Evaluating and Selecting Testing Tools

ROBERT M. POSTON and MICHAEL P. SEXTON
Programming Environments

In these tough economic times, software-development managers are pushing to get more and better testing done faster. Most recognize that automated testing tools facilitate higher quality and more productive testing, but acquiring such tools is often quite complicated.

Managers typically make the first move toward purchasing tools by assigning someone to review them. Initially, the people assigned are enthusiastic about their responsibility. On the surface, the job seems interesting and easy to complete — a simple matter of surveying and comparing commercial tools and reporting results.

Unfortunately, tool evaluations seldom end happily. Managers often ignore or reject recommendations, complaining that evaluators have failed to do their homework and cannot quantify their findings. Managers are reluctant to purchase tools on the basis of incomplete information. Evaluators, on the other hand, may blame managers for not clearly specifying the information expected in evaluation reports. When managers do buy a tool, it may not be used. People don’t want to advertise their part in buying expensive tools that sit on the shelf; so few write follow-up reports on savings or losses. Instead, the company with a nonperforming tool suffers in silence.

Some companies consistently buy testing tools that serve their software testers and developers well. What is their secret? Part of it may be that their managers and evaluators use a systematic data-collection method — usually with forms or checklists to evaluate tools.

Ever since testing tools were invented, people have circulated poorly written and
badly organized survey forms throughout their companies under the guise of collecting useful tool data. No wonder people had difficulty making wise tool selections when data was collected in such a manner. A data-collection system that leads a company to successful tool selections must be carefully devised.

Such a system was recently created for industry-wide use, if indeed, in a slightly unorthodox way. In 1987, a group of skilled tool users and software developers came together to develop a Reference Model for Computing System-Tool Interconnections, IEEE Standard 1175, which enables tools to communicate. The goal of the working group was to define tool interconnections, but the group soon realized the importance of considering interconnections during tool evaluations. As 1175 describes, interconnections affect and are affected by how a tool works in an organization, the platforms it runs on, and how it shares information with other tools. If any of these interconnection aspects are ignored, successful tool implementation will be thwarted. Other evaluation criteria are necessary, but they must be supplemented by interconnection criteria.

Besides their knowledge of interconnections, those working on 1175 shared many experiences about subtle distinctions in tools and what makes one slightly better than another in a certain situation. When the group completed its work on 1175 in December 1991, it also introduced a spin-off of its work—a useful tool-evaluation system. Of the companies that supported work on 1175, several are experimenting with this system, and four have used it to purchase and implement selected tools successfully.

One reason for this success is the system's many organizational features and safeguards. It ensures that evaluators record information only once in one place, protecting against redundant or overlapping information. It provides a comprehensive view of tools under consideration. It accounts for tool-dependent factors like functionality, performance, and ease of use, as well as environment-dependent information like how the tool affects or is affected by the organization using it, the platform it runs on, and other tools. Finally, it minimizes subjectivity. The information evaluators enter is quantified, allowing comparison by measurement. When the tools are in place, an evaluator can use these statistics to reevaluate them.

Naturally such a system is only as effective as the forms it uses. We have found a combination of forms works best for evaluation: some from 1175, a needs-analysis form, and forms for tool-selection criteria and tool classification to organize selection criteria and data about the tool. Criteria weighting, rating, and summary forms are also needed once data is collected.

With these forms, evaluators have a reasonably accurate and consistent system for:

- identifying and quantifying user needs,
- establishing tool-selection criteria,
- finding available tools, and
- selecting tools and estimating return on investment.

As Figure 1 shows, these forms encompass more than just tool selection. Our data-collection system is also suitable for evaluating software tools other than testing.
ANALYZING USER NEEDS

When managers make an educated guess (usually prompted by the technical staff) that employees can benefit from using automated testing tools, they have identified a possible need. As a tool evaluator, your first responsibility, before looking at any tools, is to find out if the managers' perceived need is an actual need.

We assume that evaluators perform all activities including needs analysis. Some companies may assign people other than evaluators to the activities we describe or even partition the evaluation process differently. But our data-gathering system should give you a comprehensive tool evaluation, even allowing for organizational differences.

Needs analysis is important to the credibility of all concerned with tool evaluation and will be referenced many times throughout the evaluation. A well-prepared needs analysis can prevent a variety of purchasing mistakes.

