Busting the Myths of Programmer Productivity

William Nichols

Software Engineering Institute

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM20-1140

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Measurement of Productivity Differences Replicates!

"One of the most replicated results in software engineering research is the 10-fold difference in productivity and quality between different programmers with the same levels of experience." – Steve McConnell [McConnell, 2002]

Sources	
[Sackman, 1968]	[Card 1987]
[Curtis 1981]	[Boehm and Papaccio 1988]
[Mills 1983]	[Valett and McGarry 1989],
[DeMarco and Lister 1985]	[Boehm 1975, 2000]
[Curtis et al. 1986]	[Schwartz, 1968]

Reported Productivity ranges of 5-1 to 28-1

Why Does Productivity Range Matter?

Huge productivity ranges between programmers should affect

- Hiring - we want the best!
- Training – we make the best!
- Evaluation – we only keep the best!
- Promotion – we recognize the best!
- Pay – we pay the best!
- Planning – we need the best!

Questions

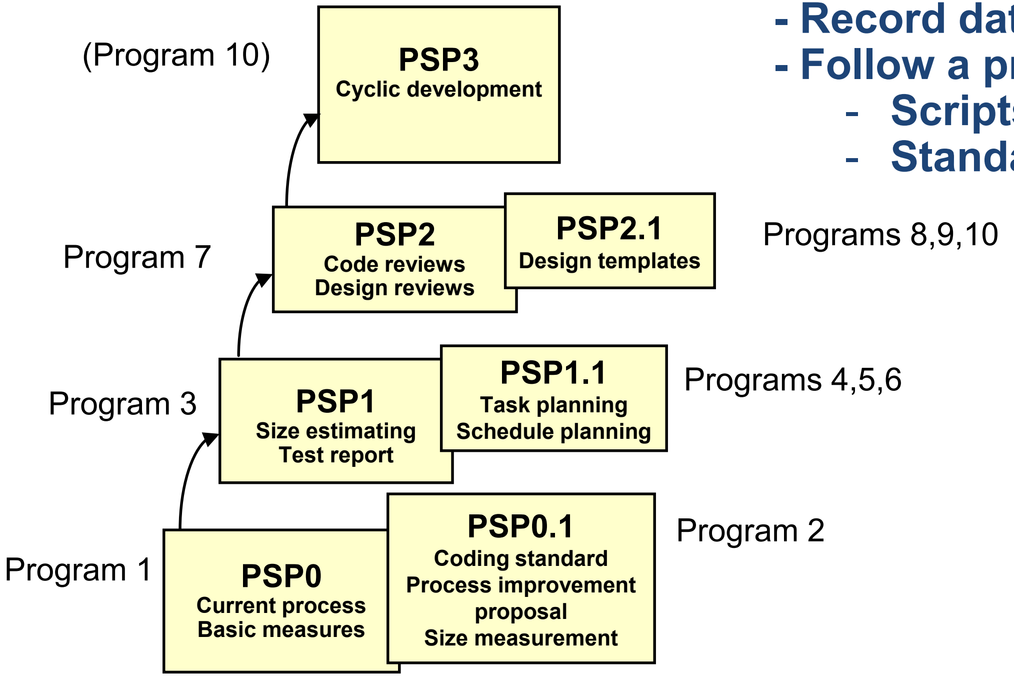
1. What factors (experience, education, the individual) account for the variation?
2. Does our data replicate the scale of performance range?
3. What practical effect will the differences have?

Approach

Sponsored by the SCOPE SEI research project, to investigate potentially controllable causes of software development cost.

Use the performance data collected during the Personal Software Process (PSP) course.

PSP Course Students Write 10 Programs



PSP components
- **Record data with forms and logs**
- **Follow a process with defined**
    - **Scripts, and**
    - **Standards**

Programs vary in size, but typically take an afternoon to complete.

PSP Data

This version PSP course was taught for about 10 years, primarily to professionals.

Each programmer developed the same 10 programs.

The programming data includes

- 3140 developers and 31,140 programs
- 3,355,882 lines of code
- 123,996.53 hours of work
- 221,346 defects

We also collected demographic data for experience, education, and so forth.

Of these, 494 include complete sets of 10 programs written in “C”.

