Submission to IEEE Transactions on Software Engineering

An empirical study of working speed differences between software engineers for various kinds of task

Lutz Prechelt (prechelt@computer.org)
Fakultät für Informatik
Universität Karlsruhe
D-76128 Karlsruhe, Germany
+49/721/608-4068, Fax: +49/721/608-7343

February 16, 2000

Abstract

How long do different software engineers take to solve the same task? In 1967, Grant and Sackman published their now famous number of 28:1 interpersonal performance differences, which is both incorrect and misleading.
This article presents the analysis of a larger dataset of software engineering work time data taken from various controlled experiments. It corrects the false 28:1 value, proposes more appropriate metrics, presents the results for the larger dataset, and further analyzes the data for distribution shapes and effect sizes.

1 Introduction

All software development managers and most software engineers are aware that large differences in the capabilities between individual softw are engineers exist. A less talented one may take several times as long for solving the same task as a very talented one — and will often still produce a less reliable, harder-to-maintain program. From a management perspective, such differences are very important. First, it is insufficient to know the average performance of a set of software engineers, because for instance the size of the task assigned to each one needs to be compatible with his or her speed etc. Second, if the performance of the individual staff members is unknown, knowing the size of the differences allows for assessing the risk incurred by assigning tasks randomly. Third, since part of the differences is due to different knowledge and training (rather than being hard-wired into the individual), the size of the differences is useful information for judging the possible benefits from additional training or from coaching etc.

In 1967, Grant and Sackman published a controlled experiment [9] in which they stated a ratio of 28:1 for the work time required by the slowest versus the fastest programmer (from a set of 12) each solving the same task. This number has since become quite famous and is being quoted over and over [7]. The present work validates and corrects this claim, using a broader and more solid foundation, namely data from a larger number of programmers, more different task types, and experiments that are not as old.

1.1 Article overview

The next section will describe the historical motivation for the present work, namely the Grant/Sackman experiment and its weaknesses. Section 3 introduces the dataset used in the present study and discusses several caveats regarding the validity and interpretation of the results.

The results of the present study are discussed in Sections 4 through 7. The main topic, interpersonal variability of work times in software engineering experiments, will be treated in Section 4. Variability statistics characterize one aspect of the work time distribution by a single number. Section 5 will complement these statistics by an assessment of the overall *shape* of the distributions. An third aspect, the size of the difference of average performance *between* groups (the so-called “effect size”), is discussed in Section 6. The comparative performance of different statistical tests is shortly discussed in Section 7.

Finally, the main observations are summarized and consequences are derived in Section 8.

2 The 1967 Grant/Sackman experiment

In 1967, Grant and Sackman published a controlled experiment for comparing online and offline programming [9]. Six subjects each debugged a program using an interactive session at a text

terminal ("online") and six others debugged the program using batch operation with fixed two-hour turnaround time ("offline"). The program, called *Maze*, had to compute the only way through a 20-by-20-maze. Each program to be debugged had been designed and coded by the same person just before. All twelve subjects were experienced professional programmers. The groups were then switched and the exercise was repeated with a different program, called *Algebra*, that computed the value of algebraic expressions.

In their article [9], Grant and Sackman state that "perhaps the most important practical finding of this study, overshadowing on-line/off-line differences, concerned the large and striking individual differences in programmer performance." They reported the now-famous ratio of the total working time required for the debugging process from the slowest to the fastest subject of 28:1 (170 hours versus 6 hours).

There are three problems with this number:

1. It is wrong.
2. Comparing the fastest with the slowest is inappropriate.
3. One should use more data to make such a claim.

I will now discuss each of these problems in order.

2.1 28:1 just isn't true!

The original publication [9] contains a complete table of the raw data, but appeared in a not-so-well-known journal, the *IEEE Transactions on Human Factors in Electronics.* The second, more well-known and more accessible source from *Communications of the ACM* [24] does not contain this raw data. Possibly this is the reason, why two methodological mistakes in the derivation of the 28:1 value are still not widely known. Dickey published about these mistakes in 1981 [7], but was too late to eradicate the false quotations.

What is wrong with 28:1? First, it refers to the union of the groups *debug Maze online* and *debug Maze offline*. Since systematic group differences between online and offline groups exist, this increases the ratio. If we consider the groups separately, the maximum difference is only 14:1. Second, three of the twelve subjects did not use the recommended high-level language JTS for solving the task, but rather programmed in assembly language instead. Two of these three in fact required the longest working times of all subjects. One might argue that the decision for using assembly is part of the individual differences, but presumably most programmers would not agree that doing the program in assembly is the same task as doing it in a high-level language. If we ignore the assembly programmers, the maximum difference drops to 9.5:1. Similar reductions occur for the other three pairs of groups in the experiment. Hence, the real differences are only half as large as claimed, but still justify the oft-stated "order of magnitude difference".

	Coding		Debugging	
	C Alge.	C Maze	D Alge.	D Maze
from [9]	16	25	28	26
SF_0	12.6	12.5	14.2	12.5
SF_{25}	7.0	8.0	5.8	4.2
SF_{50}	3.7	7.2	3.3	2.4

Table 1: Various measures of variability for each of the four pairs of groups from the Grant/Sackman experiment. The raw data are taken from [9]. Each entry in lines 2 to 4 represents the group with the higher variability; sometimes "online", sometimes "offline". Each entry in line 1 refers to SF_0 for the union of both groups.

2.2 How should we measure variability?

The second problem is much more obvious. If we compare the fastest to the slowest, we will obtain almost arbitrarily high ratios if only we have enough subjects at hand: somebody may always take still a little longer.