The first step in needs analysis is to visit the quality-assurance department, which may have productivity and quality statistics already compiled. You may end up visiting several other departments before completing the analysis.

Figure 2 is a form for organizing needs-analysis data. To use it, you must know what your company means by "testing" and what activities are involved. Many people assume that software testing includes both failure detection and elimination (debugging), but some progressive organizations now consider testing to be failure detection only.

The next step is to determine how much testing is being done now. First, see how many staff months were expended in testing on the most recently completed software project (you usually won't find that many statistics on older projects) and then look at how many projects are scheduled in the next year or two. By using statistics from the most recent project and predicting testing work on upcoming projects, you can estimate the number of staff months required for near-term testing. This estimate is the predicted cost of testing in staff months, assuming the testing staff uses no new tools. The estimate serves as a baseline for comparison with tool-assisted testing.

Test quality is also important, with the customer being the ultimate judge. Find out how many failures customers have uncovered in the last software release. Such failures are often reported in terms of failure density or number of failures per thousand lines of code. Reports on recent failure densities are useful for inferring failure densities for future projects.

Besides examining and projecting from failure histories, you should be alert to anticipated changes that could influence test effectiveness. Examples are planned activities to reduce failures, like manual inspections, and upcoming staff reorganizations.

Estimated test quality is meant to show predicted test quality if no new testing tools are introduced. Like the productivity estimate, the quality estimate will be referred to later in the evaluation.

Several techniques to predict software productivity and quality are more accurate than the simple extrapolations we suggest. Function points, publicized by Capers Jones, and test-quality measurement, developed by Programming Environments, are two such techniques. But for tool evaluation, these techniques are overly sophisticated. Ballpark estimates with error margins as high as even 20 percent should be adequate to justify most tool evaluations.

Figure 3 shows typical failure thresholds - number of allowed failures -
compiled from Business Week's special issue on quality. If predicted staff-month expenditures for testing are very low, or if predicted failure-density numbers are below a company-established threshold, you should go back to your manager because the testing need may not warrant a full-scale tool evaluation.

**ESTABLISHING SELECTION CRITERIA**

Once you have quantified the company's need for testing tools, your next job is to establish criteria for tool selection. In the form in Figure 4, selection criteria are arranged in four groups: general, environment-dependent, tool-dependent functional, and tool-dependent nonfunctional. Each group serves as a filter for subsequent groups. In that way, only the tools most likely to succeed are considered further.

After you enter the selection criteria, the final process in establishing criteria is to weight them.

### Group 1: General criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Value</th>
<th>Weighting factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Productivity gain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2 Quality gain</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Group 2: Environment-dependent criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Value</th>
<th>Weighting factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Testing tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2 Organizational changes for new or revised tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.1 Policies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.2 Techniques</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.3 Work-product standards</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.4 Measurements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.5 Training courses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3 Platform changes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.1 Hardware</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.2 Operating system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.3 Database system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.4 Language system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.5 Communications system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.6 Human-interface system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3.7 Information-exchange facilities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4 Tool-interconnect change</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4.1 Information transfer utilities</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Group 3: Tool-dependent functional criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Value</th>
<th>Weighting factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Tool Function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2 Tool Function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3 Tool Function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4 Tool Function</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Group 4: Tool-dependent nonfunctional (characteristic) criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Value</th>
<th>Weighting factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Performance (response time)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2 Human factors (user friendliness)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2.1 Time to learn (casual and dedicated users)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2.2 Time to use (casual and dedicated users)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.3 Reliability (mean time between failures)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 4. Form for recording tool-selection criteria.*
**General criteria.** A company wants to see higher development productivity and software quality as a result of incorporating new testing tools. As Figure 4 shows, these general criteria are the first filter a tool will be subjected to. The basic question is will the amount of potential improvement in testing productivity and quality satisfy the company? 

Productivity and quality thresholds are dictated sometimes by company policy, sometimes by project managers. Some companies (or managers) demand very high thresholds, as Figure 3 shows; others are more lenient. It is important to know exactly what your company’s threshold is.

To use the form in Figure 4, you record under Criteria Value the minimum acceptable thresholds, usually in percentages, that will justify tool purchase. For example, the company may require a 50-percent minimum productivity gain and a 40-percent minimum quality gain.