We will only look at these data.

Data Selection And Analysis

Limit context to reduce confounding.

- Focus only on Programming Effort (limit the scope).
- Limit sample to C Programmers (constrain confounders).

Analysis

- Apply causal search techniques (find direction of causality).
- Apply Linear Hierarchical Models and SEM to measure effect size and quantify variation.

Causal Search Found "Something about Students"

1) There were no correlations (let alone causation) between experience, education level, or statistics background and productivity.
2) Evidence that Student (j) caused outcome on Assignment (i)

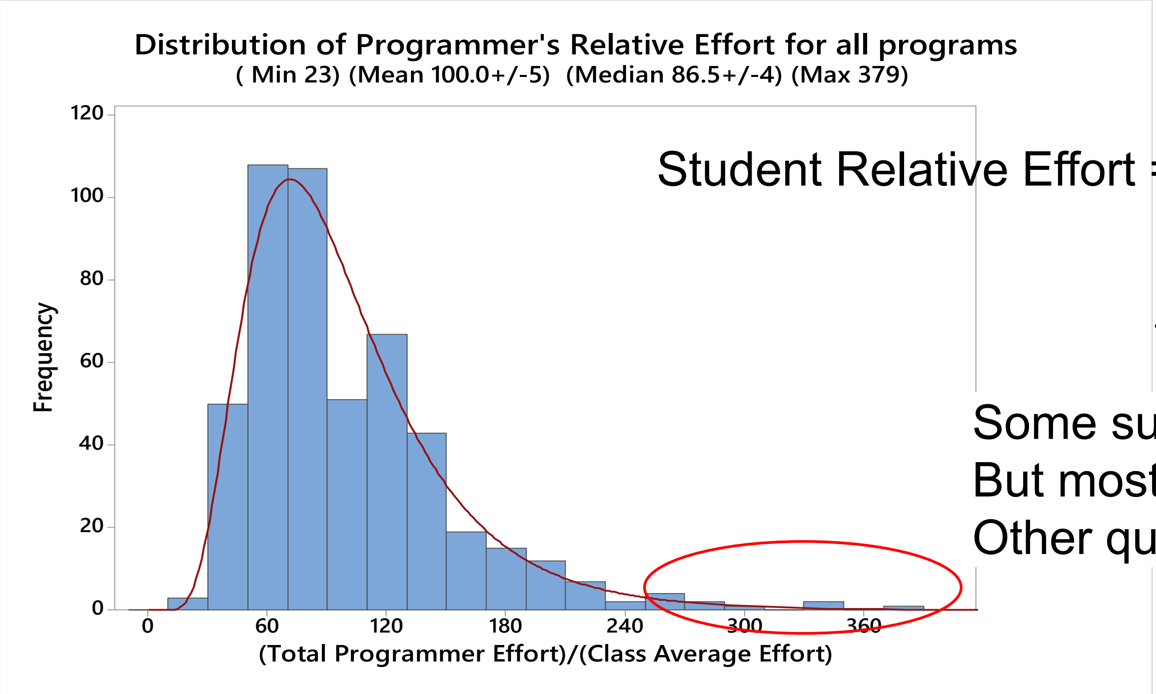
$$ProgramSize[LOC]_{ij} = ReqSize_i \times StudentSizeFactor_j$$
$$Defects_{ij} = ProgramingEffort_{ij} \times StudentDefectRate_j$$
$$ProgramEffort_{ij} = RequirementSize_i \times StudentEffortFactor_j$$

3) But the "within programmer" variation was surprisingly strong.

Error terms from mixed model (ANOVA) and SEM showed individual programmer results were only almost as variable as between programmers.

Followed up with closer look at the variation.

