to assume that gains at the larger scale will have the biggest impact. But as we go to finer granularity, it's not obvious where the ROI drops off. Some examples, moving from coarse to fine:

1. The macro point that you started with is: programming isn't just thinking; it's thinking plus tactical activities like editing code. Editing faster means more time for thinking.
2. But editing code costs more than just the time spent typing! Programming is highly dependent on short-term memory. Every pause to edit is a distraction where you can forget the details that you're juggling. Slower editing effectively weakens your short-term memory, which reduces effectiveness.
3. But editing code isn't just hitting keys! It's hitting keys plus the editor commands that those keys invoke. A more efficient editor can dramatically increase effective code editing speed, even if you type at the same WPM as before.
4. But each editor command doesn't exist in a vacuum! There are often many ways to make the same edit. A Vim beginner might type "hhhhxxxxxxxx" when "bdw" is more efficient. An advanced Vim user might use "bdw", not realizing that it's slower than "diw" despite having the same number of keystrokes. (In QWERTY keyboard layout, the former is all on the left hand, whereas the latter alternates left-right-left hands. At 140 WPM, you're typing around 14 keystrokes per second, so each finger only has 70 ms to get into position and press the key. Alternating hands leaves more time for the next finger to get into position while the previous finger is mid-keypress.)

We have to choose how deep to go when thinking about this. I think that there's clear ROI in thinking about 1-3, and in letting those inform both tool choice and practice. I don't think that (4) is worth a lot of thought. It seems like we naturally find "good enough" points there. But that also makes it a nice fence post to frame the others.

## Appendix: more examples

- Velocity doesn't matter, from Julia Evans, who I believe has been the most widely read programming blogger since about 2015
- In the comments on a post where Ben Kuhn notes that he got 50% more productive by allocating his time better, people are nearly uniformly negative about the post and say that he works too much. Although Ben clarified in multiple comments as well as in the post that not all time tracked was worked, the commenters are too busy taking the moral high ground to actually respond to the contents of the post
- Comments on Jamie Brandon's "Speed Matters"
    - Working quickly is pointless because you will be forced to do more work
    - Speed doesn't matter if you're doing the right thing, and also, if such a thing as speed did exist, it would be unmeasurable and therefore pointless to discuss
    - Thinking about productivity is unhealthy. One should relax instead
    - You can only choose 2 of "good, fast, cheap", therefore it is counterproductive to work on speed
    - A large speedup is impossible
    - "The author mistakes coding for typing"
    - etc.
    - As with Ben's post, virtually all of these comments are addressed in the post itself. I'm going to stop noting when this is true because it is generally true of the posts referred to here.
- The #3 comment on a post by Michael Malis on "How to Improve Your Productivity as a Working Programmer ": "Fuck it, the entire work environment seems designed to decrease productivity . . . Why should I bother . . ."
    - #4 comment: "What if I don't want to improve my productivity ? Just take time."
        - After the initial indignation, this comment goes on and proves that the commenter missed the point entirely, as the rest of the comment explains how the commenter works productively, which the commenter apparently is ok with as long as it's not phrased as a way to work productively, because one is supposed to be morally outraged by someone wanting to be productive and sharing techniques about how to be productive with other people who might be interested in being productive
        - In the responses, someone points out that someone who's more productive would be able to spend more time on leisure; that comment is uniformly panned because "work expands so as to fill the time available for its completion", as if how one spends time is some sort of immutable law of nature and not something under anyone's control
    - Another comment: "Alright. What are we optimizing for? Productivity? Or the end-goals of any of: achieving more, climbing the corporate ladder, making more money, etc..?"
- Comments on a post by antirez about productivity
    - The article is talking about the 10x programmer universe, not the normal universe most people live in
    - It's pointless to work on productivity since your environment determines productivity
    - Productive programmers are selfish, don't mentor, etc., and are bad for their teams because their increased productivity always comes from neglecting more important things, so anyone who's productive as a programmer is actually counterproductive for the team

        - If you read the entire comments to the post, you'll see that this is a common theme
- Comments on Alexy Guezy's thoughts on productivity
    - "Serious question: Is anything less productive than reading other people's productivity thoughts? It's a combination of procrastination and finding out what works for someone who is presumably more productive than you (ie: guilt)."
    - An anti-productivity article titled "Against Productivity
- Typing speed doesn't matter because you only spend 0.5% to 1% of your time typing
    - Despite the talk about 8-hour work days, I think people who get 4 hours of real work in a day are generally considered extremely productive. 0.5% to 1% of 4 hours is 1.2 minutes to 2.4 minutes a day or, for someone who types 100 wpm, 240 total words across slack, JIRA, email, actual code, commit messages, design docs, comments on design docs, documentation, etc.; I don't believe I know any professional programmers who type that little

etc.

Some positive examples of people who have used their productivity to "fund" things that they value include Andy Kelley (Zig), Jamie Brandon (various), Andy Matuschak (mnemonic medium, various), Saul Pwanson (VisiData), Andy Chu (Oil Shell). I'm drawing from programming examples, but you can find plenty of others, e.g., Nick Adnitt (Darkside Canoes) and, of course, numerous people who've retired to pursue interests that aren't work-like at all.