A groupsize-independent measure of variability is the ratio s/m of the standard deviation s and the mean m. It is independent of group size, but still has the disadvantage that extreme values in the group influence the value a lot.

But if we partition the group into a slower and a faster half, we can consider the ratio of the medians of these halfs. This value is robust against outliers as well as easy to interpret: How many times longer did the average "slow" subject compared to the average "fast" subject? We might call this value the slow/fast ratio SF. Mathematically, this is the ratio of the 75% quantile of the time distribution to the 25% quantile: $SF = SF_{50} := q_{75}/q_{25}$.

We may also ignore the middle half of the subjects and compare the medians of the slowest and fastest quarters $SF_{25} := q_{87,5}/q_{12,5}$. Using this notation, the ratio of maximum to minimum would be $SF_0 := q_{100}/q_0$.

I suggest to use SF_{50} and SF_{25} as robust and easily interpretable measures of interpersonal variability.

The values of these measures for the four tasks from the Grant/Sackman experiment are shown in Table 1. As we see, the typical representative of the faster half of the subjects is about two to seven times as fast as the typical slower half subject. Note that SF_{25} is not robust against even a single outlier in this particular case, because with a group size of only 6 persons, the second fastest is already $q_{16.67}$, but we are using $q_{12.5}$, so that the result contains fractions of the times required by the fastest (and slowest) subject. Only with groups of size 9 or more will SF_{25} be completely free of any direct influence of the fastest and smallest person.

Figure 1 shows the individual data points of each person in each group, including a corresponding box plot (explained in the caption). We clearly see that the high values of SF_0 usually stem from only a single person that was very slow.

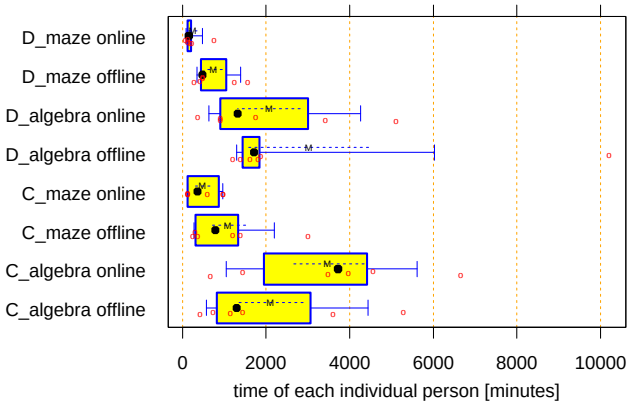

Figure 1: Work times of the individual subjects in the groups of the Grant/Sackman experiment. The box plot indicates the quantiles at 10%/90% (whiskers), 25%/75% (box), and 50% (fat dot), the mean (M), and the standard error of the mean (dashed line).

2.3 Only 12 people even after 30 years?

The possibly most amazing fact about the Grant/Sackman data is that despite the large interest in the 28:1 figure, nobody so far has ever systematically collected a larger amount of data about this phenomenon. Everybody still refers to that one case, based on only 12 programmers.

From time to time, individual researchers address the interpersonal variability for one of their own datasets (Curtis once even wrote a short article just for this purpose [3]), but as far as I know, an investigation analyzing data from several different experiments together did not exist before the current work.

3 `variance.data`: A larger dataset

The rest of this report presents the analysis of work time data collected from 61 different controlled experiments (or parts of experiments) in software engineering. These data were obtained either directly from experiments conducted by our research group, from tables printed in journal articles or technical reports, or were sent to me by one of the 17 researchers that I contacted by email. Data was used only if all persons of a group solved the same task under roughly the same conditions — the Grant/Sackman experiment with its heterogeneity of programming languages is in fact an unusual example in this respect.

3.1 Contents and structure

The dataset contains data from the following experiments: [1], [2], [3], [4], [5], [8], [9, 24], [10], [11], [12, 19, 21], [14], [15, 16], [17, 18], [20], [22], and [25]. Data from several other experiments was not available because the authors either did not answer my request or said they no longer had access to the data.

The data from the original datasets was translated into a common format, which I will now describe. `variance.data` contains one data point for each work time measurement of a complete (sub)task performed by an experiment participant. The dependent variable of each data point is the work time in minutes. Each work time is described by the following independent variables:

- *source*: a reference to the article or techreport in which the experiment is described.
- *id*: a symbolic identifier for the experiment participant. Multiple measurements for the same person within one experiment use the same id.
- *group*: A short name for the experiment conditions (except for task, see below) used for this measurement. In most cases this simply refers to either the experiment group or the control group, but in a few cases there are more than two groups being compared.
- *task*: a symbolic name for the task to be solved. Each experiment group within `variance.data` is hence described by a unique combination of *source*, *task*, and *group*.
- *type*: the kind of task. This value is always identical for all persons within one group. This variable partitions the whole dataset into parts with somewhat homogeneous properties. The following values exist:
  - maintain (understanding and modifying/extending),
  - understand (i.e., answering specific questions),
  - test/debug (testing or debugging or both),
  - review (inspecting for finding defects),
  - program (design, implementation, and testing/debugging),
  - design,
  - code (implementation).

  In case of doubt, either “program” or “maintain” will be used. Since the types “design” and “code” are rare, they will be included into type “program” in all analyses presented below.
- *seq*: the sequence number of this task for this person in this experiment (e.g. 2 if this is the second task performed by this person). This variable allows to assess sequence effects, but will not be used for the analyses shown here.

Overall, the dataset contains 1491 observations made for 614 different subjects from 137 experiment groups ranging in size from 2 to 38 persons (mean 10.9, see also Figure 2) and working on 61 different tasks. 14 groups consisting of less than 5 subjects each will subsequently be ignored, resulting in a total group size mean of 11.9 and median of 9.

