Characteristics and generative mechanisms of software development productivity distributions

Magne Jørgensen[1]

[1]Simula Metropolitan Center for Digital Engineering, Oslo, Norway
magnej@simula.no

Abstract

Context: *There is considerable variation in the productivity of software developers. Better knowledge about this variation may provide valuable inputs for the design of skill tests and recruitment processes.* **Objective**: *This paper aims to identify properties of software development productivity distributions and gain insight into mechanisms that potentially explain these productivity differences.* **Method**: *Four data sets that contain the results of software developers solving the same programming tasks were collected. The properties of the productivity distributions were analyzed, the fits of different types of distributions to the productivity data were compared, and potential generative mechanisms that would lead to the types of distributions with the best fit to the productivity data were evaluated.* **Results**: *The coefficient of variance of the productivity of the software developers was, on average, 0.55, with the top 50% of developers having average productivity that was 2.44 times higher than the bottom 50% of developers. All productivity samples were right-skewed, with an average skew of 1.79. About 30% of the observed productivity variance was explained by non-systematic, i.e., within-developer, variance. The distributions with the best fit to the empirical productivity data were the lognormal and power-law-with-an-exponential-cutoff distributions. The analysis of the mechanisms leading to productivity differences found no support for the "rich-getting-richer" explanation proposed for other disciplines. Instead, it suggests a constant productivity difference with increasing experience.* ***Conclusion***: *The substantial difference in productivity among software developers solving programming tasks indicates that a thorough evaluation of skill in the recruitment process can be rewarding. In particular, the long tail towards higher productivity values demonstrates the large gains that can be achieved by detecting and recruiting developers with very high productivity. More research is needed to understand the mechanisms leading to the large productivity differences.*

Keywords: software development, productivity, distributions, generative mechanisms

1. Introduction

Interest in the differences in productivity among software developers dates back to at least 1968 when Sackman, Erikson et al. (1968) investigated this topic. A reanalysis of the data from that study[1] suggests a ratio of, on average, 4:1 between the effort usage of the slowest and fastest developers when examining groups of 4 or 5 developers solving the same task in the same development environment. Similar results are reported by (Prechelt 1999, Figure 4) based on the aggregation of 61 programming performance experiments. That study reports a ratio of around 5:1 for the effort used on the same task of the median developers in the lower and upper productivity quartiles.[2]

Few studies have compared individual productivity differences in real-world software development contexts. A notable exception is the study reported in (Bryan 2012). That study analyzes task-solving data collected over 12 years from almost 200 programmers working on the same software application in the same company. In that company, the most productive 25% of the programmers completed 77% of the tasks, and the single top programmer completed as many as 8.3% of the tasks. Despite some potential analysis challenges in that study, e.g., not all developers worked on the system full time, the results support the claim that productivity differences among software developers can be substantial and that some developers may have extraordinarily high productivity.

The study reported in this paper aims to add to the knowledge about individual productivity differences and differs from previous studies on this topic in at least two ways. The first difference is that while previous studies focus on the differences in the *effort* usage of individual developers solving the same task, this study reports on the corresponding differences in *productivity*, where productivity is defined as the output divided by the effort used to create the output. While some measures of individual differences will not be affected by this difference, other measures will be affected, and the effort and productivity distributions will, in general, not have the same shapes. This enables a comparison of the results from this study with productivity results from other domains. The second difference is the study's aim to identify the *types of distributions that best fit* the empirical software development productivity data. To the author's knowledge, there is no prior research on that topic.

More knowledge about the shape and other characteristics of the productivity distribution of software developers may have practical consequences. In particular, it may guide the process of hiring software developers. If, for example, it is found that software productivity distributions tend to have a long tail toward higher productivity values, this may motivate a company to invest more in skill tests that aim at separating developers with very high productivity from those with just moderate to high productivity. Better knowledge about the shape of the productivity distribution may also provide information about the *generative mechanisms* of the productivity (Mitzenmacher 2004), i.e., information about the mechanisms that cause the productivity differences among software developers. This may have practical implications for, among other things, training and resource management in software companies.

The remainder of this paper is organized as follows: Section 2 formulates the research questions and describes the measures and methods used to analyze the differences in software development productivity. Section 3 uses these measures and methods to analyze four data sets containing individual developers' productivity data, solving 21 programming tasks. Section 4 discusses the results on productivity differences, distributional shapes, and the empirical support of different potential productivity-difference-generating mechanisms. It also includes a discussion of the findings' potential practical implications and the limitations of the analyses. Finally, Section 5 concludes the paper.

2. Research questions and analysis method

The two main research questions (RQs) addressed in the paper are the following:
RQ1: What are the essential characteristics of the software development productivity variance?
RQ2: What types of distributions, with connected generative mechanisms, best fit the empirical software development productivity data?

For analyses related to RQ1, a set of measures characterizing the empirical productivity variance are introduced in subsection 2.1. To support the analysis of RQ2, types of distributions that are candidates for giving the best fit to the observed variance in productivity and their generative mechanisms are selected and described in subsection 2.2. The method used to identify the type of distribution with the best fit to the empirical productivity data ("data pitting") is described in subsection 2.3.

2.1. Measures characterizing productivity variance

The measurement of the variance of software development productivity is not straightforward. As pointed out in (Prechelt 1999, Nichols 2019), it is not very informative to claim that there is a 10:1 difference between the most and least productive developers without further information about the context. This type of productivity difference can easily be made very high by recruiting from a population with a high productivity variance, assigning highly complex tasks, and evaluating many developers. Similarly, the difference can be reduced by using a recruitment process that mainly recruits medium-skilled developers, assigning simple programming tasks, and evaluating just a few developers. There is consequently a need for productivity measures that are more robust than those based on comparing the best and worst performances.

Productivity is typically defined as the ratio of the output to the input. In a software development context, this may be operationalized as the amount of software produced divided by the effort used to produce it. There has been substantial research on how to measure productivity, see for example (Petersen 2011) for an overview. The previously proposed productivity measures, such as function points divided by work effort (Symons 2019)), may be useful for indicating productivity. They are, however, severely hampered by a lack of agreement on how to define the amount of software output in a way that would enable a fair comparison of individual, team, or project differences in software development productivity.

Similar to prior studies using the differences in *effort usage* for the same task to indicate performance differences, the measurement of productivity in this paper is based on contexts in which the developers solve the same task. The output may then be said to be the same, and the effort used is what differs. Productivity in such contexts may be expressed as follows:

$$Productivity = \frac{1}{Effort}$$

This expression is simply the consequence of setting the output size to unity (1) and using the standard assumption that productivity is the ratio of the output to the input, where the input is defined as the effort used. The choice of setting the output to unity is arbitrary, and the output could be set to any constant without changing any of the results of the productivity analyses in this paper.

A potential limitation of this productivity measure, and of prior studies that used the effort on the same task as a performance measure, is that the solutions provided by different developers may differ in terms of essential quality characteristics, e.g., maintainability and robustness. In this study, this is addressed by requiring that all solutions have an acceptable quality, which in practice means that the solutions must be properly tested and returned to the developer for further improvement if they are not submitted with all the required functionalities and acceptable technical quality.

This paper uses the following measures of the productivity *difference* among software developers solving the same task:

$$\text{CoV} = \text{coefficient of variance of productivity} = \frac{\text{standard deviation of productivity}}{\text{mean productivity}}$$

$$\text{Gini}^{3} = \text{Gini coefficient} = \frac{\sum\sum|\text{productivity}_i - \text{productivity}_j|}{2n^2 \cdot \text{mean productivity}}\text{, where i and j are the indexes of n developers}$$

$$\text{PXPY} = \text{productivity ratio of the PX and PY percentiles} = \frac{\text{PX percentile of the productivity distribution}}{\text{PY percentile of the productivity distribution}}$$

$$\text{RelX\%}^{4} = \text{effort- adjusted relative production of the X\% most productive} = \frac{\overline{\text{effort}}_{1-\text{X\% least productive}}}{\overline{\text{effort}}_{\text{X\% most productive}}}$$

$$\text{Skew} = \frac{n}{(n-1)(n-2)}\sum\left(\frac{\text{productivity}_i - \text{mean productivity}}{\text{standard deviation of productivity}}\right)^3\text{, where i is the index of n developers}$$

The coefficient of variance (CoV) is a normalized measure of the standard deviation. It is commonly used to characterize productivity and work-effort distributions, see for example, (Anda, Sjøberg et al. 2008). This measure is strongly affected by large deviations from the mean productivity, which motivates the inclusion of the Gini coefficient (Gini). This measure has a defined minimum (0) and maximum (1) and may be interpreted as a measure of (half of) the *mean absolute difference in productivity* normalized by the mean productivity. The Gini coefficient has not been used much for the analysis of productivity differences, but see, for example, (Halffman and Leydesdorff 2010) for an exception. Its focus on assessing inequalities, and its higher robustness towards outliers, may be useful in the contexts analyzed in this paper. An additional potential advantage of using the Gini coefficient is that it enables a comparison of the differences in productivity with other differences or inequalities. For example, differences in productivity can be compared with differences in the distribution of wealth within a country. For example, the Gini coefficient of the wealth distribution in Scandinavian countries is 0.3, in the US it is 0.4, and in Brazil it is 0.5.[5] The interpretation of the size of the productivity differences may consequently be supported by comparing the observed Gini coefficients and the distributions of wealth in countries with similar Gini coefficients for wealth.

CoV and Gini are measures of the *general* spread of a productivity distribution and are not designed to describe the tail properties of distributions. For that purpose, as proposed by for example (Berlingieri, Blanchenay et al. 2017), a comparison of percentiles may be useful. The measure PXPY is a ratio-based measure that compares percentiles. The two variants of this measure used for the analyses in this paper are P90P10 and P80P20. The first gives the ratio of the P90 percentile to the P10 percentile, and the second is the ratio of the P80 and P20 percentiles.

