Collecting Data for Comparability: Benchmarking Software Development Productivity

Katrina D. Maxwell, *Datamax*

Collecting comparable benchmarking data is not a straightforward task. The author shares her experience, acquired over eight years, in collecting, validating, analyzing, and benchmarking software development projects.

Whether you are benchmarking an organization or simply a project, it all boils down to one thing—data. Do you have the necessary data in your company, and is that data valid and comparable? How can you access data from other organizations? To help you answer these questions and avoid some common serious mistakes in the benchmarking process, I've summarized my practical real-life experiences with software project data collection and benchmarking efforts in the following guidelines.

Planning for data collection

You can't benchmark data if you haven't collected it. Writing a questionnaire and having people complete it is not enough—you need a vision. It's similar to getting the requirement specifications right before developing software: if you learn that something is wrong with the data after collecting and analyzing it, your conclusions are meaningless, and you have to redo your work. I have wasted months trying to make sense of data collected without a clear purpose and without statistical analysis requirements in mind. If you work for a large company, consider asking the market or operations research department to help design your benchmarking questionnaire. Software managers know about software; data analysts know about questionnaire development. Collecting the right data for your purposes might require a multifunctional team effort.

Regardless of whether the data concerns chocolate bar sales, financial indicators, or software projects, the old maxim "garbage in equals garbage out" applies uniformly. Make sure that the variable definitions and responses are clear before you collect the data. Typical questions I ask when validating software project databases include: What does a zero mean? Does it mean none, is it a missing value, or was a number close to zero rounded to zero? And if a value is missing, does that indicate *no value*, *don't know*, or, if the question involved choosing from a list, was the correct response missing? Lists that include *Other* as a choice are also problematic, especially when collecting data for benchmarking. For example, let's assume that your company questionnaire includes a list of case tools. The case tool used on your project does not appear in the list, so you select *Other*. This

0740-7459/01/$10.00 © 2001 IEEE

category will thus include many diverse tools and will mean different things for different organizations.

When I receive a new software project database, I usually need to spend much more time understanding and validating the data than I do actually analyzing it. You can greatly reduce the risk of collecting the wrong data and the effort spent validating it if you spend more time up-front defining what variables to collect and how to measure them. Think about how you collect data in your own company. How careful are you? Do you ensure that everyone understands the definitions? How do you ensure uniformity over the years? Has your definition of effort evolved over time? Have you always counted support staff effort and tracked management time? If the person initially in charge of collecting the data has left the company, is the current person collecting the data in exactly the same way, using the same definitions? Even assuming that you have a high-quality data collection process for estimating cost and comparing project productivity within your company, if you want to benchmark against other companies, the critical question is: Is your data comparable?

Benchmarking and data comparability

You can't benchmark software development productivity if you have not collected size and effort data for your software projects. Productivity is typically defined as output divided by the effort required to produce that output. Although not perfect, we traditionally use software size as a measure of output for software development productivity (size/effort)—for example, 0.2 function points per hour. This should not be confused with the project delivery rate, which is also sometimes referred to as productivity but is actually the reciprocal of productivity (effort/size)—five hours per function point.[1] Remember to verify the units!

You can measure size in either lines of code or function points. How do you define lines of code—do you include comments, reused code, and blank lines? Additionally, a variety of function-point counting methods exist, including IFPUG, Mark II, 3D, Asset-R, Feature Points, Experience, and Cosmic.[2–5] How are you going to count them? Variation can also occur when different people do the counting—even if it's for the same project with the same function-point counting method.

Another question involves effort. Will you measure it in hours or months? If you use months, note that the definition of a work month can vary in other countries and companies. Also, will you include managerial time, support staff time, or just developer time? Will you include unpaid overtime? Did the customer provide significant effort, and will you count it? How many phases are you including—requirements specification through installation or feasibility study through testing?

Effort is notoriously difficult to measure accurately, even within a company. In addition to the problems already mentioned, other sources of error include late time sheets, missing cost codes, or misallocation of time for various reasons. In a recent article,[6] Martin Shepperd and Michelle Cartwright recount the experience of assisting one organization with its effort-estimating practices. The total effort data available for the same project from three different sources in the company differed in excess of 30 percent.