**Environment-dependent criteria.** Group 2 contains criteria about the tool’s environment. Under Criteria Value, you enter the most the company is willing to spend for each item. The degree of this commitment will critically affect a tool’s eventual use. If commitment to every item in Group 2 is low, the company is probably not serious enough about tool implementation to justify further tool evaluation.

If you can’t get any commitments on dollar amounts for these items, the evaluation is not lost. You simply proceed as though the items have no budget restrictions until you identify one or two tools that meet all other criteria. Then you backfill from the estimated tool cost to Group 2 criteria.

As the form in Figure 4 shows, Group 2 items include the maximum allowable cost for testing tools, organizational changes, platform changes, and tool-interconnection changes.

**Testing tool.** This criterion reflects how much money the company has available to buy testing tools. The company might not have any money in its current budget to fund testing tools, but it may intend to allocate funds during its next fiscal period.

You can sometimes get a forecast to fill in here if money is not yet budgeted. With an actual or projected maximum amount, you spend time considering only tools the company can afford.

**Organizational changes.** Items in this section are also referred to as organizational support elements because they represent the tool’s organizational environment or context.

Tool evaluators and managers often give little consideration to these support elements, because they do not understand how important they are to a successful implementation. Studies completed as early as 1984 show that implementation usually fails when companies try to incorporate a new method or tool without accounting for these elements.

These items are defined in Table 1 as follows:

- **Policies.** Also called directives, instructions, or methodologies, policies are written descriptions of what must be performed in activities in which life-cycle phase using which tools. Under Criteria Value in Figure 4, you enter the most your company could spend to create or modify policies to account for new tools. If a new tool enables testers to create test cases in the first life-cycle phase of development when they usually create them in the last life-cycle phase, company directives or instructions to software engineers must be rewritten to reflect the change.

- **Measures.** Also called metrics, measurements are written descriptions of how to evaluate work products quantitatively. When testers measure how many test cases have passed, failed, or have not been exercised, they have measured a work product quantitatively. Testers also need to measure things like test coverage and software reliability. Suppose a company wants to buy a testing tool that creates test cases. Does the new tool change how the company counts passed and failed test cases? If it does, you enter under Criteria Value how much money management can allocate to make those changes.

- **Training.** This item represents experience in applying other support elements.

- **Techniques.** Also called methodologies or procedures, techniques are written descriptions of how to perform an activity. Under Criteria Value, you enter the most your company could spend to create or modify techniques. Most companies do not try to create their own testing techniques but simply use well-regarded techniques from the software industry. Modern testing tools usually support popular techniques, so new tools are unlikely to cause changes in techniques. Occasionally, however, an innovative tool builder will incorporate a new technique that will cause testers to change how they work.

Tools that require a company to change its testing techniques (work procedures) will have a cost attached that you should account for.

- **Work-product standards.** Also called documentation standards, work-product standards are written descriptions of the items (documents, code, or data) that must be produced in an activity.
while using tools. When a new testing tool is purchased, testers will probably require training in three parts: They need to learn how to operate the tool, how to prepare tool input and how to use tool output. Most testing tools are easy to operate. Training usually amounts to a quick walkthrough of a menu system and can often be accomplished on the job. But learning to derive input for sophisticated tools and how to handle output efficiently requires classroom time. The amount you enter under Criteria Value is how much the company is willing to pay for training.

Platform changes. This section of the form in Figure 4 deals with criteria for selecting hardware and software platforms. Under Criteria Value, you enter the amount the company is willing to spend for changes to these platforms. Managers often say that a new software testing tool must run on existing hardware. You enter a zero when that restriction is imposed. You may also find similar restrictions on other platform components (items 2.3.1-2.3.7). However, because platform configurations are in flux in most companies, you should check that information is up to date. An investment allowance of zero for a platform component one week may change to a large allowance the next week.

Tool-interconnection changes. This section of the form has to do with how the new tool will affect communications among tools. Questions include does a new tool require a new repository? Must import and export capabilities be added to new tools or to already purchased tools to facilitate necessary communications? Must a transfer utility or tool interface be built? This area is highly specialized, and you'll probably need to talk to expert tool users.

You may also want to consider tools that conform to 1175 because tools that read from and write to 1175's Semantic Transfer Language will require little investment for tool-interconnection changes.

Tool-dependent functional criteria. In Group 2 of Figure 4, all criteria are related to the tool's environment (context); in Group 3, all criteria are related to the tool itself.