Distribution of Total Programmer Relative Effort



Student Relative Effort = $\dfrac{\text{Individual Effort}}{\text{Average Effort}}$

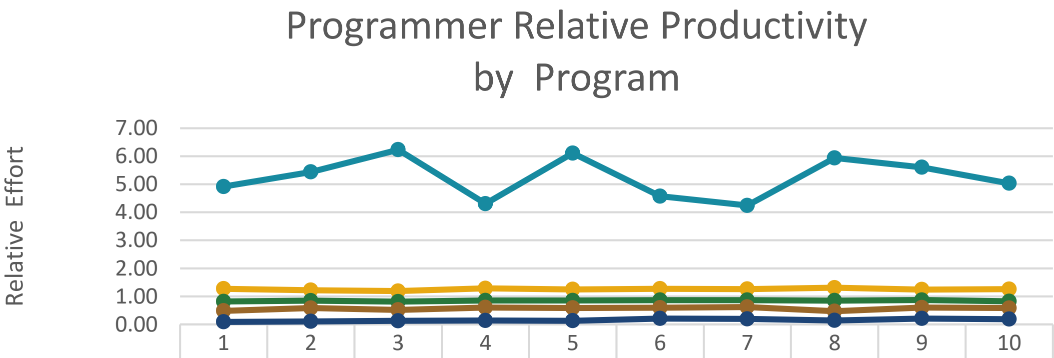
Total for all programs

Some support for x10 range
But mostly relies on outliers.
Other questions raised.

	Mean	Standard deviation	5%	25%	Median	75%	95%
Relative effort	1	0.51	0.44	0.63	0.86	1.24	1.93

Quartiles of Relative Effort for the 10 Programs
Run Chart

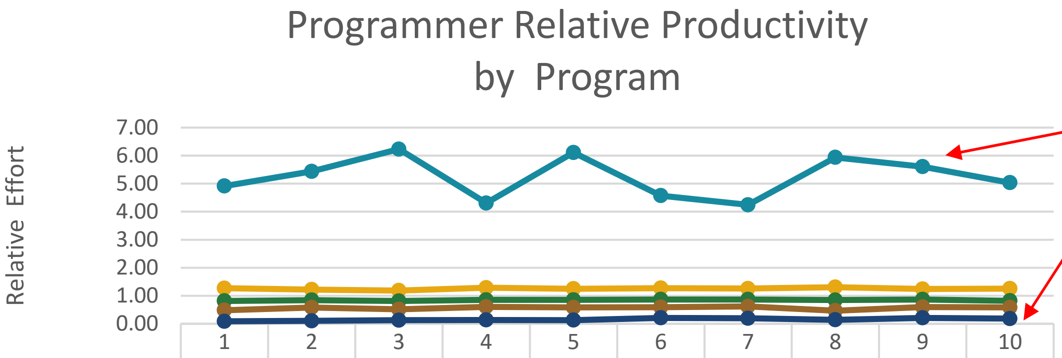
$$\textbf{Relative Effort} = \frac{\text{Individual Effort}}{\text{Average Effort}}$$



	1	2	3	4	5	6	7	8	9	10
Maximum	4.92	5.44	6.23	4.31	6.11	4.57	4.24	5.94	5.60	5.03
Q3	1.27	1.22	1.19	1.29	1.25	1.27	1.26	1.31	1.24	1.26
Median	0.82	0.85	0.81	0.85	0.85	0.87	0.87	0.85	0.87	0.82
Q1	0.48	0.58	0.52	0.61	0.59	0.59	0.62	0.47	0.60	0.59
Minimum	0.09	0.11	0.13	0.14	0.13	0.21	0.20	0.14	0.21	0.19
Max/Min	54.9	51.1	48.5	31.1	46.5	21.8	21.5	41.5	26.6	26.9
Q3/Q1	2.6	2.1	2.3	2.1	2.1	2.1	2.0	2.8	2.1	2.1

Mean = 1.0

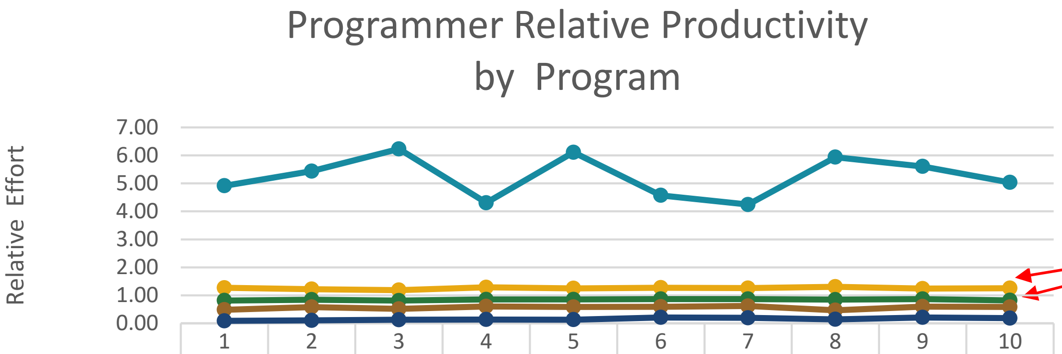
Min/Max of Relative Effort by Program Are Outliers