The measure RelX% is included to compare the effort-adjusted total production of the top X% of developers with that of the others.[6] As before, the analysis is based on the assumption that all developers produce the same amount of output. The effort-adjusted production of the top 20% of developers in terms of productivity (Rel20%) is then, relative to the remaining 80% of developers, $\frac{0.2/0.8}{0.05/0.95} = \frac{0.25}{0.053} = 4.75$, i.e., the top 20% of developers in terms of productivity produce, per effort unit, 4.75 times more than the remaining 80%.[7] Two measures of this type were implemented. Rel20% and Rel50% compare, respectively, the effort-adjusted total production of the top 20% and top 50% of developers in terms of productivity with that of the remaining 80% and 50% of developers.

The measure of the sample skewness is the adjusted Fisher-Pearson coefficient of skewness. If the productivity data are symmetric, the skewness will be near zero. A positive skew value means that the right tail of the distribution is long relative to the left tail, while a negative skew value means the opposite. The skew may be useful for, among other things, identifying whether one can expect more, less, or a similar proportion of developers with very high productivity compared to the proportion of developers with very low productivity. As a supporting element for the analysis of the distributional skew, the productivity distributions are visualized as histograms.

2.2. Distributional shapes and generative mechanisms

As pointed out in "The common patterns of nature" (Frank 2009), a small set of probability distributions seem to be able to describe the majority of the distributions observed in nature. It is argued in (Joo, Aguinis et al. 2017) that the following distributions are likely to cover most productivity observations: i) the power-law distribution (Power law), ii) lognormal distribution (Lognormal), iii) exponential distribution (Exp), iv) power-law-with-an-exponential-cutoff distribution (ExpCut), iv) normal distribution (Normal), and v) Weibull distribution (Weibull).

For the analyses of generative mechanisms, the introduced set of six distributions is divided into four types based on their tail properties. The distribution types analyzed, listed according to the decreasing potential thickness of the upper tail, are I) power-law-function distributions, II) lognormal distributions, III) distributions with an exponential tail, and IV) distributions that are or potentially can be symmetric. This division into distribution types is similar to that used for productivity analyses in other work domains, see for example (Joo, Aguinis et al. 2017).

Below, each distribution type's central characteristics and potential generative mechanisms are outlined. Note that even if a described mechanism is *able to* generate a certain type of distribution and that distribution is observed, it is not certain that this mechanism actually generated the distribution, i.e., there may be other mechanisms or combinations of mechanisms that also lead to the same distribution type. The primary use of the analysis of generative mechanisms is mainly to make some generative mechanisms more or less plausible.

I) Power law distributions: "The rich get much richer" and "self-organized criticality"

Power law distributions can be written as $p(x) \propto x^{-\alpha}$, where α determines the thickness of the upper tail. This distribution type has been shown to provide a good fit to, for example, the empirical distributions of wealth, the citations of academic papers, the sizes of nations by population, and the frequency of occurrence of unique words in novels (Andriani and McKelvey 2009, Clauset, Shalizi et al. 2009). A power law distribution has also been shown to be a good fit for software project cost overruns, see (Budzier and Flyvbjerg 2013). In the context of productivity, the power law distribution has been associated with the presence, and even dominance in terms of the output, of star performers (Aguinis and O'Boyle Jr 2014), i.e., the presence of workers who contribute a disproportionately large amount of the output production. An example consistent with a power law distribution of productivity is when the productivity distribution follows the Pareto principle (the connected Pareto distribution is a type of power-law distribution), i.e., when the top 20% of workers in terms of productivity deliver 80% of the output and hence are on average 16 times more productive than the other workers. The study presented in (Jørgensen 2015) indicates the existence of the exceptional productivity of some software developers. In that study, one developer completed a web development project in less than a week and delivered a product of good quality. In contrast, other developers spent several weeks and even months on the same project. See also (Bryan 2012) for documentation of the presence of star performers in software development.

Mechanisms that generate a power law distribution, see for example (Andriani and McKelvey 2009, Joo, Aguinis et al. 2017), and candidates for explaining the differences in software development productivity include the following two:

"The rich get much richer": If those who already have a productivity advantage ("the rich") tend to further strongly increase this advantage ("get much richer"), a power-law distribution of productivity may be observed. This may for example happen if the developers who already have high productivity are given much better opportunities to learn and further increase their productivity compared to other developers. This may also happen when different factors, such as skill and motivation, have a very strong *synergy* effect. The organizational advantage or synergy effect must be stronger than a multiplicative or proportional effect to create a power-law distribution.

"Self-organized criticality (event bursts)": The mechanism of self-organized criticality was borrowed from physics and has been used to understand several phenomena, such as landslides (Hergarten and Neugebauer 1998), activities in the brain (Hesse and Gross 2014),

and human behavior (Ramos, Sassi et al. 2011). In the context of productivity differences among individuals, this mechanism may be thought of as a system in which a few individuals experience unpredictable and large productivity bursts (large productivity increases), and these productivity bursts require a certain state (the critical state) to be achieved. Observations of developers with much higher productivity than other developers may be examples of productivity bursts, made possible because these developers already had high productivity (had reached a critical state) and somehow managed to find a very smart solution for the required work, which was not predictable from the developers' previous performance.[8]

II) Lognormal distributions: "The rich get proportionally richer" and "multiplicative aggregation"

A lognormal distribution can be written as $p(x) \propto e^{(-\frac{(\ln(x)-\mu)^2}{2\sigma^2})}$, where σ determines the thickness of the upper tail. μ is the mean and σ is the standard deviation of the corresponding normal distribution. This distribution can provide a good fit to, for example, the lengths of chess games, stock prices, and city sizes.[9] A connection between generative processes and the lognormal distribution can be found by applying the *central limit theorem* to the logarithms of values. If the generative process is based on *multiplying* independent variables with positive values (for example, productivity factors), the distribution of these aggregated values will approach a lognormal distribution.[10] The suggestion that lognormal distributions tend to be a good fit for the software development effort distribution, see for example (Halkjelsvik and Jørgensen 2018), suggests that the productivity distribution may be lognormal as well. This is a consequence of the fact that productivity is defined as the inverse of the effort for a particular amount of output and that the inverse of a lognormal distribution is also lognormal.[11]

Generative mechanisms that potentially lead to lognormal distributions include the following two:

"The rich get proportionally richer": If all software developers experience the same rate (proportion) of productivity improvement, then initially small differences in productivity will increase in terms of their absolute values, i.e., the most productive will increase their productivity more than the less productive. If having higher productivity provides an advantage in terms of further improvement of productivity, and if this advantage is proportional to the productivity already achieved, a lognormal distribution of productivity will be observed.

"Multiplicative aggregation": Many factors potentially impact how productive software developers are and will become in the future. If the aggregated effect of these factors is dominantly multiplicative, a lognormal distribution of productivity will be observed. This may, for example, mean that the aggregated effect of skill and motivation is higher than if these factors were aggregated in an additive manner. The multiplicative aggregation of factors may, for example, occur with feedback loops, as shown by (Guerrero and Garcia-Baños 2020).

As pointed out in (Mitzenmacher 2004), minor deviations from a lognormal generative process may produce a power-law distribution, and it may sometimes be challenging to separate these distributions.

III) Distributions with an exponential tail: "The rich get richer, but with diminishing returns" and "incremental differentiation effects"

Two types of distributions with similar generative mechanisms and similar tail properties are addressed: i) the pure exponential distribution $p(x) \propto e^{-\lambda x}$, where λ determines the thickness of the tail[12], and ii) the power-law-with-an-exponential-cutoff distribution (truncated power-law distribution) $f(x) \propto e^{-\lambda x} x^{-\alpha}$, which is the product of a power law and an exponential distribution; the exponential part dominates for high x-values.[13] These two distributions are right-skewed, and they have a thinner upper tail than the power-law and lognormal distributions and a thicker upper tail than the normal and Weibull distributions. Distributions with an exponential tail have been observed in several studies on productivity; for example, they were observed in the study of team productivity performed by (Bradley 2017). They have also been studied in other professional work domains (Joo, Aguinis et al. 2017).

Possible generative mechanisms for this type of distribution include the following two:

"*The rich get richer, but with diminishing returns*": If the advantage of already having high productivity is not as large as in a situation leading to the power-law or lognormal distributions, but larger than for an additive aggregation of productivity factors, a distribution with an exponential tail may be observed. A central characteristic of this mechanism is that there are diminishing returns (as measured using the percentage increase) from having higher productivity. This may, for example, be related to organizational limitations that make it hard to improve productivity above a certain level. Formally, if the increase in productivity is a *linear* function of the already achieved (accumulated) productivity, an exponential-tail type of distribution may be observed (Joo, Aguinis et al. 2017).

"*Incremental differentiation*": If the factors connected to productivity are aggregated linearly, but those who score high on one factor obtain a higher linear addition (higher aggregated productivity), an exponential distribution may be observed (Joo, Aguinis et al. 2017). If this mechanism is mainly valid among those with higher productivity, the upper tail will be exponential, but not necessarily the whole distribution.

IV) Symmetric distributions: "Homogenization" and "additive effects"

Two distributions are addressed here: i) the normal distribution $f(x) \propto e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$ and ii) the Weibull distribution $f(x) \propto (\frac{x}{\lambda})^{\beta-1} e^{-(\frac{x}{\lambda})^{\beta}}$. The tails of both distributions are potentially thinner than those of the previous distributions. The normal distribution is always symmetric, while the Weibull distribution can be left-skewed, symmetric, or right-skewed. The normal distribution may, according to the central limit theorem, result from the addition of many independent random variables. The additive effects of the genes determining human height lead, for example, to a close-to-normal height distribution (Weedon, Lango et al. 2008). The Weibull distribution is more flexible (has more parameters) than the normal distribution and covers a variety of thin-tailed distributions.

