CHAPTER 1

The Mythical 10x Programmer

Lutz Prechelt, Freie Universität Berlin, Germany

Are some programmers indeed ten times more productive than others, as some people claim? To a shocking degree, the answer depends on what exactly the question is intended to mean. In this chapter, we will work our way toward this insight by way of a fictious dialogue that is based on actual programming research data.

Alice: "I've heard the claim that 'Some programmers are ten times as productive as others.' Sounds a bit exaggerated to me. Do you happen to have data on this?"

Bob: "Indeed I do." (Bob is an evidence buff.)

Some Work Time Variability Data

Bob (pointing at Figure 1-1): "Look at this plot. Each circle shows the work time of one person for a particular small program, and each of the programs solves the same problem. The box indicates the 'inner half,' from the 25th percentile to the 75th percentile, leaving out the lower and upper fourth of the data points. The fat dot is the median (or a 50/50 split point), the *M* shows the mean and its standard error, and the whiskers extend from minimum to maximum."

C. Sadowski and T. Zimmermann (eds.), *Rethinking Productivity in Software Engineering*,
https://doi.org/10.1007/978-1-4842-4221-6_1

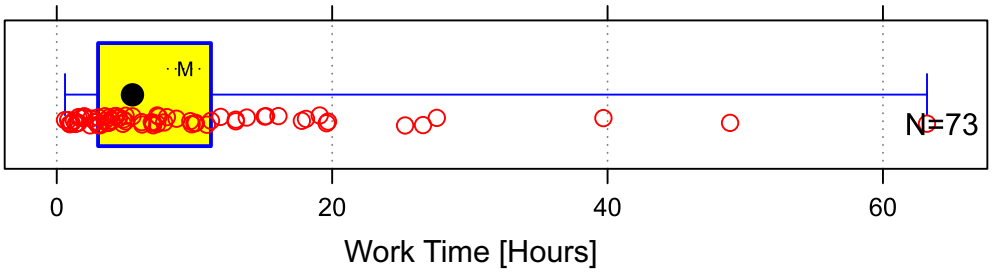

Figure 1-1. *Distribution of work times for 73 developers for the same small program*

Alice: "Wait. Not so fast. Are all these implementations working correctly?"

Bob: "23 of them have minor defects left in them; 50 work perfectly. All are more than 98 percent reliable and can be considered acceptable."

Alice: "I see. So min to max…that is how much?"

Bob: "Minimum is 0.6 hours; maximum is 63. That's a **105x** ratio."

Insisting on Homogeneity

Alice: "Wow, impressive. And are these data points indeed comparable?"

Bob: "What do you mean, comparable?"

Alice: "I don't know. Um, for instance…were these solutions all written in the same programming language? Maybe some languages are better suited to the problem than others. What type of problem is that anyway?"

Bob: "It's an algorithmic problem, a search-and-encode task. The data set mixes seven different languages, and some of those are indeed less suitable for the task than others."

Alice: "So, could we kick those out, please?"

Bob (showing Figure 1-2): "We can do even better because one of the seven groups provides 30 percent of the whole. This is what it looks like for only the Java solutions."

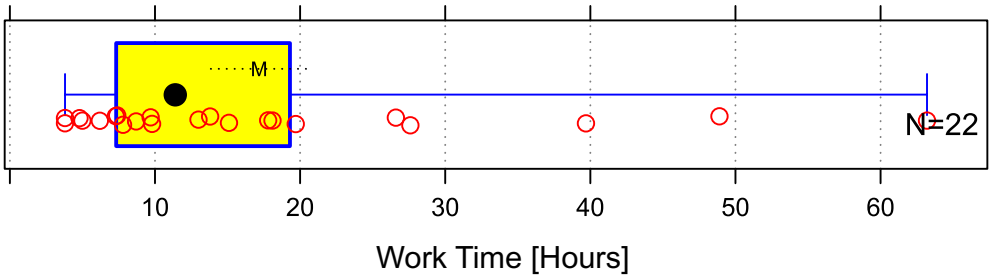

Figure 1-2. *Distribution of work times for 22 developers for the same small Java program*

Alice: "Uh-huh. Five of the six slowest are still there, but many of the fastest are not. So, that is still how much? 20x?"