	1	2	3	4	5	6	7	8	9	10
Maximum	4.92	5.44	6.23	4.31	6.11	4.57	4.24	5.94	5.60	5.03
Q3	1.27	1.22	1.19	1.29	1.25	1.27	1.26	1.31	1.24	1.26
Median	0.82	0.85	0.81	0.85	0.85	0.87	0.87	0.85	0.87	0.82
Q1	0.48	0.58	0.52	0.61	0.59	0.59	0.62	0.47	0.60	0.59
Minimum	0.09	0.11	0.13	0.14	0.13	0.21	0.20	0.14	0.21	0.19
Max/Min	54.9	51.1	48.5	31.1	46.5	21.8	21.5	41.5	26.6	26.9
Q3/Q1	2.6	2.1	2.3	2.1	2.1	2.1	2.0	2.8	2.1	2.1

Program Max/Max extreme and variable

Min/Max Effort by Program Are Outliers

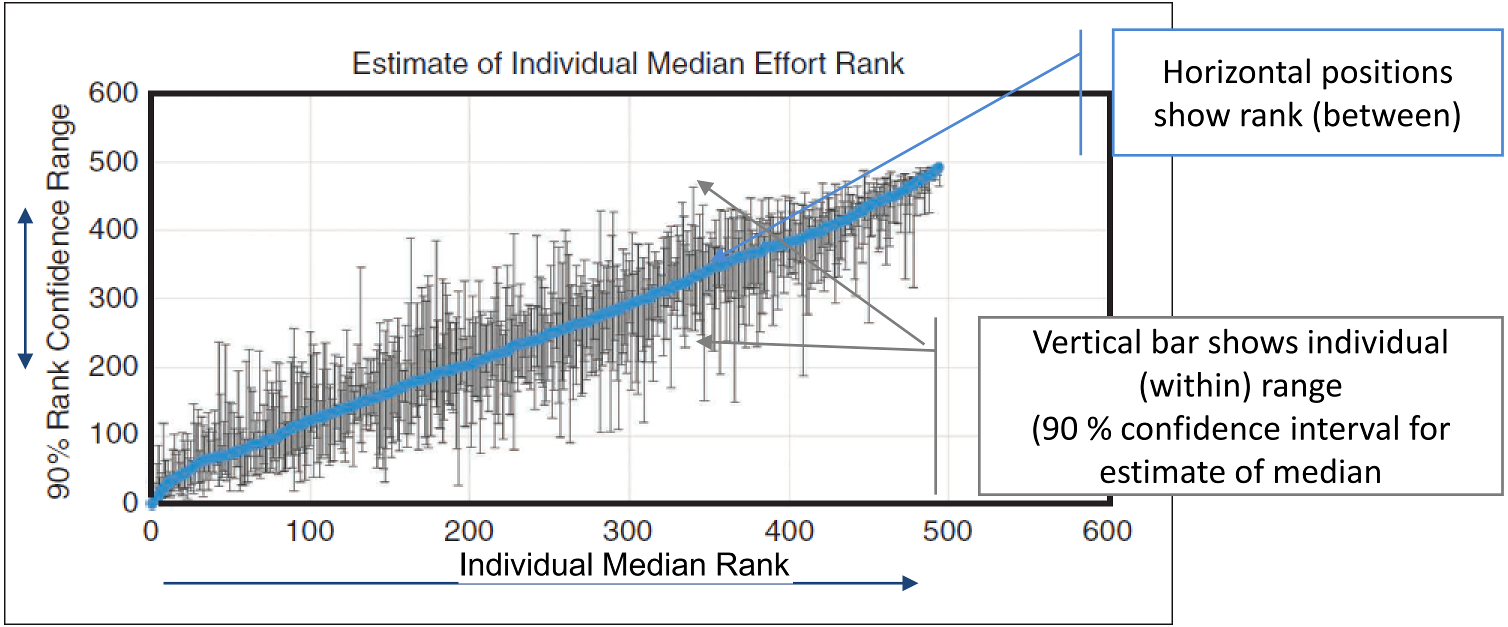


