The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really?

Lutz Prechelt (prechelt@ira.uka.de)
Fakultät für Informatik
Universität Karlsruhe
D-76128 Karlsruhe, Germany
+49/721/608-4068, Fax: +49/721/608-7343
http://wwwipd.ira.uka.de/EIR/

Technical Report 1999-18

December 20, 1999

Abstract

How long do different programmers 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 report presents the analysis of a much larger dataset of software engineering work time data with respect to the same question. It corrects the false 28:1 value, proposes more appropriate metrics, presents the results for the larger dataset, and presents results of several further analyses: distribution shapes, effect sizes, and the performance of various significance tests.

Contents

1 The 1966 experiment of Grant and Sackman 3
1.1 28:1 just isn't true! . . . 3
1.2 How should we measure variability? . . . 3
1.3 Only 12 programmers even after 30 years? . . . 5
1.4 Overview of this report . . . 5

2 `variance.data`: A larger dataset 5
2.1 Contents and structure . . . 5
2.2 Warning about the interpretation of the data . . . 7

3 The size of interpersonal work time differences 7
3.1 Slowest versus fastest individual . . . 7
3.2 Slowest versus fastest quarter . . . 8
3.3 Slower versus faster half . . . 8
3.4 Standard deviation relative to the mean . . . 10
3.5 Summary . . . 10

4 The shape of work time distributions 10
4.1 The normal distribution assumption . . . 11
4.2 Examples of actual distributions . . . 11
4.3 Estimating "natural" work time distributions . . . 12
4.4 Summary . . . 12

5 Effect sizes 13
5.1 Definition . . . 13
5.2 Expectations about effect size . . . 14
5.3 Effect size distribution . . . 14

6 Assessing different statistical tests for group differences 15
6.1 The set of tests considered . . . 16
6.2 p-values obtained . . . 16
6.3 Estimation of power and type I error . . . 18
6.4 Power and type I error of t-Test vs. Wilcoxon-Test . . . 18
6.5 Power and type I error overview . . . 20

7 Summary and consequences 21

Bibliography 24

1 The 1966 experiment of Grant and Sackman

In 1966, Grant and Sackman performed a controlled experiment for comparing online and offline programming. 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 [10], 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 a 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). This value of 28:1 interpersonal performance differences has since become quite famous. It is being quoted over and over [7].

There are three problems with this number:

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

We will now discuss each of these problems in order.

1.1 28:1 just isn't true!

The original publication [10] contains a complete table of the raw data, but appeared in a not-so-well-known journal. The second, more readily accessible source [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. The real differences are only half as large as claimed, but still justify the oft-stated "order of magnitude difference".

1.2 How should we measure variability?

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

A more appropriate 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.

	Coding		Debugging	
Wert	C Algebra	C Maze	D Algebra	D Maze
from [10]	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 [10]. 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.

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 as used historically would be $SF_0 := q_{100}/q_0$.

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

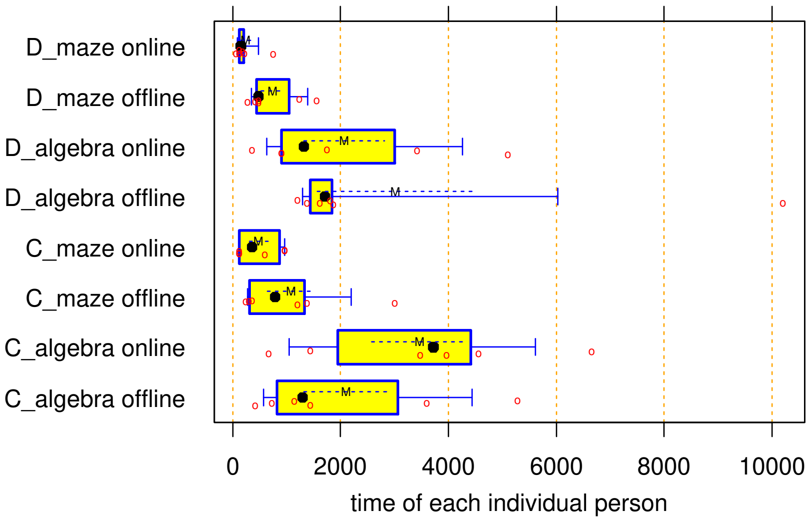

Figure 1: Work times of the individual subjects in the groups of the Grant/Sackman experiment. The box plot indicates the quantiles at 10%/90% (whiskers), 25%/75% (box), and 50% (fat dot), the mean (M), and the standard error of the mean (dashed line).

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

1.3 Only 12 programmers even after 30 years?

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

1.4 Overview of this report

The main subject of this report, interpersonal variability of work times in software engineering experiments, will be treated in Section 3, right after the dataset on which the investigation will be based has been introduced in Section 2.

Variability statistics characterize one aspect of the work time distribution by a single number. Section 4 will complement these statistics by an assessment of the overall *shape* of the distributions.

Another aspect, the size of the difference of average performance *between* groups (the so-called "effect size"), is discussed in Section 5.

Section 6 then discusses another related matter: the behavior and performance of various statistical tests (that compare means or medians of work times between two subject groups) for the pairs of groups found in `variance.data`.

Finally, the main observations are summarized and consequences are derived in Section 7.

2 `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.

2.1 Contents and structure

The dataset contains data from the following experiments: [15, 16], [13, 21, 19], [20], [22], [18, 17], [14], [25], [11], [10, 24], [5], [2], [1], [9], [3], [4], and [12]. 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 dataset 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 such value 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 is simply refers to either the experiment group or the control group, but sometimes there are more than two groups being compared.

- *task*: a symbolic name for the task to be solved. Groups of values that can be compared directly (as groups) are identical for *source* and *task* and are different for *group*.

- *type*: the kind of task. This value is always identical for all persons within one task. This variable partitions the whole dataset into experiment parts with similar properties. The following values exist:

  - maintain (understanding and modifying/extending),
  - understand (i.e., answering specific questions),
  - test/debug,
  - review (reviewing for finding defects),
  - program (design, implementation, testing/debugging),
  - design,
  - code (implementation).

  In case of doubt, either "program" or "maintain" will be used.

- *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 in this report.

In the analysis presented below, the data for task types "design" and "code" will be included in "program" to avoid too small type groups or dubious type group boundaries.

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.