Bob: "3.8 to 63, so it's **17x**."

Deciding What We Even Mean

Alice (shaking her head): "Okay, but I think I see the problem now. I said 'faster than *other* programmers,' but if those others are the worst possible ones, the difference can be *any* size because some people may need an arbitrarily long time."

Bob: "I agree. The experimenters for this data had expected this to be a half-day task for most people and a full day for the slower ones, but apparently the slowest ones instead came back every day for a week. Dogged folks!"

Alice: "So, I think what the statement really ought to mean is 'faster than *normal* programmers.'"

Bob: "And 'normal' is just the average? No, I don't agree with that definition. The comparison group then would include everybody and also those who are fast or even very fast. Would anybody expect to be 9x faster nevertheless?"

Alice: "Good point. So, then the statement should mean 'faster than ordinary-not-so-great programmers'?"

Bob: "Probably. And that means what?"

Alice: "Hmm, I suggest those are the slower half of all."

Bob: "Sounds fair to me. And how are they represented, by the slower-half mean or the slower-half median?"

Alice: "Median. Or else a single super-obstinate slow person taking 1,000 hours could still make it easy to be 10x as fast."

Bob: "Okay. The median of the slower half is the 75th percentile. That's simply the right edge of the box. That leaves 'some.'"

Alice: "Excuse me?"

Bob: "What do we mean by '*some* programmers?'"

Alice: "Ah, yes. There should be more than one."

Bob: "How about the top 2 percent?"

Alice: "No, that is almost irrelevant in practice. We need to have a few more of these people before it starts to matter that they exist. I'd say we take the top 10 percent. Programmers overall need to be pretty intelligent people, and to be among the top 10 percent of those is quite elite. Where does that get us?"

Bob: "The median of the top 10 percent is the 5th percentile. For the Java people, that comes out at 3.8 as well. And the 75th percentile is 19.3. That's a **5x** ratio."

Alice: "Ha! I knew it! 10x is just too much. On the other hand…"

Alice stares into the distance.

Uninsisting on Homogeneity

Bob: "What?"

Alice: "Who picked the programming language used?"

Bob: "Each programmer decided this for him or herself."

Alice: "Then the suitability of the language and all its effects should be part of the performance we consider. Insisting on a fixed language will artificially dampen the differences. Let's go back to the complete data. What's the ratio then?"

Bob: "The 5th percentile is 1; the 75th percentile is 11. An **11x** ratio."

Alice (shaking her head): "Gosh. Over ten again—a wild ride."

Questioning the Base Population

Alice: "So, maybe I was wrong after all. Although…who *were* these people?"

Bob: "Everybody essentially. It is a diverse mix from students to seasoned professionals, people with much language experience to little, scruffy ones and neat, and what-have-you. The only thing similar about them is their motivation to take part in the experiment."

Alice (looking hopeful): "So, can we make the set a little more homogeneous?"

Bob (grinning sardonically): "Based on what? Their productivity?"

Alice: "No, I mean…there must be *something*!"

Her face lightens up. "I bet there are freshmen and sophomores among the students?"

Bob: "No. All seniors or graduate students. Besides, many places in industry have some people with no formal computer science training at all!"

Alice: "So, you mean this is an adequate population to study our question?"

Bob: "Probably. At least it is unclear what a better one ought to look like."

Alice: "So 11x is the answer?"

Bob: "At least approximately, yes. What else?"

Alice thinks hard for a while.

It's Not Only About Development Effort

Alice: "Oops."

Bob: "Oops what?"