	1	2	3	4	5	6	7	8	9	10
Maximum	4.92	5.44	6.23	4.31	6.11	4.57	4.24	5.94	5.60	5.03
Q3	1.27	1.22	1.19	1.29	1.25	1.27	1.26	1.31	1.24	1.26
Median	0.82	0.85	0.81	0.85	0.85	0.87	0.87	0.85	0.87	0.82
Q1	0.48	0.58	0.52	0.61	0.59	0.59	0.62	0.47	0.60	0.59
Minimum	0.09	0.11	0.13	0.14	0.13	0.21	0.20	0.14	0.21	0.19
Max/Min	54.9	51.1	48.5	31.1	46.5	21.8	21.5	41.5	26.6	26.9
Q3/Q1	2.6	2.1	2.3	2.1	2.1	2.1	2.0	2.8	2.1	2.1

Interquartile range is narrow and stable

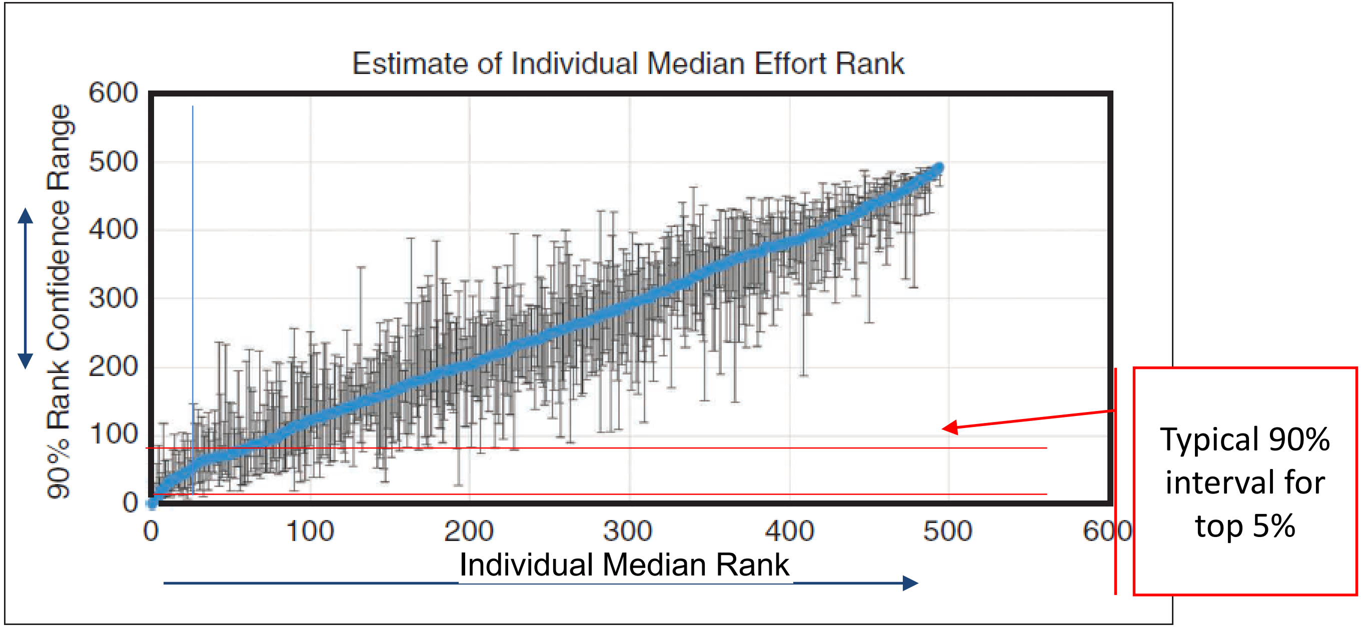
Programmer performance category changes by program.

