INTRODUCTION
Ten years ago, the most effective method for testing software was to bang away at the code. Occasionally, tools were built as part of the development effort to test a specific aspect of the software. A few capture/playback recorders were available, but they were cumbersome and time-consuming to use. A few debugging tools existed. That was the extent of it ten years ago.

Today, not only do commercial testing tools exist, but multiple classes of testing tools are heavily marketed throughout the software development industry. There are improved test tools for debugging programs, for regression testing, for performance testing, and for Web sites. In fact, over the past ten years, so many different test tools in so many different classifications have hit the market, that evaluating and selecting the best test tool set for an organization can became a full-time job for one or more employees for several weeks. That investment, along with the annual license fees for the tools selected, may seem steep, but the risks associated with bringing the wrong tools into an organization can be even more expensive.

To minimize risk and decrease research and evaluation time, there are two important decisions to make. The first is whether an organization has a clear picture in mind as to what the benefit of the tool will be within the organization. The second is the...
size of the investment the organization is committed to making in bringing one or more test tools into the organization. Having an understanding of the classes of test tools available today can help in making these decisions.

CLASSES OF TEST TOOLS
The first step in tool selection requires an understanding of the classifications of the tools. While most companies are up-front in explaining what their tool sets will and will not do, they tend to make the tools sound easier to deploy and maintain than they really are. This means that, as a consumer of the tools, it is important to know and ask the right questions. Start internally by asking yourself where test tools will have the greatest impact on quality improvement.

- Is it in the development environment? Unit testing? String testing?
- Is it in code coverage?
- Is it in system workload? Testing transaction frequency? Transaction volume? Response time?
- Is it in system functionality? Repeatability of frequently run tests?
- Is it in capturing results? Proven performance? Benchmarking? Capturing evidence of completed tests for legal and other purposes?
- Is it in defect tracking?

Knowing where the greatest benefit will come from will point to the class of tools needed.

Debuggers
Debuggers are tools used in the development environment at the unit test and string test levels. They work by allowing the developer to identify break points in the operation of the program so that the execution of the program is interrupted and the output at a specific intermediate point of operation can be checked. The checking requires the user of the tool to be able to interpret the results in machine language and to have the ability to validate the machine language results to the anticipated results. Most often, a baseline run of the program is captured, deemed correct, and then used as a point of comparison whenever any changes to the program are made.

In deciding the benefit level an organization will reach from debugging tools, it is important to consider the programming language, the number of programs, the skill level of the programmers who will be using the tool, and the frequency of planned regression testing. For instance, serious software development companies with very senior programmers who are supporting frequent multiple-releases of maintained programs on multiple platforms will benefit more from the use of
debuggers than a retail operation where development and maintenance is limited to modest updates on a two or five-year schedule.

**Code Coverage Tools**

Coverage tools provide an automated method for identifying what paths have been tested and what paths remain to be tested. These tools work by executing individual test cases and logging the specific paths through the code that are read to generate the output.

Coverage tools decrease edit, compile, and debug times in the development environment by providing a comprehensive picture of what has and has not been tested, how many times a particular line of code has been exercised, when select lines of code have been tested in a block of code, and lines of code that have not been tested by the test cases used. If the test cases are comprehensive, code coverage tools identify code not used by the program at all.

These types of tools — with a tool manufactured by Platinum taking the lead — have been heavily used by organizations in finding how complete the test scenarios are.

**Performance Tools**

Performance tools are used to measure the workload that the system is capable of handling under normal conditions and under worst-case conditions. Here, the term automated test tool is somewhat of an oxymoron. Most of these tools require some degree of programming, referred to as scripts, in UNIX, PERL, or some other proprietary language. The decision to use such tools must consider the trade-offs in training and programming time in relationship to workload benefits.

An example for putting performance testing into perspective is the retail industry. Retailers’ point-of-sale software must offer near-immediate responsiveness with their price scanning, credit card swiping software, and customer look-up applications to provide their customers with acceptable service. The capability of the software to quickly process data scanned simultaneously by several users within each store serviced by the software must be tested to ensure it will handle the normal workload and worst-case volume and frequency demand.