The generative mechanisms that may lead to a symmetric or thin-tail type of distribution include the following two:

"*Homogenization*": If there are "ceilings and floors," i.e., mechanisms or actions that put restrictions on higher or lower productivity, thin-tailed distributions such as the normal and Weibull distributions may be observed (Aguinis, Ji et al. 2018).

"*Additive aggregation*": If the factors connected to productivity are additive and independent, a normal productivity distribution will, in accordance with the central limit theorem, be observed. Even if the factors are not independent, symmetric distributions may be observed. Given the additive aggregation, the tails will tend to be thinner than those of the previous types of distributions.

2.3. Analyses of distributional fit ("data pitting")

To identify the type of distribution with the best fit to the empirical productivity data, the method of data pitting, introduced in (Joo, Aguinis et al. 2017), is used. The data pitting method consists of three decision rules. The *first decision rule* is based on the results of pairwise statistical tests of the differences in the distributions' fits to the empirical productivity data. For this purpose, the R package *Dpit*[14] was used. *Dpit* gives the pairwise log-likelihood ratio and the statistical significance of observing the measured difference (or a difference larger than the measured difference) in the distributional fit to the empirical productivity data. A p-value of 0.1 is used as a threshold that indicates that one distribution has a statistically significant better fit to an empirical distribution than another. If one distribution type has a log-likelihood ratio statistically significantly better than those of *all* other types of distributions, using pairwise testing, this distribution type is selected as the distribution type with the best fit. Otherwise, it is moved on to the next two decision rules.

The second and the third decision rules are based on the principle of parsimony (Occam's razor), i.e., one should select the simplest model that explains a phenomenon similarly well. The *second decision rule* says that nested distributions are less parsimonious than the non-nested distributions they include. This study includes two nested distributions of this type, i.e., the power-law-with-an-exponential-cutoff (the power-law distribution and the exponential distribution are included) and Weibull (the exponential and the normal distributions are included) distributions. If it is still impossible to decide which distribution has the best fit after using the second decision rule, it is moved on to the *third decision rule*. This decision rule says that distributions with more flexibility in shape are less parsimonious than less-flexible distributions. Based partly on the number of parameters and partly on the variance in shapes enabled by the distribution type, it is considered that the lognormal and Weibull distribution types are more flexible than the power-law, exponential, power-law-with-an-exponential-cutoff, and normal distribution types. For example, the lognormal and Weibull distributions include distributional shapes close to the normal distribution, while the opposite is not necessarily true.

As can be seen, the above decision rules do not guarantee the selection of a distribution type with the best fit. The conclusion may be that it is impossible to determine the distribution with the best fit to the empirical productivity data based on the "data pitting" method.

3. Data sets and results

This section presents the productivity data sets (3.1), analyzes the characteristics and distributional shapes of the productivity data (3.2), and discusses the potential generative mechanisms leading to the productivity distributions (3.3).

3.1. The data sets

Table 1 describes the properties of the four data sets used in the analyses. All data sets are from contexts where software developers have individually solved the same task. All data sets, except data set C, are from contexts with quality criteria and system testing in place to ensure that the solutions had an acceptable quality, i.e., submitting solutions with an unacceptable quality would lead to the need for the correction and re-submission of the solution. For data set C, submitted solutions with an unacceptable quality were removed from the analysis. All data sets were collected for other research purposes and are part of other research papers, but they have never been used for the research purpose of this paper.

Table 1 Productivity data sets

ID	*Original study*	*Sample*	*Recruitment process*	*Task type and size*	*Original study purpose*
A	(Jørgensen and Gruschke 2009)	20 developers solving the same 5 tasks.	Hired experienced, professional software developers from several Norwegian companies. Median length of experience of 6 years. Hourly paid consultancy work.	Changes to an existing web system written in Java. Average effort spent per task was 4 to 12 hours.	Improvement of effort uncertainty assessments.
B	(Nichols 2019)	494 developers	Developers, mainly software professionals, participating in a	Data manipulation and other types of	Training to improve

		solving the same 10 tasks.	course on the Personal Software Process (PSP) in the US. Median length of experience of 4 years. The analysis includes those who completed all ten tasks and used the C programming language.	tasks using the C programming language. Average effort spent per task was 3 to 7 hours.	software development work.
C	(Jørgensen, Bergersen et al. 2020)	100 developers solving the same 5 tasks.	Hired professional software developers from two medium-large Eastern Europe offshoring companies. Median length of experience of 6 years. Only tasks completed with an acceptable quality are included in the analysis. Hourly paid work.	Updates on existing Java programs and other types of Java programming tasks. Average effort spent per task was 8 to 40 minutes.	Identifying indicators of programming skill.
D	(Jørgensen and Grov 2021)	16 developers working on the same small project.	Hired professional software developers working for offshoring companies in Europe and Asia. All hired developers had high satisfaction scores from previous clients and satisfactory performance on a trial task organized by us. Median length of experience of 7 years.	One small web-development project to be programmed using Java. Average effort spent on the project was 41 hours.	Use of work-sampling to identify development skills.

3.2. The productivity distributions

This subsection presents the variance in the productivity for the data sets and tasks (3.2.1), an analysis of the systematic and unsystematic productivity variance, i.e., the between-developer variance as opposed to the within-developer variance (3.2.2), and the types of distributions that gives the best fit to the empirical productivity data (3.2.3).

3.2.1. Characteristics of the productivity distributions

Table 2 presents the characteristics of the productivity distributions according to the measures introduced in Section 2. Figure 1 visualizes the productivity distributions as histograms.

Table 2 Characteristics of the empirical productivity distributions

Data sets	*CoV*	*Gini*	*P90P10*	*P80P20*	*Rel20%*	*Rel50%*	*Skew*
A: Task 1 (n = 20)	0.43	0.22	2.85	1.92	2.26	1.83	1.04
A: Task 2 (n = 20)	0.40	0.22	2.83	2.05	2.12	2.05	0.26
A: Task 3 (n = 20)	0.40	0.22	2.77	2.03	2.07	2.09	0.42
A: Task 4 (n = 20)	0.53	0.27	3.52	2.50	2.54	2.15	1.20
A: Task 5 (n = 20)	0.49	0.26	3.25	2.41	2.49	2.31	0.61
B: Task 1 (n = 494)	0.84	0.41	6.45	3.44	4.35	3.18	2.17
B: Task 2 (n = 494)	0.72	0.33	4.36	2.56	3.03	2.56	3.24
B: Task 3 (n = 494)	0.63	0.32	4.33	2.68	3.10	2.72	1.96
B: Task 4 (n = 494)	0.59	0.30	4.12	2.52	2.74	2.42	2.17
B: Task 5 (n = 494)	0.61	0.31	4.36	2.53	2.84	2.45	2.12
B: Task 6 (n = 494)	0.58	0.30	4.00	2.57	2.81	2.42	1.47
B: Task 7 (n = 494)	0.53	0.28	3.76	2.39	2.60	2.30	1.42
B: Task 8 (n = 494)	0.70	0.36	5.71	3.41	3.68	2.96	1.55
B: Task 9 (n = 494)	0.51	0.27	3.48	2.41	2.53	2.33	1.24
B: Task 10 (n = 494)	0.55	0.29	4.08	2.57	2.66	2.47	1.57
C: Task 1 (n = 96 of 100)[i]	0.56	0.26	2.69	2.00	2.25	1.92	2.47
C: Task 2 (n = 66 of 100)	0.41	0.22	2.66	2.10	2.09	1.87	1.06

C: Task 3 (n = 74 of 100)	0.73	0.26	2.79	1.99	2.09	1.90	5.91
C: Task 4 (n = 43 of 100)	0.42	0.22	2.70	1.92	2.04	1.80	1.41
C: Task 5 (n = 15 of 100)	0.15	0.07	1.26	1.13	1.80	1.18	2.39
D: Project (n = 16)	0.71	0.34	3.84	3.16	3.43	2.82	1.92
Mean	**0.55**	**0.27**	**3.98**	**2.57**	**2.83**	**2.44**	**1.79**

i: There were 100 developers participating in study C. Only the developers who provided acceptable-quality solutions are included.