Alice: "We've overlooked a big part of the question. We've assumed development time is all there is to productivity because the resulting programs are all equivalent. But you said it was an algorithmic problem. What if the program is run often or with large data in a cloud computing scenario? Then the programs could have wildly different execution costs. High cost means the program is less valuable; that must be factored into the productivity."

Bob: "Good thinking."

Alice: "But I guess your data does not contain such information?"

Bob: "In fact it does. For each program there is a benchmark result stating run time and memory consumption."

Are Slower Programmers Just More Careful?

Alice: "Fantastic! I bet some of the slower programmers have spent time on producing faster and leaner programs, and once we factor that in, the productivity becomes more even. Can we please look at a scatterplot with work time on the x-axis and memory consumption multiplied by run time on the y-axis? Both those latter factors produce proportional execution cost increases in the cloud, so they ought to be multiplied."

Bob (showing Figure 1-3): "Here we are. Note the logarithmic axes. Some of those costs are extreme."

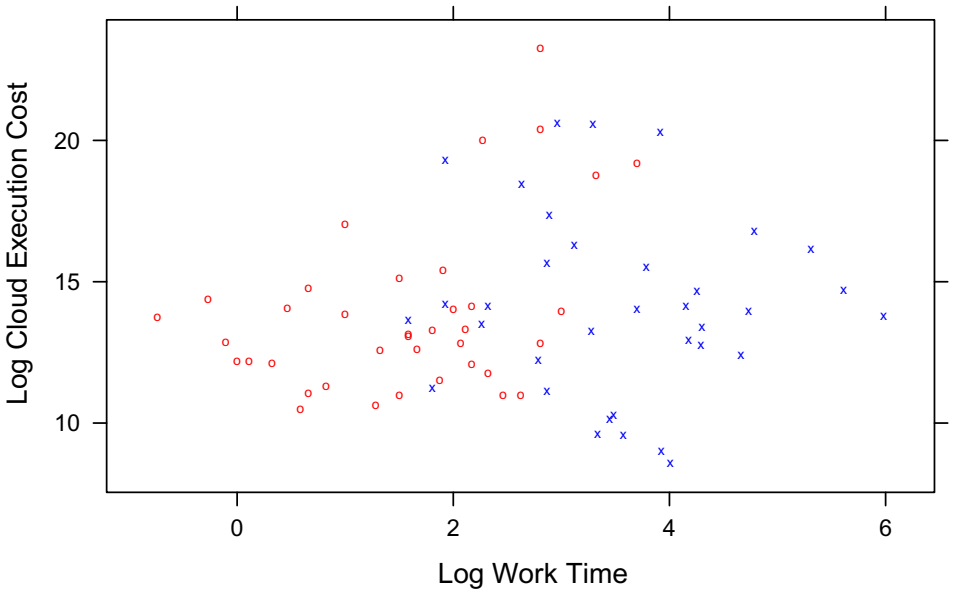

Figure 1-3. *Work time versus cloud execution cost (memory consumption times run time), log scale*

Alice: "Oh, there's hardly any correlation at all. I wouldn't have expected this."

Bob: "Do you still think the ratio will go down?"

Alice: "No, I guess not."

Secondary Factors Can Be Important

Alice: "By the way, what's the difference between the plot symbols?"

Bob: "The circles represent programs written in a dynamically typed scripting language; the Xs are statically typed programs."

Alice: "The scripts tend to be written much faster, so picking a scripting language was a clever move."

Bob: "Yes. That's because scripts get only half as long. This is what drove up the ratio compared to the Java-only group."

Alice: "Interesting. Yet scripts compete okay in terms of execution cost."

Bob: "Except against the very best nonscripts, yes."

The Productivity Definition Revisited

Alice: "But back to our question. Let's incorporate this execution cost idea: productivity is value per effort. Effort is our work time. Value goes down as cost goes up; so, value is the inverse of cost. Can you show that?"

Bob (showing Figure 1-4): "Sure. Here's the resulting plot."