Typically, people in an organization get together and brainstorm about the functions and features they want in new testing tools. The brainstormers try to produce a wish list to give to evaluators. The informal requirements on the wish list are essential for tool evaluation, but the list is usually incomplete and may contain definitions that are inconsistent with industry-accepted or vendor definitions. Often the list will contain overlapping or redundant requirements.

Testers and managers can use Figure 5, which gives testing-tool classifications, as a checklist for selecting jobs (functions) they want testing tools to do. Within each job category is a list of smaller jobs or tasks. Testers and managers can check or number the boxes at the left of the task names to assign task priorities. When this checklist is filled in, it will contain more complete, organized, and substantive information than can be found in most wish lists.

You can plug the information from this checklist into Group 3 of Figure 4. Beside 3.1 to 3.4, there is space to enter four tool-function names. You record the most requested function from the checklist as 3.1 followed by other functions in order of preference. If you are asked to compare more than four functions among tools, you can extend the Group 3 list, but it is easier to locate appropriate tools if you concentrate on a few, high-priority functions.

As the evaluation progresses, testers and managers may change their minds.
about the tool functions they want most. The function in 3.1 may become 3.2 and so on. If necessary, you can use the space in Criteria Value to assign the new priorities.

**Tool-dependent nonfunctional criteria.** Like Group 3, Group 4 is concerned with the tool itself, but Group 3 deals with functions, while Group 4 deals with a tool's measurable properties, attributes, and characteristics. Performance, often measured in response time, is the most widely referenced characteristic. Other measurable characteristics include reliability and human factors, such as the time it takes to learn a tool and use it.

In this section of the form in Figure 4, you enter one minimum acceptable response time that applies to each function listed in Group 3. Response time must be defined according to a standard load or benchmark. For example, a function that compares expected and actual outputs must compare two 100,000-byte files (load) in less than one minute (response time).

**Human factors is a very tricky area in tool evaluation.** Nearly every wish list will include a requirement that testing tools be easy to user friendly. These terms are red flags! A user can be frequent, casual, left-brained, right-brained, a touch typist, or a hunt-and-pecker. "Easy to use" is too vague to work with.

Also, at this point, you shouldn't worry about mice, menu systems, touch panels, or any of the interface features people think a tool must have to be easy to use. Instead, concentrate on quantifiable human factors like the time it takes to learn and use a tool.

**Under Criteria Value in Section 4.2,** you enter the most time the company allocates for a tester to learn to use a tool (4.2.1) and the most time the company will give a trained tool user to perform a testing function (4.2.2).

**Under Criteria Value in Section 4.3,** which deals with reliability, you enter how many tool breakdowns (number of times the tool is inoperable) the company will tolerate. Testing tools are supposed to make other software products reliable or failure free, so it seems appropriate to judge testing tools for their reliability.

Occasionally, when examining very similar tools, you may consider characteristics in addition to those listed in Group 4, like robustness or maintainability. These and other tool-dependent characteristics are detailed in a technical report from the Software Engineering Institute. If other characteristics are important, simply expand this section, but remember that expansion is an exception. More is generally not better in most sections.

**Weighting.** After you have filled out all criteria values in Figure 4, you are ready to meet with the testers who will use the new tools to reach a consensus on the relative importance, or weight, of the criteria. The rules for assigning weights are simple. Every criterion must have a weight, and no two criteria may have the same weight. If two or more criteria are weighted equally, common sense should tell you that weighting begins to lose its usefulness. The most important criterion gets the greatest weight; the least important criterion gets the lowest weight.

Putting weights on criteria is an easy but significant activity because it brings tool users to agreement on which requirements are most important. And it makes every potential tool user part of the evaluation.

**SEARCHING FOR TOOLS**

If you've followed the preceding recommendations, you are well prepared for tool shopping. You've completed a needs analysis, enlisted management's help to establish benefit and cost thresholds, and made sure that the testers sanction the tools' requirements and generally agree on what is most important.