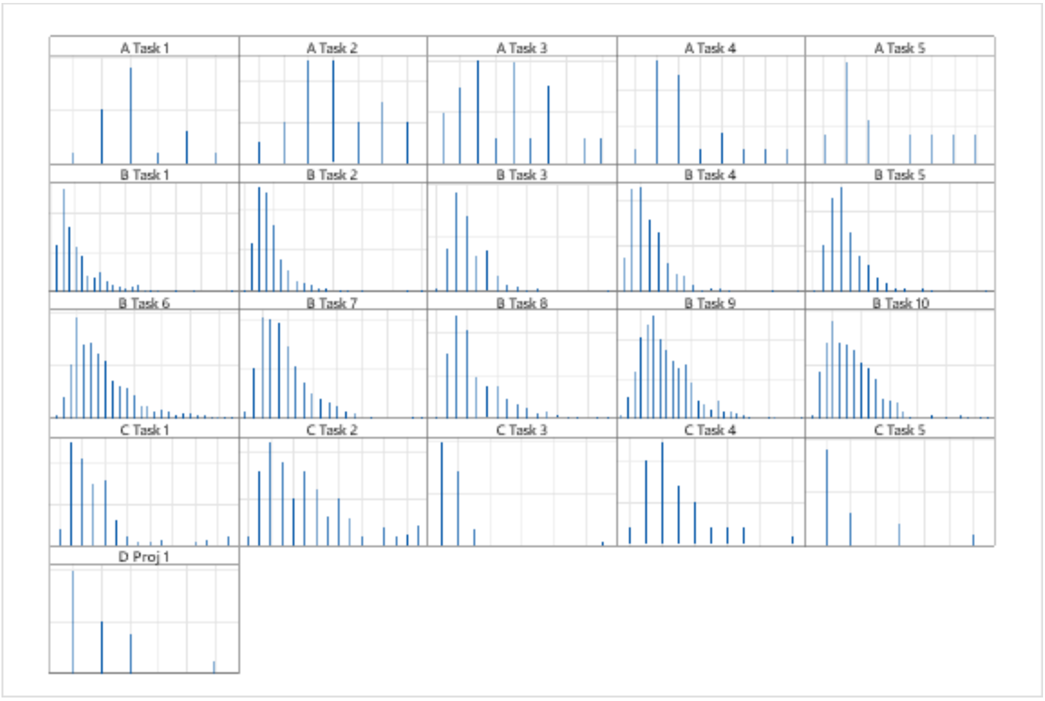

Figure 1 Histograms of the empirical productivity distributions

Table 2 indicates that there is a substantial difference in programming productivity. The mean coefficient of variance is 0.55, the mean *Gini* coefficient is 0.27[15], the mean productivity of a programmer in the 90th percentile is 3.98 times higher than that of a programmer in the 10th percentile, the mean productivity of a programmer in the 80th percentile is 2.57 times higher than that of a programmer in the 20th percentile, the mean productivity of the top 20% of developers in terms of productivity is 2.83 times higher than the that of the remaining 80% (which implies that the top 20% of developers would produce around 40% of the output with the same effort), and the mean productivity of the top 50% of developers is 2.44 times higher than that of the remaining 50% (which implies that the top 50% of developers in terms of productivity would produce 70% of the output with the same effort). It is clear from the skew and the histograms in Figure 1 that all the productivity distributions are substantially skewed towards higher values (right-skewed), with an average skew value of 1.79. This suggests that one will find more software developers with very high than very low productivity. It also means that a symmetric type of productivity distribution, particularly the normal distribution, is less plausible.

The productivity variance differed between the data sets and within the same data set. Possible contextual influences on productivity differences include the following:

The selection of only highly skilled developers, as is the case for data set A, may decrease the productivity variance. Similarly, excluding those with non-satisfactory solutions, which is connected with a lower skill level, results in a lower variance in productivity. For example, although Task 5 of data set C was more difficult than the other tasks (only 15% of the 100 developers managed to complete it with a satisfactory quality), those who completed it had a lower productivity variance.

Higher task complexity is connected with higher productivity variance. For example, although the software developers of data set D were carefully selected based on their previous client experience and type of experience, the productivity spread was among the largest out of all the data sets.[16]

The variance in the productivity in the different data sets may consequently have been affected by both the developer recruitment process and the task complexity. This is similar to what is reported for other domains, see for example (Schmidt, Hunter et al. 1979, Buzacott 2002).

3.2.2. Between- and within-variance in productivity

As pointed out in (Nichols 2019, Jørgensen and Escott 2022), not all of the performance variance among software developers is systematic, i.e., some of the productivity differences are non-repeatable and due to non-systematic variance (randomness) in how well a developer performs. To examine the size of the non-systematic variance in productivity, a linear mixed model on the productivity data sets was created. The logarithm of the actual productivity was used as the response variable, the developer identification number as the random variable, and the estimation task as the fixed effect. This was only done on data sets A[17] and B, since the other data sets had only one task (data set D) or a varying number of developers completing the tasks (data set C). The logarithm of the productivity was used because the productivity distributions were strongly right-skewed and likely to result in strongly non-normal residuals. The variance estimation was based on the restricted maximum likelihood, and the tests of fixed effects used the Kenward-Roger degrees of freedom approximation. The linear mixed model produced close to normally distributed conditional residuals, suggesting a reasonably good model. The residual variance of the random effects of the linear mixed model is the productivity variance that is *not* explained by the differences between the developers or by task differences. This residual variance may consequently be interpreted as the within-developer (unsystematic) variance, assuming a reasonably good model fit. The results of the mixed model for data sets A and B are displayed in Tables 3 and 4.

Table 3 Mixed model parameters and results for data set A

Dependent variable	*ln(productivity), where productivity is the actual productivity on a task*			
Random effects	**Variable**	**Variance**	**Percentage of the total variance**	**Test**
	Developer ID (1..20)	0.13	64% [Between-developer variance]	Z-value = 2.77, *p*-value = 0.003
	Residual	0.07	36% [Within-developer variance]	Z-value = 6.16, *p*-value < 0.001
Fixed effect	**Variable**	**Tests**		
	Task (1..5)	F-value = 36.74, *p*-value < 0.001		

Table 4 Mixed model parameters and results for data set B

Dependent variable	*ln(productivity), where productivity is the actual productivity on a task*			
Random effects	**Variable**	**Variance**	**Percentage of the total variance**	**Test**
	Developer ID (1..494)	0.22	63% [Between-developer variance]	Z-value = 14.8, *p*-value < 0.001

	Residual	0.13	37% [Within-developer variance]	Z-value = 47.1, *p*-value < 0.001
Fixed effect	**Variable**	**Tests**		
	Task (1..10)	F-value = 415.0, *p*-value < 0.001		

Tables 3 and 4 show that a substantial part (36% in data set A and 37% in data set B) of the observed difference in productivity is not between-developer variance but rather a within-developer variance. The observed variance in productivity is consequently not a good measure of the true (inherent) difference in the productivity (skill difference) between the developers. To report the true between-developer variance in productivity, one needs, for example, to reduce the coefficient of variance (CoV) by the measured within-developer variance.[18] Assuming a within-developer variance of 36% and a CoV of 0.55 (the mean of all CoVs in the data sets), one consequently needs to reduce the CoV to 0.35 ($0.55 \cdot (1 - 0.36)$) to find the between-developer variance in the productivity. This reduction in variance from the observed productivity variance to the between-developer variance is important to keep in mind when interpreting the values in Table 2, i.e., about one-third of the variance is due not to differences in skill but to the unsystematic (random) variance in productivity.

3.2.3. Distributional fit

The data suggests that the distribution of productivity is right-skewed. While this observation may exclude the normal distribution, it does not tell us what type of distribution will best fit the empirical data. For this purpose, the data pitting method was applied to the types of distributions introduced in Section 2. The results from the use of the data pitting method are presented in Table 5. As described in Section 2, the first decision rule requires a distribution to provide a statistically significant better fit ($p < 0.1$) than all the other distributions in pairwise tests to be selected as the best distribution type. If two or more distributions are statistically significantly better than the others but not statistically significantly better than each other, the second decision rule (non-nested distributions are preferred over nested distributions) is used to choose between them. If no selection is possible using that rule, the third decision rule (less-flexible distributions are preferred over more-flexible distributions) is used.

Table 5 Distribution pitting

Sample	*Distributions with best fit*[i,ii]	*After first decision rule*	*After second decision rule*	*After third decision rule*
A: Task 1 (n = 20)	Cut, Log, Wei	Not determined	Not determined	Cut
A: Task 2 (n = 20)	Cut, Log, Wei	Not determined	Not determined	Cut
A: Task 3 (n = 20)	Cut, Log, Wei	Not determined	Not determined	Cut
A: Task 4 (n = 20)	Cut, Log, Wei	Not determined	Not determined	Cut
A: Task 5 (n = 20)	Cut, Log, Wei	Not determined	Not determined	Cut
B: Task 1 (n = 494)	Log, Cut, Wei	Log	-	-
B: Task 2 (n = 494)	Log, Cut, Wei	Log	-	-
B: Task 3 (n = 494)	Log, Cut, Wei	Not determined	Not determined	Cut
B: Task 4 (n = 494)	Log, Cut, Wei	Log	-	-
B: Task 5 (n=494)	Log, Cut, Wei	Log	-	-
B: Task 6 (n = 494)	Log, Cut, Wei	Log	-	-
B: Task 7 (n = 494)	Log, Cut, Wei	Log	-	-
B: Task 8 (n = 494)	Log, Cut, Wei	Log	-	-
B: Task 9 (n = 494)	Log, Cut, Wei	Log	-	-
B: Task 10 (n = 494)	Log, Cut, Wei	Log	-	-
C: Task 1 (n = 96 of 100)	Log, Cut, Wei	Log	-	-
C: Task 2 (n = 66 of 100)	Cut, Log, Wei	Cut	-	-
C: Task 3 (n = 74 of 100)	Cut, Log, Exp	Cut	-	-
C: Task 4 (n = 43 of 100)	Cut, Log, Wei	Cut	-	-

C: Task 5 (n = 15 of 100)	PL, Cut, Norm	Not determined	Not determined	PL
D: Project (n = 16)	Cut, Log, Wei	Cut	-	-

i: PL = power law, Log = lognormal, Exp = exponential, Cut = power law with an exponential cutoff, Norm = normal, Wei = Weibull.
ii: The selection and ranking, represented by the sequence, are based on the number of times a distribution gave a better fit (based on the log-likelihood ratio) in the pairwise comparisons.

Table 5 shows that the distributions with the best fit to the empirical productivity distributions are the lognormal distribution (Log) for ten tasks and the power-law function with an exponential cutoff (Cut) for ten tasks. Only for Task 5 of data set C (which only 15% of the developers completed) did another distribution, the power-law distribution, give the best fit. The results in Table 3 also show that the type of distribution with the best fit is the same for all, or nearly all, tasks within the same data set. This indicates that the recruitment process of the developers, which varies between but not within the data sets, is a central factor in the determination of the shape of the productivity distribution.

Complementing the analysis with a visual investigation of the empirical distributions in Figure 1 suggests that the distributions have essential similarities in their shapes. All distributions have a long tail towards higher productivity values, and none are symmetric or short-tailed. On the other hand, the tail is not as long and thick as one might expect if productivity follows a power law distribution. Consequently, the skewness and thickness of the upper tail seem to be in between, but not including, a pure power law distribution and a normal distribution.

