December 12, 2017

Managing Programmer Productivity

By **Tim Ottinger**
8 min read

Culture · Learning · Mob Programming · Tech Safety

Several decades ago, the software industry was obsessed with counting lines of code per programmer per month. That metric practice famously drove teams to create bloated, defect-ridden software products. After only a few years, the practice was abandoned as detrimental.

Sadly, the biases and flaws behind that practice are with us still, influencing our practices and policies in subtle ways.

There is a persistent myth that programming is mostly typing, that programmers know exactly what to type in to get the desired effect, and that there isn't that much thinking, reading, and design involved.

What would it mean to us if it is not so?

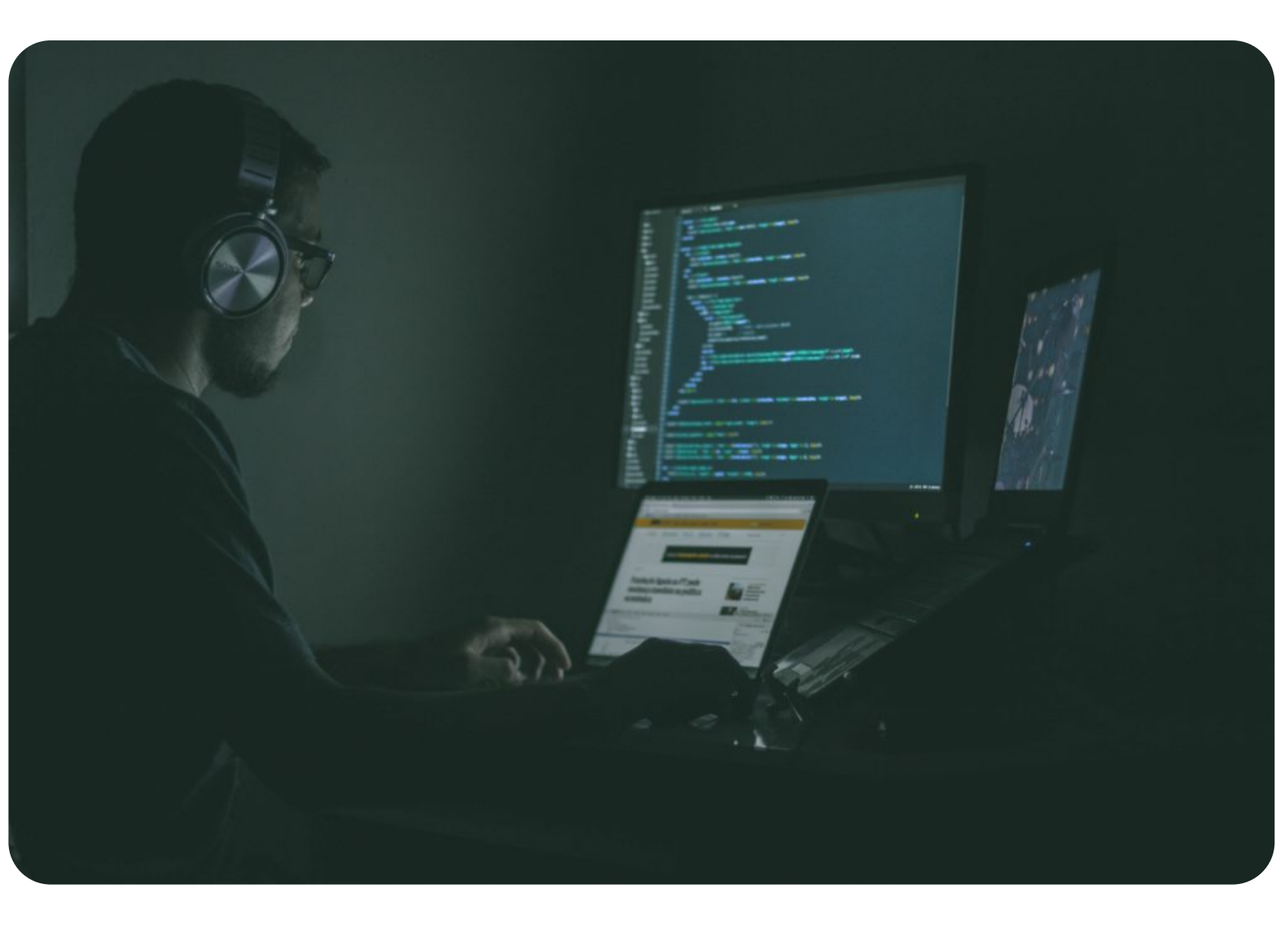
If programming were (mostly) typing, then we'd replace programmers with typists. Typists are much faster at typing than programmers are at programming. We'd have them typing as fast as they can in order to get the maximum output.

