It's OK to Want to Have a Good Time

Luke Church
University of Cambridge, UK
Vibe Robotics Inc

Mariana Marasoiu
Gradient Control Laboratories

Abstract

We reflect on our decade or so of experience advising students, corporate teams and policy makers, many of whom really just want to have a good time at work and want others to as well. However, they seem to be embarrassed to say so, they dress their work in the mantle of productivity, and mostly damage it in the process.

In this short opinion piece, we argue that it is long past time to say goodnight to Taylorism and ask what models of research aligned to something humane might replace it?

2012 ACM Subject Classification Software and its engineering → Designing software

Keywords and phrases productivity, happiness, Hegel

Digital Object Identifier 10.4230/OASIcs.Programming.2025.5

1 The status quo

For years, programming languages, frameworks, libraries, and environments have been marketed along the lines of developer productivity, that is, that it will take less time for a developer to solve the problem with the New Fancy Language (™) than the Old Crufty One (e.g. [7, 1, 9]). Timing studies have been a staple of UX research since the beginning of the field (see for example [11]). Influential research frameworks such as Cognitive Dimensions of Notations originated partially from questions about claims made about the productivity of visual programming languages compared to textual ones [4].

In the past, we too have participated in these conversations, from reporting times taken to use new APIs [5], to finding things in code completions [6] and code reviews [10]. And why wouldn't we, surely this is in everyone's best interests? Developers don't like it when it takes a long time to get things done, companies don't like paying them for it, and customers don't like paying the companies for it.

Of course, productivity isn't a uniform thing, we don't all try and build the same software. The authors have spent years building data visualisations and CAD tools, and are reasonably productive at it; but if we needed to write DSP code for a music synthesis tool we wouldn't be very quick at it. This is part of what gives rise to hyperbolic descriptions of performance-associated "rockstars", "ninjas", "10x developers" and the like. There are some problems where a certain threshold of experience is needed to be able to make progress on solving the problem, and it doesn't appear to be possible to create a cumulative effect of inexperienced people to compensate for experienced ones. Hardly a surprise, really.

Very crudely speaking, this concern of the application of scientific methods to improve the productive output per hour of software developers is known as Scientific Management, or Taylorism [13]. It is implicitly embedded in a lot of the software industry, from the productivity tools we've alluded to above, to work management strategies like Agile.

2 What's the problem then?

So, productivity seems to be a sensible thing to think about, creates mutually aligned benefits for customers-developers-employers, and is all pervasive. What is there to complain about?

The short answer is that we don't think many people actually [behave like they] believe in it. What they actually want to do is to make decisions based on aesthetics, ethics and taste. They want to have a good time programming. It's just they're embarrassed to say so, and so they dress up their preferences in the garments of productivity in order to seem serious. Let's stop doing that.

A key motivation for this essay originates from a recent discussion with a colleague considering the next steps in their career. They showed the first author a visual programming language in the node and arc design strategy, and argued that an iteration on the interaction paradigm was going to result in better productivity.

We think there are good reasons to be sceptical about this. Node and arc programming languages are as old as the hills – the oldest we could find [2] was published at the same time as Pygmalion, one of the first and foundational visual programming environments [12]. The ensuing surge in the design of new visual languages, node and arc and otherwise, has been motivated by claims of being "better" than textual programming languages; yet the empirical studies that led to Cognitive Dimensions showed serious productivity issues [4]. When the first author worked on the product that would become Autodesk's Dynamo visual language [3], he spent a lot of time and effort trying to overcome these pitfalls by exploiting the isomorphism between visual and textual representations (a feature called "Node to Code" [8]). Ultimately this work, novel at its time, was mostly useless.

Based on many hours of sitting with people programming in Dynamo, it was clear that many of them preferred the visual presentation to the textual version (or other supported textual languages). But they didn't prefer it because it was quicker for them to use, it often wasn't, they preferred it because it *felt* better. They described it as being comfortable – they could apply their extensive visual reasoning experience to work with the material. Sure there was a lot of clickity-clicking to do to fix things – but who cares, they're architects, they're used to needing to perform manual labour to manipulate visual material. They would rather move arcs and nodes around than deal with the insane pedantry of a compiler, or the obscure pseudo-linguistic pseudo-mathematical soup that the textual languages gave them. Architects, seemingly unlike programmers, were sufficiently organisationally effective to be able to negotiate for a working style they preferred rather than one that was better for their employers.

Going back to our colleague, when we poked at the argument, they didn't actually seem to fundamentally be motivated by productivity either. They'd already built the new system, not because they wanted to be productive – but just because they wanted to use it! They were now looking for a way to justify it, but didn't feel that saying they preferred it was likely to get them the funding they needed to continue working on it.

3 Productivity claims are (mostly) about politics, not developers

But it doesn't stop here. Not only are people feeling embarrassed and pressured to dress up their arguments as productivity claims, productivity claims get bandied around as if they're scientific facts. In our experience, they're mostly not.

Talking to another colleague, they reported a conversation with their management that went something along the lines of "one of our competitors has just posted on X that they rewrote their codebase of a million lines in half an hour with AI Tool Of the Moment (™). Why can't we do that?"