Initially performance tools will help by identifying high-usage bottlenecks and allowing fine-tuning of the software. Eventually, the performance tools can be used to emulate a given number of users and volume and frequency of transactions. Without the aid of performance tools, testing whether the throughput will adequately handle transactions becomes nearly impossible to determine.

Implementing software without sufficient testing carries high risks. Imagine the impact at a retail store on the Friday after Thanksgiving if the
software was sluggish. There would be long lines and very unhappy customers.

The same performance level does not carry the same implications for the back-end software used by retailers to set prices, maintain inventory, track shipments, record returns, complaints, etc. Many price change programs for retail chains are batch jobs that run overnight. There is some time margin allowable. This is not to say that failure of the software to recognize a temporary override of default prices with sale prices on time would not cause some amount of havoc when the store opens on sale day, but it would be a more manageable situation.

The retail example represents the two ends of a line on which it is possible to scale the weight of an organization’s software performance needs from nice-to-have to critical. The point on the scale should dictate the business significance of installing and training personnel to use performance tools. Note that there are many different considerations associated with the testing of Web sites. Thus, the decisions related to use of performance tools for Web sites and Web-based products are handled differently and should not be rated on this type of scale.

Recorders

Test tools that capture and play back keystrokes allowing black box test scenarios to be run and then rerun as needed, are called recorders. Recorders are used to ensure that testing of software functionality is a repeatable process.

There was a time when scripts and cryptic coding needed to be prepared in advance of running recorders. Today, however, most recording test tools have been simplified to the point that users click a record icon, execute a predefined test scenario involving one or more transactions, and click ‘stop’ when the test scenario is complete. The same scenario can then be run over and over again by replacing the recorded script. The first few times the recorder is used, if a bug is identified, both the test scenario and the steps used during the recorder setup must be analyzed to determine if the bug exists in the software or the testing process.

Many organizations today scoff at the need for recorders, opting for manual testing because they do not feel the tests will be rerun a sufficient number of times to offset the seat licensing costs for the tools. According to Boris Beizer in his book, Black-Box Testing, that type of thinking supports a great myth. The myth says manual test execution works. Beizer contends that the error rate of manual testing skews the test results.

The extent of data entry and processing errors is greatly reduced when the test steps are captured, analyzed, and refined, and the same test process can be run each time. Because the test can be rerun automatically, errors caused by a tester forgetting a step or a condition can be controlled and eliminated. In the process, the integrity of all future tests
run is higher — because the recorder will reproduce the exact same test over and over again.

The other benefit of automated recording tools is that the test results are automatically captured and compared with the previously recorded correct answers. Evidence of the test results can be preserved for future reference, for legal matters, and most importantly, for analysis in fixing bugs and upgrading future software releases.

Capturing Results Manually
So what happens when an organization needs to test and a recording test tool is not used to automatically run and capture the test results? It only means that the organization had better have some methods and controls put into place that will ensure that a valid test is run and that results are captured in some other way. One way is to use an inexpensive tool designed to capture and store screen prints, such as Hyper Snap by Hydrionics.

Another is to use a desktop database to link to the application’s database tables and copy the records in all database tables expected to be affected during the test scenario execution. The records must be copied both before and after test execution to provide sufficient information. The copied record can be pasted in a spreadsheet and the spreadsheet can be used to determine if the results are correct. The method is cumbersome, but it is not as time-consuming as it would be to try to prove the test worked without having evidence of the results.

Of course, word processing tools are a must for any testing effort. Not only are word processors needed for the construction of the test scenario, but also for providing a place for documenting test tracking and status.

Defect Tracking
Identifying and documenting defects during the testing effort is a critical part of testing. Tracking the types of defects, attributing them to the correct version of software, having sufficient information to be able to recreate the defect, and being able to report the status of outstanding and closed defects must be supported. Many organizations use home-grown tools to accomplish this. Of all of the methods and tools available for defect tracking, word processors are the least effective for this task. The best approach for a home-grown defect tracking tool is a desktop database application such as Access.

Better than a home-grown Access application are defect tracking tools that are sold with configuration management tools. The reason is simple. Defect tracking tools manufactured to be integrated with configuration management tools are sophisticated enough to help manage the testing of multiple releases, multiple platforms, and multiple versions of custom code. Shrink-wrapped defect tracking tools that are integrated with ver-
sion control and release management tools provide assistance in managing the testing and subsequent releases of the software.