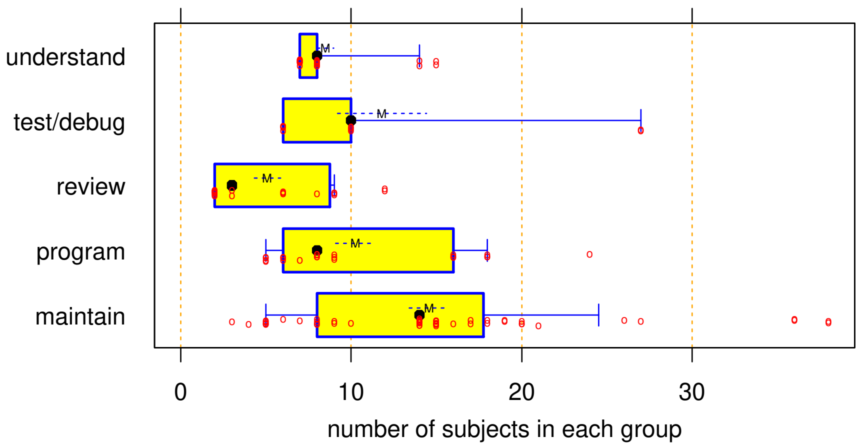

Figure 2: Distribution group sizes in `variance.data`. There is the following number of groups in each task type category: understand:28, test/debug:10, review:22 (but mostly very small), program:23, maintain:54.

2.2 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 work time is the only quantitative variable that is available for all of the 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 accuracy is hard to estimate.

The main message of the above is the following: Only in a few experiments have all participants worked until their task was solved completely and correctly. Therefore only in a few experiments is the work time information sufficient to characterize subject performance. In contrast, most experiments also contain data points in which the subjects reduced their work time at the expense of product quality — whether that was a conscious choice or not. On the other hand, we find similar behavior in real software development as well, hence we may consider the effect (and thus the time differences) realistic.

3 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.

3.1 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.

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. And before we even start to interpret too much into this figure, let us switch to a more meaningful representation: The ratio of the slowest to the fastest quarter.

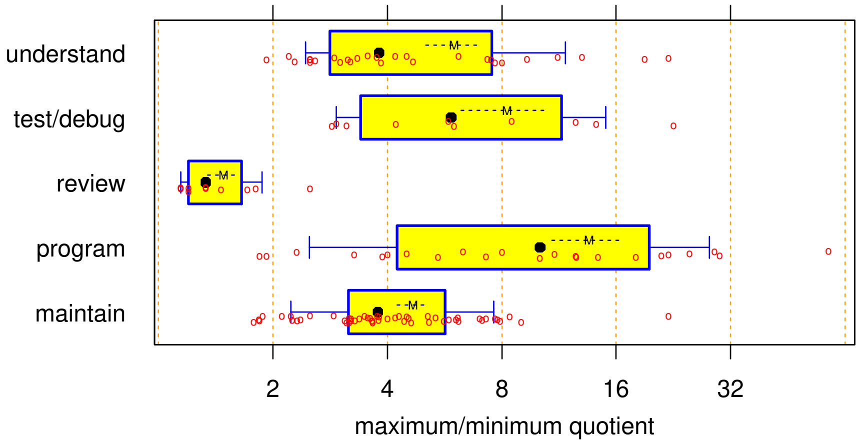

Figure 3: Distribution of the ratio SF_0 of the slowest to the fastest subject within each group. Each point represents one group of measurements for the same task under the same experimental conditions. Note the logarithmic scale.

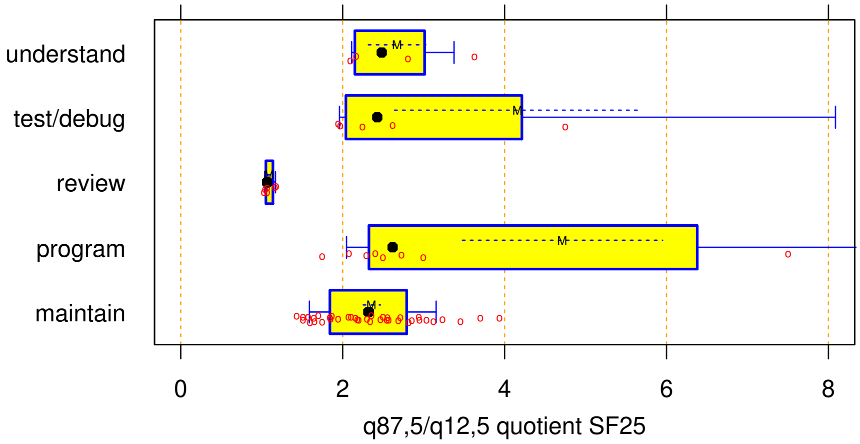

Figure 4: Distribution of the ratio SF_{25} of the medians of the slowest and fastest quarter of each group, for all groups with 9 or more subjects. One point of "test/debug" is at 11.4 (that is, outside of the plot); likewise two points of "programming" at 10.9 and at 12.

3.2 Slowest versus fastest quarter

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

3.3 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.

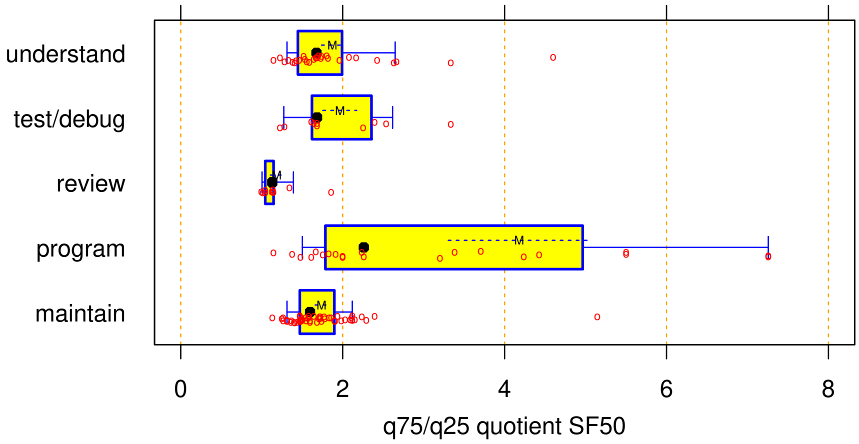

Figure 5: Distribution of the ratio SF_{50} of the medians of the faster and slower half of the subjects for each group.

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. I conclude that these values may be misleading and atypical for more recent tasks and subject populations. However, a different explanation is possible as well. 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 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 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.