Caterpillar Plot of Effort Rank for Within and Between



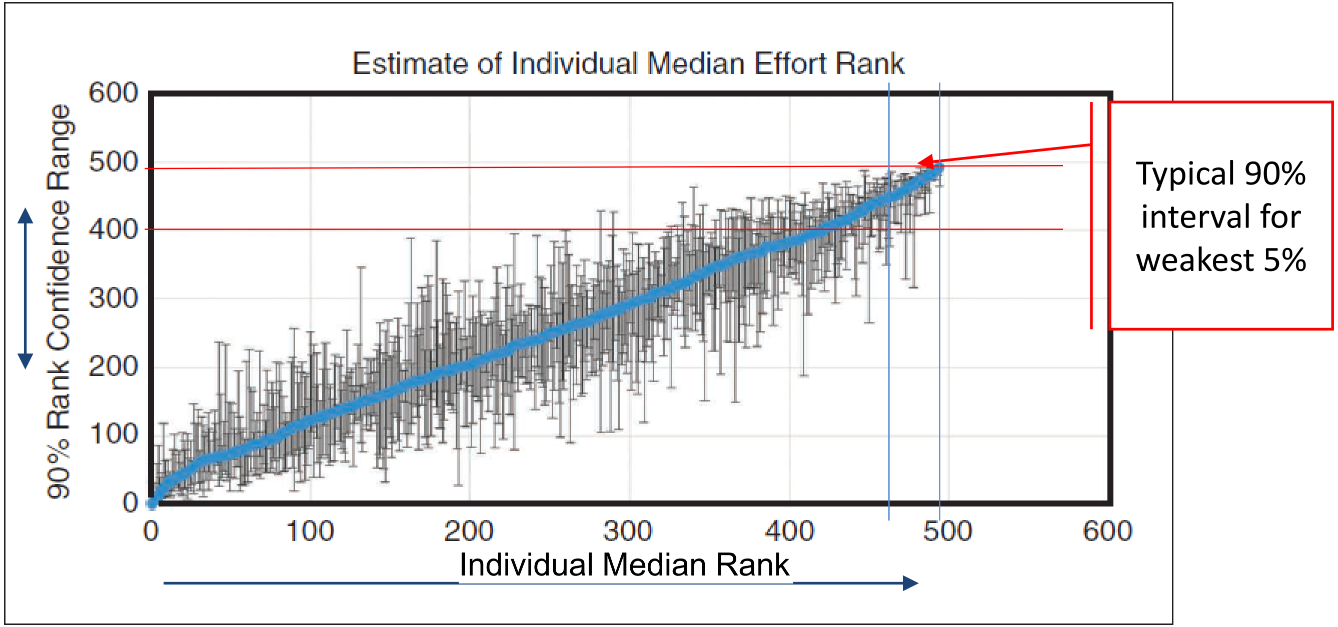
90% confidence in individual rank spans half the sample.