The integration of defect tracking tools with configuration management tools ensures that correct versions of code are tested with the correct release. This also ensures that correct versions of the software are tested and that these versions, when tested, are reported against correctly. When set up correctly, these defect tracking tools can reduce the margin for error in testing the wrong version of a program, and the probability of using the wrong data in a test is reduced.

A bigger problem associated with defect tracking is that many organizations have yet to implement a configuration management program. This issue causes difficulties in the testing process due to insufficient testing of multiple versions of code, and errors in tracking defects from one environment to another and one platform to another.

Finding bugs is the primary purpose of testing. Documenting found bugs is the only way to ensure that the software that caused the bug gets fixed. Most organizations understand this and have a process for defect tracking. Unfortunately, the professional defect tracking tools available in today's market are not widely used.

The value of professionally developed, shrink-wrapped defect tracking tools is their comprehensive recording and extensive reporting features. These tools are designed to meet the needs of varying test groups in all different types of organizations. In some instances, the home-grown defect tool developed for a project even neglects to record the application name. The information captured becomes useless for future reference. Future referencing is a necessary part of continuous process improvement.

Commercially developed defect tracking tools provide a consistent method for defect tracking across software projects. The customization features of a product designed specifically to support defect tracking jog the mind, causing it to think of what needs to be tracked. Organizations do not end up with databases full of defects without knowing which version or release the software defects were identified.

Getting the bugs out of the defect tracking process used during testing efforts is important in the big picture. If each project sets up its own rules for grading defects and recording them, it is much more difficult to improve the testing processes of the company. Commercially available defect tracking tools can be used at all testing levels from debugging through enterprise testing. And, they can be integrated with the configuration management tools at a point in time that an organization decides to install and use one.

GUIDELINES FOR SELECTION
Now that the areas for improvement have been identified and the general classifications of test tools are understood, give each area a score of 1
to 5. Scoring is different from prioritization. The objective of scoring is to obtain a solid understanding of how one or more test tools will increase the value of the product being developed. A possible definition of each score is shown below and in Exhibit 1.

- A score of 5 for a defined area of development and testing indicates that major benefits would result from using a test tool in the area. Major benefits include increased productivity, increased testing without schedule impact, increased number of defects identified, and perhaps a decrease in testing budget over time. Justification to management will be easy, regardless of any initial impacts related to tool implementation.
- A score of 4 indicates that significant improvements in the product would result from using the test tool. These improvements may include all of the benefits for a score of 5 but to a lesser degree, or they may include only one or two of those benefits but to a much greater degree. Justification to management may require a little more work but would be worth it, regardless of any initial impacts related to tool implementation.
- A score of 3 indicates there are identifiable benefits but also reflects one or more downsides to bringing in a test tool in a given area. The downside might include schedule impacts while testers are trained and test tool scripts are being written, or the lack of testing processes within an organization, which would make the switch over to tool use difficult. Justification to management will have to emphasize why bringing a tool in at this point is not just awash in the testing arena.
- A score of 2 indicates there are at least a few benefits, not the least of which may be that the manufacturer is providing a significant discount on the product as long as it is purchased at the same time as other test tool products that have been scored at 4 or higher.
- A score of 1 indicates that implementation of a test tool will not hurt anything even if there are no apparent benefits; and, besides, the manufacturer is throwing the tool in for free.

---

**EXHIBIT 1 — Guidelines for Selection of Testing Tools**

<table>
<thead>
<tr>
<th>Score</th>
<th>Indication</th>
<th>Justification to Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Major benefits result from using the test tool</td>
<td>Easy</td>
</tr>
<tr>
<td>4</td>
<td>Significant improvement in the product from using the test tool</td>
<td>Requires work, but worth the effort regardless of impact related to implementation</td>
</tr>
<tr>
<td>3</td>
<td>Identifiable benefits, but downsides to bringing in the test tool</td>
<td>Must emphasize benefits of bringing in the test tool</td>
</tr>
<tr>
<td>2</td>
<td>Few benefits</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Test tool not needed</td>
<td></td>
</tr>
</tbody>
</table>
• A score of 0 indicates that the tool is not needed, cannot be used, or is already in place. One could not use it even if the manufacturer provides it at no cost.