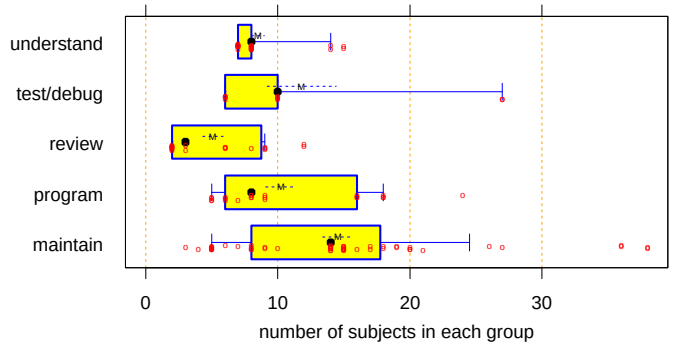

Figure 2: Distribution of group sizes in `variance.data`. There is the following number of groups in each task type category: understand:28, test/debug:10, review:22 (but mostly very small), program:23, maintain:54.

3.2 Internal and external validity: Warning about the interpretation of the data

Some caveats have to be kept in mind in order to avoid misinterpretations:

- In principle, the dataset contains only data for which no explicit time limit was mentioned to have been enforced. It is possible, though, that such a limitation was present but was not mentioned by the authors or was overlooked by me.
- If multiple subjects started their work in the same room at the same time, social pressure may have reduced the work time differences ("Oh, somebody's already finishing. I should hurry.")
- For most of these experiments, work time was not the only variable that describes the performance of the subjects. However, other measures that may warrant or explain very short or very long work times are not considered here, because no quantitative variable other than work time is available for any large number of experiments.
- For some of the experiments work time was not an important performance variable.
- The resolution and accuracy of the time measurement is often unknown. In most cases, resolution is either one minute or five minutes. In many cases, the times were recorded by the subjects themselves, hence assessing accuracy is difficult.

The main message of the above is the following: Only in a few experiments all participants have worked until their task was solved completely and correctly. Therefore only in a few experiments the work time information is sufficient to characterize subject performance.

With respect to the generalizability (external validity) of the results found in the present work, consider the following: No doubt some subjects will have reduced their work time at the expense of product quality (whether that was a conscious choice or not). It is difficut to assess the extent to which that happened. However, a similar statement is true for real software development as well: Given typical schedule and market pressures, work time is often cut down in a way that decreases product quality. The extent to which this happens, however, can only be judged on a case-by-case basis.

Hence, it is left to the reader to decide when the results from this study apply to some specific software engineering reality and when they do not. But at least it appears sensible to assume that the results will hold for similar controlled software engineering experiments.

4 The size of interpersonal work time differences

This section discusses the core of the matter. To improve the homogeneity of the data, we partition it according to task type.

4.1 Slowest versus fastest individual

Let us start with the maximum/minimum ratio as used by Grant and Sackman — except that we will properly discriminate different experimental conditions. This statistic is shown in Figure 3.

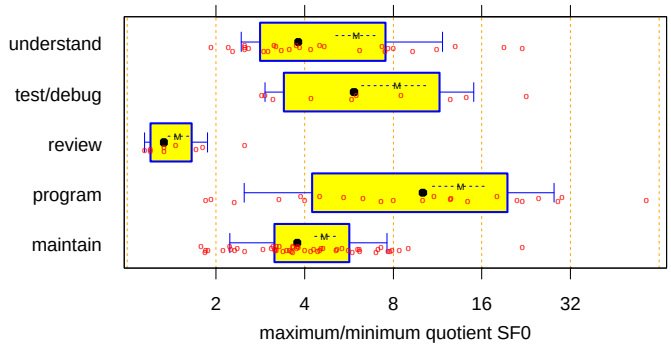

Figure 3: Distribution of the ratio SF_0 of the slowest to the fastest subject within each group. Each point represents one group of measurements for the same task under the same experimental conditions. Note the logarithmic scale.

As we see, large individual variations similar to those seen in the Grant/Sackman experiment (that is, 12 to 14 or more) *do* occur, but are not at all typical. Except for the task type "programming", which exhibits the largest variation of variations, interpersonal differences of more than factor 10 are rare. Before we interpret too much into this figure, let us switch to a more robust representation: The ratio of the slowest to the fastest quarter.

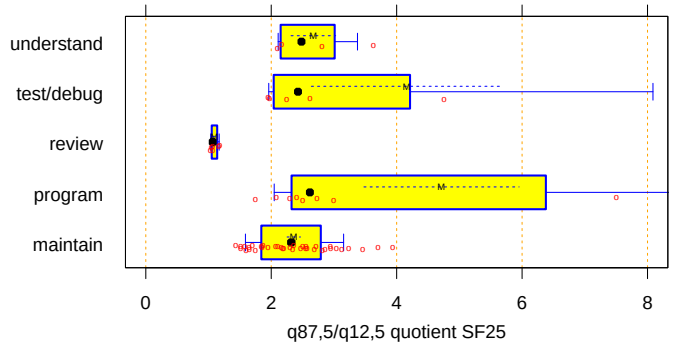

Figure 4: Distribution of the ratio SF_{25} of the medians of the slowest and fastest quarter of each group, for all groups with 9 or more subjects. (SF_{25} is not robust for groups smaller than 9, because the minimum and maximum value would then still influence the result.) One point of "test/debug" is outside of the plot at 11.4; likewise two points of "programming" at 10.9 and at 12.