Best Performers Usually Finished in the First Quintile



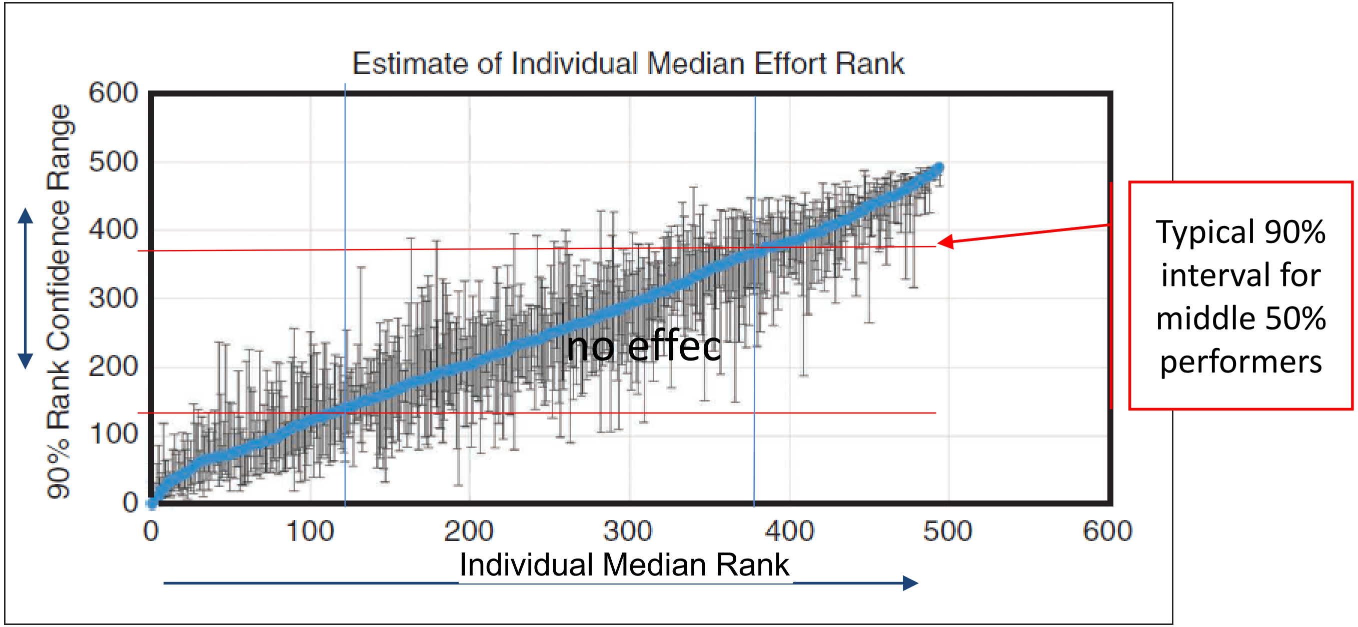
90% confidence in individual rank spans half the sample.

Weakest Performers Were a little More Variable



90% confidence in individual rank spans half the sample.

Most Individual **Effort Rank** Varied by 50% of the Population



90% confidence in individual rank spans half the sample.

Is there at least an Order of Magnitude range in programmer productivity?

Is there at least an Order of Magnitude range in programmer productivity?

With caveats.

The question either isn't very precise, or doesn't mean the same thing to all people.

What do we mean?

Programmers have about a x5 range, but

Most are within about x2

There may be very few much much slower, than average, but

There are not many more than twice average speed

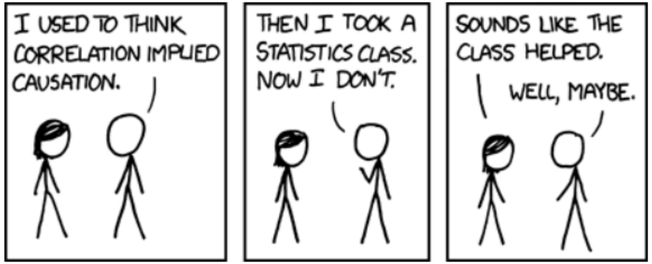
This is for small programs that should not exceed normal capability.

Other findings

Experience didn't matter – *I have 15 years of experience, (or maybe 1 year 15 times)*

We had years of experience and amount of code written, but no effect

Highest degree obtained,  no effect

College level statistics class 

actually, no

Summary, Similar within and between productivity variation

1) IntraCluster Correlation 0.6 (40% of variation within developer)
2) **New** Within variation had not previously been reported
3) Wide min/max range remains, but
4) Most developers are within a much narrower “x5” total productivity range, best and average about “x2”
5) Single point rather than repeated measures
   - Are not reliable (high variation)
   - Exaggerate range
6) Can get what ever Min/Max ratio you want by varying sample size and other assumptions. Min/Max isn’t very useful by itself.
7) Older studies replicated, but chased the noise

Limitations of This Study

This study only used C programmers (15% of the sample).

Programs are not broadly representative.

- Mostly implement math and numeric
- Small
- Simple and well defined

Drop out bias? (common problem among the studies)

Selection bias for entry?

Some “re-use” affects individual program results.

Significant differences in productivity remain.

Implications

This has implications for Industry.