## Appendix: another reason to avoid being productive

An idea that's become increasingly popular in my extended social circles at major tech companies is that one should avoid doing work and waste as much time as possible, often called "antiwork", which seems like a natural extension of "tryhard" becoming an insult. The reason given is often something like, work mainly enriches upper management at your employer and/or shareholders, who are generally richer than you.

I'm sympathetic to the argument and agree that upper management and shareholders capture most of the value from work. But as much as I sympathize with the idea of deliberately being unproductive to "stick it to the man", I value spending my time on things that I want enough that I'd rather get my work done quickly so I can do things I enjoy more than work. Additionally, having been productive in the past has given me good options for jobs, so I have work that I enjoy a lot more than my acquaintances in tech who have embraced the "antiwork" movement.

The less control you have over your environment, the more it makes sense to embrace "antiwork". Programmers at major tech companies have, relatively speaking, a lot of control over their environment, which is why I'm not "antiwork" even though I'm sympathetic to the cause.

Although it's about a different topic, a related comment from Prachee Avasthi about avoiding controversial work and avoiding pushing for necessary changes when pre-tenure ingrains habits that are hard break post-tenure. If one wants to be "antiwork" forever, that's not a problem, but if one wants to move the needle on something at some point, building "antiwork" habits while working for a major tech company will instill counterproductive habits.

Thanks to Fabian Giesen, Gary Bernhardt, Ben Kuhn, David Turner, Marek Majkowski, Anja Boskovic, Aaron Levin, Lifan Zeng, Justin Blank, Heath Borders, Tao L., Nehal Patel, and Jamie Brandon for comments/corrections/discussion

---

1. When I look at the productiveness of meetings, there are some people who are very good at keeping meetings on track and useful. For example, one person who I've been in meetings with who is extraordinarily good at ensuring meetings are productive is Bonnie Eisenman. Early on in my current job, I asked her how she was so effective at keeping meetings productive and have been using that advice since then (I'm not nearly as good at it as she is, but even so, improving at this was a significant win for me). [return]

2. 10x might sound like an implausibly large speedup on writing, but in a discussion on writing speed on a private slack, a well-known newsletter author mentioned that their net writing speed for a 5k word newsletter was a little under 2 words per minute (WPM). My net rate (including time spent editing, etc.) is over 20 WPM per doc.

   With a measured typing speed of 110 WPM, that might sound like I spend a small fraction of my time typing, but it turns out it's roughly half the time. If I look at my writing speed, it's much slower than my typing test speed and it seems that it's perhaps half the rate. If I look at where the actual time goes, roughly half of it goes to typing and have goes to thinking, semi-serially, which creates long pauses in my typing.

   If I look at where the biggest win here could come, it would be from thinking and typing in parallel, which is something I'd try to achieve by practicing typing more, not less. But even without being able to do that, and with above average typing speed, I still spend half of my time typing!

   The reason my net speed is well under the speed that I write is that I do multiple passes and re-write. Some time is spent reading as I re-write, but I read much more quickly than I write, so that's a pretty small fraction of time. In principle, I could adopt an approach that involves less re-writing, but I've tried a number of things that one might

expect would lead to that goal and haven't found one that works for me (yet?).

Although the example here is about work, this also holds for my personal blog, where my velocity is similar. If I wrote ten times slower than I do, I don't think I'd have much of a blog. My guess is that I would've written a few posts or maybe even a few drafts and not gotten to the point where I'd post and then stop.

I enjoy writing and get a lot of value out of it in a variety of ways, but I value the other things in my life enough that I don't think writing would have a place in my life if my net writing speed were 2 WPM.

[return]

3. Another strategy would be to write shorter docs. There's a style of doc where that works well, but I frequently write docs where I leverage my writing speed to discuss a problem that would be difficult to convincingly discuss without a long document.

   One example of a reason that my docs is that I frequently work on problems that span multiple levels of the stack, which means that I end up presenting data from multiple levels of the stack as well as providing enough context about why the problem at some level drives a problem up or down the stack for people who aren't deeply familiar with that level of the stack, which is necessary since few readers will have strong familiarity with every level needed to understand the problem.

   In most cases, there have been previous attempts to motivate/fund work on the problem that didn't get traction because there wasn't a case linking an issue at one level of the stack to important issues at other levels of the stack. I could avoid problems that span many levels of the stack, but there's a lot of low hanging fruit among those sorts of problems for technical and organizational reasons, so I don't think it makes sense to ignore them just because it takes a day to write a document explaining the problem (although it might make sense if it took ten days, at least in cases where people might be skeptical of the solution).

   [return]

4. Of course, promotions are highly unfair and being more productive doesn't guarantee promotion. If I just look at what things are correlated with level, it's not even clear to me that productivity is more strongly correlated with level than height, but among factors that are under my control, productivity is one of the easiest to change. [return]

← *What to learn*

*The value of in-house expertise* →

*Archive*

*Twitter*