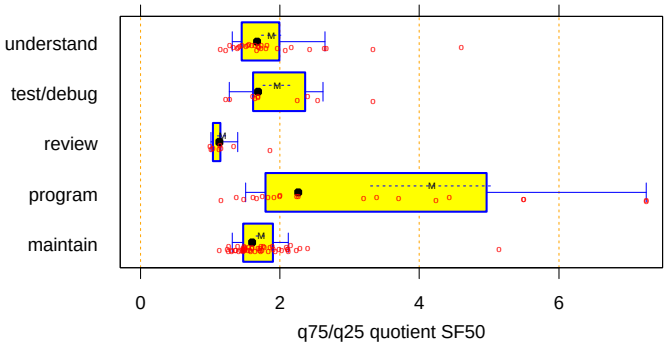

Figure 5: Distribution of the ratio SF_{50} of the medians of the faster and slower half of the subjects for each group.

4.2 Slowest versus fastest quarter

SF_{25} is plotted in Figure 4. It appears that 2 to 3 is the typical median for this performance ratio (for all task types except review). Thus, if we ignore the most extreme cases, the differences between the slowest and the fastest individuals are by far not as dramatic as the 28:1 figure suggests. The large values of 4 to 8 found for the Grant/Sackman experiment are unusual compared to most of the rest.

We note a rather narrow distribution for reviews. Apparently the individual capability differences will be converted mostly into quality differences rather than time differences for this task type. There are two possible reasons: First, in contrast to any other task type, one can declare an inspection finished at essentially any time. Second, in some of the experiments, a certain review speed (in lines per hour) was recommended to the subjects.

4.3 Slower versus faster half

Next, see Figure 5: If we consider the slower and faster half instead of the slowest and fastest quarter of each group, the differences shrink further.

The difference between the slower and faster half is usually less than factor two, again except for task type "programming". And again, the high values in the Grant/Sackman data (2.4 to 7.2, as seen in Table 1) are rather unusual. How can those high values be explained? The programming education of programmers in 1967 was certainly more inhomogeneous than the education of CS students in the 1980s and 1990s (which most of the subject populations of the other experiments come from). Perhaps this inhomogeneity has caused the higher variance. Since in some practical software engineering contexts the background of programmers is becoming more diverse again (because increasingly more of them do not have a formal CS or SE education), this explanation suggests that in practical situations the variability may often be larger than the distributions shown in Figure 5 suggest.

Another explanation could be based on the observation that the absolute work times of the Grant/Sackman experiment are longer than the work times of the other experiments. Maybe larger tasks result in higher variability? No, the plot of SF_{50} versus mean work time for all experiment groups (Figure 6) indicates no such trend at all — the opposite appears to be more likely.

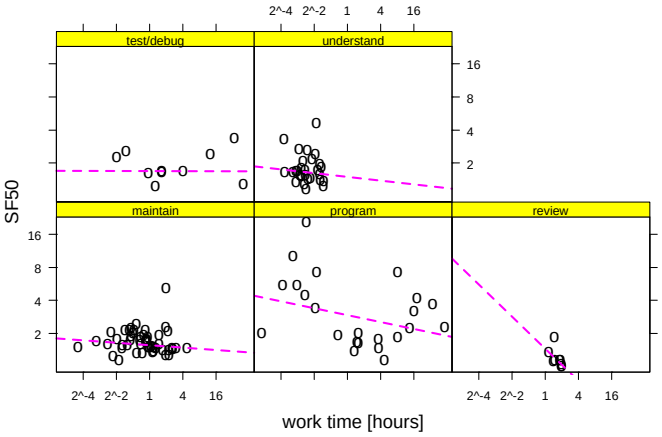

Figure 6: SF_{50} depending on the mean work time of each experiment group. The dashed trend line is a robust L_1 (minimum absolute distance) regression line. Both axes are logarithmic.

A median SF_{50} of about 2 matches well with the results of the fascinating 1984 field study of DeMarco and Lister [6]. In that study 166 professional developers from 35 different organizations had solved the same task (of type "program") under very different local work conditions and SF_{50} was 1.9.

4.4 Standard deviation relative to the mean

For sake of completeness, let us have a look at the ratio of standard deviation and mean, which also characterizes variability. Figure 7 shows that this ratio is typically about 0.5. This means that on the average a programmer will take about 50 percent more or less time than the average programmer does.

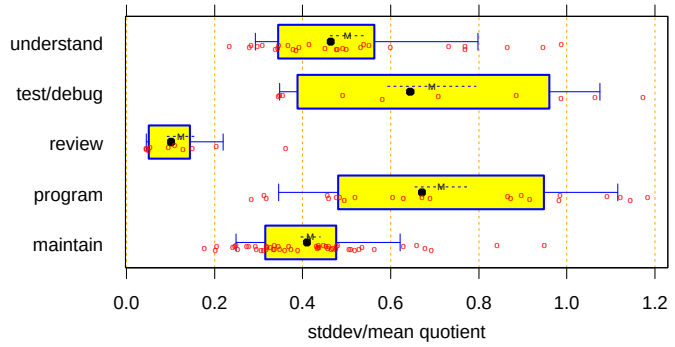

Figure 7: Distribution of the ratio of standard deviation and arithmetic mean for each group.

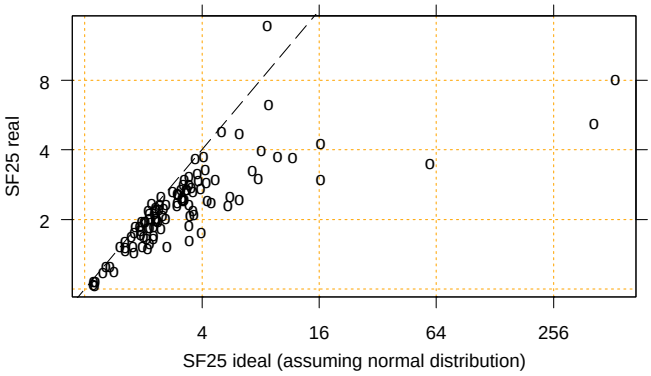

Figure 8: Comparison of the actual value of SF_{25} for each experiment group with a theoretical value of SF_{25} based on a normal distribution assumption and computed from mean and standard deviation. The axes are logarithmic.