As many times as I've made that statement, no managers have been tempted to try it.

We recognize that programming requires special skills, knowledge, and dedication. Yet our policies often reflect an underlying belief in "programming as typing." As a result, our policies and practices remain sub-optimal.1

What is Programming Like?

The work of programming does produce lines of code, and those lines of code are produced by typing into the computer. However, 11/12ths of the work programmers do is not typing but thinking.



Typing is not the bottleneck.

Developers spend nearly all of their time trying to understand the existing code so that they may introduce new features without violating any existing rules or corrupting any data paths. Reading, understanding, and discussing the existing design is a primary activity.

Most corporate programmers spend at least 25% of their time in meetings. The goal of these meetings is usually to help make good decisions about the design and purpose of new features. They refer to these as "the good meetings."

Some developers I have polled reported spending upwards of 50% of their time in meetings, and added that most of the time they would rather have been writing code or fixing bugs instead.

There is a constant risk of making mistakes. A mistake in programming may be called a "defect", "design flaw", or "bug" (among other names). Typically, corporate programmers report spending 60% to 80% of non-meeting time fixing defects. Often these are discovered late in the process when there is little recovery time; "hardening sprints" sometimes drag on for weeks, preventing release of software.

Yes, for those doing the math, most programmers spend less than 30% of their total time writing new features. They tend to divide their feature coding time between 3-5 different features that they're trying to manage at once, so a given feature may receive 6-10% of a single programmer's attention (assuming individual work assignments).

No wonder it feels like feature production is going slowly, eh?

Returning to the 60% of time that goes into defect-fixing: why are so many defects present?

Making mistakes in programming is a lot like making mistakes in management policy. It all seems right and well-intentioned, and then it goes horribly wrong when it meets actual use and practice – when you find it conflicts with other values and policies in a surprising way. If the programmer knew ahead of time that they were making a mistake, they would not have made the mistake.

A defect is simply a decision that did not work out. Some of these are tiny decisions ("what kind of integer should I use here?") and others are larger decisions ("how should I construct the database for efficient query", "how can we do this in 1/2 the round-trips to the server", "what cloud architecture should we use"). Programming involves many hundreds of decisions every day.

Decisions tend to be made in a hurry, and often are made by an individual in isolation. It's not surprising that defects persist, especially in the pressurized world of software deadlines.

What If Programming Is Mostly Thinking?

If the goal is for programmers to successfully and carefully apply principles of good design into a complicated existing system with lots of rules and potential side-effects, then our industry has been going about things in a sub-optimal way1.

We probably don't want programmers sitting alone, working in isolation, cranking out the maximum lines of code (or story points or feature count) per week. That way optimizes typing time and defect production.

If programming is a kind of complicated knowledge work, then we want people combining their knowledge of the system, their knowledge of systems in general, their understanding of the hardware and operating system, their knowledge of the programming language, and their collective insight. We want them to produce "the right code" rather than "more code."

Perhaps we should think about producing code
at *the best speed*
rather than *the maximum speed.*

If errors are taking more than half of our programming time, then we can go twice as fast if we had no defects at all. This would not require increased headcount or faster typing.

Defect reduction is a process of ongoing experimentation and education.