3.3. Generative mechanisms

In subsection 3.2, the lognormal and the power-law-function-with-an-exponential-cutoff distributions gave the best fit to the observed productivity distributions. As pointed out in subsection 2.2, these distributions may be generated by aggregation mechanisms that are stronger than additive aggregation and weaker than the power-law type of aggregation. The generative mechanisms that were identified for the lognormal and power-law-with-an-exponential-cutoff types of distributions were called "multiplicative aggregation" and "incremental differentiation.", with the following two types of general underlying potential reasons for the generative mechanisms identified:

The rich get richer: Productivity differences are caused by workers with higher productivity receiving advantages compared to those with lower productivity. This leads to a system in which the differences in productivity increase over the years. Several studies claim this generative mechanism to explain observed productivity differences, see for example (Allison and Stewart 1974, Aguinis, O'Boyle Jr et al. 2016, Joo, Aguinis et al. 2017, Ruocco, Daraio et al. 2017).

Productivity factors: Productivity differences are mainly caused by a combination of contextual and personal characteristics, such as task-solving skills and motivation. This explanation is supported by findings reported by, for example, (Shockley 1957, Huber 2001, Fletcher, Baines et al. 2008, Pluchino, Biondo et al. 2018, Van Iddekinge, Aguinis et al. 2018).

An essential difference between these two explanations is that the first explanation predicts that the productivity differences will increase over time, i.e., there will be larger differences in productivity among those with more experience compared to those with less experience. The remaining part of this subsection analyzes and discusses which of the above two explanations can better explain the variance in the productivity of software developers by examining to what extent the differences in productivity increase with more experience.

For this purpose, the programmers were divided into four equally sized categories based on their length of experience as programmers. These four categories (quartiles) are termed Q1, Q2, Q3, and Q4. Then, the variance in the productivity within each of these quartiles using the previously introduced measures was analyzed. Here, an aggregated productivity measure across all tasks per developer is used, i.e., the total productivity is defined as 1/(the total effort of the tasks within the data set). Only data sets A and B are used since they were the only data sets with enough developers to enable a meaningful division into and analysis of four groups. In data set B, there were 41 developers (out of the total of 494) with an unknown amount of experience. These developers were removed from the analysis. Table 6 displays the values of the productivity variance measures. It also includes a measure of the median normalized productivity, which is defined as the total productivity divided by the median productivity of the data set.

Table 6 Productivity differences for different experience levels

Data set	*Amount of experience*	*Norm. mean prod.*	*CoV*	*Gini*	*P90P10*	*P80P20*	*Rel80%*	*Rel50%*	*Skew*
A	Q1: [2.0–4.8] years	1.39	0.42	0.21	2.48	1.92	2.06	1.67	0.00
	Q2: (4.8–6.0] years	1.31	0.50	0.22	2.35	1.84	2.27	1.79	1.52
	Q3: (6.0–8.3] years	0.95	0.38	0.17	1.91	1.47	1.73	1.27	1.71
	Q4: (8.3–18] years	1.18	0.31	0.15	1.82	1.37	2.22	1.73	0.49
B	Q1: [0–1.0] years	1.14	0.41	0.20	3.14	2.12	2.12	2.04	0.70
	Q2: (1.0–4.0] years	1.26	0.40	0.24	3.08	1.99	2.20	2.19	0.25
	Q3: (4.0–10.0] years	1.08	0.43	0.24	3.00	2.23	2.21	2.17	0.74
	Q4: (10.0–39.0] years	0.81	0.58	0.26	2.59	1.83	2.14	1.83	3.53

Table 6 shows no systematic increase in the normalized mean productivity with more experience. The data instead suggest a weak tendency towards a *decrease* in productivity with more experience. The correlations (Pearson correlation coefficient) between productivity and amount of experience were -0.22 ($p=0.34$) for data set A and -0.27 ($p<0.001$) for data set B. A t-test of the productivity difference between the Q1 and Q4 quartile gives a statistically significant difference ($p<0.001$) for data set B, but not for data set A ($p=0.52$). The effect size (normalized difference in productivity between the quartiles) is similar between the data sets.

No correlation or just a weak correlation between the amount of experience and productivity among software programmers has been reported in several other studies, see for example (Jørgensen and Sjøberg 2002, Bergersen and Gustafsson 2011, Bryan 2012, Dieste, Aranda et al. 2017).[19]

This finding weakens the plausibility of the first explanation, which is based on a "rich-getting-richer" type of mechanism, since an increase in the productivity difference over time is not likely if the productivity does not increase with more experience.

The "rich-getting-richer" explanation is also not supported by the other measures of productivity differences. The coefficient of variation suggests no change or even a decrease in the productivity variance with more experience for data set A. For data set B, the coefficient of variance shows no large difference for different experience categories, except that it indicates a higher variance among those in the category with the most experience (Q4). Combining this observation with the observation that those in Q4 for data set B, on average, had a productivity that was 20% lower than the mean productivity suggests that the empirical data can hardly be explained by a "rich-getting-richer" type of explanation. The more outlier-robust Gini coefficient shows even less variance in the productivity between the experience quartiles. The P90P10 (and P80P20) measures suggest that when comparing the productivity of the 90th and 10th percentiles (80th and 20th percentiles) of the productivity distribution, the productivity differences decreased for data set A, while there was no clear trend for data set B. As before, no support for increased productivity differences with more experience was found. Even when including the entire upper 20% tail (and 50% tail), as is done for Rel80% (and Rel50%), no systematic increase in the productivity variance for either of the two data sets is observed.

4. Discussion, implications, and limitations

4.1. Essential characteristics of the variance in productivity (RQ1)

Many claims have been made about the variance in software developers' productivity. In particular, there have been claims about the variance between the most and least productive developers. Statements of this type are, however, difficult to interpret, and they are not based on robust measures of productivity differences. For example, the difference between the most and least productive developers in a group of five developers should be interpreted differently from the difference for thirty developers.[20] For an insightful discussion on the problems of productivity claims based on comparing the most and least productive developers, see (Nichols 2019).

This paper introduces and applies measures of productivity differences that are more robust and easier to interpret than measures based on comparing the most and least productive developers. This includes normalized measures of the spread (the coefficient of variance and the Gini coefficient), measures showing the ratio between developers in different percentiles of the empirical productivity distribution, and measures indicating the effort-adjusted total production of the top X% of developers in terms of productivity relative to the other developers. These measures have been used only to a limited extent or not in prior studies related to software development productivity differences.

This paper finds, based on empirical productivity data from four data sets, mean values of the coefficient of variance of 0.55, a Gini coefficient of 0.27, a ratio of the developers in the 90th and 10th percentiles (P90P10) of 2.98, and a ratio of the developers in the 80th and 20th percentiles (P80P20) of 2.57. The study also finds results consistent with that the top 20% of the developers would produce about 40%, and the top 50% about 70% of the total output. While this suggests a substantial variance in the productivity among software professionals, the variance is, for example, not as large as indicated by claims of a 10:1 difference in productivity[21] or as implied by the application of the Pareto principle, which implies that 20% of the developers do 80% of the work. In addition, only about two-thirds of the observed difference is systematic, i.e., a difference that one can expect to be repeated for future tasks. The systematic differences in individual productivity among software developers are consequently substantially lower than the differences observed in this and other studies. The true (systematic) coefficient of variance of productivity seems, for example, to be only around two-thirds of what is observed.

Comparing the coefficient of variance found in this study with those reported in previous studies,[22] this study finds both lower and higher values. A lower value is reported in (Anda, Sjøberg et al. 2008), who reports a coefficient of variance of the effort of four companies completing the same project of 0.29. Including only four projects means that the robustness of that result may be low. When looking at the reported cost estimates of the set (n = 35) of companies competing for the project in that study, the coefficient of variance is 0.65, i.e., a value similar to what was found in this study. The review study of (Prechelt 1999, Section 3.4) reports that the typical (median) coefficient of variance value in the set of tasks from that paper was around 0.5, which is similar to what was found in this study. The reanalysis of the data presented in (Grant and Sackman 1967) gives a mean coefficient of variance of 0.58, i.e., a value very close to what is reported in this paper. This may indicate that even though the technology and development environment, and probably the overall productivity of software developers, has changed significantly since 1967, the *differences* in productivity among software developers may have changed much less. In general, there seems to be a high degree of similarity in the values of the productivity variance, as measured by the coefficient of variance, reported in this study and other studies.

There are also similarities when comparing the results in this paper with the few previous results on PXPY-based (percentile-based) measures.[23] The data sets in (Prechelt 1999, Section 3.1) suggest a P87.5P12.5 ratio of around 2.5. This is only slightly lower than this study's comparable P90P10 ratio of 2.95.

All data sets, except data set D, are conducted in contexts where the participants knew they were part of an experiment (data sets A and C), or in a training context (data set B). Therefore, a short survey with software professionals as participants was conducted to validate our results with the productivity variance in more realistic software development settings. The survey was conducted at a cost estimation seminar for software professionals. In total, 74 highly skilled software developers participated in the survey. In that survey, a development situation similar to that of data set D was described, and the following question was asked:

Assume a programming task related to the extension of a web application. Both the web application and the task have normal (medium) complexity. There is a team of ***seven software developers****, all with at least one year of experience with the application. Assume a normal variance in competence between these seven developers. Consider a – hypothetical – situation where all seven developers have completed the task, individually and independently of each other, and that all have delivered acceptable quality of their work. The software developer who used the* ***middle*** *(medium) amount of effort of the seven spent* ***40 work hours*** *(effective hours) to complete the task. How many work hours (effective hours) do you think that the developer who spent the most and the least effort spent on the task. Consider all effort spent on the task, including that on the error corrections following errors found in the test.* ***Base your judgment on what you have experienced as typical differences in the use of effort between the most and least productive software developers in similar contexts****.*

The responses gave that the most productive developer in a team of seven developers was experienced to be, on average, 4.27 times more productive than the least productive. Comparing the most and least productive developers in a group of seven is similar to comparing the 86% (=6/7) and 14% (=1/7) percentile (P86P14), which is close to our P90P10-measure. Comparing this with our results in Table 2, we see that this value (4.27) is higher, but not much higher, than our mean P90P10 value (3.98) and is similar to several of the individual task values. In particular interesting, the experienced productivity difference is identical to that found for data set D (3.98), which had the most similar context to the context described in our survey. Furthermore, the responses gave, similar to what we found in our data sets, an expectation of a strongly right-skewed productivity distribution. While the most productive developer in a team of seven developers would have, according to the respondents' experience, a productivity 112% higher than the average developer, the least productive developer would have a productivity only 41% lower than the average developer. Fitting a lognormal distribution to the expectation of productivity variance in the survey data (using the tools @Risk) gave a Rel20%-value of 4.5 and a Rel50%-value of 3.7, which may indicate that the software developer participating in the survey on average experienced an even longer tail towards the higher productivity values than observed in the data in this study. The responses to the survey question support that the productivity results presented in this paper are representative of the industrial contexts experienced by the participants. If anything, the survey data suggests that the variance and right-skewness are typically higher than reported in this paper.