At this point, you'll be relying on a number of standards, articles, and surveys. We suggest you begin with the following:


---

**Tool-to-Organization Interconnection Standard Profile**

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization interconnections</td>
<td>Name of applicable standards</td>
</tr>
<tr>
<td>Job function</td>
<td></td>
</tr>
<tr>
<td>Primary use</td>
<td></td>
</tr>
<tr>
<td>Secondary use</td>
<td></td>
</tr>
<tr>
<td>Final user</td>
<td></td>
</tr>
<tr>
<td>Life cycle</td>
<td></td>
</tr>
<tr>
<td>Phase of initial use</td>
<td></td>
</tr>
<tr>
<td>Phase of intermediate use</td>
<td></td>
</tr>
<tr>
<td>Phase of final use</td>
<td></td>
</tr>
<tr>
<td>Support elements</td>
<td></td>
</tr>
<tr>
<td>Policies</td>
<td></td>
</tr>
<tr>
<td>Techniques (methodologies)</td>
<td></td>
</tr>
<tr>
<td>Work-product standards</td>
<td></td>
</tr>
<tr>
<td>Measurements</td>
<td></td>
</tr>
<tr>
<td>Training courses</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Tool-to-Organization interconnection profile. This form can also be used for creating an organization profile by replacing "Tool Name" with "Organization Name."

Tertiaryparation, Exemtion, and dowl-ryF TmhRqmt, Software Technology Support Center, 1992. This survey, which is annually updated, will be of particular interest to the US defense community. Contact Greg Daich, STSC, O0-ALC/FTSE, Hill Air Force Base, UT 84056; (801) 777-7703 or Defense Service No. 438-7703.

Graham, Dorothy R., Computer-Aided Software Testing: The CAST Report, Unicorn Seminars, Uxbridge, Middlesex, UK, 1991. This tool description and survey is particularly useful to European testers. Contact Unicorn Seminars, Brunel Science Park, Cleveland Road, Uxbridge, Middlesex, UB8 3PH; (08-95) 56484.

Creating organization profiles. While waiting for these publications to arrive in the mail, you can use the forms in Figures 6 through 8 to profile or outline the organization in which tools will operate. You need only delete "Tool Name" at the top of the form and insert "Organization Name." Names of standards documents go in the blanks on the right side. If no document references are available, refer to the items in Group 2 (environment-dependent criteria).

In filling out the form in Figure 7, again enter the applicable standards. If the references called for are not available, 1175 (pp. 13 and 14) provides a list of recommended standards for each category.

In using surveys, after using the forms in Figures 6 through 8 to create organizational profiles, you are ready to use them as tool surveys. Usually tool surveys will list many more tools than you care to consider. Also, they're often out of date because they're not published frequently enough to keep up with the changes in the tool market. You may be able to screen some tools out immediately, such as those that do not perform any itemized functions. Your goal is to compile a short list of candidate tools from the long survey lists. You can make your short list more complete by searching for new-product announcements in trade journals. Many times your evaluation may be underway when promising new tools come on the market.

Contacting vendors. When you are satisfied with the short list, the next step is to request current information about the candidate tools from vendors. You can write a form letter asking vendors for:
- current product description or brochure,
- price list,
• completed set of tool-profile forms (send blank forms in Figures 6-8 for vendors to complete),
• list of current users who will spend time talking to potential users, and
• case studies or references to case studies that document productivity or quality gains.

The kind of reply you receive can tell you a lot about the vendor. A prompt, complete reply often means good service ahead; a late, incomplete, or inaccurate response may foreshadow unsatisfactory future service.

You can verify vendor-provided information by checking with people who have used or studied the vendor's tool. With the corroborated tool information, you are ready to compare what a tool offers with what the organization needs. The closer the tool profile correlates with the organizational profile, the higher the probability that the tool will serve the company well.

Using other evaluations. Beyond the information the vendor provides, you may find published evaluations of a tool. Sometimes people within your own company have already reported on a tool. Many tool evaluations appear in trade magazines.

However, magazines often claim their evaluations have been performed by expert tool users. But is the expert tool user also an expert tester? Can the expert identify the merits of a reliability testing tool as opposed to a control-flow testing tool? Does the expert know the difference between a terminal-based execution tool and a host-based execution tool? Unless the expert understands testing well enough to make these kinds of distinctions, the evaluation in the literature may not be a good source. The general rule here is to be wary of accepting any published evaluation or in-house report without question.

SELECTING TOOLS

With vendor feedback and published evaluations at hand, you are ready to begin rating tools. A rate is a number between zero and one, much like a correlation factor in statistical analysis. You derive it by comparing vendor-supplied and other collected tool information to the Criteria Value information in Figure 4. If tool information completely matches a criterion value, the rate is 1. If there is no tool information for a criterion, the rate is 0. If tool information does not map to a criterion precisely, you must calculate (or estimate) how close the correlation is. For example, suppose a vendor's productivity gain is documented as 40 percent in Group 1 and the company requires a 50-percent gain. You would enter a rate of 0.8 (40 percent is 80 percent of 50 percent).

