Language level critical for DSI and function points

To the editor:

In Barry Boehm's otherwise excellent article, "Improving Software Productivity," in the September issue of Computer, the discussion on delivered source instructions (DSI) as a productivity metric unfortunately omits the primary weakness of DSI; that is, it tends to move backwards as economic productivity improves.

Domains outside of software have recognized for more than 200 years that if a manufacturing process involves a significant amount of fixed costs, and there is a reduction in the number of units manufactured, then the cost per unit must go up.

A significant portion of software development costs are either fixed or inelastic, in that they do not relate in any way to DSI. Activities such as requirements, initial design, architecture, user documentation, and design reviews stay essentially constant regardless of DSI, and the costs of these activities range from 35 percent to more than 60 percent of the total cost of a large software system. Coding itself and code-related costs may total less than 25 percent of all development—especially for large military projects following military specifications, with their enormous paperwork requirements.

DSI originated in the early days of software when coding was the primary task and code was the primary deliverable. Since the early 1970s, coding itself has dropped in overall percentage of costs while paperwork has expanded. Clearly, DSI is not a suitable metric for paper-intensive activities.

To illustrate the paradoxical regression of DSI as a productivity metric, consider the results of two similar projects in terms of functionality, with one of the projects being written in Macro Assembler and the second in Ada (see table).

<table>
<thead>
<tr>
<th>Project comparison</th>
<th>Macro Assembler Version</th>
<th>Ada Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code size</td>
<td>70,000 DSI</td>
<td>25,000 DSI</td>
</tr>
<tr>
<td>Development costs</td>
<td>$1,043,000</td>
<td>$590,000</td>
</tr>
<tr>
<td>DSI per man-month</td>
<td>335</td>
<td>211</td>
</tr>
<tr>
<td>Cost per source statement</td>
<td>$14.90</td>
<td>$23.60</td>
</tr>
<tr>
<td>Function point per man-month</td>
<td>1.65</td>
<td>2.92</td>
</tr>
<tr>
<td>Cost per function point</td>
<td>$3,023</td>
<td>$1,710</td>
</tr>
</tbody>
</table>

Clearly, neither DSI per man-month nor cost per source statement match the economics of the situation—a two to one improvement in economic productivity on the side of Ada.

In this example, the IBM function-point technique developed by A.J. Albrecht gives a much better fit to economic productivity than DSI.

Boehm correctly points out that the function point method itself is not completely error-free, and many of the terms are ambiguous and prone to varying interpretations. In my own side-by-side measurements of software projects, I record both the DSI and function point sizes and also the counting rules in effect. My observations indicate errors in excess of 80 percent in counting DSI but only 20 percent in counting function points. Unlike DSI, which has remained unstandardized for more than 40 years, function points have an active forum of users (the International Function Point Users' Group) and a much better prognosis for achieving a satisfactory level of consistency than DSI has ever had.

Software productivity rates measured with DSI have exhibited the largest variation of any human activity. Much of that variation is spurious and attributed to the paradoxical behavior of the DSI metric itself. Careful side-by-side measurements of projects with both function points and DSI are starting to favor function points for consistency, and certainly for coming to grips with economic productivity.

Capers Jones
Software Productivity Research, Inc.

Author's reply:

Capers Jones' letter makes some very good points. Comparing productivity rates in DSI across different language levels (assembly, Ada, fourth-generation languages) does not make much sense. DSI users need to account for their language levels to avoid the paradoxes shown in Jones' letter. In general, organizations can avoid seriously misleading themselves by adopting a guideline of not using a lower level language where a higher level language can do the job.

For all their good aspects, function points have their paradoxes as well. A common example involves software maintenance, where one can make a lot of useful changes to a program's code without making any change in its inputs, outputs, etc. Thus, there is no change in function points, and one gets no productivity credit at all.

All of this points to the need for caution in interpreting productivity figures and for further experimentation and careful definition in the area of software productivity metrics.

Barry W. Boehm
TRW Defense Systems Group
Letraset pricing clarified

To the editor:

On p. 90 of the October issue of Computer ("Letraset upgrades design software"), you state that ImageStudio, Letraset’s new image processing software for the Macintosh, costs $495 and is available to registered Ready Set Go! users for $75. The suggested retail price of ImageStudio is $495, as stated. However, there is no special offer whereby registered Ready Set Go! users can purchase ImageStudio for $75. I hope this will clarify our pricing for your readers.

Michael Rennert
Letraset USA

Abstract data types here to stay

To the editor:

Although you have, no doubt, received several letters in the same vein, the following point is so important that I cannot in conscience leave the task to those of a like mind and risk that everyone else also defer.

I am thankful that the authors of "The Effect of Abstract Data Types on Program Development" (Jeffrey Mitchell, Joseph E. Urban, and Robert McDonald, Computer, August 1987, pp. 85-88) did not suggest that we abandon abstract data types altogether, but one could infer from the conclusion in their article that if one lacks an automatic ADT generator (to presume that such an animal exists!), then one should avoid the use of ADTs in programming. I assert that such an idea is penny-wise and pound-foolish.

It is certainly beyond the scope of their study to discover the true value of ADTs. Since coding constitutes just a small portion of the software life cycle, they leave unexplored the possibility that the code written with the use of ADTs will be more maintainable, more extensible, and easier to analyze and reuse in other applications. One ADT module, properly written, can be employed in a whole raft of applications, and even several times within one application. The whole idea of keeping the data type as abstract as possible (while still exhibiting its defining characteristics) makes this reuse possible.

Perhaps the time spent by the ADT team on programming the ADT portion of the project should be amortized over a dozen or so hypothetical future applications. In that case, I have no doubt that the ADT team would emerge as the true winner in this contest.

Thomas H. Hildebrandt
Raleigh, N.C.

Author’s Reply:

We agree that abstract data types positively affect much of the software development and maintenance process, and we do not suggest abandoning abstract data types. Our empirical evidence shows that the additional effort necessary to implement abstract data types prolongs the development effort. As Thomas Hildebrandt suggested, this development cost could be amortized over several projects, provided the supporting code for the abstract data types was generic enough to be used in the different applications.

It is apparent that the development effort can be streamlined if the implementation of the abstract data type is somewhat independent of the project’s coding phase. This can be accomplished either by generating the abstract data types automatically (B. Belkhouche and J. E. Urban, "Direct Implementation of Abstract Data Types from Abstract Specifications," IEEE Trans. Software Eng., May 1986, pp. 649-661) or by implementing the abstract data types independently.

Jeffrey Mitchell
University of Southwestern Louisiana

FREE FACTS. FAST!

We've just sped up the response time to our Reader Service cards, so now you can get information more quickly about advertisers' products and services, and products we list in the New Products section. Under our new system, the company will have your name and address just days after you send out the card!

While you're indicating the products that interest you, please take a moment to circle the articles and departments that you liked in this issue. That will let us serve your interests better.

We'd like to hear from you!

COMPUTER