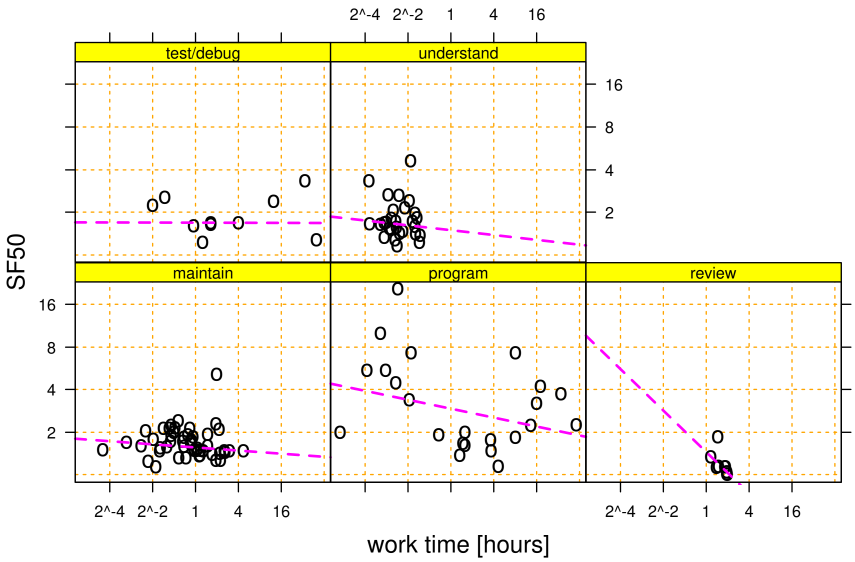

Figure 6: SF_{50} depending on the mean work time of each experiment group. The dashed trend line is a robust L_1 (minimum absolute distance) regression line. Both axes are logarithmic.

A median SF_{50} of about 2 matches well will 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 under very different local work conditions and SF_{50} was 1.9.

3.4 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.

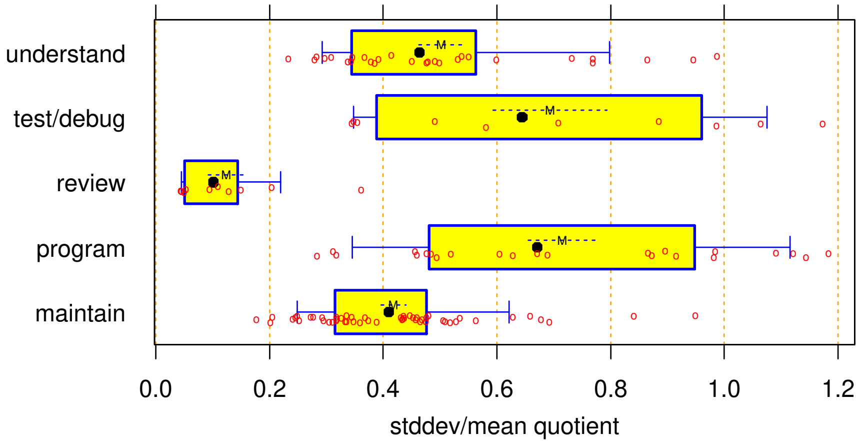

Figure 7: Distribution of the ratio of standard deviation and arithmetic mean for each group.

3.5 Summary

In summary we can say that *typical* work time differences between slower and faster programmers are more on the order of 2:1 — much less impressive than the oft-cited 28:1. Still, these differences are much larger than the typical differences between two different experimental conditions (see Section 5) and are hence clearly worthy our attention in terms of improving the software development process.

4 The shape of work time distributions

Beneath summary indicators of variability such as the ones presented, a more general question is lurking: 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 correct and efficient statistical inference in controlled experiments, more accurate schedule and staff planning in projects, improved risk detection in projects, etc.

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

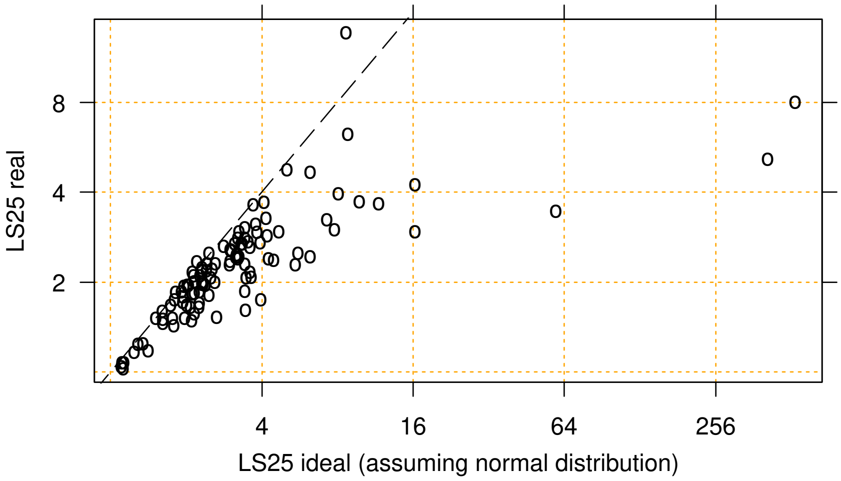

Figure 8: Comparison of the actual value of SF_{25} for each experiment group with a theoretical value of SF_{25} based on a normal distribution assumption and computed from mean and standard deviation. The axes are logarithmic.

4.1 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. Obviously, a normal distribution assumption is often not warranted.

4.2 Examples of actual distributions

So 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 [13].

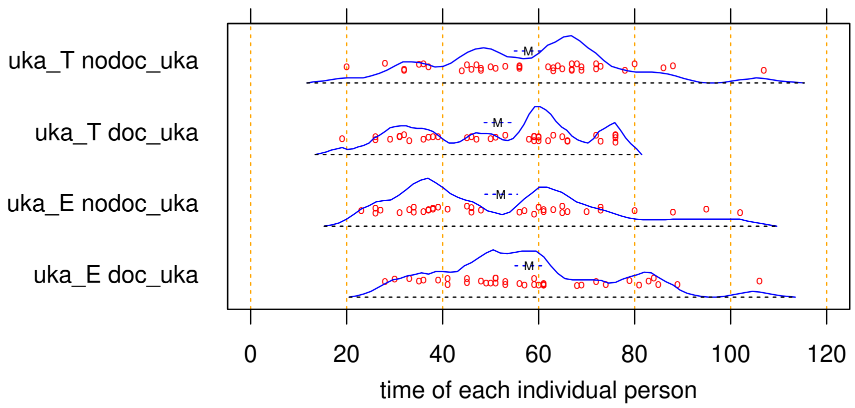

Figure 9: Estimated probability density functions for four groups from [13], each having between 36 and 38 subjects. Each dot represents one work time value.

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, fairly interesting phenomena appear — different ones for different experiment groups: While the first (uppermost) and fourth distribution could be accepted as normal, the second one has too much weight on the right side and the third one even looks like it might have two peaks. 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.