Teams must find their errors, consider the conditions and origins of the errors, and then construct safer habits and code constructs.

Team Productivity

Uncle Bob Martin suggests that our industry is doubling in size about every five years. That suggests that in an average company, 50% of the employees have less than 6 years of experience.

We find that experienced programmers tend to have better habits than new programmers, and deeper knowledge of domains, platforms, languages, and operating systems.

Newer programmers tend to have more enthusiasm and more knowledge of recently-emerging technologies. Due to typically having fewer non-work responsibilities (spouses, children, etc) they tend to be available for longer work hours as well, making them attractive to hiring managers.

Typical companies have a mix of the two, and yet most organizations tend to keep the experienced developers and the new developers working on individual, assigned tasks with minimal interaction and knowledge-sharing. When any interaction takes place, it tends to be the experienced developer coming to the aid (or rescue) of the junior, rather than continual peering and mentoring.

Each team member brings their own experience and knowledge to the work, and none of them is as careful, intelligent, quick, and prudent as all of them.

Jessica Kerr noted that programmers should be recognized less for their individual productivity ("Sue gets 20% more done than her teammates") but rather for their *generativity* ("Everyone on the team gets 20% more done now that Sue is there").

This work is intense and requires that people maintain mental freshness. We find we can increase productivity by taking regular breaks, and we find that during these breaks we have the majority of our breakthrough ideas.

Better Ways

Existing, flawed models of programmer productivity are held in place mostly by good intentions and confirmation bias, which is complicated because staff tries so very hard to please their managers that even awful decisions sometimes end in improved results … at least temporarily, and at the cost of unsustainably increased effort.

The best way we know to *decrease* productivity is to have people work alone, without tests, for long hours without breaks, in separate branches, producing the largest numbers of lines of code per day possible. This style looks productive from a "raw typing" point of view but it tends to create many integration problems and other errors. It also keeps them hidden until it is too late to make a successful release.

The best ways we know to *sustainably improve productivity* in programming teams involves working in groups (pair programming and/or mob programming), continuously integrating code and using test-driving techniques (Test-Driven Development, Behavior-Driven Development, Design By Contract) to prevent or detect errors early in the process. It requires rapid feedback loops with users in order to ensure that the product being built has the desired outcome. It is only possible if there is the necessary psychological safety to keep learning and constant integration of ideas alive.

Of course, if programmers' time were not divided across so many tasks, they may complete each task more quickly.

You can find a summary of the motivating ideas for productive teams at the modern agile website, and you will find these concepts foremost in the training and coaching you receive from Industrial Logic professionals (as well as other organizations who have chosen these values).

If you are the one leading a team, what should be your policy on how programmers work?

I await your comments, below.

Notes:
1 "Sub-optimal" is a the author's polite way of saying "lousy."

Share

in f X

Our Author

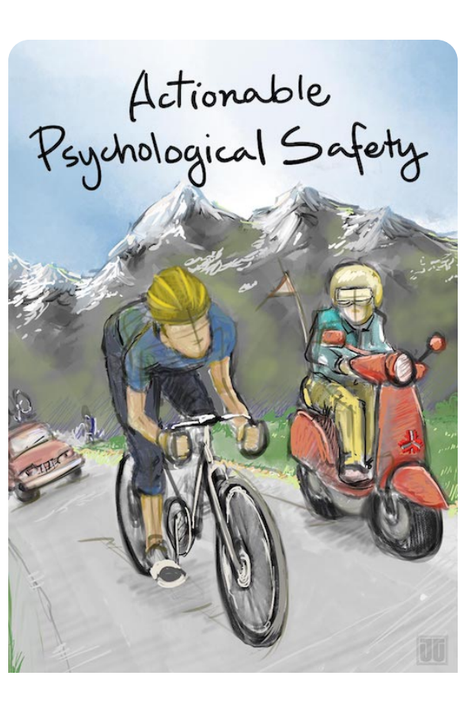
Tim Ottinger
Senior Consultant
Edinburgh, Scotland - UK

