KLOCs Matter

https://devSteve.com/blog/posts/klocs-matter.html

September 02, 2025

Assertion

Despite popular opinion, KLOCs (or "thousands of lines of code") are the best metric for measuring developer productivity.

Definitions

My assertion applies to "software developers" or "people who are paid to write code". My assertion does not apply to "technical leads", "data scientists", "software architects", "project managers", "qa engineers", or any other software-adjacent job title.

If you're still uneasy with the "*paid to write code*" definition, then how about "*a person who's role entails >=50% time allocation to writing code for a product or service within a company/organization*"? This is what I mean by "software developer", and to whom my KLOC assertion applies.

Another point to make clear is that "most productive" is a relative term. Different teams can have different constraints that can affect developer productivity. We grade on a curve, and by "most productive", I mean the developers on a team who ship more code than other developers on the *same team*.

Lastly, by "shipping code" I mean deploying code to production. If you want to give partial credit for merging code to dev or QA, be my guest, but full credit only goes to the code that makes it to production. Also, lines-of-code *removed* count at least as much as lines-of-code added. There's no need to flood the comment section with remarks about how "*removing unused code is just as valuable as adding new code*". I'm referring to the total lines-of-code **delta**, not specifically lines-of-code **added**.

Observations

The most productive developers always seem to ship the most code (relative to the other members of their teams). This has been true for 100% of my 20+ year career. There has never been an exception.

"*But aren't the best software developers the ones who can solve the hardest problems?*" Yes, and being able to solve more problems allows one to ship more code. The code you don't know how to write doesn't get shipped.

"But don't the best software developers use DRY (Don't-Repeat-Yourself) patterns and elegant solutions that require fewer lines of code than their sub-optimal counterparts?" Yes, more experienced developers tend to write more succinct code, but because they solve problems more efficiently, they tend to have more bandwidth to solve a larger number of problems.

Taking a step back, a software project usually breaks down into three phases:

1. Make it work.
2. Make it work correctly.
3. Make it fast and efficient.

Developers don't understand a problem until they've had time to form a mental model in their heads. The mental model will evolve over time as their understanding of the problem improves.

This lack of full understanding at the onset of a software project is what leads to the aforementioned three-phase approach to software development. In phase one you just want get something (anything) to work. The first working version or "MVP" is rarely completely optimized. Besides, premature optimization is the root of all evil, as the old saying goes.

Getting to the first working version will require code to be written that may or may not be completely DRY with perfect abstractions and no extraneous code. If the code *is* completely optimal in its first commit, congratulations, but lines of code were still committed so my assertion that *"the person who ships the most code is the most productive developer on the team"* still holds true.

Just because Developer A might write more lines of code to solve Problem X than Developer B, that doesn't mean that Developer A's overall lines-of-code shipped will exceed Developer B's. Based on real-world experience, quite the contrary has been true. Developer B is more likely to ship more code than Developer A because competent experienced developers often ship more code, and thus are more productive, than inexperienced junior developers.

Exception

There's a clear exception to the line *"competent experienced developers often ship more code, and thus are more productive, than inexperienced junior developers"* from the previous section, and that is that some people are just plain lazy.

There are people in this world who work their hardest to do the bare minimum - as ironic as that sounds. I've seen it first hand. A developer will seem completely aloof until someone tries to get them to work on a hard problem, then it's game-on baby! They'll come up with all sorts of reasons why someone else should be assigned the work with a level of tenacity and vigor rarely exhibited.

AI Agents

There's another caveat to the line "*competent experienced developers often ship more code, and thus are more productive, than inexperienced junior developers*", and that caveat is AI code generation.

With AI tools like Claude Code, even someone with little-to-no prior software development experience can generate working code. In order for the code to count as "productive" by our definition, it still needs to be merged and shipped to production.

"Throw away" code that never gets merged does not count toward the lines-of-code metric, though it could be argued that any learnings or knowledge gleaned from the throw-away experiments should still count for *something*.

For the AI-generated code that *does* get merged and shipped, then those lines of code count just like any others. There is no distinction between human-generated code and AI-generated code with respect to my assertion. Shipped lines-of-code are shipped lines-of-code.

Alternatives

The most commonly used alternative methods of measuring developer productivity I've seen are:

- Number of tickets closed.
- Number of story points completed.
- Number of LLM tokens used.

The LLM token usage approach is obviously fairly new, and in my opinion pales in comparison to measuring lines of code *shipped* (rather than lines of code *generated by an LLM* - which may or may not get merged and shipped).

"Tickets closed" is another interesting method because it rewards the procedural over the empathetic. When someone reaches out for help with an urgent issue, responding with a cold "have you filed a ticket yet?" clearly establishes the business-nature of the interaction, and downplays the more human side of helping someone.

In a setting that measures productivity on the number of tickets closed, the procedural employee will appear more productive, even if an empathetic employee solves more problems but didn't file a ticket for each problem solved.

The "story points completed" metric has never seemed quite right because there's always someone on the team that either pads/over-values their story points (ex: puts 8 points where others might put 4 or even 2), or over-itemizes their tasks so instead of one 4 point story they end up with four 2 point stories. The people who use this approach rarely seem to ship the most code.

The people who advocate for these alternative productivity metrics often argue that lines-of-code isn't a valid metric either because developers can simply game the system by writing more verbose code thus producing more lines of code. This argument seems weak for two reasons:

1. Code reviews are a thing. If someone was gaming the system it would be immediately obvious to the rest of the team.
2. Tools/scripts can be written to remove comments and whitespace to give a more consistent view of the code being committed.

Conclusion

Admittedly, I've never worked with another software developer who has agreed that KLOCs are the best metric for measuring developer productivity. Almost all of them have used the argument that "*better developers solve problems in fewer lines of code, and therefore the lines-of-code metric should not be used*".

What's interesting is that *all* developers seem to make this argument - even the ones who actually *ship the most code*! Perhaps it's because they don't want to reduce their perceived worth down to some trivial-sounding metric like lines-of-code.

There are obviously other things that go into being a highly productive software developer. Being able to help brainstorm ideas, design new features, fix bugs, improve performance, maintain code-quality, etc. are all valuable contributions, but these are *ancillary* job functions. The *primary* job function of a software developer is to *ship software*. In the words of Linus Torvalds: "*Talk is cheap, show me the code*."