4.2. Distributional shape and generative mechanisms (RQ2)

Prior studies, see for example (Prechelt 1999), report that the effort usage of software developers completing the same task is right-skewed, i.e., it has a long tail towards higher effort values. A long tail towards higher effort values implies that the worst performers are further from the median performer than the best performers. This insight has been used, see for example (Sackman, Erikson et al. 1968), to argue that it is even more important to detect and avoid recruiting the poorest-performing developers than it is to detect and hire very-high-performing developers. This study finds that the productivity distributions were right-skewed[24], i.e., the best performers were further from the median than the worst performers. This may be used to argue that detecting and recruiting the best developers is more important than detecting and avoiding the worst developers, i.e., the results imply the opposite of the effort-based performance analysis.

Observing simultaneously, right-skewed distributions of effort usage and productivity may appear contradictory, but it is not. Applying statistical theory, it is straightforward to show this. If, for example, effort is lognormally distributed with mean *m* and standard deviation *s*, then productivity measured as 1/effort will also be lognormally distributed with the same standard deviation and skew as the corresponding normal distribution, see for example, en.wikipedia.org/wiki/Log-normal_distribution.

This simultaneous presence of right-skewed effort and productivity distributions may also be supported by analyzing the data used in this paper. When back-transforming the data to effort data (Effort = 1/Productivity), a right-skewness of the effort for all tasks, with skewness values between 1.28 and 1.79, is observed, i.e., skewness values that are not far from those observed for the productivity data.

The apparent contradiction in finding a long tail towards lower-performing developers (as in the effort-based analysis) and higher-performing developers (as in the productivity-based analysis), may also be resolved by distinguishing between two software development contexts. These two contexts differ regarding what is kept as a constant factor and what is considered the variable factor. If one considers the output as a constant, then the right-skewness of the *effort usage* is what matters. Constant output is, for example, present in a situation where one would like to recruit one or more developers to complete a task or a project. In that case, there is more to gain, measured in terms of the difference in the *effort* spent compared to a medium-performing developer, from avoiding the worst developers than from detecting and recruiting the best developers. If, on the other hand, one keeps the input (the effort) constant, then the right-skewness of the productivity (output produced in a constant time unit) is what matters. A constant amount of effort is, for example, present when a company wants to recruit a developer for a permanent position to work for a certain number of work hours every year. In that case, there is more to gain, measured in terms of the difference in the *output* produced compared to a medium-performing developer, from detecting the best developers than from avoiding the worst developers. In other words, both results are correct, but they have practical implications in different contexts. In most real-life contexts, there will probably be a focus on both avoiding the worst developers and finding the best developers. Still, the above observation should motivate a shift in the evaluation focus depending on in which of the two contexts recruitment is occurring.

This paper reports, using the method of "data pitting," that the best fit to the software productivity data was achieved by the power law with an exponential cutoff and by the lognormal distribution. The lack of prior work on fitting distributions to empirical productivity data in software development contexts means that it is only possible to compare the results from this study with those from other domains. The analysis of 229 productivity data sets in (Joo, Aguinis et al. 2017) suggests similar findings. In that extensive analysis, as many as 75% of the data sets were dominated by distributions with an exponential tail, such as the power law with an exponential cutoff. The lognormal distribution provided the best fit for about 5% of the data sets. Other studies report similar results, i.e., that either the exponential (Huber 2001) or lognormal distributions provide the best fit for productivity data (Shockley 1957). It is expected, based on the observed variation between the data sets, that the type of distribution with the best fit for software development productivity will vary from context to context. Nevertheless, it seems to be a robust observation that productivity is right-skewed and that the upper tail tends to be thicker than that of a normal distribution but typically not as thick as that of a pure power-law type of distribution (such as a Pareto distribution).

The identification of the type of distribution with the best fit to the productivity data was motivated by a desire to provide input to a discussion of the mechanisms potentially generating the observed productivity differences. This is a difficult task, as many possible generative mechanisms may explain the same type of distribution. Through the analyses in this paper, it was possible to provide evidence that some previously proposed mechanisms are less plausible as central explanations for productivity differences. An explanation based on the advantages of already having high productivity (the "rich-getting-richer" type of explanation) is for example less plausible because of the observation that the productivity difference did not increase over time. Explanations based on strong ceilings and floors on productivity (homogenization) or the additivity in factors leading to productivity are also less plausible. If these had been plausible explanations, one would have expected to find a better fit of the productivity data to the normal or Weibull types of distributions.

The most plausible, quite general, explanation is that there is a non-additive, either multiplicative- or "incremental differentiation"-based aggregation of productivity factors. Two important productivity factors, well-documented in studies from other domains, see the review in (Van Iddekinge, Aguinis et al. 2018), are related to skill and motivation. There seem to be synergy effects between the productivity factors. These are weaker than those predicted by a power law, and exclude additive aggregation. More research is needed to identify and operationalize productivity factors and to examine how they are connected to create software development productivity differences.

4.3. Practical and research implications

Possible practical and research implications of results reported in this paper include the following:

Companies hiring software developers may benefit from implementing recruitment processes that not only enable the separation of developers with low and high performances but also enable the separation of high- and very-high-performing software developers. This is a consequence of the long tail toward higher productivity values.

A potential means of separating "very good" software developers from "good" software developers may be the use of more extensive work sample tests or "trialsourcing" (Jørgensen 2015). The use of work sample tests in recruitment is also motivated by the findings in this and previous studies that the amount of experience and other indicators typically present in developer *curriculum vitae* are poor indicators of productivity. There is a need for more research on the underlying reasons for this lack of improved productivity with more experience. More insight into this may lead to improved learning environments in software development contexts.

In contexts in which a company selects one or more software developers to complete a particular task, rather than hiring them for a period of time or a permanent position, the gains from avoiding the poorer performers dominate, i.e., selection processes should have a particular focus on avoiding the poorest performers. This is because there is a long tail toward higher effort usage among software developers solving the same task. Notice that the simultaneous observation of a long tail towards higher effort values and a long tail towards higher productivity values is neither statistically nor practically contradictory; these observations are relevant in different contexts.

Finding star performers, i.e., software developers consistently showing very high productivity, is likely to be difficult. This is because much of the observed variance in productivity is expected to be non-systematic for those with the highest observed productivity.[25] There also seem to be very few star performers among software developers, as indicated by the poor fit of the power-law types of distributions to the productivity data.

Productivity factors, such as skill and motivation, are likely to be connected in a non-additive manner (i.e., with synergy effects) to determine productivity, as suggested by the types of distributions that best fit the observed productivity data. If companies tend to assume an additive effect, as found by, for example, (Duma 2021), this may cause them to underestimate the actual combined effect of productivity factors in the assessment of software developers.

4.4. Limitations

Several limitations reduce the external and internal validity of the software development productivity results presented in this paper:

The analyses of productivity include only programming-related activities. Other activities, such as design, communication with clients, and coordination with other developers, are not included. This means it is still being determined how well the developers would perform in more realistic settings with a broader set of activities, working in teams, and with a weaker or stronger motivation to perform well. Supporting the external validity of the results is, however, the fact that programming is the most central part of what software developers do, and a large part of software development work is typically based on individual programming. In addition, one of the data sets (data set D) included a larger task (a project) involving elements other than just the programming, e.g., communication with clients and system testing. The results from that data set were similar to those from the other data sets.

Most of the data sets are from contexts where the developers knew they were part of an experiment (data sets A and C) or a training program (data set B). It is not clear whether or not they would have behaved the same way in more realistic settings. However, the author of this paper's experience from conducting many experiments of this type is that most software developers take experimental programming tasks seriously and do their best. The developers are sometimes even more motivated to perform well and have a stronger focus on task-solving when they participate in experiments than in real-life tasks. Furthermore, the experience of software professionals, as reported in the survey, gave very similar results as those found in the data sets. This also supports the external validity of the results on productivity differences and the right-skewed shape of the productivity distribution.

The tasks are quite small, and even the largest of the tasks require only a few days of work. This means, among other things, that it needs to be discovered how well the developers can build upon previously developed software to build larger applications, with work covering weeks and months instead of hours and days. The data from data sets A and B address this challenge to some extent since these tasks involve extensions and modifications of existing systems. While the validity would be improved by using productivity data from larger projects, one would then face the challenge of a need for more commonly agreed-upon measures of the output of software development that can be used to compare the productivity across projects. To ask developers or teams of developers to complete the same, large project would hardly be feasible. For this reason, it may be hard to avoid using smaller tasks for studies on software development productivity.

Many of the distributional types included in the analysis of the fit to the empirical data are quite flexible in terms of the thicknesses and lengths of their tails and shapes. This increases the risk of overfitting, especially for data sets with fewer observations. This study has tried to address this challenge using "data pitting," which combines statistical tests of the differences in the fit with the principle of selecting the most parsimonious (least flexible) distribution when there is more than one distribution with a good fit. The danger of overfitting is nevertheless still present, and other distribution types may give the best fit in other contexts. There are nonetheless reasons to believe in the robustness of the main results on the distributional shape, i.e., that of right-skewed productivity distributions and with a thickness of the upper tail between, but not including, a power-law distribution and a normal distribution, as this is consistent with all the data sets in this study, the responses by the survey respondents and the results in several other domains.