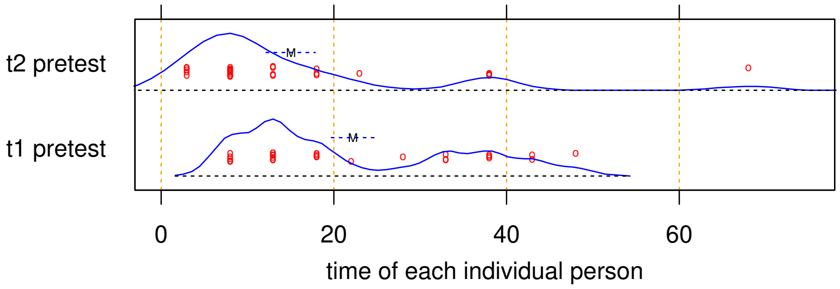

Figure 10: Estimated probability density functions for the two groups from [3], 27 subjects each. The times were reported as ranges instead of with a precision of one minute.

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. 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.

4.3 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 accross a diverse set of programmers 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 show 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 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).

4.4 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.

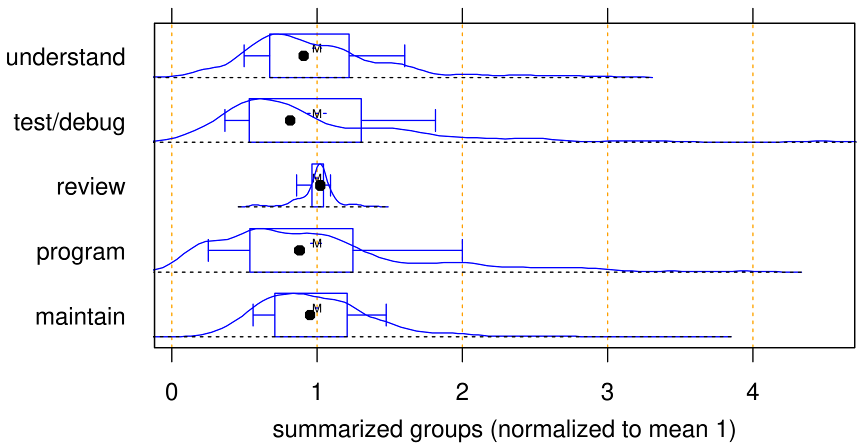

Figure 11: Estimated probability density functions for virtual groups created by normalizing each group mean to 1. These virtual groups consist of (top to bottom) 238, 118, 86, 236, and 780 data points, respectively.

- The work time variability tends to be larger for task type "test/debug" (mean $SF_{50} = 2.4$, $SF_{25} = 3.2$) and even more for "programming" (mean $SF_{50} = 2.4$, $SF_{25} = 7.1$) than it is for "maintain" (mean $SF_{50} = 1.7$, $SF_{25} = 2.4$) or for "understand" (mean $SF_{50} = 1.8$, $SF_{25} = 2.9$).
- Inspections are special. They exhibit both low variability (mean $SF_{50} = 1.1$, $SF_{25} = 1.3$) and low skewness.

5 Effect sizes

Viewed from an engineering perspective, the purpose of experiments is identifying better ways of software development. Therefore, we want to find 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.

5.1 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 one method to another. Furthermore, Greenland [23, S. 671f] 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.

5.2 Expectations about effect size

We might expect the following:

- For one, some experiments do not find an expected work time effect and for many experiments the effect 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 comparison 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 time required for coding a database query in SQL versus in assembler. 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.

5.3 Effect size distribution

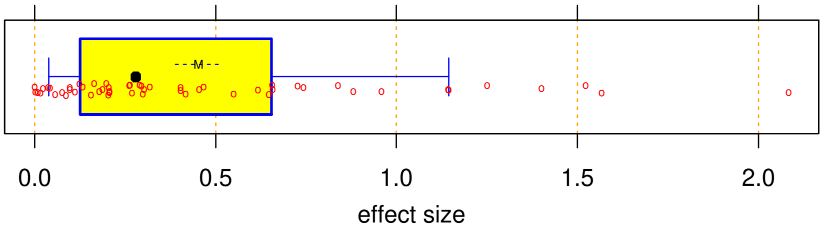

Figure 12: Effect sizes: Each point represents the quotient-minus-1 of the mean work times of the slowest versus the fastest group for each experiment task.

Given this expectation, 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.

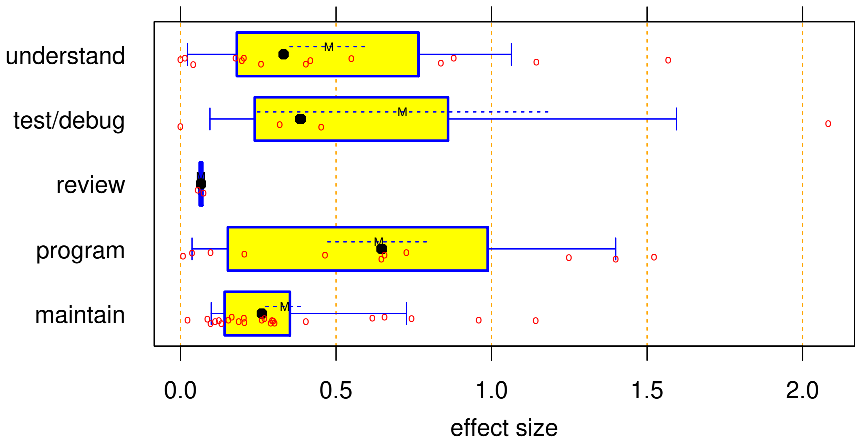

Figure 13: Work time effect sizes by task type.

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.

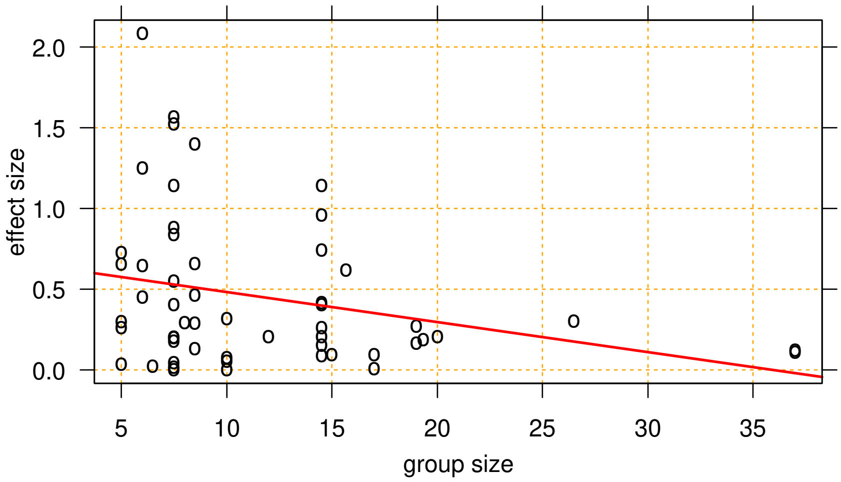

Figure 14: Effect size depending on group size. The effect has a random component which is larger in smaller groups, as the regression line shows.