Using this scoring scheme, if internal evaluation within an organization developing a mainframe application pointed to the areas of performance and functionality as being the areas that would yield the highest return on the investment of one or more test tools, performance tools and recorders would receive a score of 5. In that same organization, defect tracking might already be set up and running with the organization’s configuration management system. Defect tracking would receive a score of 0.

Each organization should use the bulleted scoring definitions or create definitions that better reflect its own needs. What is critical is to evaluate and score the need before contacting the manufacturers. This internal evaluation and scoring will ensure the best investment of corporate dollars.

TOOLS AND TOOL SUITES
Once scoring is complete, it is time to move the evaluation process outside the organization and take a look at vendors. Many companies selling shrink-wrapped test tools offer complete sets of tools, referred to as test suites. Among the leaders in test suite offerings are Mercury Interactive Incorporated, Rational Software Corporation, and Segue. Their tools can be purchased separately or as test suites, and all of the companies have training programs and customer support programs established to ensure maximum benefits from their tools and tool suites. Exhibit 2 shows the classes of testing tools available from these three vendors. Keep in mind that the companies listed allow these tools to be purchased separately or together, and offer additional testing tool products and processes.

Test tool suites are individual test tools integrated to work together in the various areas of testing. For example, Mercury’s Test Director product combines the need for defect tracking with the overall need for managing test scenarios. It does this by:

• providing comprehensive status information about the scenarios
• driving test scenarios using their WinRunner and XRunner recorder tools, allowing tests to be set to run at specific times and under specific conditions

Rational’s SQA Manager performs a similar function using their SQA Robot (to be renamed Rational Robot in the spring of 1999) recorder. Rational’s defect tracking tool is ClearQuest. ClearQuest can be used as a stand-alone tool for defect tracking, or it can be used as an integrated component of Rational’s ClearCase configuration management tool.
Segue's line of Silk test tools includes SilkDeveloper for code coverage, SilkPerformer for performance testing, SilkTest for capture/playback/recorder, and SilkRadar for defect tracking.

For performance testing, Mercury Interactive, Inc. offers a product called LoadRunner, readily recognized as the current leader in client/server environments. Rational offers two products with two different intents for performance testing: Performance Studio is used to test transaction workloads, and Visual Quantify is used to test the execution time and number of calls in an application.

### THE TESTING PROCESS

All of the test tools available through Mercury Interactive, Inc. and Rational Software are designed to support an organization's testing processes. For example, Segue also offers a product called LiveQuality that is composed of a Producer, Realizer, and Delivery designed to provide an organization with a nearly turnkey testing process including tool usage. These companies, along with others such as Micro Focus, will assist an organization in establishing testing processes and deploying test tools to support quality goals throughout the software development life cycle. These processes and the use of tools do make a difference when the organization's needs and goals are evaluated along with the test tools.

---

**EXHIBIT 2 — Test Tools and Test Tool Classifications**

<table>
<thead>
<tr>
<th>Company &amp; Product</th>
<th>Test Tool Classifications</th>
<th>Code Coverage</th>
<th>Performance Recorders</th>
<th>Defect Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mercury Interactive, Inc.</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>LoadRunner</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WinRunner</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xrunner</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Director</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rational Software Corp.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rational Purify</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual PureCoverage</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>VisualQuantify</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance Studio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOA Robot</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SOA Test Manager</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClearQuest</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Segue Corp.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SilkDeveloper</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SilkPerformer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SilkTest</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SilkRadar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LiveQuality</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CONCLUSION
The implementation of test tools in any organization needs to be considered carefully. Just issuing a directive that test tools will be used is not the right road to success and increased product quality. Evaluating the internal testing needs is as important as evaluating the tools themselves. With solid planning and firm commitment, bringing testing tools into the development environment will result in better overall testing and meaningful regression testing at an affordable cost.

Notes

Polly Perryman Kuver has more than 19 years of computer experience, including 12 in management positions. As a process engineer, her areas of expertise are national and international software engineering and documentation standards, quality assurance, configuration management, and data management. Currently, she is a consultant in the Boston, MA, area.