Software development tools—Acquisition considerations—
A position paper

by LEON G. STUCKI
Boeing Computer Services
Seattle, Washington

A NEED FOR TOOLS

The literature abounds now with many guidelines on current programming methodology. Today’s software artisan is, however, often commissioned with more tasks to perform in a finite amount of time than can be accomplished using strictly manual techniques. Therefore, a knowledge of currently available automated programming tools is of key concern to today’s software analysts and programming managers. The spectrum of tools currently available impacts the software life cycle at various points, for example:

1. As an aid in ensuring that program specifications and standards are met,
2. As a debugging aid,
3. As a means of maintaining current and up to date documentation and configuration control,
4. As an auditing aid for recording and checking acceptance test results, and
5. As a performance measurement tool aimed at improving program efficiency.

GUIDELINES FOR EVALUATING TOOLS

The following guidelines are suggested for consideration when evaluating various generic classes of automated tools for a given project or for a given programming environment.

1. Start early. Tool acquisition should be considered early in the software development cycle. Software tools cannot be commissioned in one day and installed on the next. Unforeseen problems often occur when installing tools at the user’s site. Many tools must be customized for each new programming environment.
2. Select easy to use tools. Care must be taken to select tools that are easy for the applications programmers to use. System interfaces and procedures must be easy to use, understand, and modify.
3. Place tools in proper perspective. The use of tools must always be viewed in terms of supplementing, and not replacing, a good common sense approach to software development. The formulation of carefully designed test plans and good desk checking procedures are still of paramount concern.
4. Remember that tools consume resources. In many cases, the application of automated tools initially increases machine costs. Some tools are very powerful and can be misused by applying them to classes of problems that could better be solved by simpler means.
5. Select tools with a payoff. The utility of each potential tool should be understood by analyst and management alike. The use of an automated tool should assist in increasing the quality of the subject software, increasing user confidence, or increasing program performance in some way. Users should understand the benefits and cost associated with each tool in order to more fully optimize their utilization.

A SAMPLE EVALUATION OF SELECTED TOOLS

Table I was developed in the course of a lengthy investigation conducted in 1975–1976 by the author. This table is by no means complete. It is simply offered as an example of some of the evaluation criteria involved in examining several of the more sophisticated automated programming tools.

Much work remains to be done in this area. We are still, for the most part, benchmarking tools to find their costs and guessing as to their potential benefits. Little empirical data has been gathered on the actual measured utility of applying tools. One recent set of experiments by the author at UCLA pointed out some of the pitfalls often encountered in trying to assess the utility of tools. At the same time, however, it offered a prospect of some very interesting potential payoffs.

This presentation will offer a glimpse into the world of automated tools. Generic classes of tools will be considered. General remarks will be offered on various factors of tool utility. Specific mention will be made of selected findings from evaluation activities conducted by the author.
TABLE 1—Summary of Test Results

<table>
<thead>
<tr>
<th>TOOL</th>
<th>IMPLEMENTATIONS</th>
<th>SOURCE LANGUAGES</th>
<th>STATIC / DYNAMIC</th>
<th>OPERATING COST</th>
<th>DATE OF USE</th>
<th>FEATURES</th>
<th>LIMITATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADG</td>
<td>UNIVAC</td>
<td>FORTRAN</td>
<td>STATIC</td>
<td>1</td>
<td>1</td>
<td>COSTS IMPOSED ON PROGRAMMERS TO PRODUCE OFFICIAL TEST CASES</td>
<td></td>
</tr>
<tr>
<td>FUTURE</td>
<td>IBM</td>
<td>FORTRAN</td>
<td>STATIC</td>
<td>0</td>
<td>1</td>
<td>NO LONGER SUPPORTED</td>
<td></td>
</tr>
<tr>
<td>CAPEX</td>
<td>PARSINGER, A.</td>
<td>CDC</td>
<td>STATIC</td>
<td>1</td>
<td>2</td>
<td>SUPPORTS TEST ENVIRONMENTS</td>
<td></td>
</tr>
<tr>
<td>HAC</td>
<td>UNIVAC</td>
<td>FORTRAN</td>
<td>STATIC</td>
<td>0</td>
<td>1</td>
<td>COSTS IMPOSED ON PROGRAMMERS TO PRODUCE OFFICIAL TEST CASES</td>
<td></td>
</tr>
<tr>
<td>PPS</td>
<td>IBM &amp; ALL</td>
<td>FORTRAN</td>
<td>DYNAMIC</td>
<td>1</td>
<td>2</td>
<td>SUPPORTS TEST ENVIRONMENTS</td>
<td></td>
</tr>
<tr>
<td>STRUCTURE</td>
<td>IBM</td>
<td>FORTRAN</td>
<td>STATIC</td>
<td>0</td>
<td>1</td>
<td>SUPPORTS TEST ENVIRONMENTS</td>
<td></td>
</tr>
<tr>
<td>SCADEX</td>
<td>FUPID</td>
<td>FORTRAN</td>
<td>STATIC</td>
<td>0</td>
<td>1</td>
<td>SUPPORTS TEST ENVIRONMENTS</td>
<td></td>
</tr>
<tr>
<td>JOOSE</td>
<td>CDC</td>
<td>FORTRAN</td>
<td>STATIC</td>
<td>0</td>
<td>1</td>
<td>SUPPORTS TEST ENVIRONMENTS</td>
<td></td>
</tr>
<tr>
<td>CASE</td>
<td>UNIV. OF COLORADO</td>
<td>FORTRAN</td>
<td>STATIC</td>
<td>0</td>
<td>1</td>
<td>SUPPORTS TEST ENVIRONMENTS</td>
<td></td>
</tr>
</tbody>
</table>

FUTURE NEEDS

Much work remains to be done in the area. One current need is for more test environments (Software Engineering Laboratories) where various tools and techniques can be selectively applied and assessed under controlled conditions. A set of cost/benefit matrices is needed to aid in the quantitative evaluation of automated tools.

REFERENCES


From the collection of the Computer History Museum (www.computerhistory.org)