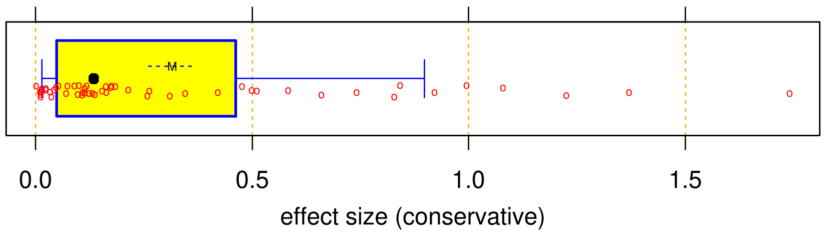

Figure 15: Effect size after approximate removal of the random component.

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, is made in rather small steps.

6 Assessing different statistical tests for group differences

For a majority of all controlled experiments (in software engineering as well as elsewhere) a statistical hypothesis test for differences between the group means or the group medians is the core of the statistical evaluation. Even though work time is not always the most important (let alone the only) performance measure, `variance.data` provides a nice opportunity to review the p-values returned by various tests. What fraction of tests will be successful? How do the results for different kinds of tests differ?

6.1 The set of tests considered

In this and the following section, we will consider five different tests for comparing two samples x and y:

- The standard t-Test for differences in the means of independent samples. This test assumes the samples are each taken from a normal distribution and that the variances of these distributions are equal. These assumptions will more or less inappropriate for some of the group pairs.
- The "optimized" t-Test. This is like the standard t-Test, except that the either the sample values themselves or their square roots or their logarithms will be used for the testing, whichever transformation leads to the best approximation for the normality assumption (as indicated by lower resulting p-values). Furthermore if the variances of the (possibly transformed) samples differ by more than factor 2, a Welch correction for unequal variance ($t = \frac{\bar{x}-\bar{y}}{\sqrt{\sigma(x)/|x|+\sigma(y)/|y|}}$) will be applied to the test.
- The Wilcoxon Rank Sum Test (also known as Mann/Whitney U-Test) for differences in the median of two samples. The only assumption is that both samples must come from a continuous distribution.
- A Bootstrap-based test for differences in the group means. This test, like the Wilcoxon-Test, works without any strong assumptions but, unlike the Wilcoxon-Test, still compares means, not medians. It works by repeatedly drawing samples (with replacement) of size $|x|$ from x and of size $|y|$ from y, computing an empirical Bootstrap distribution for the difference of their means, and reading the p-value from this distribution as the area beyond zero. This is the so-called percentile method for Bootstrap tests. Confidence intervals for the difference can also be determined directly from this distribution. For an introduction into Bootstrap techniques, see [8].
- The "optimized" Bootstrap-based test. Like above, but tests the square roots or logarithms of the values if the optimized t-Test recommends it.

Although the t-Test is known to be fairly robust against deviations from normality in the data, such deviations will make the test less sensitive (i.e. reduce its power). Many researchers thus use the Wilcoxon-Test if they have reason to believe such deviations exist. However, the Wilcoxon-Test compares medians, not means, which is often of less interest. Furthermore, the Wilcoxon test has allegedly lower power (i.e. a lower probability of actually rejecting a false null hypothesis). Overall, it is quite interesting to compare the t-Test and the Wilcoxon-Test directly for a number of pairs of work time samples from the software engineering domain and to compare the other tests to both as possible alternatives.

6.2 p-values obtained

Let us first compare the t-Test and Wilcoxon-Test directly; see Figure 16. There are severe differences in individual cases, but on the average the two tests perform quite similarly.

Figure 17 shows the distribution of p-values obtained from two-sided tests for all those pairs of groups whose effect size is shown in Figure 12. The success rate is not very large. Only about 20% to 30% of all tests come out below the most common threshold of $p < 0.05$. If one-sided tests are used, we must divide the plotted values by 2 and about 30% to 40% of all tests will be successful.

The Bootstrap-Tests typically return lower p-values than both t-Tests and Wilcoxon-Test and will hence reject the null hypothesis more frequently. This can be seen more clearly in Figure 18. The Wilcoxon-Test exhibits

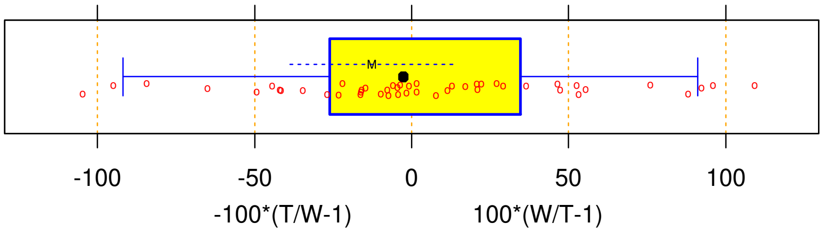

Figure 16: Comparison of the p-values returned by the t-Test (T) and the Wilcoxon-Test (W). The plots shows the difference (in percent) of the larger relative to the smaller value. This differences is plotted to the positive side if W>T, to the negative side otherwise. A few extreme values lie outside of the plotted area.

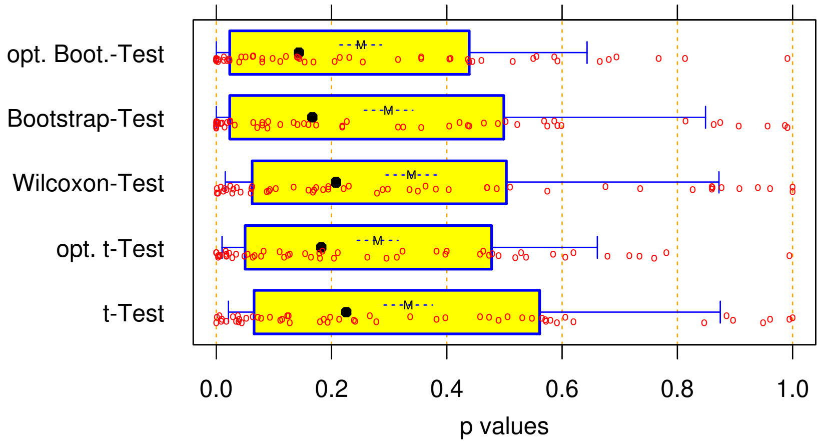

Figure 17: Distribution of the p-values returned by the various (two-sided) tests. Each point represents the comparison of one pair of groups.