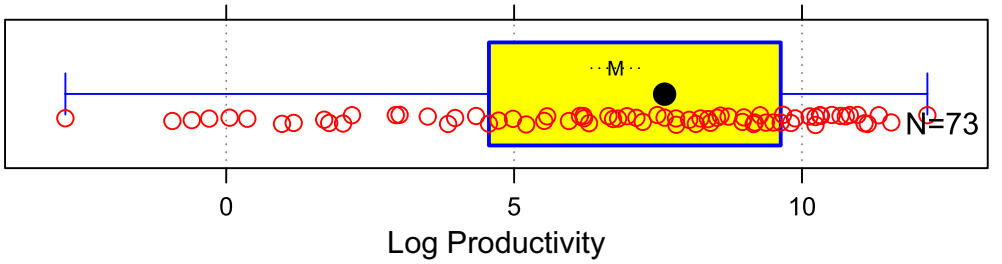

Figure 1-4. *"Productivity" for 73 developers for the same small program*

Bob: "It's hopeless without the logarithm and has a really strange unit of measurement, so it is difficult to make sense of intuitively. Larger is better now, so for our ratio we look at the 95th percentile, which is 2200, and the 25th percentile, the left box edge, which is 23.6, which makes the ratio **93x**. I guess you should get used to the fact that 10x differences exist."

How Would Real People Work?

Alice: "Perhaps. On the other hand, I now recognize that even with our refined understanding of what the question should mean we are asking the wrong question."

Bob: "Why is that?"

Alice: "I see two reasons. First, in a real scenario, one would not assign a task with cost implications as big as this one has to a developer from the lower half. Few people would be so shortsighted. Let's ignore the lower half."

Bob: "And instead of the 25th percentile of everybody take the 25th percentile of the upper productivity half?"

Alice: "Hmm, nobody can know that exactly in advance, but for simplicity's sake let's say yes."

Bob: "That would be the 62.5th percentile then. That's 385 and leads to a ratio of **6x**."

Alice: "Aaaaah, that sounds a lot more reasonable to me."

Bob: "I'm always happy to help."

Alice: "But that's not all. Second, if you build a solution with very high execution cost, you will go and optimize it. And if the original developer is not capable enough to do that properly, somebody else will come to the rescue. Or should at least. Productivity is about teams, really, not individuals!"

So What?

The next day, Bob runs into Alice in the kitchen.

Bob: "That was a really interesting discussion yesterday. But what is your take-home message from it?"

Alice: "My answer to the question of whether some programmers are indeed 10x more productive than others?"

Bob: "Yes."

Alice: "My answer is that is a misleading question. Other productivity facts are way more useful."

Bob: "And that would be which?"

Alice: "First, as the data showed, the low end of productivity can be *reeeeeally* low. So, do your best not to have such people on your team. Second, productivity is a lot about quality. There was not much information about this in your particular data set, but in the real world, I am strongly convinced that it makes little sense to talk about effort without talking about quality as well. Third, my personal conclusion is to assign critical tasks to the best engineers and noncritical tasks however they fit. Finally, although the data didn't have a lot to say about this, I firmly believe in improving a product over time. Productivity differences are a fact of life, but if you invest incrementally where it matters, they will not hurt very much."

The End.

Key Ideas

Here are the key ideas from this chapter in a nutshell:

- The low end of productivity can be really low.
- Quality matters, too, not only raw development speed.

- Assign critical tasks to your best engineers.
- Do your best not to have very weak engineers on your team at all.

References

The original study for the data used in this chapter is [1]. You can find a shorter report at [2] but will miss the add-on analyses. The data itself can be downloaded from [3].

[1] Lutz Prechelt. "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program." Technical Report 2000–5, 34 pages, Universität Karlsruhe, Fakultät für Informatik, March 2000. http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprtTR.pdf

[2] Lutz Prechelt. "An empirical comparison of seven programming languages." IEEE Computer 33(10):23–29, October 2000.

[3] Lutz Prechelt. http://page.mi.fu-berlin.de/prechelt/packages/jccpprtTR.csv