4.5 Summary

In summary we can say that *typical* work time differences between slower and faster individuals are more on the order of 2:1 if we consider the slower versus faster half (SF_{50}) or 3:1 to 4:1 if we consider the slowest and fastest quarters only. In any case, the differences are much smaller than the oft-cited 28:1. Still, these differences are much larger than the typical differences between two different experimental conditions (see Section 6) and are hence clearly worthy our attention in terms of improving the software development process.

5 The shape of work time distributions

The summary indicators of variability such as the ones presented above address but a single aspect of a more general question: What is the *shape* of a work time distribution in software engineering?

The answer to this question is a key to solving a number of problems such as more accurate schedule and staff planning in large projects, improved risk detection in projects, correct and efficient statistical inference in controlled experiments, etc.

This section will therefore investigate the shape of the work time distributions present in `variance.data` for the various task types.

5.1 The normal distribution assumption

For each group of values, we take their mean and standard deviation and compute how large SF_{25} would be, if this group of values was exactly normally distributed. Figure 8 compares these theoretical values to the actual ones. As we see, the normal distribution assumption will almost always over-estimate the actual SF_{25} variability of a group, often by far. Thus, a normal distribution assumption is probably often not warranted.

5.2 Examples of actual distributions

Just to get a feel for the issues involved, let us look at a few examples of what such distributions actually look like. For small experiment groups we cannot get a clear view of the actual distribution, but for larger groups, a density estimator function (in our case based on Gaussian kernels) can provide us with a reasonable approximation of the actual distribution. Figure 9 shows the thus-estimated distributions of four groups from [12].

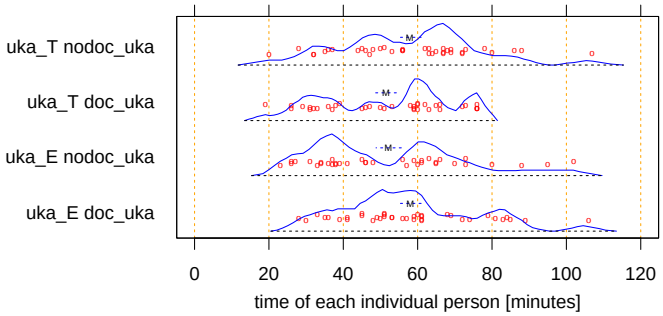

Figure 9: Estimated probability density functions for four groups from [12], each having between 36 and 38 subjects. Each dot represents one work time value.

The estimation uses a common rule for thumb for selecting the amount of smoothing in the estimation: For a sample X, the standard deviation of the Gaussian kernel functions is $2 \cdot (\max(X) - \min(X)) / \log_2(|X|)$. For a sample of which we expect that it may be normally distributed, but don't take this assumption for granted, this amount of smoothing reduces irritating discontinuities in the data, but still allows to see severe deviations from normality, if present. In our case, several interesting phenomena appear — different ones for different experiment groups: While the first (uppermost) distribution could be accepted as normal, the second one has too much weight on the right side, the third one even looks like it might have two peaks, and the fourth is too heavy on both sides of the peak. Unfortunately, we cannot be sure that these phenomena are real

for the underlying distribution, because samples of the given size sometimes look quite weird even if they *do* come from a normal distribution. However, a two-sided composite Kolmogorov/Smirnov goodness-of-fit test (KS-test) for normality returns p-values in the range 0.04 to 0.12 for the latter three distributions, suggesting that a normality assumption would be risky at least.

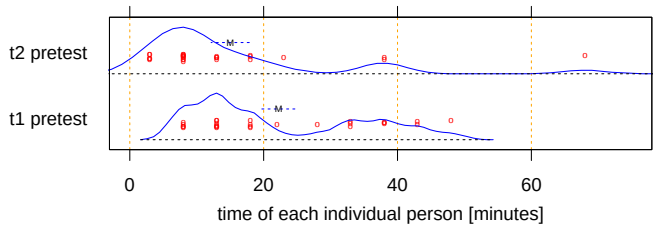

Figure 10: Estimated probability density functions for the two groups from [3], 27 subjects each. These times were originally reported as a histogram with 5-minute ranges only, hence the granularity.

So let us look at another example. The two groups from Figure 10 both have a right tail that is longer than in a normal distribution (KS-test $p < 0.002$). This is called positive skewness and is a common phenomenon for time data, because one can take arbitrarily long, but not arbitrarily short (0 is the limit). The lower distribution also looks like it might perhaps have two peaks. This could mean that the different subjects applied either of two different methods for solving the task. Each of the two methods results in a normal distribution, but for the less appropriate method, the mean of this distribution is higher than for the other.

It is plausible that positive skewness is typical of software engineering work time data. But so far this was only an assumption.

5.3 Estimating "natural" work time distributions

The large amount of data in our dataset, however, allows producing a fairly accurate estimate of the actual shape of the average work time distribution across a diverse set of individuals and tasks. For obtaining this estimation we normalize all our data by dividing each work time value by its experiment group average. The resulting values thus have a mean of 1 and will exhibit a certain "natural" variance structure — if such a thing exists.