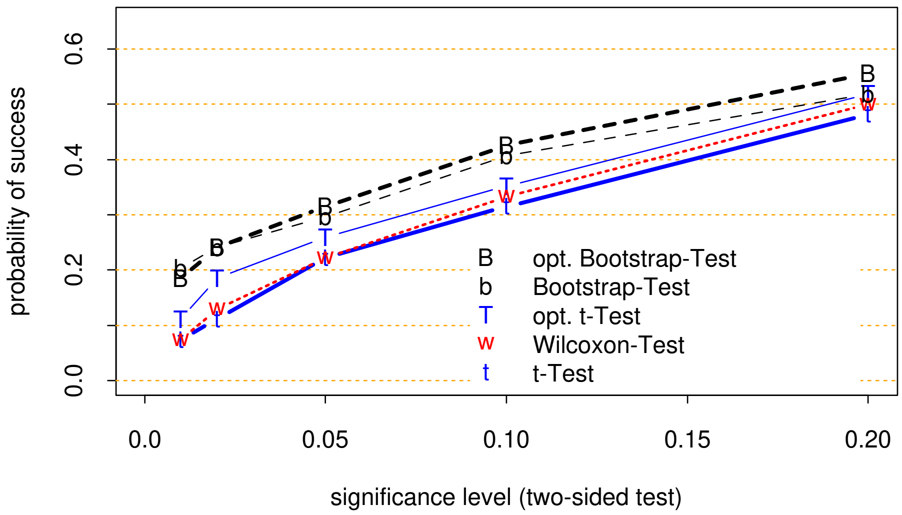

Figure 18: Fraction of tests that reject the null hypothesis on different significance levels (0.01; 0.02; 0.05; 0.1; 0.2). Each point represents the success rate of all tests for all pair-wise group comparisons.

almost the same success rate than the standard t-Test. Unlike the claims often seen in experimental papers, experiments will *on the average* not lose power if the Wilcoxon-Test is used instead of the t-Test.

6.3 Estimation of power and type I error

There are two ways how a statistical test may give an incorrect result:

- Type I Error: The null hypothesis is being rejected although there is no real difference indeed.
- Type II Error: The null hypothesis is not being rejected although a real difference in fact exists.

The probability that either of these errors occurs in a statistical test is often called α or β, respectively.

α is identical to the chosen significance level if the test functions perfectly. In practical situations, however, there is usually some difference, because the assumptions of the test are fulfilled only approximately.

β decreases with increasing α, increasing sample size and and increasing appropriateness for the test with respect to the actual distribution of the data. $1 - \beta$ is called the *power* of the test.

For a given pair of samples, we can use Bootstrapping to compute both α and $1 - \beta$ (power) in the following fashion. Assume we want to assess the t-test for two samples x and y with different mean. Now the trick is to consider these samples to be the population itself, so that we *know* that a means differences in the population does exist and thus also know that the test should reject the null hypothesis. To compute the power, we do now repeatedly (say, 500 times) draw a resample from x (that is, a sample of size $|x|$ drawn from x with replacement) and a resample from y, apply the t-test to the resample, and count in which fraction of the cases the test rejects the null hypothesis for the resamples at a given fixed significance level. This fraction is the Bootstrap estimate for the power of the test. Likewise, we can compute a modified sample $y' := y - \bar{y} + \bar{x}$ such that we know that no means difference between the populations x and y' exists (so that the test should never reject the null hypothesis) and apply the same process described above to the samples x and y' to produce an estimate for the type I error α.

Note that the values for α and power obtained by this method will only be approximations, because for small sample sizes such as ours, equating the sample with the population is a rather crude operation.

For assessing the Wilcoxon test in this manner, there are two complications. First, we must align the medians (instead of the means) for the estimation of α. Second, we must add a small amount of random noise to each data value of each resample, because the Wilcoxon test in its default form cannot cope with ties in a sample but resamples usually contain ties. The noise must be smaller than the minimum difference between any two values from the unified sample $x \cup y$. Alternatively, we can use a modified form of the Wilcoxon test that applies a normal-theory approximation to accomodate the presence of ties in the samples. The difference of the results of these two methods is quite small.

6.4 Power and type I error of t-Test vs. Wilcoxon-Test

Now let us apply this method to the various pairs of samples in our dataset and analyze the results.

Let us first consider a direct comparison of t-Test and Wilcoxon-Test again. For power (Figure 19) the behavior is quite similar to that we found for the p-values. Large differences exist in individual cases, but the average performance is almost the same for both tests.

The Bootstrap estimates of the actual type I error, however, are surprising; see Figure 20. First, most errors are fairly far away from the nominal 0.05. Second, in most cases the differences are rather different for the two tests; the correlation is only 0.42. The lesson to be learned from this plot is that relying on a single kind of test is dangerous.

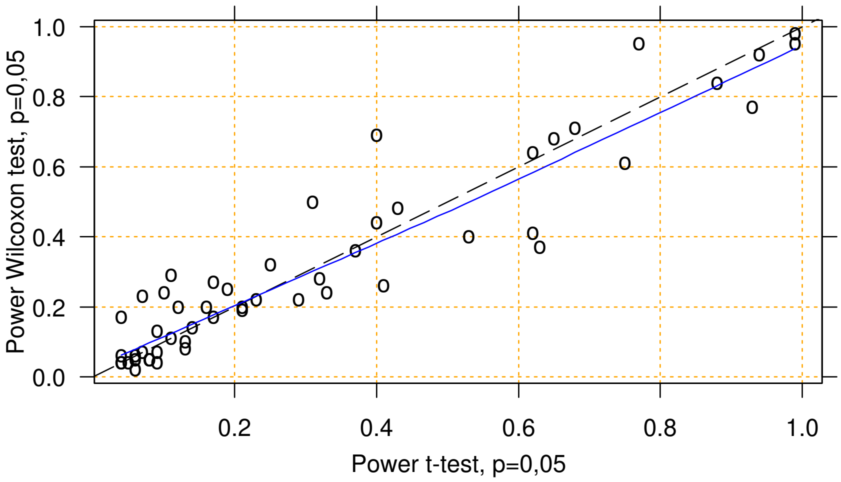

Figure 19: Comparison of the Bootstrap-estimated power of the t-Test and the Wilcoxon-Test for $p = 0.05$. Each point represents one pair of groups that were compared. The dashed line indicates $x = y$, the full line is a local regression trend line.