Needless to say, if they do not pay attention to data comparability, two companies measuring the same project can end up with different sizes and efforts. As productivity is calculated by dividing these two error-prone terms, benchmarking productivity is potentially extremely inaccurate. For example, let's assume that Companies A and B have developed the exact same insurance software application and used exactly the same effort. However, Company A uses the IFPUG 4.0 method,[7] which doesn't count algorithms, and Company B uses the Experience 2.0 function point method,[5] which does count them. This results in a 20 percent greater function-point count for Company B. In addition, Company B does not count the effort of installation at the customer site, whereas company A does, and this results in a 20 percent lower effort for Company B. So, for Company A, 100 function points divided by 400 hours equals .25 function points per hour. For Company B, 120 function points divided by 320 hours equals .375 function points per hour. Because Company B divides a 20 percent larger size by a 20 percent smaller effort, it calculates its productivity as 50 percent higher than Company A.

You can't benchmark software development productivity if you have not collected size and effort data for your software projects.

Productivity rates are highly variable across the software development industry.

Obviously, you need to beware of comparability errors. Unfortunately, we are less likely to ask questions and more likely to believe a result when it proves our point. If you think that comparability errors exist, rather than calculate a single productivity value, calculate a probable range of productivity values assuming an error in both terms.

If you want a dependable benchmark of software development productivity, make every effort possible to measure in exactly the same way. One way to compare your data to similar benchmarking data is to collect effort in hours by phase and staff type and to keep the detailed breakdown of the function-point count so that you can create the different effort and function-point metrics. Another way is to decide in advance which benchmarking database you want to use and to collect your data using its definitions. If benchmarking is something you plan to do on a regular basis, you should collect your data with a tool used by other companies that also want to benchmark. In addition, verify that the benchmarking database you use contains projects that the data collector has carefully validated.

Benchmarking and project comparability

Even if you are measuring productivity in exactly the same way, you must also benchmark against similar projects. It is not enough to measure a project's size and effort and compare it with a large database's average productivity. Productivity rates are highly variable across the software development industry. Business sector, requirements volatility, application language, hardware platform, case tool use, start year, and hundreds of other parameters can affect productivity. The identification and interaction of these factors makes comparing productivity rates very difficult. This is why software development databases should be statistically analyzed to determine the factors that contribute most to the specific database's productivity variation. Once you've identified the variables—or combinations of variables—that explain most of the database's productivity variation, you can limit your comparisons to projects similar to your own.

For example, if you developed a project using Cobol on a mainframe, and language and platform are important factors in explaining productivity differences in the database, then you should only benchmark your productivity against other projects using Cobol on a mainframe platform. On the contrary, if your project uses case tools and using the tools does not explain the differences in productivity of the database projects, there is no point in limiting your comparisons to other projects that also use case tools. So, either verify that the benchmarking service statistically analyzes the data and informs you of the key factors, or that it provides you with the raw data so that you can do so yourself. Also, pay attention to how many projects the benchmark is based on for each subset of data. You might consider a benchmark more reliable if it is based on 20 projects rather than four. Benchmarking against up-to-date data is also important.

Benchmarking data availability

Although many companies would like to benchmark projects, few contribute data to multicompany databases. We need data on a regular basis to keep these services up-to-date. Although large companies with well-established metrics programs, high project turnover, and data analysis competency might be content to benchmark projects internally, smaller companies do not have this option. These companies must look to benchmarking services for access to numerous recent, comparable projects. (See the "Sources of Software Project Benchmarking Data" sidebar for some useful sources.) In addition, most cost estimation tool vendors also have databases that you can use for benchmarking.

Sources of Software Project Benchmarking Data

- *Experience Benchmarking*: www.datamax-france.com
- *European Space Agency/INSEAD*: http://xrise.insead.fr/risenew/rise_esa.html
- *International Software Benchmarking Standards Group*: www.isbsg.org.au
- *Rubin Systems*: www.hrubin.com
- *Software Productivity Research*: www.spr.com