Figure 11 shows the work time distributions of the resulting virtual groups by task type. With the exception of "review", all task types show a clearly positive skewness, which ranges from 1.25 for "maintain" (the bootstrapped 90% confidence interval is 0.86 to 1.64) up to 1.95 for "test/debug" (confidence interval 1.17 to 2.49). The skewness of "review" tends to be negative, but is unsure (confidence interval -1.42 to 0.19).

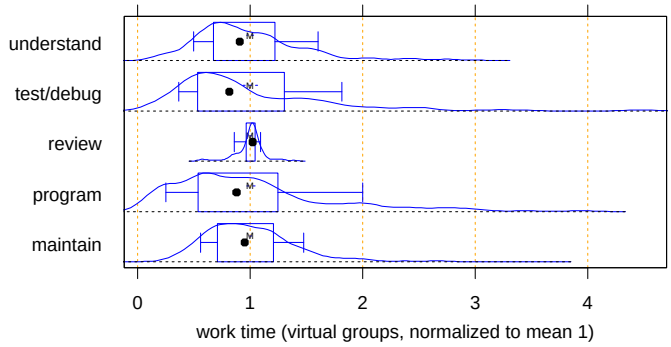

Figure 11: Estimated probability density functions for virtual groups created by normalizing each group mean to 1. These virtual groups consist of (top to bottom) 238, 118, 86, 236, and 780 data points, respectively.

5.4 Summary

The following can be said about the working time distributions found in `variance.data`:

- A long right tail is indeed a typical phenomenon for software engineering work time distributions. The typical skewness is about 1 to 2.
- The work time variability tends to be larger for task type "test/debug" ($SF_{50} = 2.4$, $SF_{25} = 3.2$) and even more for "programming" ($SF_{50} = 2.4$, $SF_{25} = 7.1$) than it is for "maintain" ($SF_{50} = 1.7$, $SF_{25} = 2.4$) or for "understand" ($SF_{50} = 1.8$, $SF_{25} = 2.9$).
- Task type "review" is special. It exhibits both low variability ($SF_{50} = 1.1$, $SF_{25} = 1.3$) and low skewness.

Note that in contrast to the means or medians from Figures 4 and 5, the above values of SF_{50} and SF_{25} correctly reflect the different group sizes within `variance.data`.

6 Effect sizes

Viewed from an engineering perspective, the purpose of experiments is identifying better ways of software development. Therefore, we want to know not just that one group performs better than the other, but also how large that difference actually is. If the difference is too small, the better technique may not be worth the effort required for introducing it. `variance.data` allows for asking "How large are the effects of the experiment variable on the average work time?" and to analyze a whole distribution of such effect sizes. We will look at these results in the present section.

6.1 Definition

What is the effect size? Given two experiment groups A and B and their work time measurement vectors t_A and t_B, let us

assume that A is faster on average. Then we can define effect size in two different ways: Either the relative difference of the means

$$E_1 : E := \frac{\overline{t_B}}{\overline{t_A}} - 1$$

or the absolute difference of the means in proportion to the pooled standard deviation

$$E_2 : E := \frac{\overline{t_B} - \overline{t_A}}{\sigma(t_{A \cup B})}$$

In the statistical literature, E_2 is more common, because it is closely related to the power of statistical tests. For practical purposes, however, E_1 is more relevant, because it directly tells us how much we can expect to gain by switching from method A to B. Furthermore, Greenland [23, p. 671] argues that E_2 can mislead in effect size comparisons. Hence, I will use E_1 below. However, we should be aware that higher variability in the samples increases the stochastic error when measuring E_1.

6.2 Expectations about effect size

We might expect the following:

- Some experiments do not find an expected work time effect and for many experiments the effect of the independent variable does not (or not mainly) influence work time. Therefore, we should expect to find a large proportion of small effect sizes.
- Furthermore, if we had a situation with a giant effect, we would not need a controlled experiment to assess it. Hardly anybody, for instance, would come up with the idea of comparing the working time required for coding a database query in SQL versus in assembler. It is just too clear that (and why) we will find a huge difference. Therefore, we should expect that the size of the largest effects in our effect size distribution is rather modest. Effect sizes of 0.1 to 0.4 appear realistic; much larger ones should be rare.

6.3 Effect size distribution

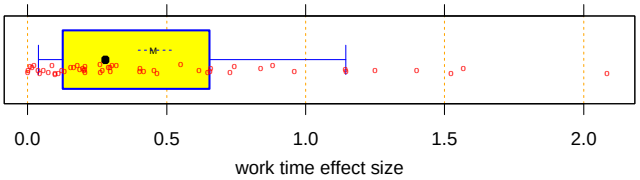

Figure 12: Effect sizes: Each point represents the quotient-minus-1 of the mean work times of the slowest versus the fastest group for each experiment task.

Given these expectations, the actual distribution of effect sizes, as shown in Figure 12 is surprising. Small effects exist, but are not as frequent as expected. Also, about one third of all effects is larger than 0.5, 10 percent are even larger than 1.0. Figure 13 shows the same data partitioned by task type.

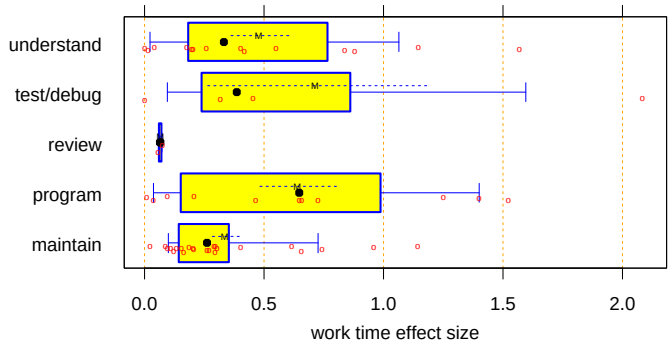

Figure 13: Work time effect sizes by task type.