The analysis of generative mechanisms was more complex and less conclusive than anticipated. This was partly caused by the finding that productivity did not increase as the amount of experience increased, which caused us to look for other types of explanations in addition to those related to the "rich getting richer." The main contribution may, consequently, be to show the low plausibility of the "rich-getting-richer" types of explanations and to point out the need for more research on factors and mechanisms explaining the differences in productivity.

5. Conclusions and future work

The study presented in this paper focused on the software development productivity variance and is based on an analysis of the productivity of software developers solving small, well-defined programming tasks. A substantial variation in productivity was found for all four data sets, which included 21 tasks in total. The variation increased, as expected, with fewer skill restrictions on the recruitment of developers and with the complexity of the task. The coefficient of variation (CoV) of productivity varied between 0.15 and 0.84, with a mean value of 0.55. The productivity ratio of the top 20% of developers compared to the remaining 80% varied from only 1.18 to as much as 3.18, with a mean of 2.83. It was found that not all of the observed differences in productivity should be interpreted as a systematic difference in productivity, since about one-third of the variance in productivity seems to be due to non-systematic, i.e., not repeatable, variance.

An additional goal of the paper was to study the shapes of productivity distributions. The productivity distributions for all tasks were found to have a long tail towards higher productivity values, i.e., they were right-skewed. The distribution type with the best fit to the empirical productivity data, according to the "data pitting" analysis method, was, for all but one task, either the lognormal distribution or the power-law-with-an-exponential-cutoff distribution. This implies that there were fewer "star performers" compared to what would be expected if the productivity followed a pure power-law distribution type, including the Pareto distribution. This also implies that the assumption of a normally distributed productivity distribution with few high performers (a thin tail) is hardly plausible.

It was attempted to use the findings about the type of distribution with the best fit to the empirical data to identify generative mechanisms that may have produced the observed productivity differences and those that were less plausible. The first finding was that neither the productivity nor the difference in productivity increased with the amount of experience. This makes it less likely that productivity differences are caused by a "rich-getting-richer" type of aggregation process, i.e., mechanisms that give developers who already have high productivity advantages that lead to increased differences in productivity. The observation that the lognormal and power-law-with-an-exponential-tail types of distributions provide the best fit supports the idea that productivity factors, such as skill and motivation, are likely to have a synergy-based relationship in how they determine the productivity of software developers.

The long tail toward higher productivity values has practical implications for the recruitment and hiring process of software developers. Companies hiring software developers for a given period may, for example, gain much from implementing recruiting processes that separate very good from good software developers.

The current knowledge about how productivity differences among software developers are generated is limited, and there is a need for more research on this topic. In particular, the current knowledge about the underlying reasons for, i.e., the mechanisms generating the large differences in productivity, should be a subject for further research. In addition, the reasons for the lack of a positive correlation between length of experience and software development productivity may deserve more research attention. In particular, this research should address when and why do not more experienced in software development transform into higher productivity.

References

Aguinis, H., Y. H. Ji and H. Joo (2018). "Gender productivity gap among star performers in STEM and other scientific fields." Journal of Applied Psychology **103**(12): 1283.

Aguinis, H. and E. O'Boyle Jr (2014). "Star performers in twenty-first century organizations." Personnel Psychology **67**(2): 313-350.

Aguinis, H., E. O'Boyle Jr, E. Gonzalez-Mulé and H. Joo (2016). "Cumulative advantage: Conductors and insulators of heavy-tailed productivity distributions and productivity stars." Personnel Psychology **69**(1): 3-66.

Allison, P. D. and J. A. Stewart (1974). "Productivity differences among scientists: Evidence for accumulative advantage." American sociological review(August): 596-606.

Anda, B. C., D. I. Sjøberg and A. Mockus (2008). "Variability and reproducibility in software engineering: A study of four companies that developed the same system." IEEE Transactions on Software Engineering **35**(3): 407-429.

Andriani, P. and B. McKelvey (2009). "Perspective—from Gaussian to Paretian thinking: Causes and implications of power laws in organizations." Organization science **20**(6): 1053-1071.

Bergersen, G. R. and J.-E. Gustafsson (2011). "Programming skill, knowledge, and working memory among professional software developers from an investment theory perspective." Journal of individual Differences **32**(4): 201.

Berlingieri, G., P. Blanchenay, S. Calligaris and C. Criscuolo (2017). "Firm-level productivity differences: insights from the OECD's multiprod project." International Productivity Monitor **32**: 97-115.

Bradley, K. J. (2017). On the Abundance of Star Teams: Investigating the Drivers of Heavy Tails in Team Productivity Distributions, Indiana University.

Bryan, G. E. (2012). Not all programmers are created equal—Redux©. 2012 IEEE Aerospace Conference, IEEE.

Budzier, A. and B. Flyvbjerg (2013). "Making sense of the impact and importance of outliers in project management through the use of power laws." Proceedings of IRNOP (International Research Network on Organizing by Projects) **11**(June).

Buzacott, J. A. (2002). "The impact of worker differences on production system output." International Journal of Production Economics **78**(1): 37-44.

Campbell, D. T. and D. A. Kenny (2002). A primer on regression artifacts, Guilford Press.

Clauset, A., C. R. Shalizi and M. E. Newman (2009). "Power-law distributions in empirical data." SIAM review **51**(4): 661-703.

Dieste, O., A. M. Aranda, F. Uyaguari, B. Turhan, A. Tosun, D. Fucci, M. Oivo and N. Juristo (2017). "Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study." Empirical Software Engineering **22**(5): 2457-2542.

Duma, L. (2021). "The Groundless Use of Linearity in Daily Thinking and Decision-making." Periodica Polytechnica Social and Management Sciences **29**(2): 125-135.

Fletcher, S. R., T. S. Baines and D. Harrison (2008). "An investigation of production workers' performance variations and the potential impact of attitudes." The International Journal of Advanced Manufacturing Technology **35**(11): 1113-1123.

Frank, S. A. (2009). "The common patterns of nature." Journal of evolutionary biology **22**(8): 1563-1585.

Grant, E. E. and H. Sackman (1967). "An exploratory investigation of programmer performance under on-line and off-line conditions." IEEE transactions on human factors in electronics(1): 33-48.

Guerrero, F. G. and A. Garcia-Baños (2020). "Multiplicative processes as a source of fat-tail distributions." Heliyon **6**(7): e04266.

Halffman, W. and L. Leydesdorff (2010). "Is inequality among universities increasing? Gini coefficients and the elusive rise of elite universities." Minerva **48**(1): 55-72.

Halkjelsvik, T. and M. Jørgensen (2018). Time Predictions: Understanding and Avoiding Unrealism in Project Planning and Everyday Life, Springer Nature.

Hergarten, S. and H. J. Neugebauer (1998). "Self-organized criticality in a landslide model." Geophysical research letters **25**(6): 801-804.

Hesse, J. and T. Gross (2014). "Self-organized criticality as a fundamental property of neural systems." Frontiers in systems neuroscience **8**: 166.

Huber, J. C. (2001). "A new method for analyzing scientific productivity." Journal of the American Society for Information Science and Technology **52**(13): 1089-1099.

Jones, H. E. and D. J. Spiegelhalter (2009). "Accounting for regression-to-the-mean in tests for recent changes in institutional performance: Analysis and power." Statistics in medicine **28**(12): 1645-1667.

Joo, H., H. Aguinis and K. J. Bradley (2017). "Not all nonnormal distributions are created equal: Improved theoretical and measurement precision." Journal of Applied Psychology **102**(7): 1022.

Jørgensen, M. (2015). "Better selection of software providers through trialsourcing." IEEE Software **33**(5): 48-53.

Jørgensen, M., G. R. Bergersen and K. Liestøl (2020). "Relations between effort estimates, skill indicators, and measured programming skill." IEEE Transactions on Software Engineering.

Jørgensen, M. and E. Escott (2022). "Relative estimates of software development effort: Are they more accurate or less time-consuming to produce than absolute estimates, and to what extent are they person-independent?" Information and Software Technology **143**: 106782.

Jørgensen, M. and J. Grov (2021). "A field experiment on trialsourcing and the effect of contract types on outsourced software development." Information and Software Technology **134**: 106559.

Jørgensen, M. and T. M. Gruschke (2009). "The impact of lessons-learned sessions on effort estimation and uncertainty assessments." IEEE Transactions on Software Engineering **35**(3): 368-383.

Jørgensen, M. and D. I. Sjøberg (2002). "Impact of experience on maintenance skills." Journal of Software Maintenance and Evolution: Research and Practice **14**(2): 123-146.

Mitzenmacher, M. (2004). "A brief history of generative models for power law and lognormal distributions." Internet mathematics **1**(2): 226-251.

Nichols, W. R. (2019). "The end to the myth of individual programmer productivity." IEEE Software **36**(5): 71-75.

Norton, M., V. Khokhlov and S. Uryasev (2021). "Calculating CVaR and bPOE for common probability distributions with application to portfolio optimization and density estimation." Annals of Operations Research **299**(1): 1281-1315.

Petersen, K. (2011). "Measuring and predicting software productivity: A systematic map and review." Information and Software Technology **53**(4): 317-343.

Pluchino, A., A. E. Biondo and A. Rapisarda (2018). "Talent versus luck: The role of randomness in success and failure." Advances in Complex systems **21**(03n04): 1-13.

Prechelt, L. (1999). The 28: 1 grant-sackman legend is misleading, or: How large is interpersonal variation really, Univ., Fak. für Informatik, Bibliothek (page.mi.fu-berlin.de/prechelt/Biblio/varianceTR.pdf).