The reason why is pretty simple. The original claim was a lie, or in contemporary terminology "aspirational marketing". But it's a very potent lie; it can be used to distract, demotivate, destabilise your competitors.

Similarly, claims about 5-10x productivity improvement that get bandied about by frameworks and languages are ridiculous. Let's take a notional feature of a product that needs to be developed. In our experience, if in the end it takes a total of, say, 100 hours of time, 25 of those hours are spent trying to work out what it should do, 25 hours are spent in meetings motivating, coordinating, synchronising and calibrating the team, 25 hours of the hours are spent typing in code, and 25 of the hours are spent calibrating it with users, and fixing what it does. If the productivity improvement of the language was to reduce the development time to zero, then that's a 25% productivity improvement, not a 10x one. Amdahl's law again.

By far the biggest productivity problem in software development is understanding the purpose for which the software is being written, and not having to throw it away and do it again; something that studies of productivity rarely include. We're not suggesting that all developer productivity research is bad – we certainly don't think that is the case. What were are trying to do is to highlight that the total scope of the professional activities of a software engineer are wide, varied and complicated.

4 Professionality

So, in order to lend credibility to the macro argument about productivity of programming languages, we have to move out of scope most of the work of an experienced software engineer, and focus only on the coding part they do.

But even in this absurdly narrowed scope – we don't find agreement. Many developers seem to have an intrinsic dislike of being treated as workers in factories for software. They keep engaging in pesky behaviours like "building it right", "wanting to build things that matter" or "producing quality work", and no amount of explaining to them about the "net present value" of their work seems to satisfy them.

Fundamentally we think that this is because they see themselves as having an identity as a professional as well as a component in a money making machine. They see that professional commitment as a responsibility to produce some kinds of work over others, even if it isn't right for commercial reasons. Just like some prefer a visual presentation, even if it isn't productive, some prefer code that isn't a massive pile of technical debt, even if it isn't what they were asked for.

5 Why are we embarrassed about all of this?

The picture we're painting in this essay is one of highly skilled, highly motivated, creative professionals believing that their craft work has an extra-commercial justification. Why would this be something to be embarrassed about?

It seems to us that software developers feel vulnerable. Despite being in one of the most revenue generating industries in history, they feel like their enjoyment at work has to be on the basis of a capitalist intent, whilst simultaneously knowing that this is an inversion of the role of markets and policies – they're invented tools to act as human coordination mechanisms not a priori necessities.

But this feels like it can only be part of the story. We would like to invite a discussion as to how come we've allowed taste, personal preference, purpose and ethics to get relegated so far down the priority list.

6 What to do?

It's a fine question. Answers on a post-it note, but perhaps, at least to start, we could be a little bit more sceptical about both how hard it is to optimise the nature of being a cog in a factory – we're pretty good cogs already – and whether that is the appropriate aspiration for someone with 7 years of professional training, or whether perhaps it's acceptable to want to enjoy your work instead.

References

1 Lars Bak and Kasper Lund. Dart for the Entire Web. `https://news.dartlang.org/2015/03/dart-for-entire-web.html`, March 2015. Accessed: 2025-3-29.

2 Jack B Dennis. First Version of a Data Flow Procedure Language. Technical Report MIT-LCS-TM-061, Laboratory for Computer science, MIT., May 1975.

3 Dynamo BIM. Dynamo. `https://dynamobim.org/`. Accessed: 2025-5-18.

4 Thomas R G Green and Marian Petre. When visual programs are harder to read than textual programs. In *Human-Computer Interaction: Tasks and Organisation, Proceedings ECCE-6 (6th European Conference Cognitive Ergonomics)*, volume 57. academia.edu, 1992.

5 Andrew Macvean, Luke Church, John Daughtry, and Craig Citro. API Usability at Scale. In *Proceedings of the 27th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2016)*, 2016.

6 Mariana Marasoiu, Luke Church, and Alan Blackwell. An empirical investigation of code completion usage by professional software developers. In *Proceedings of the 26th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2015)*, pages 71–82, 2015.

7 Rob Pike. Go at Google: Language Design in the Service of Software Engineering. `https://go.dev/talks/2012/splash.article`, 2012. Accessed: 2025-3-29.

8 Dynamo Primer. DesignScript Syntax - Node to Code. `https://primer.dynamobim.org/07_Code-Block/7-2_Design-Script-syntax.html#node-to-code`. Accessed: 2025-5-18.

9 Daniel Rosenwasser. TypeScript 2.0 is now available! `https://devblogs.microsoft.com/typescript/announcing-typescript-2-0/`, September 2016. Accessed: 2025-3-29.

10 Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. Modern code review: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '18, pages 181–190, New York, NY, USA, May 2018. Association for Computing Machinery. `doi:10.1145/3183519.3183525`.

11 M E Sime, Thomas R G Green, and D J Guest. Psychological evaluation of two conditional constructions used in computer languages. *International journal of human-computer studies*, 51(2):125–133, August 1972. `doi:10.1006/ijhc.1972.0302`.

12 David Canfield Smith. *Pygmalion: A Creative Programming Environment*. PhD thesis, Stanford University, 1975.

13 Wikipedia contributors. Scientific management. `https://en.wikipedia.org/wiki/Scientific_management`, May 2025.