What should we think of the effect size distribution? The modest fraction of small effects can partially be explained by the small size of most experiment groups as follows. The measured effect size is the sum of the true effect (caused by switching the experimental condition) and a random effect (caused by incidental group differences). The contribution of the random effect grows with increasing variability within the groups and with decreasing group size, because individual differences have less chance to balance out in smaller groups. And indeed we find that the effects tend to be larger for smaller groups; see Figure 14.

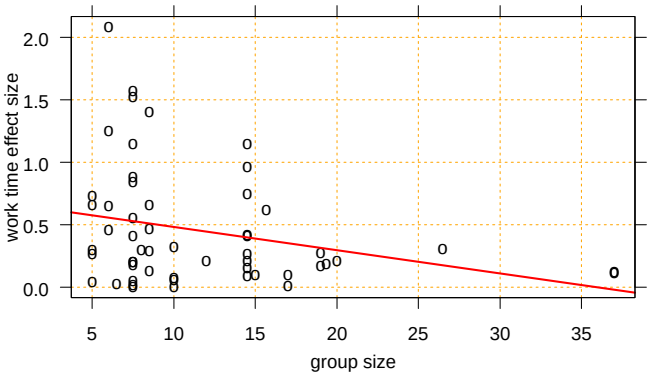

Figure 14: Effect size depending on group size. The effect has a random component which tends to be larger in smaller groups, as the regression line suggests.

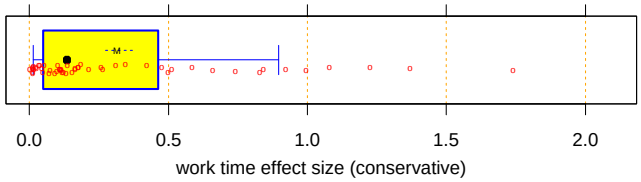

Figure 15: Effect size after approximate removal of the random component.

We can subtract the mean random effect (standard error of the effect size) from the effect size. This results in approximately the effect size distribution shown in Figure 15, which comes much closer to our expectations.

However, the very large effects in this distribution are quite impressive even after this correction. Such a large effect suggests that either the concrete task to be solved in the experiment was biased towards one of the experiment conditions or a very narrow aspect of software engineering has been measured in an unrealistically pure form.

But whatever we think of the extremes of the effect size distribution, its median can be used to make an important point. Only half of all experiments found an effect that was larger than 14 percent. Thus, an experimenter should not usually expect to see a larger effect and reviewers should not reject experimental research work just because its effect size is small. Most progress in software engineering, as everywhere, comes in rather small steps.

7 On the performance of statistical tests

In addition to the analyses discussed above, one can use the data for studying the behavior of different statistical tests on such software engineering data. Such tests form the core of the statistical evaluation in most articles about controlled experiments today[1] — in particular the t-test (which compares the means of two samples using a normal distribution assumption) or the Wilcoxon Rank Sum test (also known as Mann/Whitney U-test, which compares the medians of two samples assuming only that the distribution is non-discrete).

I have performed a comparative analysis for p-value distributions, actual type I errors, and the power of five different tests using the group pairs from `variance.data` as the empirical basis. Since this analysis has many technical caveats, I will not delve into the details here (please refer to [13] instead) and just point out the main conclusion: Depending on the actual data samples, any single test can sometimes mislead and it is therefore advisable to use several tests at once and present their results side-by-side in order to provide a broader and more reliable basis for understanding the data.

In particular, the Wilcoxon test is not just an inferior replacement for the t-test to be used if sample normality is violated or dubious, but rather is always a useful (sometimes even superior) complement. Since the Wilcoxon test assesses something quite different (namely medians instead of means), both tests should routinely be used together.

Such an approach also suggests that researchers should not be transfixed by a certain significance threshold such as 0.05. More reasonably, p-values should only be taken as one indicator among others and conclusions should be derived from a set of observations and analyses (numerical, graphical, and qualitative) that is as diverse as possible in a given situation.

[1] Note, that whenever clear group differences exist, confidence intervals should be provided, because they are much more useful information than bare hypothesis test results.

8 Summary and conclusions

The main findings from this investigation of the dataset `variance.data` can be summarized as follows:

- The interpersonal variability in working time is rather different for different types of tasks.
- More robust than comparing the slowest to the fastest individual is a comparison of, for example, the slowest to the fastest quarter (precisely: the medians of the quarters) of the subjects, called SF_{25}.
- The ratio of slowest versus fastest quarter is rarely larger than 4:1, even for task types with high variability. Typical ratios are in the range 2:1 to 3:1. The data from the Grant/Sackman experiment (with values up to 8:1) is rather unusual in comparison.
- Caveat: Maybe most experiments represented in `variance.data` underestimate the realistic interpersonal variability somewhat, because in practical contexts the population of software engineering staff will often be more inhomogeneous than the populations (typically CS students) used in most experiments.
- Still only little is known about the shape of working time distributions. However, `variance.data` exhibits a clear trend towards positive skewness for task types with large variability.
- The effect size (relative difference of the work time group means) is very different from one experiment to the next. The median is about 14%.
- The oft-cited ratio of 28:1 for slowest to fastest work time in the Grant/Sackman experiment is plain wrong. The correct value is 14:1.

As a consequence of these results, I suggest the following:

1. Although the data showed the individual variation to be lower than previously assumed, it is still much larger than the effect of the experimental variables. Therefore, it would be valuable to have simple and reliable tests that predict the performance of an individual for a certain kind of task. Experimenters could use such tests for proper grouping, blocking, or matching of their subjects, thus increasing the sensitivity of their experiments. Practitioners could use the same tests to optimize task assignments to project staff. After some interest in such tests in the 1960s (mostly for trainee selection), nobody appears to be working on this question any more.