Ramos, R., R. Sassi and J. Piqueira (2011). "Self-organized criticality and the predictability of human behavior." New Ideas in Psychology **29**(1): 38-48.

Ruocco, G., C. Daraio, V. Folli and M. Leonetti (2017). "Bibliometric indicators: the origin of their log-normal distribution and why they are not a reliable proxy for an individual scholar's talent." Palgrave Communications **3**(1): 1-8.

Sackman, H., W. J. Erikson and E. E. Grant (1968). "Exploratory experimental studies comparing online and offline programming performance." Communications of the ACM **11**(1): 3-11.

Schmidt, F. L., J. E. Hunter, R. C. McKenzie and T. W. Muldrow (1979). "Impact of valid selection procedures on work-force productivity." Journal of Applied Psychology **64**(6): 609.

Shockley, W. (1957). "On the statistics of individual variations of productivity in research laboratories." Proceedings of the IRE **45**(3): 279-290.

Symons, C. (2019). The COSMIC Method for Measuring the Work-Output Component of Productivity. Rethinking Productivity in Software Engineering, Springer**:** 191-204.

Van Iddekinge, C. H., H. Aguinis, J. D. Mackey and P. S. DeOrtentiis (2018). "A meta-analysis of the interactive, additive, and relative effects of cognitive ability and motivation on performance." Journal of Management **44**(1): 249-279.

Weedon, M. N., H. Lango, C. M. Lindgren, C. Wallace, D. M. Evans, M. Mangino, R. M. Freathy, J. R. Perry, S. Stevens and A. S. Hall (2008). "Genome-wide association analysis identifies 20 loci that influence adult height." Nature genetics **40**(5): 575-583.

Endnotes

[1] The focus of the experiment described in that paper was not on performance differences but rather on comparing the *debugging effort in different environments*. As pointed out by for example (Prechelt 1999), the reported effort differences (such as the 28:1 difference in the use of effort) are therefore the combination of individual differences in performance and in the development environment. The re-analysis in this paper, which uses the data reported by (Grant and Sackman 1967), is based on the following method: i) define the effort used as the sum of the effort spent on coding and debugging, and ii) compare the developer performances only within the same context, i.e., within the same experimental conditions and the same programming language (excluding a few developers using assembly language). This led to the formation of four groups of 4–5 developers completing the same task in the same development environment.

[2] The study reports, however, lower effort usage differences for non-programming activities related to software development, such as reviewing and understanding code. Additionally, the median effort usage difference (which is around 2.5:1) in the study was substantially lower than the mean difference, suggesting that a few very large differences have had a large impact on the mean difference in the effort usage.

[3] The utilized formulation of the Gini coefficient is mathematically equivalent to the more common formulation based on the Lorenz curve. It was considered that the formulation used would be more operational and somewhat easier to interpret.

[4] The expression is derived as follows (starting with the relative output divided by the relative effort of the most productive):

$$RelX\% = \frac{\frac{\sum Output_{X\%\ most\ productive}}{\sum Output_{1-X\%\ least\ productive}}}{\frac{\sum Effort_{X\%\ most\ productive}}{\sum Effort_{1-X\%\ least\ productive}}} = \frac{\frac{N_{X\%}\overline{Output}_{X\%\ most\ productive}}{N_{1-X\%}\overline{Output}_{1-X\%\ least\ productive}}}{\frac{N_{X\%}\overline{Effort}_{X\%\ most\ productive}}{N_{1-X\%}\overline{Effort}_{1-X\%\ least\ productive}}} = \frac{\frac{0.2 \cdot 1}{0.8 \cdot 1}}{\frac{0.2\ \overline{Effort}_{X\%\ most\ productive}}{0.8\ \overline{Effort}_{1-X\%\ least\ productive}}} = \frac{\overline{Effort}_{1-X\%\ least\ productive}}{\overline{Effort}_{X\%\ most\ productive}}.$$

[5] worldpopulationreview.com/country-rankings/gini-coefficient-by-country

[6] This type of measure (typically based on so-called super-quantiles) is increasingly being used for risk analyses; see, for example, (Norton, Khokhlov et al. 2021). Its focus on the expected value of the tail means that the information of the *whole* tail is included instead of just including one point of the tail, such as in percentile-based measures.

[7] The RelX% value may easily be compared to results of the following type: X% of the people do Y% of the work, where Y = RelX%/(RelX% + (1-X)/X). In the example with the relative production ratio of the top 20% of 4.75, Y = 4.75/(4.75 + (1-0.2)/0.2) = 0.54, i.e., the top 20% of developers in terms of productivity would complete 54% of the work.

[8] The author of this paper was in charge of recruiting and following up with the developers of this study, which was ordinary client work for the developers. As part of the recruitment process, the developers' previous work and the satisfaction of previous clients was analyzed.

[9] https://en.wikipedia.org/wiki/Log-normal_distribution

[10] The central limit theorem implies that *the sum* of random, independent variables approaches a normal distribution. The log of a product (multiplication) of positive values is the sum of the log of these values. Therefore, the product will approach a lognormal distribution. This is sometimes called Gibrat's law (see en.wikipedia.org/wiki/Central_limit_theorem).

[11] Assume that the effort (E) follows a lognormal distribution with a mean *m* and standard deviation *s*, where *m* and *s* refer to the parameters of the corresponding normal distribution, and that the productivity is defined as 1/E. log(1/E) = log(1) - log(E) = -log(E). This shows that 1/E is lognormal with a mean -*m* and standard deviation *s*.

[12] The pure exponential distribution may also be defined as the distribution of the waiting time for the first occurrence of an event (or the waiting time between events) in a Poisson process (equal event probability for each time unit).

[13] There is an interesting similarity between the power law with an exponential cutoff and the gamma distribution, i.e., the gamma distribution is essentially also a product of the power law and an exponential function (Frank 2009). A gamma function may be generated as the aggregated waiting time for the n-th event (as opposed to the exponential distribution, which is the waiting time for the first event). For the software development productivity, one will consequently observe a power-law-function-with-an-exponential-cutoff type of distribution, if one consider the productivity at one point in time as the aggregation of productivity increments, where the time between each increment is exponentially distributed (follows a Poisson process, with a certain probability of observing a productivity increment within a period of time). This situation may be thought of as a situation in which software developers differ in their probabilities of experiencing a productivity increment over sequences of time units, but the individual probabilities that increase within a period of time are relatively stable.

[14] cran.r-project.org/web/packages/Dpit/index.html

[15] This corresponds to the distribution of wealth in the Scandinavian countries, which are among the countries with the most equally distributed wealth in the world.

[16] The higher variance in the productivity of more complex projects compared to tasks is also supported by the company data provided by (Jørgensen 2015), where the same requirement specifications were given to six carefully selected offshoring companies. The coefficient of variation of the effort spent by these six companies was 0.79. The difference in productivity between the most and least productive companies in that study was as high as 18:1.

[17] Productivity is, in this case, with the same output size given by all developers, a linear combination of effort. This analysis will consequently give the same within- vs between-developer variance results as the analysis of (Jørgensen and Escott 2022).

[18] Variance is additive, which enables the subtraction of variance.

[19] Notice that several of these studies give a more balanced picture; for example, there may be a productivity increase only in the first few years of experience, the effect of experience may be indirect and mediated through programming knowledge, or the amount of the most relevant experience (with certain tools or applications) may matter.

[20] Assume, for example, the lognormal distribution Lognorm(4.5, 0.5), which has a coefficient of variance equal to that observed in this paper's analyses, i.e., 0.55. The expected productivity difference when five developers are selected from this distribution is about 4:1, which increases to about 9:1 when thirty developers are selected.

[21] As pointed out earlier, it is not clear what the often-stated claim of a 10:1 difference in productivity means. Clearly, there are developer pairs that show this difference in productivity, but as shown in this paper, this is not the "typical" difference,

e.g., the productivity ratio between developers in the 90th and 10th percentiles is not close to 10:1; instead, this ratio is on average less than 3:1, and this is partly due to random variation in the productivity.

[22] Previous studies measured the CoV of the effort spent by developers to solve the same task, while this paper measures the productivity. A direct comparison of the coefficient of variance of effort (on the same task) and productivity assumes in this context that the underlying distribution is approximately lognormal. As shown earlier, if X is lognormally distributed, its reciprocal (1/X) is a lognormal distribution with the standard deviation of the corresponding normal distribution (*s*). The coefficient of variance of a lognormal distribution may be expressed as $\sqrt{e^{s^2}-1}$ (see, for example, en.wikipedia.org/wiki/Log-normal_distribution), which implies that the coefficient of variance does not change when examining 1/X (productivity) instead of X (effort) when X is lognormally distributed. As reported in Section 3, the lognormal distribution gave the best or second-best fit for all but one of the tasks in this paper, which suggests that the coefficients of variance of the effort data of prior studies can be meaningfully compared with the productivity-based coefficients of variance in the study in this paper.

[23] PXPY types of measures from studies of variance in the effort usage on the same task are directly comparable with the productivity results in this paper, regardless of the distribution. This is because the ratio of the productivity of developers X and Y is equal to the ratio of their effort, i.e., $\frac{\frac{1}{Effort_X}}{\frac{1}{Effort_Y}} = \frac{Effort_Y}{Effort_X}$.

[24] By transforming the productivity data into effort data (Effort = 1/Productivity), one find a right-skewed distribution of effort, which is similar to what is reported in prior studies.

[25] As reported earlier, this paper found that around one-third of the variance was non-systematic. For developers with a very high productivity on a task, the non-systematic (random) part is likely to be even larger, i.e., the systematic productivity (skill) is likely to be substantially lower than the average. This is a consequence of the regression-towards-the-mean effect, which increases in strength with increasing distance between the score and the mean score (Campbell and Kenny 2002, Jones and Spiegelhalter 2009).