- Max/Min ratio doesn’t tell us much because it’s mostly from the low end. We are measuring outliers not the main effects.
- Hard to get enough “top” programmers to affect a large project performance.
- Don’t over react to short term variation.
    - Look for controllable sources of variation (test, design, reviews).
    - Recognize outlier events for intervention.
    - Short term variation swamps other productivity effects.
- Don’t focus just on productivity! Other factors matter too!, environment, design, tooling, reviews, and quality. These matter, are more controllable, and are trainable. Don’t focus excessively on short term productivity.

Future work

Linear mixed models are suggestive.

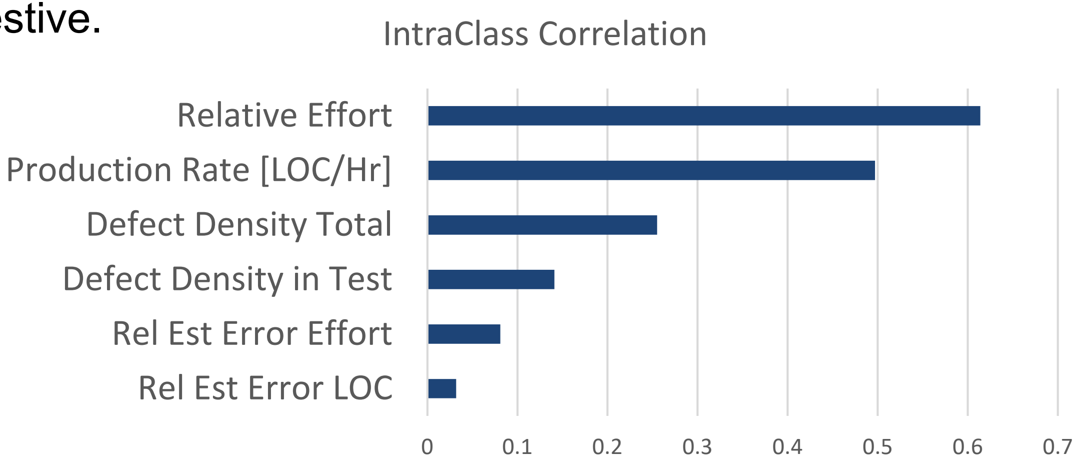
Relative Effort had the highest intra class correlation of these!

Look at effects across programming languages.

Test other long held beliefs

How much does programming language matter?

Is Quality free?



	Rel Est Error LOC	Rel Est Error Effort	Defect Density in Test	Defect Density Total	Production Rate [LOC/Hr]	Relative Effort
Series1	0.032	0.081	0.141	0.255	0.497	0.614

Data Link: http://ieee-dataport.org/1783

To Use the PSP Data

PSP STUDENT ASSIGNMENT DATA

Permalink: http://ieee-dataport.org/open-access/psp-student-assignment-data

DOI Link: http://dx.doi.org/10.21227/a5vb-cf02

Short Link: http://ieee-dataport.org/1783

Citation: William Nichols, Watts Humphrey, Julia Mullaney, James McHale, Dan Burton, Alan Willett, "PSP Student Assignment Data", IEEE Dataport, 2019. [Online]. Available: http://dx.doi.org/10.21227/a5vb-cf02.

The PSP and TSP course materials are available on the SEI Digital Library at https://www.sei.cmu.edu/go/tsp for use under the terms of the Creative Commons license

Acknowledgements

Mike Konrad and SCOPE project for supporting this work

Tim Menzies for his guidance and editing for the IEEE article

PSP Instructors and Students who contributed their data

The PSP team

References

Glass, R. L. **Facts and Fallacies of Software Engineering**. (2002). Boston, MA: Addison Wesley.

McConnell, S. Chapter 30 . *What Does 10x Mean ? Measuring Variations in Programmer Productivity*. **Making software : what really works, and why we believe it** (2010**).**

Sackman, H., W. I. Erikson, and E. E. Grant. 1968. "Exploratory Experimental Studies Comparing Online and Offline Programming Performances." Communications of the ACM, Jan.

Schwartz, Jules. 1968. "Analyzing Large-Scale System Development." In Software Engineering Concepts and Techniques, Proceedings of the 1968 NATO Conference, edited by Thomas Smith andKaren Billings. New York: Petrocelli/Charter.

Data Distributions Roughly Log-Normal

Size [LoC]

Effort [Minutes]

Defects

Student (j)

Assignment (i,j)