2. Robust measures of variability such as SF_{25} and SF_{50} should be used more frequently for describing software engineering data. Reducing performance variability across persons can be an important contribution of a method or tool.

Acknowledgements

Thanks to all who offered or gave support obtaining datasets. Kudos to the owners of several other datasets for publishing their data. Thanks to Michael Philippsen for carefully ripping apart a draft of this article.

References

[1] Victor R. Basili, Scott Green, Oliver Laitenberger, Filippo Lanubile, Forrest Shull, Sivert Sørumgård, and M. Zelkowitz. The empirical investigation of perspective-based reading. *Empirical Software Engineering*, 1(2):133–164, 1996.

[2] Michelle Cartwright. An empirical view of inheritance. *Information & Software Technology*, 40(4):795–799, 1998. http://dec.bournemouth.ac.uk/ESERG.

[3] Bill Curtis. Substantiating programmer variability. *Proceedings of the IEEE*, 69(7):846, July 1981.

[4] John Daly. *Replication and a Multi-Method Approach to Empirical Software Engineering Research*. PhD thesis, Dept. of Computer Science, University of Strathclyde, Glasgow, Scotland, 1996.

[5] John Daly, Andrew Brooks, James Miller, Marc Roper, and Murray Wood. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering*, 1(2):109–132, 1996.

[6] Tom DeMarco and Timothy Lister. Programmer performance and the effects of the workplace. In *Proc. 8th Intl. Conf. on Software Engineering*, pages 268–272, London, UK, August 1985. IEEE CS Press.

[7] Thomas F. Dickey. Programmer variability. *Proceedings of the IEEE*, 69(7):844–845, July 1981.

[8] Pierfrancesco Fusaro, Filippo Lanubile, and Guiseppe Visaggio. A replicated experiment to assess requirements inspections techniques. *Empirical Software Engineering*, 2(1):39–57, 1997.

[9] E. Eugene Grant and Harold Sackman. An exploratory investigation of programmer performance under on-line and off-line conditions. *IEEE Trans. on Human Factors in Electronics*, 8(1):33–48, March 1967.

[10] Christian Krämer. Ein Assistent zum Verstehen von Softwarestrukturen für Java. Master's thesis, Fakultät Informatik, Universität Karlsruhe, June 1999.

[11] Christopher Lott. A controlled experiment to evaluate online process guidance. *Empirical Software Engineering*, 2(3):269–289, 1997.

[12] Lutz Prechelt. An experiment on the usefulness of design patterns: Detailed description and evaluation. Technical Report 9/1997, Fakultät für Informatik, Universität Karlsruhe, Germany, June 1997. ftp.ira.uka.de.

[13] Lutz Prechelt. The 28:1 grant/sackman legend is misleading, or: How large is interpersonal variation really? Technical Report 1999-18, Fakultät für Informatik, Universität Karlsruhe, Germany, December 1999. ftp.ira.uka.de.

[14] Lutz Prechelt and Georg Grütter. Accelerating learning from experience: Avoiding defects faster. *IEEE Software*, .(.):., . . Submitted April 1999.

[15] Lutz Prechelt and Walter F. Tichy. A controlled experiment measuring the impact of procedure argument type checking on programmer productivity. Technical Report CMU/SEI-96-TR-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, June 1996.

[16] Lutz Prechelt and Walter F. Tichy. A controlled experiment to assess the benefits of procedure argument type checking. *IEEE Trans. on Software Engineering*, 24(4):302–312, April 1998.

[17] Lutz Prechelt and Barbara Unger. A controlled experiment measuring the effects of Personal Software Process (PSP) training. *IEEE Trans. on Software Engineering*, .(.):., . . submitted September 1999.

[18] Lutz Prechelt and Barbara Unger. A controlled experiment on the effects of PSP training: Detailed description and evaluation. Technical Report 1/1999, Fakultät für Informatik, Universität Karlsruhe, Germany, March 1999. ftp.ira.uka.de.

[19] Lutz Prechelt, Barbara Unger, Michael Philippsen, and Walter Tichy. Two controlled experiments assessing the usefulness of design pattern information during program maintenance. *IEEE Trans. on Software Engineering*, .(.):., . . submitted August 1999.

[20] Lutz Prechelt, Barbara Unger, Michael Philippsen, and Walter F. Tichy. A controlled experiment on inheritance depth as a cost factor for maintenance. *IEEE Trans. on Software Engineering*, .(.):., . . submitted September 1999.

[21] Lutz Prechelt, Barbara Unger, and Douglas Schmidt. Replication of the first controlled experiment on the usefulness of design patterns: Detailed description and evaluation. Technical Report wucs-97-34, Washington University, Dept. of CS, St. Louis, December 1997. http://www.cs.wustl.edu/cs/cs/publications.html.

[22] Lutz Prechelt, Barbara Unger, Walter F. Tichy, Peter Brössler, and Lawrence G. Votta. A controlled experiment in maintenance comparing design patterns to simpler solutions. *IEEE Trans. on Software Engineering*, January 1999. Submitted. http://wwwipd.ira.uka.de/~prechelt/Biblio/.

[23] Kenneth Rothman and Sander Greenland. *Modern Epidemiology*. Lippincott-Raven, Philadelphia, PA, 2nd edition, 1998.

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

[25] Rainer Typke. Die Nützlichkeit von Zusicherungen als Hilfsmittel beim Programmieren: Ein kontrolliertes Experiment. Master's thesis, Fakultät für Informatik, Universität Karlsruhe, Germany, April 1999. http://wwwipd.ira.uka.de/EIR/.