An example of a criterion that may call for an estimated rate is training. The company may be considering a test-execution (capture/playback) tool. The vendor provides free one-day training on how to operate the tool but does not offer training on how to prepare tool input or use output. The testing staff will train itself in I/O on the job. You must estimate how much this self-training will cost the company and rate it accordingly.

The next step is to score each criterion by multiplying weight by rate. The tool with the highest total score is the tool you'll usually recommend.

Now comes a summit meeting with managers and testers. You are not looking for a rubber stamp on your choice at this time. Rather, you should encourage interaction and discussion about the results. When everyone has a chance to express an opinion and vote on which tools should be selected, tool implementations are more likely to succeed.

In this meeting some testers may fer-
vently support a particular tool. These testers are champions or agents of change. Tests who strongly believe in a tool will push its implementation. Managers should note these champions. They should be the first users of the tool they advocate.

Sometimes champions will not emerge right away. Even with a complete paper evaluation, testers may want to test drive a tool before casting a vote. Many vendors will supply a trial-use or evaluation copy of a tool for a limited time so that testers can try out a tool in their own environment. Trial use sounds like a good idea, but it often leads to wrong decisions. The biggest problem is time. Often testers do not have enough time to complete their regular work assignments, let alone evaluate a new tool. Usually testers can do only a surface evaluation, especially if the tool is powerful and sophisticated. Another problem with trial use is the differing levels of user expertise. If a project has 10 experienced testers and one inexperienced tester, the new tester is often the one with the most time to try out the tool because the experienced people are tied up in the project's critical path. Consequently, managers get a report from someone who doesn't have the background to understand all the tool's implications or possibilities.

If testers can devote enough time and appropriate expertise to complete a thorough trial use, they should request an evaluation copy. Otherwise, they should make selections on the basis of the well-researched evidence you present.

**REERRORATATION**

When a tool is selected, the initial evaluation is finished. The selected tool goes on to be implemented. When a tool promises big increases in quality and productivity and tool selectors have great confidence in a tool, a company may implement that tool on many projects simultaneously. Other companies will elect a more cautious approach, implementing a tool one project at a time.

The first project in which a tool is implemented is often called a pilot project. On a pilot project, the tool's champions should be brought in to use the real product for the first time.

After the tool has been used on a pilot project, it should be reevaluated. Every time a testing tool is introduced on a pilot project, new technology is inserted into an organization with the expectation that it will improve software quality and development productivity. Reevaluation tells management if the tool performed as expected.

Because what we are describing is a system, with a process as well as forms, it gives managers and tool evaluators a reliable way to identify the tool that best fits their organization's needs. As an added benefit, evaluation can continue after the tool has been used for a while to measure how it lives up to expectations. The process we have described can be used again and again, and the forms are easy to incorporate into spread sheets for continual use. Needs analysis is particularly useful. Managers can refer to this form for productivity and quality statistics gathered before tools were implemented and compare them to numbers documented after tools are used.

**ACKNOWLEDGMENTS**

As we developed this paper, Gregory T. Daich of SAIAC at Hill Air Force Base, Utah, carefully reviewed each section. We thank him for his thought-provoking comments.

**REFERENCES**


---

Robert M. Poston is president of Programming Environments and the originator of the T software-testing tool. He has also chaired the standards effort to develop IEEE Standard 1175. Poston has received numerous professional and industry achievement awards and has lectured worldwide on software-engineering subjects. He received a BSEE from California State Polytechnic University and has been prominent in IEEE activities for 25 years.

Michael P. Sexton is a software engineer at Programming Environments, where he co-developed the test-execution tool called Runner. He is also a lieutenant in the US Army reserves in the 911th Chemical Co.

Sexton graduated from Worcester Polytechnic Institute in Massassachusetts with a BS in computer science. He is pursuing a master's degree in computer science at Monmouth College in West Long Branch, New Jersey. He is a member of the IEEE.

Address questions about this article to Poston at Programming Environments, 4043 State Hwy. 33, Tinton Falls, NJ 07753; (908) 918-0110; fax (908) 918-0111.