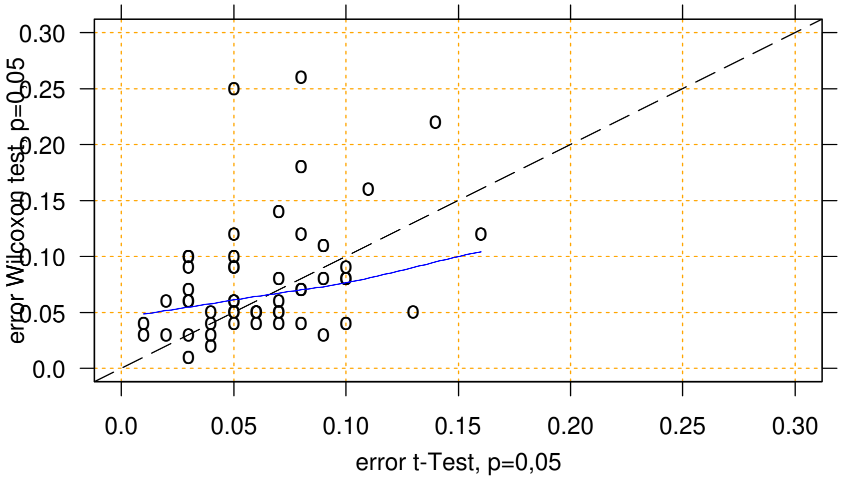

Figure 20: Comparison of the Bootstrap-estimated actual type I errors made by the t-Test and the Wilcoxon-Test applied with a nominal significance level of $p = 0.05$. Each point represents one pair of groups that were compared. The dashed line indicates $x = y$, the full line is a local regression trend line.

6.5 Power and type I error overview

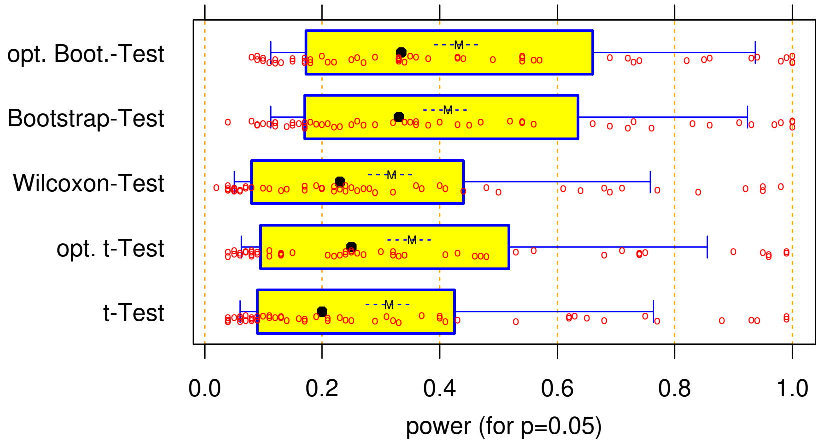

Figure 21: Distribution of the Bootstrap estimates of the power of the different tests at a nominal significance level of $p = 0.05$. The estimates are based on 100 pairs of Bootstrap resamples. Each point represents one pair of groups that were compared.

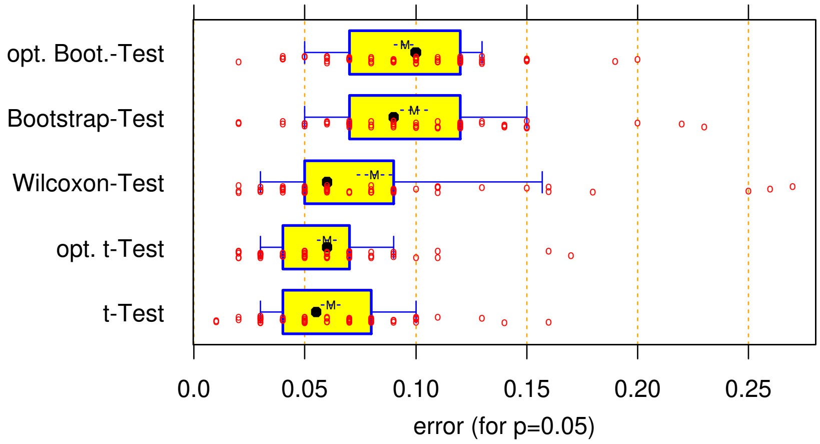

Figure 22: Distribution of the Bootstrap estimates of the actual type I errors for the different tests at a nominal significance level of $p = 0.05$. The estimates are based on 100 pairs of Bootstrap resamples. Each point represents one pair of groups that were compared.

Looking at the distribution of power values (at nominal signficance level 0.05) for all tests at once (Figure 21) we arrive at the same conclusions as after the analysis of the p-values obtained. The power of the Boostrap-based tests is indeed somewhat higher than the power of the conventional tests. Due to the large number of cases, for example the difference in median power between Bootstrap-Test and optimized t-Test is highly significant with $p = 0.001$.

However, looking at the type I errors (for the same nominal significance level of 0.05, shown in Figure 22) we see the downside of this difference: Along with the high power of the Bootstrap-based test we buy a lower reliability. The actual error probabilities obtained by, for instance, the optimized Bootstrap-Test are almost twice as large as the expected ones. The other tests are too optimistic as well (which may indicate some bias introduced by the Boostrap error estimation procedure), but less so.

For all tests the variability in the actual error is rather conspicous. To some degree this can be explained by the small size of the groups in many of the experiments in our dataset. Small groups result in higher fragility of both the statistical tests themselves and the Bootstrap resampling used for computing the errors.

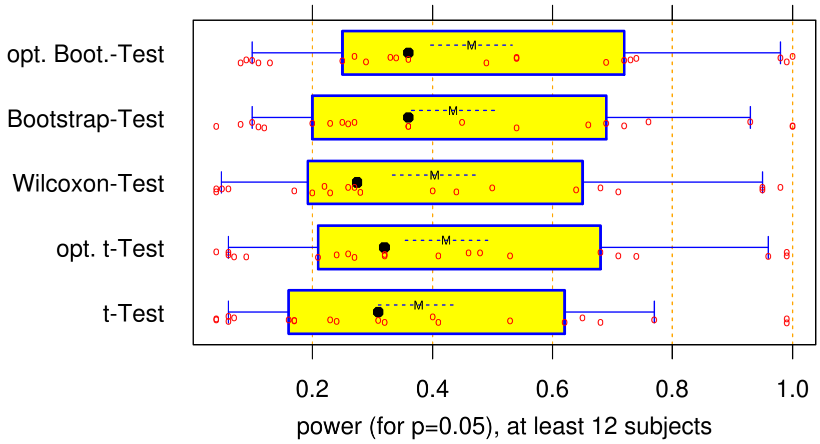

Figure 23: Like Figure 21, but shows only the data for groups of 12 or more subjects.