More About Tim Ottinger

Upcoming Workshops

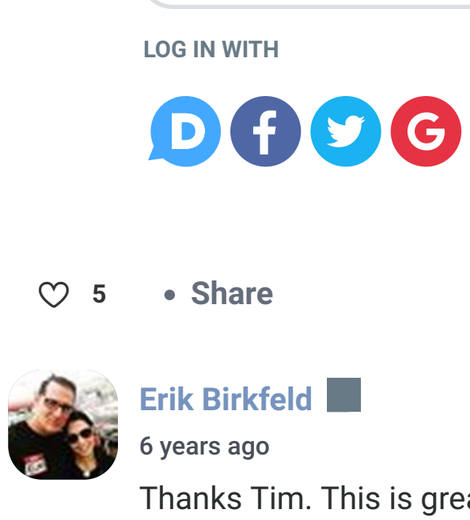
Technical Excellence Workshop - US
January 8 - February 29

Technical Excellence Workshop - EU
January 8 - February 29



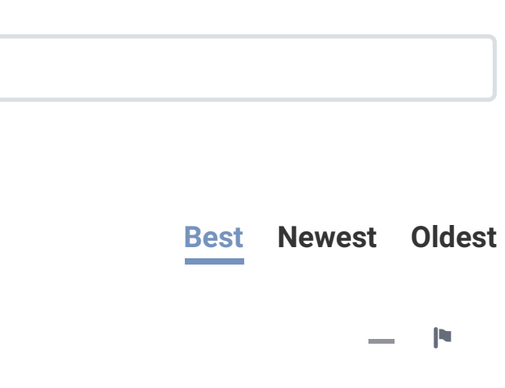
Actionable Psychological Safety Workshop
September 25 - November 2

Browse eLearning



Take a Workshop

Read a Blog



9 Comments Login ▾

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

♡ 5 • Share Best Newest Oldest

Erik Birkfeld
6 years ago
Thanks Tim. This is great and I also love Jessica's point,
" but rather for their generativity ("Everyone on the team gets 20% more done now that Sue is there")."

That collective success is a paradigm shift and a game changer.

4 0 Reply • Share ›

> **Art Gorr** → Erik Birkfeld
> 5 years ago
> I believe the generativity lever is much greater than the individual 10X lever.
>
> 1 0 Reply • Share ›

alexkates
2 years ago
Evergreen post. Re-reading now after a few years and I'm having all the same aha moments all over again!

Thank you, Tim!

1 0 Reply • Share ›

Elder Morris
4 years ago edited
I would add CLOSE CONTACT WITH THE USER as a key element. Without that, half the time we deliver a well designed, bug free, completely documented program that turns out to be useless because doesn't do what the user needs.

0 0 Reply • Share ›

> **Tim Ottinger** → Elder Morris
> 4 years ago
> See the above mention of "rapid feedback loops with users in order to ensure that the product being built has the desired outcome."
>
> So clearly we agree!
>
> 0 0 Reply • Share ›

Vaibhav Gawali
5 years ago
Great article. The best way for an experienced developer is to lead by example, if he is lousy then soon juniors join the path.

0 0 Reply • Share ›

BMK LAKSHMINARAYANAN
6 years ago
Thanks Tim, great article and it is timely as well :)

0 0 Reply • Share ›

david putman
6 years ago
Great article: "80% of non-meeting time fixing defects" but what about the 95% of meeting time spent discussing defects, regardless of the original intent of the meeting?
Deming famously pointed out decades ago that around 85% of our productivity is determine by the system we are working in, which includes our colleagues. Only 15% or so by our personal efforts.

0 0 Reply • Share ›

> **Tim Ottinger** → david putman
> 6 years ago edited
> * -94% belong to the system (the responsibility of management)
> * – 6% are attributable to special causes.
>
> "The New Economics" 1994 – Ch. 2 -The Heavy Losses-, page 33
>
> 3 0 Reply • Share ›

Subscribe Privacy Do Not Sell My Data DISQUS