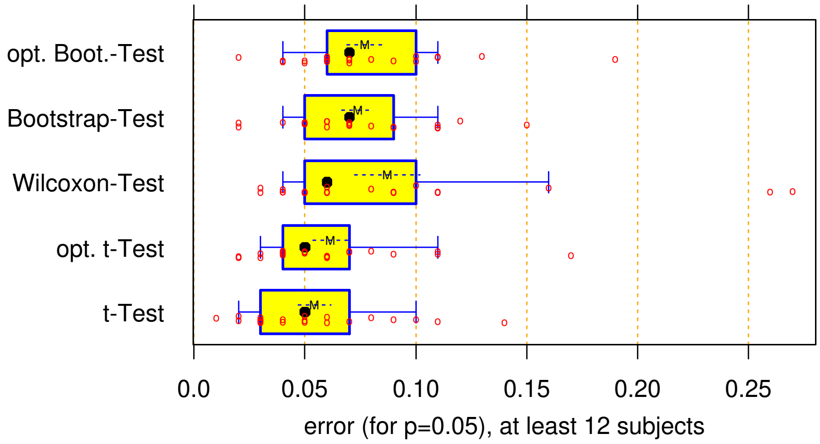

Figure 24: Like Figure 22, but shows only the data for groups of 12 or more subjects.

If we remove the small groups from the evaluation and look at the results for groups of at least 12 subjects only, we obtain somewhat higher power (Figure 23) and more modest distributions of error (Figure 24). The average power, however, is still below 0.4. Only a minority of all experiments can have found significant differences for working time.

These trends stay fairly consistent if we change the nominal significance level, as Figures 25 (for power) and Figure 25 (for type I error) show. There is one exception, though: The Wilcoxon test performs much better with respect to its error at smaller nominal significance levels (strict tests) compared to larger levels (loose tests).

7 Summary and consequences

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

- 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.
- More appropriate 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}.

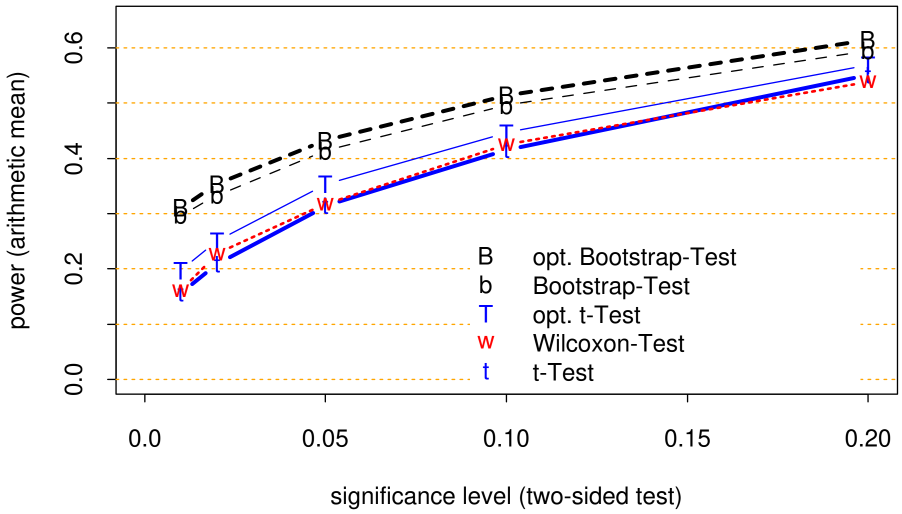

Figure 25: Mean power of the tests for different nominal significance levels. Each point represents the arithmetic mean from all pairwise group comparisons using one kind of test.

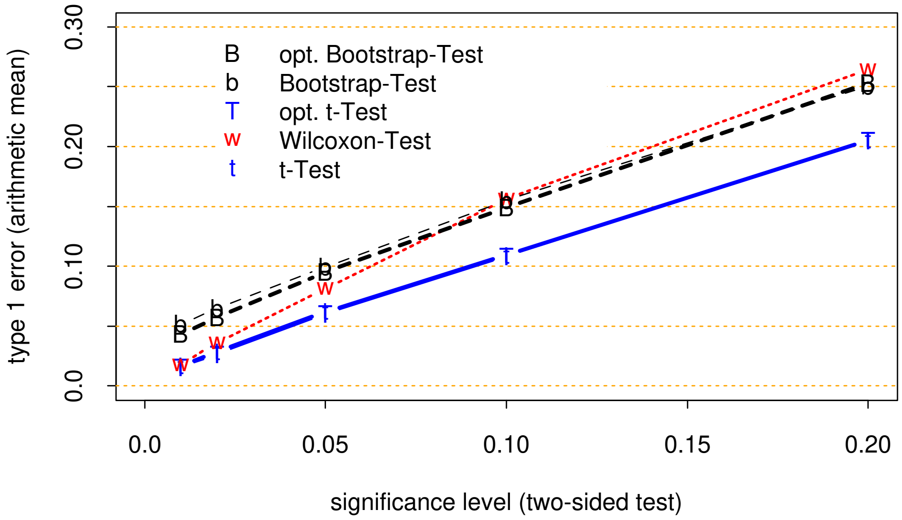

Figure 26: Mean size of the type I error for all tests at different nominal significance levels. Each point represents the arithmetic mean from all pairwise group comparisons using one kind of test.

- The variability in interpersonal performance is rather different for different types of tasks.
- SF_{25} is rarely larger than 4, even for task types with high variability The data from the Grant/Sackman experiment (with values up to 8) looks almost like an outlier in comparison.
- Caveat: Maybe the experiments represented in `variance.data` underestimate the interpersonal variability somewhat, because in practical contexts the programmer population 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. For some tasks we may expect to see two-peaked or n-peaked distributions due to different approaches with non-equivalent suitability taken by different persons. 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 de-biased median is 14%.

- The Boostrap-based test for differences of group means appears to have (on average) higher power but also higher type I error than conventional tests such as t-Test and Wilcoxon Rank Sum Test.
- The type I error of all tests varies a lot from one application to another — not necessarily in the same way for each kind of test.

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.
3. The dubious reliability of any single kind of statistical test seen in our dataset suggests to always present the results of several different test side-by-side when analyzing work time data.
4. For the same reason, 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 (both quantitative and qualitative) that is as diverse as possible in a given situation.

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] Bradley Efron and Robert Tibshirani. *An introduction to the Bootstrap*. Monographs on statistics and applied probability 57. Chapman and Hall, New York, London, 1993.

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

[10] 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.

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

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

[13] 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.

[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/.